Life is Short

January 2016Life is short, as everyone knows. When I was a kid I used to wonder

about this. Is life actually short, or are we really complaining

about its finiteness? Would we be just as likely to feel life was

short if we lived 10 times as long?Since there didn't seem any way to answer this question, I stopped

wondering about it. Then I had kids. That gave me a way to answer

the question, and the answer is that life actually is short.Having kids showed me how to convert a continuous quantity, time,

into discrete quantities. You only get 52 weekends with your 2 year

old. If Christmas-as-magic lasts from say ages 3 to 10, you only

get to watch your child experience it 8 times. And while it's

impossible to say what is a lot or a little of a continuous quantity

like time, 8 is not a lot of something. If you had a handful of 8

peanuts, or a shelf of 8 books to choose from, the quantity would

definitely seem limited, no matter what your lifespan was.Ok, so life actually is short. Does it make any difference to know

that?It has for me. It means arguments of the form "Life is too short

for x" have great force. It's not just a figure of speech to say

that life is too short for something. It's not just a synonym for

annoying. If you find yourself thinking that life is too short for

something, you should try to eliminate it if you can.When I ask myself what I've found life is too short for, the word

that pops into my head is "bullshit." I realize that answer is

somewhat tautological. It's almost the definition of bullshit that

it's the stuff that life is too short for. And yet bullshit does

have a distinctive character. There's something fake about it.

It's the junk food of experience.

[1]If you ask yourself what you spend your time on that's bullshit,

you probably already know the answer. Unnecessary meetings, pointless

disputes, bureaucracy, posturing, dealing with other people's

mistakes, traffic jams, addictive but unrewarding pastimes.There are two ways this kind of thing gets into your life: it's

either forced on you, or it tricks you. To some extent you have to

put up with the bullshit forced on you by circumstances. You need

to make money, and making money consists mostly of errands. Indeed,

the law of supply and demand insures that: the more rewarding some

kind of work is, the cheaper people will do it. It may be that

less bullshit is forced on you than you think, though. There has

always been a stream of people who opt out of the default grind and

go live somewhere where opportunities are fewer in the conventional

sense, but life feels more authentic. This could become more common.You can do it on a smaller scale without moving. The amount of

time you have to spend on bullshit varies between employers. Most

large organizations (and many small ones) are steeped in it. But

if you consciously prioritize bullshit avoidance over other factors

like money and prestige, you can probably find employers that will

waste less of your time.If you're a freelancer or a small company, you can do this at the

level of individual customers. If you fire or avoid toxic customers,

you can decrease the amount of bullshit in your life by more than

you decrease your income.But while some amount of bullshit is inevitably forced on you, the

bullshit that sneaks into your life by tricking you is no one's

fault but your own. And yet the bullshit you choose may be harder

to eliminate than the bullshit that's forced on you. Things that

lure you into wasting your time have to be really good at

tricking you. An example that will be familiar to a lot of people

is arguing online. When someone

contradicts you, they're in a sense attacking you. Sometimes pretty

overtly. Your instinct when attacked is to defend yourself. But

like a lot of instincts, this one wasn't designed for the world we

now live in. Counterintuitive as it feels, it's better most of

the time not to defend yourself. Otherwise these people are literally

taking your life.

[2]Arguing online is only incidentally addictive. There are more

dangerous things than that. As I've written before, one byproduct

of technical progress is that things we like tend to become more

addictive. Which means we will increasingly have to make a conscious

effort to avoid addictions to stand outside ourselves and ask "is

this how I want to be spending my time?"As well as avoiding bullshit, one should actively seek out things

that matter. But different things matter to different people, and

most have to learn what matters to them. A few are lucky and realize

early on that they love math or taking care of animals or writing,

and then figure out a way to spend a lot of time doing it. But

most people start out with a life that's a mix of things that

matter and things that don't, and only gradually learn to distinguish

between them.For the young especially, much of this confusion is induced by the

artificial situations they find themselves in. In middle school and

high school, what the other kids think of you seems the most important

thing in the world. But when you ask adults what they got wrong

at that age, nearly all say they cared too much what other kids

thought of them.One heuristic for distinguishing stuff that matters is to ask

yourself whether you'll care about it in the future. Fake stuff

that matters usually has a sharp peak of seeming to matter. That's

how it tricks you. The area under the curve is small, but its shape

jabs into your consciousness like a pin.The things that matter aren't necessarily the ones people would

call "important." Having coffee with a friend matters. You won't

feel later like that was a waste of time.One great thing about having small children is that they make you

spend time on things that matter: them. They grab your sleeve as

you're staring at your phone and say "will you play with me?" And

odds are that is in fact the bullshit-minimizing option.If life is short, we should expect its shortness to take us by

surprise. And that is just what tends to happen. You take things

for granted, and then they're gone. You think you can always write

that book, or climb that mountain, or whatever, and then you realize

the window has closed. The saddest windows close when other people

die. Their lives are short too. After my mother died, I wished I'd

spent more time with her. I lived as if she'd always be there.

And in her typical quiet way she encouraged that illusion. But an

illusion it was. I think a lot of people make the same mistake I

did.The usual way to avoid being taken by surprise by something is to

be consciously aware of it. Back when life was more precarious,

people used to be aware of death to a degree that would now seem a

bit morbid. I'm not sure why, but it doesn't seem the right answer

to be constantly reminding oneself of the grim reaper hovering at

everyone's shoulder. Perhaps a better solution is to look at the

problem from the other end. Cultivate a habit of impatience about

the things you most want to do. Don't wait before climbing that

mountain or writing that book or visiting your mother. You don't

need to be constantly reminding yourself why you shouldn't wait.

Just don't wait.I can think of two more things one does when one doesn't have much

of something: try to get more of it, and savor what one has. Both

make sense here.How you live affects how long you live. Most people could do better.

Me among them.But you can probably get even more effect by paying closer attention

to the time you have. It's easy to let the days rush by. The

"flow" that imaginative people love so much has a darker cousin

that prevents you from pausing to savor life amid the daily slurry

of errands and alarms. One of the most striking things I've read

was not in a book, but the title of one: James Salter's Burning

the Days.It is possible to slow time somewhat. I've gotten better at it.

Kids help. When you have small children, there are a lot of moments

so perfect that you can't help noticing.It does help too to feel that you've squeezed everything out of

some experience. The reason I'm sad about my mother is not just

that I miss her but that I think of all the things we could have

done that we didn't. My oldest son will be 7 soon. And while I

miss the 3 year old version of him, I at least don't have any regrets

over what might have been. We had the best time a daddy and a 3

year old ever had.Relentlessly prune bullshit, don't wait to do things that matter,

and savor the time you have. That's what you do when life is short.Notes[1]

At first I didn't like it that the word that came to mind was

one that had other meanings. But then I realized the other meanings

are fairly closely related. Bullshit in the sense of things you

waste your time on is a lot like intellectual bullshit.[2]

I chose this example deliberately as a note to self. I get

attacked a lot online. People tell the craziest lies about me.

And I have so far done a pretty mediocre job of suppressing the

natural human inclination to say "Hey, that's not true!"Thanks to Jessica Livingston and Geoff Ralston for reading drafts

of this.Korean TranslationJapanese TranslationChinese Translation

Do Things that Don't Scale

Want to start a startup? Get funded by

Y Combinator.

July 2013One of the most common types of advice we give at Y Combinator is

to do things that don't scale. A lot of would-be founders believe

that startups either take off or don't. You build something, make

it available, and if you've made a better mousetrap, people beat a

path to your door as promised. Or they don't, in which case the

market must not exist.

[1]Actually startups take off because the founders make them take off.

There may be a handful that just grew by themselves, but usually

it takes some sort of push to get them going. A good metaphor would

be the cranks that car engines had before they got electric starters.

Once the engine was going, it would keep going, but there was a

separate and laborious process to get it going.RecruitThe most common unscalable thing founders have to do at the start

is to recruit users manually. Nearly all startups have to. You

can't wait for users to come to you. You have to go out and get

them.Stripe is one of the most successful startups we've funded, and the

problem they solved was an urgent one. If anyone could have sat

back and waited for users, it was Stripe. But in fact they're

famous within YC for aggressive early user acquisition.Startups building things for other startups have a big pool of

potential users in the other companies we've funded, and none took

better advantage of it than Stripe. At YC we use the term "Collison

installation" for the technique they invented. More diffident

founders ask "Will you try our beta?" and if the answer is yes,

they say "Great, we'll send you a link." But the Collison brothers

weren't going to wait. When anyone agreed to try Stripe they'd say

"Right then, give me your laptop" and set them up on the spot.There are two reasons founders resist going out and recruiting users

individually. One is a combination of shyness and laziness. They'd

rather sit at home writing code than go out and talk to a bunch of

strangers and probably be rejected by most of them. But for a

startup to succeed, at least one founder (usually the CEO) will

have to spend a lot of time on sales and marketing.

[2]The other reason founders ignore this path is that the absolute

numbers seem so small at first. This can't be how the big, famous

startups got started, they think. The mistake they make is to

underestimate the power of compound growth. We encourage every

startup to measure their progress by weekly growth

rate. If you have 100 users, you need to get 10 more next week

to grow 10% a week. And while 110 may not seem much better than

100, if you keep growing at 10% a week you'll be surprised how big

the numbers get. After a year you'll have 14,000 users, and after

2 years you'll have 2 million.You'll be doing different things when you're acquiring users a

thousand at a time, and growth has to slow down eventually. But

if the market exists you can usually start by recruiting users

manually and then gradually switch to less manual methods.

[3]Airbnb is a classic example of this technique. Marketplaces are

so hard to get rolling that you should expect to take heroic measures

at first. In Airbnb's case, these consisted of going door to door

in New York, recruiting new users and helping existing ones improve

their listings. When I remember the Airbnbs during YC, I picture

them with rolly bags, because when they showed up for tuesday dinners

they'd always just flown back from somewhere.FragileAirbnb now seems like an unstoppable juggernaut, but early on it

was so fragile that about 30 days of going out and engaging in

person with users made the difference between success and failure.That initial fragility was not a unique feature of Airbnb. Almost

all startups are fragile initially. And that's one of the biggest

things inexperienced founders and investors (and reporters and

know-it-alls on forums) get wrong about them. They unconsciously

judge larval startups by the standards of established ones. They're

like someone looking at a newborn baby and concluding "there's no

way this tiny creature could ever accomplish anything."It's harmless if reporters and know-it-alls dismiss your startup.

They always get things wrong. It's even ok if investors dismiss

your startup; they'll change their minds when they see growth. The

big danger is that you'll dismiss your startup yourself. I've seen

it happen. I often have to encourage founders who don't see the

full potential of what they're building. Even Bill Gates made that

mistake. He returned to Harvard for the fall semester after starting

Microsoft. He didn't stay long, but he wouldn't have returned at

all if he'd realized Microsoft was going to be even a fraction of

the size it turned out to be.

[4]The question to ask about an early stage startup is not "is this

company taking over the world?" but "how big could this company

get if the founders did the right things?" And the right things

often seem both laborious and inconsequential at the time. Microsoft

can't have seemed very impressive when it was just a couple guys

in Albuquerque writing Basic interpreters for a market of a few

thousand hobbyists (as they were then called), but in retrospect

that was the optimal path to dominating microcomputer software.

And I know Brian Chesky and Joe Gebbia didn't feel like they were

en route to the big time as they were taking "professional" photos

of their first hosts' apartments. They were just trying to survive.

But in retrospect that too was the optimal path to dominating a big

market.How do you find users to recruit manually? If you build something

to solve your own problems, then

you only have to find your peers, which is usually straightforward.

Otherwise you'll have to make a more deliberate effort to locate

the most promising vein of users. The usual way to do that is to

get some initial set of users by doing a comparatively untargeted

launch, and then to observe which kind seem most enthusiastic, and

seek out more like them. For example, Ben Silbermann noticed that

a lot of the earliest Pinterest users were interested in design,

so he went to a conference of design bloggers to recruit users, and

that worked well.

[5]DelightYou should take extraordinary measures not just to acquire users,

but also to make them happy. For as long as they could (which

turned out to be surprisingly long), Wufoo sent each new user a

hand-written thank you note. Your first users should feel that

signing up with you was one of the best choices they ever made.

And you in turn should be racking your brains to think of new ways

to delight them.Why do we have to teach startups this? Why is it counterintuitive

for founders? Three reasons, I think.One is that a lot of startup founders are trained as engineers,

and customer service is not part of the training of engineers.

You're supposed to build things that are robust and elegant, not

be slavishly attentive to individual users like some kind of

salesperson. Ironically, part of the reason engineering is

traditionally averse to handholding is that its traditions date

from a time when engineers were less powerful — when they were

only in charge of their narrow domain of building things, rather

than running the whole show. You can be ornery when you're Scotty,

but not when you're Kirk.Another reason founders don't focus enough on individual customers

is that they worry it won't scale. But when founders of larval

startups worry about this, I point out that in their current state

they have nothing to lose. Maybe if they go out of their way to

make existing users super happy, they'll one day have too many to

do so much for. That would be a great problem to have. See if you

can make it happen. And incidentally, when it does, you'll find

that delighting customers scales better than you expected. Partly

because you can usually find ways to make anything scale more than

you would have predicted, and partly because delighting customers

will by then have permeated your culture.I have never once seen a startup lured down a blind alley by trying

too hard to make their initial users happy.But perhaps the biggest thing preventing founders from realizing

how attentive they could be to their users is that they've never

experienced such attention themselves. Their standards for customer

service have been set by the companies they've been customers of,

which are mostly big ones. Tim Cook doesn't send you a hand-written

note after you buy a laptop. He can't. But you can. That's one

advantage of being small: you can provide a level of service no big

company can.

[6]Once you realize that existing conventions are not the upper bound

on user experience, it's interesting in a very pleasant way to think

about how far you could go to delight your users.ExperienceI was trying to think of a phrase to convey how extreme your attention

to users should be, and I realized Steve Jobs had already done it:

insanely great. Steve wasn't just using "insanely" as a synonym

for "very." He meant it more literally — that one should focus

on quality of execution to a degree that in everyday life would be

considered pathological.All the most successful startups we've funded have, and that probably

doesn't surprise would-be founders. What novice founders don't get

is what insanely great translates to in a larval startup. When

Steve Jobs started using that phrase, Apple was already an established

company. He meant the Mac (and its documentation and even

packaging — such is the nature of obsession) should be insanely

well designed and manufactured. That's not hard for engineers to

grasp. It's just a more extreme version of designing a robust and

elegant product.What founders have a hard time grasping (and Steve himself might

have had a hard time grasping) is what insanely great morphs into

as you roll the time slider back to the first couple months of a

startup's life. It's not the product that should be insanely great,

but the experience of being your user. The product is just one

component of that. For a big company it's necessarily the dominant

one. But you can and should give users an insanely great experience

with an early, incomplete, buggy product, if you make up the

difference with attentiveness.Can, perhaps, but should? Yes. Over-engaging with early users is

not just a permissible technique for getting growth rolling. For

most successful startups it's a necessary part of the feedback loop

that makes the product good. Making a better mousetrap is not an

atomic operation. Even if you start the way most successful startups

have, by building something you yourself need, the first thing you

build is never quite right. And except in domains with big penalties

for making mistakes, it's often better not to aim for perfection

initially. In software, especially, it usually works best to get

something in front of users as soon as it has a quantum of utility,

and then see what they do with it. Perfectionism is often an excuse

for procrastination, and in any case your initial model of users

is always inaccurate, even if you're one of them.

[7]The feedback you get from engaging directly with your earliest users

will be the best you ever get. When you're so big you have to

resort to focus groups, you'll wish you could go over to your users'

homes and offices and watch them use your stuff like you did when

there were only a handful of them.FireSometimes the right unscalable trick is to focus on a deliberately

narrow market. It's like keeping a fire contained at first to get

it really hot before adding more logs.That's what Facebook did. At first it was just for Harvard students.

In that form it only had a potential market of a few thousand people,

but because they felt it was really for them, a critical mass of

them signed up. After Facebook stopped being for Harvard students,

it remained for students at specific colleges for quite a while.

When I interviewed Mark Zuckerberg at Startup School, he said that

while it was a lot of work creating course lists for each school,

doing that made students feel the site was their natural home.Any startup that could be described as a marketplace usually has

to start in a subset of the market, but this can work for other

startups as well. It's always worth asking if there's a subset of

the market in which you can get a critical mass of users quickly.

[8]Most startups that use the contained fire strategy do it unconsciously.

They build something for themselves and their friends, who happen

to be the early adopters, and only realize later that they could

offer it to a broader market. The strategy works just as well if

you do it unconsciously. The biggest danger of not being consciously

aware of this pattern is for those who naively discard part of it.

E.g. if you don't build something for yourself and your friends,

or even if you do, but you come from the corporate world and your

friends are not early adopters, you'll no longer have a perfect

initial market handed to you on a platter.Among companies, the best early adopters are usually other startups.

They're more open to new things both by nature and because, having

just been started, they haven't made all their choices yet. Plus

when they succeed they grow fast, and you with them. It was one

of many unforeseen advantages of the YC model (and specifically of

making YC big) that B2B startups now have an instant market of

hundreds of other startups ready at hand.MerakiFor hardware startups there's a variant of

doing things that don't scale that we call "pulling a Meraki."

Although we didn't fund Meraki, the founders were Robert Morris's

grad students, so we know their history. They got started by doing

something that really doesn't scale: assembling their routers

themselves.Hardware startups face an obstacle that software startups don't.

The minimum order for a factory production run is usually several

hundred thousand dollars. Which can put you in a catch-22: without

a product you can't generate the growth you need to raise the money

to manufacture your product. Back when hardware startups had to

rely on investors for money, you had to be pretty convincing to

overcome this. The arrival of crowdfunding (or more precisely,

preorders) has helped a lot. But even so I'd advise startups to

pull a Meraki initially if they can. That's what Pebble did. The

Pebbles

assembled

the first several hundred watches themselves. If

they hadn't gone through that phase, they probably wouldn't have

sold $10 million worth of watches when they did go on Kickstarter.Like paying excessive attention to early customers, fabricating

things yourself turns out to be valuable for hardware startups.

You can tweak the design faster when you're the factory, and you

learn things you'd never have known otherwise. Eric Migicovsky of

Pebble said one of the things he learned was "how valuable it was to

source good screws." Who knew?ConsultSometimes we advise founders of B2B startups to take over-engagement

to an extreme, and to pick a single user and act as if they were

consultants building something just for that one user. The initial

user serves as the form for your mold; keep tweaking till you fit

their needs perfectly, and you'll usually find you've made something

other users want too. Even if there aren't many of them, there are

probably adjacent territories that have more. As long as you can

find just one user who really needs something and can act on that

need, you've got a toehold in making something people want, and

that's as much as any startup needs initially.

[9]Consulting is the canonical example of work that doesn't scale.

But (like other ways of bestowing one's favors liberally) it's safe

to do it so long as you're not being paid to. That's where companies

cross the line. So long as you're a product company that's merely

being extra attentive to a customer, they're very grateful even if

you don't solve all their problems. But when they start paying you

specifically for that attentiveness — when they start paying

you by the hour — they expect you to do everything.Another consulting-like technique for recruiting initially lukewarm

users is to use your software yourselves on their behalf. We

did that at Viaweb. When we approached merchants asking if they

wanted to use our software to make online stores, some said no, but

they'd let us make one for them. Since we would do anything to get

users, we did. We felt pretty lame at the time. Instead of

organizing big strategic e-commerce partnerships, we were trying

to sell luggage and pens and men's shirts. But in retrospect it

was exactly the right thing to do, because it taught us how it would

feel to merchants to use our software. Sometimes the feedback loop

was near instantaneous: in the middle of building some merchant's

site I'd find I needed a feature we didn't have, so I'd spend a

couple hours implementing it and then resume building the site.ManualThere's a more extreme variant where you don't just use your software,

but are your software. When you only have a small number of users,

you can sometimes get away with doing by hand things that you plan

to automate later. This lets you launch faster, and when you do

finally automate yourself out of the loop, you'll know exactly what

to build because you'll have muscle memory from doing it yourself.When manual components look to the user like software, this technique

starts to have aspects of a practical joke. For example, the way

Stripe delivered "instant" merchant accounts to its first users was

that the founders manually signed them up for traditional merchant

accounts behind the scenes.Some startups could be entirely manual at first. If you can find

someone with a problem that needs solving and you can solve it

manually, go ahead and do that for as long as you can, and then

gradually automate the bottlenecks. It would be a little frightening

to be solving users' problems in a way that wasn't yet automatic,

but less frightening than the far more common case of having something

automatic that doesn't yet solve anyone's problems.BigI should mention one sort of initial tactic that usually doesn't

work: the Big Launch. I occasionally meet founders who seem to

believe startups are projectiles rather than powered aircraft, and

that they'll make it big if and only if they're launched with

sufficient initial velocity. They want to launch simultaneously

in 8 different publications, with embargoes. And on a tuesday, of

course, since they read somewhere that's the optimum day to launch

something.It's easy to see how little launches matter. Think of some successful

startups. How many of their launches do you remember?

All you need from a launch is some initial core of users. How well

you're doing a few months later will depend more on how happy you

made those users than how many there were of them.

[10]So why do founders think launches matter? A combination of solipsism

and laziness. They think what they're building is so great that

everyone who hears about it will immediately sign up. Plus it would

be so much less work if you could get users merely by broadcasting

your existence, rather than recruiting them one at a time. But

even if what you're building really is great, getting users will

always be a gradual process — partly because great things

are usually also novel, but mainly because users have other things

to think about.Partnerships too usually don't work. They don't work for startups

in general, but they especially don't work as a way to get growth

started. It's a common mistake among inexperienced founders to

believe that a partnership with a big company will be their big

break. Six months later they're all saying the same thing: that

was way more work than we expected, and we ended up getting practically

nothing out of it.

[11]It's not enough just to do something extraordinary initially. You

have to make an extraordinary effort initially. Any strategy

that omits the effort — whether it's expecting a big launch to

get you users, or a big partner — is ipso facto suspect.VectorThe need to do something unscalably laborious to get started is so

nearly universal that it might be a good idea to stop thinking of

startup ideas as scalars. Instead we should try thinking of them

as pairs of what you're going to build, plus the unscalable thing(s)

you're going to do initially to get the company going.It could be interesting to start viewing startup ideas this way,

because now that there are two components you can try to be imaginative

about the second as well as the first. But in most cases the second

component will be what it usually is — recruit users manually

and give them an overwhelmingly good experience — and the main

benefit of treating startups as vectors will be to remind founders

they need to work hard in two dimensions.

[12]In the best case, both components of the vector contribute to your

company's DNA: the unscalable things you have to do to get started

are not merely a necessary evil, but change the company permanently

for the better. If you have to be aggressive about user acquisition

when you're small, you'll probably still be aggressive when you're

big. If you have to manufacture your own hardware, or use your

software on users's behalf, you'll learn things you couldn't have

learned otherwise. And most importantly, if you have to work hard

to delight users when you only have a handful of them, you'll keep

doing it when you have a lot.Notes[1]

Actually Emerson never mentioned mousetraps specifically. He

wrote "If a man has good corn or wood, or boards, or pigs, to sell,

or can make better chairs or knives, crucibles or church organs,

than anybody else, you will find a broad hard-beaten road to his

house, though it be in the woods."[2]

Thanks to Sam Altman for suggesting I make this explicit.

And no, you can't avoid doing sales by hiring someone to do it for

you. You have to do sales yourself initially. Later you can hire

a real salesperson to replace you.[3]

The reason this works is that as you get bigger, your size

helps you grow. Patrick Collison wrote "At some point, there was

a very noticeable change in how Stripe felt. It tipped from being

this boulder we had to push to being a train car that in fact had

its own momentum."[4]

One of the more subtle ways in which YC can help founders

is by calibrating their ambitions, because we know exactly how a

lot of successful startups looked when they were just getting

started.[5]

If you're building something for which you can't easily get

a small set of users to observe — e.g. enterprise software — and

in a domain where you have no connections, you'll have to rely on

cold calls and introductions. But should you even be working on

such an idea?[6]

Garry Tan pointed out an interesting trap founders fall into

in the beginning. They want so much to seem big that they imitate

even the flaws of big companies, like indifference to individual

users. This seems to them more "professional." Actually it's

better to embrace the fact that you're small and use whatever

advantages that brings.[7]

Your user model almost couldn't be perfectly accurate, because

users' needs often change in response to what you build for them.

Build them a microcomputer, and suddenly they need to run spreadsheets

on it, because the arrival of your new microcomputer causes someone

to invent the spreadsheet.[8]

If you have to choose between the subset that will sign up

quickest and those that will pay the most, it's usually best to

pick the former, because those are probably the early adopters.

They'll have a better influence on your product, and they won't

make you expend as much effort on sales. And though they have less

money, you don't need that much to maintain your target growth rate

early on.[9]

Yes, I can imagine cases where you could end up making

something that was really only useful for one user. But those are

usually obvious, even to inexperienced founders. So if it's not

obvious you'd be making something for a market of one, don't worry

about that danger.[10]

There may even be an inverse correlation between launch

magnitude and success. The only launches I remember are famous

flops like the Segway and Google Wave. Wave is a particularly

alarming example, because I think it was actually a great idea that

was killed partly by its overdone launch.[11]

Google grew big on the back of Yahoo, but that wasn't a

partnership. Yahoo was their customer.[12]

It will also remind founders that an idea where the second

component is empty — an idea where there is nothing you can do

to get going, e.g. because you have no way to find users to recruit

manually — is probably a bad idea, at least for those founders.Thanks to Sam Altman, Paul Buchheit, Patrick Collison, Kevin

Hale, Steven Levy, Jessica Livingston, Geoff Ralston, and Garry Tan for reading

drafts of this.Japanese TranslationRussian TranslationFrench TranslationArabic TranslationItalian Translation

The Refragmentation

January 2016One advantage of being old is that you can see change happen in

your lifetime. A lot of the change I've seen is fragmentation. US

politics is much more polarized than it used to be. Culturally we

have ever less common ground. The creative class flocks to a handful

of happy cities, abandoning the rest. And increasing economic

inequality means the spread between rich and poor is growing too.

I'd like to propose a hypothesis: that all these trends are instances

of the same phenomenon. And moreover, that the cause is not some

force that's pulling us apart, but rather the erosion of forces

that had been pushing us together.Worse still, for those who worry about these trends, the forces

that were pushing us together were an anomaly, a one-time combination

of circumstances that's unlikely to be repeated — and indeed, that

we would not want to repeat.The two forces were war (above all World War II), and the rise of

large corporations.The effects of World War II were both economic and social.

Economically, it decreased variation in income. Like all modern

armed forces, America's were socialist economically. From each

according to his ability, to each according to his need. More or

less. Higher ranking members of the military got more (as higher

ranking members of socialist societies always do), but what they

got was fixed according to their rank. And the flattening effect

wasn't limited to those under arms, because the US economy was

conscripted too. Between 1942 and 1945 all wages were set by the

National War Labor Board. Like the military, they defaulted to

flatness. And this national standardization of wages was so pervasive

that its effects could still be seen years after the war ended.

[1]Business owners weren't supposed to be making money either. FDR

said "not a single war millionaire" would be permitted. To ensure

that, any increase in a company's profits over prewar levels was

taxed at 85%. And when what was left after corporate taxes reached

individuals, it was taxed again at a marginal rate of 93%.

[2]Socially too the war tended to decrease variation. Over 16 million

men and women from all sorts of different backgrounds were brought

together in a way of life that was literally uniform. Service rates

for men born in the early 1920s approached 80%. And working toward

a common goal, often under stress, brought them still closer together.Though strictly speaking World War II lasted less than 4 years for

the US, its effects lasted longer. Wars make central governments

more powerful, and World War II was an extreme case of this. In

the US, as in all the other Allied countries, the federal government

was slow to give up the new powers it had acquired. Indeed, in

some respects the war didn't end in 1945; the enemy just switched

to the Soviet Union. In tax rates, federal power, defense spending,

conscription, and nationalism, the decades after the war looked more

like wartime than prewar peacetime.

[3]

And the social effects

lasted too. The kid pulled into the army from behind a mule team

in West Virginia didn't simply go back to the farm afterward.

Something else was waiting for him, something that looked a lot

like the army.If total war was the big political story of the 20th century, the

big economic story was the rise of a new kind of company. And this

too tended to produce both social and economic cohesion.

[4]The 20th century was the century of the big, national corporation.

General Electric, General Foods, General Motors. Developments in

finance, communications, transportation, and manufacturing enabled

a new type of company whose goal was above all scale. Version 1

of this world was low-res: a Duplo world of a few giant companies

dominating each big market.

[5]The late 19th and early 20th centuries had been a time of consolidation,

led especially by J. P. Morgan. Thousands of companies run by their

founders were merged into a couple hundred giant ones run by

professional managers. Economies of scale ruled the day. It seemed

to people at the time that this was the final state of things. John

D. Rockefeller said in 1880

The day of combination is here to stay. Individualism has gone,

never to return.

He turned out to be mistaken, but he seemed right for the next

hundred years.The consolidation that began in the late 19th century continued for

most of the 20th. By the end of World War II, as Michael Lind

writes, "the major sectors of the economy were either organized

as government-backed cartels or dominated by a few oligopolistic

corporations."For consumers this new world meant the same choices everywhere, but

only a few of them. When I grew up there were only 2 or 3 of most

things, and since they were all aiming at the middle of the market

there wasn't much to differentiate them.One of the most important instances of this phenomenon was in TV.

Here there were 3 choices: NBC, CBS, and ABC. Plus public TV for

eggheads and communists. The programs that the 3 networks offered were

indistinguishable. In fact, here there was a triple pressure toward

the center. If one show did try something daring, local affiliates

in conservative markets would make them stop. Plus since TVs were

expensive, whole families watched the same shows together, so they

had to be suitable for everyone.And not only did everyone get the same thing, they got it at the

same time. It's difficult to imagine now, but every night tens of

millions of families would sit down together in front of their TV

set watching the same show, at the same time, as their next door

neighbors. What happens now with the Super Bowl used to happen

every night. We were literally in sync.

[6]In a way mid-century TV culture was good. The view it gave of the

world was like you'd find in a children's book, and it probably had

something of the effect that (parents hope) children's books have

in making people behave better. But, like children's books, TV was

also misleading. Dangerously misleading, for adults. In his

autobiography, Robert MacNeil talks of seeing gruesome images that

had just come in from Vietnam and thinking, we can't show these to

families while they're having dinner.I know how pervasive the common culture was, because I tried to opt

out of it, and it was practically impossible to find alternatives.

When I was 13 I realized, more from internal evidence than any

outside source, that the ideas we were being fed on TV were crap,

and I stopped watching it.

[7]

But it wasn't just TV. It seemed

like everything around me was crap. The politicians all saying the

same things, the consumer brands making almost identical products

with different labels stuck on to indicate how prestigious they

were meant to be, the balloon-frame houses with fake "colonial"

skins, the cars with several feet of gratuitous metal on each end

that started to fall apart after a couple years, the "red delicious"

apples that were red but only nominally

apples. And in retrospect, it was crap.

[8]But when I went looking for alternatives to fill this void, I found

practically nothing. There was no Internet then. The only place

to look was in the chain bookstore in our local shopping mall.

[9]

There I found a copy of The Atlantic. I wish I could say it became

a gateway into a wider world, but in fact I found it boring and

incomprehensible. Like a kid tasting whisky for the first time and

pretending to like it, I preserved that magazine as carefully as

if it had been a book. I'm sure I still have it somewhere. But

though it was evidence that there was, somewhere, a world that

wasn't red delicious, I didn't find it till college.It wasn't just as consumers that the big companies made us similar.

They did as employers too. Within companies there were powerful

forces pushing people toward a single model of how to look and act.

IBM was particularly notorious for this, but they were only a little

more extreme than other big companies. And the models of how to

look and act varied little between companies. Meaning everyone

within this world was expected to seem more or less the same. And

not just those in the corporate world, but also everyone who aspired

to it — which in the middle of the 20th century meant most people

who weren't already in it. For most of the 20th century, working-class

people tried hard to look middle class. You can see it in old

photos. Few adults aspired to look dangerous in 1950.But the rise of national corporations didn't just compress us

culturally. It compressed us economically too, and on both ends.Along with giant national corporations, we got giant national labor

unions. And in the mid 20th century the corporations cut deals

with the unions where they paid over market price for labor. Partly

because the unions were monopolies.

[10]

Partly because, as

components of oligopolies themselves, the corporations knew they

could safely pass the cost on to their customers, because their

competitors would have to as well. And partly because in mid-century

most of the giant companies were still focused on finding new ways

to milk economies of scale. Just as startups rightly pay AWS a

premium over the cost of running their own servers so they can focus

on growth, many of the big national corporations were willing to

pay a premium for labor.

[11]As well as pushing incomes up from the bottom, by overpaying unions,

the big companies of the 20th century also pushed incomes down at

the top, by underpaying their top management. Economist J. K.

Galbraith wrote in 1967 that "There are few corporations in which

it would be suggested that executive salaries are at a maximum."

[12]To some extent this was an illusion. Much of the de facto pay of

executives never showed up on their income tax returns, because it

took the form of perks. The higher the rate of income tax, the

more pressure there was to pay employees upstream of it. (In the

UK, where taxes were even higher than in the US, companies would

even pay their kids' private school tuitions.) One of the most

valuable things the big companies of the mid 20th century gave their

employees was job security, and this too didn't show up in tax

returns or income statistics. So the nature of employment in these

organizations tended to yield falsely low numbers about economic

inequality. But even accounting for that, the big companies paid

their best people less than market price. There was no market; the

expectation was that you'd work for the same company for decades

if not your whole career.

[13]Your work was so illiquid there was little chance of getting market

price. But that same illiquidity also encouraged you not to seek

it. If the company promised to employ you till you retired and

give you a pension afterward, you didn't want to extract as much

from it this year as you could. You needed to take care of the

company so it could take care of you. Especially when you'd been

working with the same group of people for decades. If you tried

to squeeze the company for more money, you were squeezing the

organization that was going to take care of them. Plus if

you didn't put the company first you wouldn't be promoted, and if

you couldn't switch ladders, promotion on this one was the only way

up.

[14]To someone who'd spent several formative years in the armed forces,

this situation didn't seem as strange as it does to us now. From

their point of view, as big company executives, they were high-ranking

officers. They got paid a lot more than privates. They got to

have expense account lunches at the best restaurants and fly around

on the company's Gulfstreams. It probably didn't occur to most of

them to ask if they were being paid market price.The ultimate way to get market price is to work for yourself, by

starting your own company. That seems obvious to any ambitious

person now. But in the mid 20th century it was an alien concept.

Not because starting one's own company seemed too ambitious, but

because it didn't seem ambitious enough. Even as late as the 1970s,

when I grew up, the ambitious plan was to get lots of education at

prestigious institutions, and then join some other prestigious

institution and work one's way up the hierarchy. Your prestige was

the prestige of the institution you belonged to. People did start

their own businesses of course, but educated people rarely did,

because in those days there was practically zero concept of starting

what we now call a startup:

a business that starts small and grows

big. That was much harder to do in the mid 20th century. Starting

one's own business meant starting a business that would start small

and stay small. Which in those days of big companies often meant

scurrying around trying to avoid being trampled by elephants. It

was more prestigious to be one of the executive class riding the

elephant.By the 1970s, no one stopped to wonder where the big prestigious

companies had come from in the first place. It seemed like they'd

always been there, like the chemical elements. And indeed, there

was a double wall between ambitious kids in the 20th century and

the origins of the big companies. Many of the big companies were

roll-ups that didn't have clear founders. And when they did, the

founders didn't seem like us. Nearly all of them had been uneducated,

in the sense of not having been to college. They were what Shakespeare

called rude mechanicals. College trained one to be a member of the

professional classes. Its graduates didn't expect to do the sort

of grubby menial work that Andrew Carnegie or Henry Ford started

out doing.

[15]And in the 20th century there were more and more college graduates.

They increased from about 2% of the population in 1900 to about 25%

in 2000. In the middle of the century our two big forces intersect,

in the form of the GI Bill, which sent 2.2 million World War II

veterans to college. Few thought of it in these terms, but the

result of making college the canonical path for the ambitious was

a world in which it was socially acceptable to work for Henry Ford,

but not to be Henry Ford.

[16]I remember this world well. I came of age just as it was starting

to break up. In my childhood it was still dominant. Not quite so

dominant as it had been. We could see from old TV shows and yearbooks

and the way adults acted that people in the 1950s and 60s had been

even more conformist than us. The mid-century model was already

starting to get old. But that was not how we saw it at the time.

We would at most have said that one could be a bit more daring in

1975 than 1965. And indeed, things hadn't changed much yet.But change was coming soon. And when the Duplo economy started to

disintegrate, it disintegrated in several different ways at once.

Vertically integrated companies literally dis-integrated because

it was more efficient to. Incumbents faced new competitors as (a)

markets went global and (b) technical innovation started to trump

economies of scale, turning size from an asset into a liability.

Smaller companies were increasingly able to survive as formerly

narrow channels to consumers broadened. Markets themselves started

to change faster, as whole new categories of products appeared. And

last but not least, the federal government, which had previously

smiled upon J. P. Morgan's world as the natural state of things,

began to realize it wasn't the last word after all.What J. P. Morgan was to the horizontal axis, Henry Ford was to the

vertical. He wanted to do everything himself. The giant plant he

built at River Rouge between 1917 and 1928 literally took in iron

ore at one end and sent cars out the other. 100,000 people worked

there. At the time it seemed the future. But that is not how car

companies operate today. Now much of the design and manufacturing

happens in a long supply chain, whose products the car companies

ultimately assemble and sell. The reason car companies operate

this way is that it works better. Each company in the supply chain

focuses on what they know best. And they each have to do it well

or they can be swapped out for another supplier.Why didn't Henry Ford realize that networks of cooperating companies

work better than a single big company? One reason is that supplier

networks take a while to evolve. In 1917, doing everything himself

seemed to Ford the only way to get the scale he needed. And the

second reason is that if you want to solve a problem using a network

of cooperating companies, you have to be able to coordinate their

efforts, and you can do that much better with computers. Computers

reduce the transaction costs that Coase argued are the raison d'etre

of corporations. That is a fundamental change.In the early 20th century, big companies were synonymous with

efficiency. In the late 20th century they were synonymous with

inefficiency. To some extent this was because the companies

themselves had become sclerotic. But it was also because our

standards were higher.It wasn't just within existing industries that change occurred.

The industries themselves changed. It became possible to make lots

of new things, and sometimes the existing companies weren't the

ones who did it best.Microcomputers are a classic example. The market was pioneered by

upstarts like Apple. When it got big enough, IBM decided it was

worth paying attention to. At the time IBM completely dominated

the computer industry. They assumed that all they had to do, now

that this market was ripe, was to reach out and pick it. Most

people at the time would have agreed with them. But what happened

next illustrated how much more complicated the world had become.

IBM did launch a microcomputer. Though quite successful, it did

not crush Apple. But even more importantly, IBM itself ended up

being supplanted by a supplier coming in from the side — from

software, which didn't even seem to be the same business. IBM's

big mistake was to accept a non-exclusive license for DOS. It must

have seemed a safe move at the time. No other computer manufacturer

had ever been able to outsell them. What difference did it make if

other manufacturers could offer DOS too? The result of that

miscalculation was an explosion of inexpensive PC clones. Microsoft

now owned the PC standard, and the customer. And the microcomputer

business ended up being Apple vs Microsoft.Basically, Apple bumped IBM and then Microsoft stole its wallet.

That sort of thing did not happen to big companies in mid-century.

But it was going to happen increasingly often in the future.Change happened mostly by itself in the computer business. In other

industries, legal obstacles had to be removed first. Many of the

mid-century oligopolies had been anointed by the federal government

with policies (and in wartime, large orders) that kept out competitors.

This didn't seem as dubious to government officials at the time as

it sounds to us. They felt a two-party system ensured sufficient

competition in politics. It ought to work for business too.Gradually the government realized that anti-competitive policies

were doing more harm than good, and during the Carter administration

it started to remove them. The word used for this process was

misleadingly narrow: deregulation. What was really happening was

de-oligopolization. It happened to one industry after another.

Two of the most visible to consumers were air travel and long-distance

phone service, which both became dramatically cheaper after

deregulation.Deregulation also contributed to the wave of hostile takeovers in

the 1980s. In the old days the only limit on the inefficiency of

companies, short of actual bankruptcy, was the inefficiency of their

competitors. Now companies had to face absolute rather than relative

standards. Any public company that didn't generate sufficient

returns on its assets risked having its management replaced with

one that would. Often the new managers did this by breaking companies

up into components that were more valuable separately.

[17]Version 1 of the national economy consisted of a few big blocks

whose relationships were negotiated in back rooms by a handful of

executives, politicians, regulators, and labor leaders. Version 2

was higher resolution: there were more companies, of more different

sizes, making more different things, and their relationships changed

faster. In this world there were still plenty of back room negotiations,

but more was left to market forces. Which further accelerated the

fragmentation.It's a little misleading to talk of versions when describing a

gradual process, but not as misleading as it might seem. There was

a lot of change in a few decades, and what we ended up with was

qualitatively different. The companies in the S&P 500 in 1958 had

been there an average of 61 years. By 2012 that number was 18 years.

[18]The breakup of the Duplo economy happened simultaneously with the

spread of computing power. To what extent were computers a precondition?

It would take a book to answer that. Obviously the spread of computing

power was a precondition for the rise of startups. I suspect it

was for most of what happened in finance too. But was it a

precondition for globalization or the LBO wave? I don't know, but

I wouldn't discount the possibility. It may be that the refragmentation

was driven by computers in the way the industrial revolution was

driven by steam engines. Whether or not computers were a precondition,

they have certainly accelerated it.The new fluidity of companies changed people's relationships with

their employers. Why climb a corporate ladder that might be yanked

out from under you? Ambitious people started to think of a career

less as climbing a single ladder than as a series of jobs that might

be at different companies. More movement (or even potential movement)

between companies introduced more competition in salaries. Plus

as companies became smaller it became easier to estimate how much

an employee contributed to the company's revenue. Both changes

drove salaries toward market price. And since people vary dramatically

in productivity, paying market price meant salaries started to

diverge.By no coincidence it was in the early 1980s that the term "yuppie"

was coined. That word is not much used now, because the phenomenon

it describes is so taken for granted, but at the time it was a label

for something novel. Yuppies were young professionals who made lots

of money. To someone in their twenties today, this wouldn't seem

worth naming. Why wouldn't young professionals make lots of money?

But until the 1980s, being underpaid early in your career was part

of what it meant to be a professional. Young professionals were

paying their dues, working their way up the ladder. The rewards

would come later. What was novel about yuppies was that they wanted

market price for the work they were doing now.The first yuppies did not work for startups. That was still in the

future. Nor did they work for big companies. They were professionals

working in fields like law, finance, and consulting. But their example

rapidly inspired their peers. Once they saw that new BMW 325i, they

wanted one too.Underpaying people at the beginning of their career only works if

everyone does it. Once some employer breaks ranks, everyone else

has to, or they can't get good people. And once started this process

spreads through the whole economy, because at the beginnings of

people's careers they can easily switch not merely employers but

industries.But not all young professionals benefitted. You had to produce to

get paid a lot. It was no coincidence that the first yuppies worked

in fields where it was easy to measure that.More generally, an idea was returning whose name sounds old-fashioned

precisely because it was so rare for so long: that you could make

your fortune. As in the past there were multiple ways to do it.

Some made their fortunes by creating wealth, and others by playing

zero-sum games. But once it became possible to make one's fortune,

the ambitious had to decide whether or not to. A physicist who

chose physics over Wall Street in 1990 was making a sacrifice that

a physicist in 1960 didn't have to think about.The idea even flowed back into big companies. CEOs of big companies

make more now than they used to, and I think much of the reason is

prestige. In 1960, corporate CEOs had immense prestige. They were

the winners of the only economic game in town. But if they made as

little now as they did then, in real dollar terms, they'd seem like

small fry compared to professional athletes and whiz kids making

millions from startups and hedge funds. They don't like that idea,

so now they try to get as much as they can, which is more than they

had been getting.

[19]Meanwhile a similar fragmentation was happening at the other end

of the economic scale. As big companies' oligopolies became less

secure, they were less able to pass costs on to customers and thus

less willing to overpay for labor. And as the Duplo world of a few

big blocks fragmented into many companies of different sizes — some

of them overseas — it became harder for unions to enforce their

monopolies. As a result workers' wages also tended toward market

price. Which (inevitably, if unions had been doing their job) tended

to be lower. Perhaps dramatically so, if automation had decreased

the need for some kind of work.And just as the mid-century model induced social as well as economic

cohesion, its breakup brought social as well as economic fragmentation.

People started to dress and act differently. Those who would later

be called the "creative class" became more mobile. People who didn't

care much for religion felt less pressure to go to church for

appearances' sake, while those who liked it a lot opted for

increasingly colorful forms. Some switched from meat loaf to tofu,

and others to Hot Pockets. Some switched from driving Ford sedans

to driving small imported cars, and others to driving SUVs. Kids

who went to private schools or wished they did started to dress

"preppy," and kids who wanted to seem rebellious made a conscious

effort to look disreputable. In a hundred ways people spread apart.

[20]Almost four decades later, fragmentation is still increasing. Has

it been net good or bad? I don't know; the question may be

unanswerable. Not entirely bad though. We take for granted the

forms of fragmentation we like, and worry only about the ones we

don't. But as someone who caught the tail end of mid-century

conformism,

I can tell you it was no utopia.

[21]My goal here is not to say whether fragmentation has been good or

bad, just to explain why it's happening. With the centripetal

forces of total war and 20th century oligopoly mostly gone, what

will happen next? And more specifically, is it possible to reverse

some of the fragmentation we've seen?If it is, it will have to happen piecemeal. You can't reproduce

mid-century cohesion the way it was originally produced. It would

be insane to go to war just to induce more national unity. And

once you understand the degree to which the economic history of the

20th century was a low-res version 1, it's clear you can't reproduce

that either.20th century cohesion was something that happened at least in a

sense naturally. The war was due mostly to external forces, and

the Duplo economy was an evolutionary phase. If you want cohesion

now, you'd have to induce it deliberately. And it's not obvious

how. I suspect the best we'll be able to do is address the symptoms

of fragmentation. But that may be enough.The form of fragmentation people worry most about lately is economic inequality, and if you want to eliminate

that you're up against a truly formidable headwind — one that has

been in operation since the stone age: technology.Technology is

a lever. It magnifies work. And the lever not only grows increasingly

long, but the rate at which it grows is itself increasing.Which in turn means the variation in the amount of wealth people

can create has not only been increasing, but accelerating. The

unusual conditions that prevailed in the mid 20th century masked

this underlying trend. The ambitious had little choice but to join

large organizations that made them march in step with lots of other

people — literally in the case of the armed forces, figuratively

in the case of big corporations. Even if the big corporations had

wanted to pay people proportionate to their value, they couldn't

have figured out how. But that constraint has gone now. Ever since

it started to erode in the 1970s, we've seen the underlying forces

at work again.

[22]Not everyone who gets rich now does it by creating wealth, certainly.

But a significant number do, and the Baumol Effect means all their

peers get dragged along too.

[23]

And as long as it's possible to

get rich by creating wealth, the default tendency will be for

economic inequality to increase. Even if you eliminate all the

other ways to get rich. You can mitigate this with subsidies at

the bottom and taxes at the top, but unless taxes are high enough

to discourage people from creating wealth, you're always going to

be fighting a losing battle against increasing variation in

productivity.

[24]That form of fragmentation, like the others, is here to stay. Or

rather, back to stay. Nothing is forever, but the tendency toward

fragmentation should be more forever than most things, precisely

because it's not due to any particular cause. It's simply a reversion

to the mean. When Rockefeller said individualism was gone, he was

right for a hundred years. It's back now, and that's likely to be

true for longer.I worry that if we don't acknowledge this, we're headed for trouble.

If we think 20th century cohesion disappeared because of few policy

tweaks, we'll be deluded into thinking we can get it back (minus

the bad parts, somehow) with a few countertweaks. And then we'll

waste our time trying to eliminate fragmentation, when we'd be

better off thinking about how to mitigate its consequences.

Notes[1]

Lester Thurow, writing in 1975, said the wage differentials

prevailing at the end of World War II had become so embedded that

they "were regarded as 'just' even after the egalitarian pressures

of World War II had disappeared. Basically, the same differentials

exist to this day, thirty years later." But Goldin and Margo think

market forces in the postwar period also helped preserve the wartime

compression of wages — specifically increased demand for unskilled

workers, and oversupply of educated ones.(Oddly enough, the American custom of having employers pay for

health insurance derives from efforts by businesses to circumvent

NWLB wage controls in order to attract workers.)[2]

As always, tax rates don't tell the whole story. There were

lots of exemptions, especially for individuals. And in World War

II the tax codes were so new that the government had little acquired

immunity to tax avoidance. If the rich paid high taxes during the

war it was more because they wanted to than because they had to.After the war, federal tax receipts as a percentage of GDP were

about the same as they are now. In fact, for the entire period since

the war, tax receipts have stayed close to 18% of GDP, despite

dramatic changes in tax rates. The lowest point occurred when

marginal income tax rates were highest: 14.1% in 1950. Looking at

the data, it's hard to avoid the conclusion that tax rates have had

little effect on what people actually paid.[3]

Though in fact the decade preceding the war had been a time

of unprecedented federal power, in response to the Depression.

Which is not entirely a coincidence, because the Depression was one

of the causes of the war. In many ways the New Deal was a sort of

dress rehearsal for the measures the federal government took during

wartime. The wartime versions were much more drastic and more

pervasive though. As Anthony Badger wrote, "for many Americans the

decisive change in their experiences came not with the New Deal but

with World War II."[4]

I don't know enough about the origins of the world wars to

say, but it's not inconceivable they were connected to the rise of

big corporations. If that were the case, 20th century cohesion would

have a single cause.[5]

More precisely, there was a bimodal economy consisting, in

Galbraith's words, of "the world of the technically dynamic, massively

capitalized and highly organized corporations on the one hand and

the hundreds of thousands of small and traditional proprietors on

the other." Money, prestige, and power were concentrated in the

former, and there was near zero crossover.[6]

I wonder how much of the decline in families eating together

was due to the decline in families watching TV together afterward.[7]

I know when this happened because it was the season Dallas

premiered. Everyone else was talking about what was happening on

Dallas, and I had no idea what they meant.[8]

I didn't realize it till I started doing research for this

essay, but the meretriciousness of the products I grew up with is

a well-known byproduct of oligopoly. When companies can't compete

on price, they compete on tailfins.[9]

Monroeville Mall was at the time of its completion in 1969

the largest in the country. In the late 1970s the movie Dawn of

the Dead was shot there. Apparently the mall was not just the

location of the movie, but its inspiration; the crowds of shoppers

drifting through this huge mall reminded George Romero of zombies.

My first job was scooping ice cream in the Baskin-Robbins.[10]

Labor unions were exempted from antitrust laws by the Clayton

Antitrust Act in 1914 on the grounds that a person's work is not

"a commodity or article of commerce." I wonder if that means service

companies are also exempt.[11]

The relationships between unions and unionized companies can

even be symbiotic, because unions will exert political pressure to

protect their hosts. According to Michael Lind, when politicians

tried to attack the A&P supermarket chain because it was putting

local grocery stores out of business, "A&P successfully defended

itself by allowing the unionization of its workforce in 1938, thereby

gaining organized labor as a constituency." I've seen this phenomenon

myself: hotel unions are responsible for more of the political

pressure against Airbnb than hotel companies.[12]

Galbraith was clearly puzzled that corporate executives would

work so hard to make money for other people (the shareholders)

instead of themselves. He devoted much of The New Industrial

State to trying to figure this out.His theory was that professionalism had replaced money as a motive,

and that modern corporate executives were, like (good) scientists,

motivated less by financial rewards than by the desire to do good

work and thereby earn the respect of their peers. There is something

in this, though I think lack of movement between companies combined

with self-interest explains much of observed behavior.[13]

Galbraith (p. 94) says a 1952 study of the 800 highest paid

executives at 300 big corporations found that three quarters of

them had been with their company for more than 20 years.[14]

It seems likely that in the first third of the 20th century

executive salaries were low partly because companies then were more

dependent on banks, who would have disapproved if executives got

too much. This was certainly true in the beginning. The first big

company CEOs were J. P. Morgan's hired hands.Companies didn't start to finance themselves with retained earnings

till the 1920s. Till then they had to pay out their earnings in

dividends, and so depended on banks for capital for expansion.

Bankers continued to sit on corporate boards till the Glass-Steagall

act in 1933.By mid-century big companies funded 3/4 of their growth from earnings.

But the early years of bank dependence, reinforced by the financial

controls of World War II, must have had a big effect on social

conventions about executive salaries. So it may be that the lack

of movement between companies was as much the effect of low salaries

as the cause.Incidentally, the switch in the 1920s to financing growth with

retained earnings was one cause of the 1929 crash. The banks now

had to find someone else to lend to, so they made more margin loans.[15]

Even now it's hard to get them to. One of the things I find

hardest to get into the heads of would-be startup founders is how

important it is to do certain kinds of menial work early in the

life of a company. Doing things that don't

scale is to how Henry Ford got started as a high-fiber diet is

to the traditional peasant's diet: they had no choice but to do the

right thing, while we have to make a conscious effort.[16]

Founders weren't celebrated in the press when I was a kid.

"Our founder" meant a photograph of a severe-looking man with a

walrus mustache and a wing collar who had died decades ago. The

thing to be when I was a kid was an executive. If you weren't

around then it's hard to grasp the cachet that term had. The fancy

version of everything was called the "executive" model.[17]

The wave of hostile takeovers in the 1980s was enabled by a

combination of circumstances: court decisions striking down state

anti-takeover laws, starting with the Supreme Court's 1982 decision

in Edgar v. MITE Corp.; the Reagan administration's comparatively

sympathetic attitude toward takeovers; the Depository Institutions

Act of 1982, which allowed banks and savings and loans to buy

corporate bonds; a new SEC rule issued in 1982 (rule 415) that made

it possible to bring corporate bonds to market faster; the creation

of the junk bond business by Michael Milken; a vogue for conglomerates

in the preceding period that caused many companies to be combined

that never should have been; a decade of inflation that left many

public companies trading below the value of their assets; and not

least, the increasing complacency of managements.[18]

Foster, Richard. "Creative Destruction Whips through Corporate

America." Innosight, February 2012.[19]

CEOs of big companies may be overpaid. I don't know enough

about big companies to say. But it is certainly not impossible for

a CEO to make 200x as much difference to a company's revenues as

the average employee. Look at what Steve Jobs did for Apple when

he came back as CEO. It would have been a good deal for the board

to give him 95% of the company. Apple's market cap the day Steve

came back in July 1997 was 1.73 billion. 5% of Apple now (January

2016) would be worth about 30 billion. And it would not be if Steve

hadn't come back; Apple probably wouldn't even exist anymore.Merely including Steve in the sample might be enough to answer the

question of whether public company CEOs in the aggregate are overpaid.

And that is not as facile a trick as it might seem, because the

broader your holdings, the more the aggregate is what you care

about.[20]

The late 1960s were famous for social upheaval. But that was

more rebellion (which can happen in any era if people are provoked

sufficiently) than fragmentation. You're not seeing fragmentation

unless you see people breaking off to both left and right.[21]

Globally the trend has been in the other direction. While

the US is becoming more fragmented, the world as a whole is becoming

less fragmented, and mostly in good ways.[22]

There were a handful of ways to make a fortune in the mid

20th century. The main one was drilling for oil, which was open

to newcomers because it was not something big companies could

dominate through economies of scale. How did individuals accumulate

large fortunes in an era of such high taxes? Giant tax loopholes

defended by two of the most powerful men in Congress, Sam Rayburn

and Lyndon Johnson.But becoming a Texas oilman was not in 1950 something one could

aspire to the way starting a startup or going to work on Wall Street

were in 2000, because (a) there was a strong local component and

(b) success depended so much on luck.[23]

The Baumol Effect induced by startups is very visible in

Silicon Valley. Google will pay people millions of dollars a year

to keep them from leaving to start or join startups.[24]

I'm not claiming variation in productivity is the only cause

of economic inequality in the US. But it's a significant cause, and

it will become as big a cause as it needs to, in the sense that if

you ban other ways to get rich, people who want to get rich will

use this route instead.Thanks to Sam Altman, Trevor Blackwell, Paul Buchheit, Patrick

Collison, Ron Conway, Chris Dixon, Benedict Evans, Richard Florida,

Ben Horowitz, Jessica Livingston, Robert Morris, Tim O'Reilly, Geoff

Ralston, Max Roser, Alexia Tsotsis, and Qasar Younis for reading

drafts of this. Max also told me about several valuable sources.BibliographyAllen, Frederick Lewis. The Big Change. Harper, 1952.Averitt, Robert. The Dual Economy. Norton, 1968.Badger, Anthony. The New Deal. Hill and Wang, 1989.Bainbridge, John. The Super-Americans. Doubleday, 1961.Beatty, Jack. Collossus. Broadway, 2001.Brinkley, Douglas. Wheels for the World. Viking, 2003.Brownleee, W. Elliot. Federal Taxation in America. Cambridge, 1996.Chandler, Alfred. The Visible Hand. Harvard, 1977.Chernow, Ron. The House of Morgan. Simon & Schuster, 1990.Chernow, Ron. Titan: The Life of John D. Rockefeller. Random House,

1998.Galbraith, John. The New Industrial State. Houghton Mifflin, 1967.Goldin, Claudia and Robert A. Margo. "The Great Compression: The

Wage Structure in the United States at Mid-Century." NBER Working

Paper 3817, 1991.Gordon, John. An Empire of Wealth. HarperCollins, 2004.Klein, Maury. The Genesis of Industrial America, 1870-1920. Cambridge,

2007.Lind, Michael. Land of Promise. HarperCollins, 2012.Mickelthwaite, John, and Adrian Wooldridge. The Company. Modern

Library, 2003.Nasaw, David. Andrew Carnegie. Penguin, 2006.Sobel, Robert. The Age of Giant Corporations. Praeger, 1993.Thurow, Lester. Generating Inequality: Mechanisms of Distribution.

Basic Books, 1975.Witte, John. The Politics and Development of the Federal Income

Tax. Wisconsin, 1985.Related:Too Many Elite American Men Are Obsessed With Work and Wealth

How to Think for Yourself

November 2020There are some kinds of work that you can't do well without thinking

differently from your peers. To be a successful scientist, for

example, it's not enough just to be correct. Your ideas have to be

both correct and novel. You can't publish papers saying things other

people already know. You need to say things no one else has realized

yet.The same is true for investors. It's not enough for a public market

investor to predict correctly how a company will do. If a lot of

other people make the same prediction, the stock price will already

reflect it, and there's no room to make money. The only valuable

insights are the ones most other investors don't share.You see this pattern with startup founders too. You don't want to

start a startup to do something that everyone agrees is a good idea,

or there will already be other companies doing it. You have to do

something that sounds to most other people like a bad idea, but

that you know isn't like writing software for a tiny computer

used by a few thousand hobbyists, or starting a site to let people

rent airbeds on strangers' floors.Ditto for essayists. An essay that told people things they already

knew would be boring. You have to tell them something new.But this pattern isn't universal. In fact, it doesn't hold for most

kinds of work. In most kinds of work to be an administrator, for

example all you need is the first half. All you need is to be

right. It's not essential that everyone else be wrong.There's room for a little novelty in most kinds of work, but in

practice there's a fairly sharp distinction between the kinds of

work where it's essential to be independent-minded, and the kinds

where it's not.I wish someone had told me about this distinction when I was a kid,

because it's one of the most important things to think about when

you're deciding what kind of work you want to do. Do you want to

do the kind of work where you can only win by thinking differently

from everyone else? I suspect most people's unconscious mind will

answer that question before their conscious mind has a chance to.

I know mine does.Independent-mindedness seems to be more a matter of nature than

nurture. Which means if you pick the wrong type of work, you're

going to be unhappy. If you're naturally independent-minded, you're

going to find it frustrating to be a middle manager. And if you're

naturally conventional-minded, you're going to be sailing into a

headwind if you try to do original research.One difficulty here, though, is that people are often mistaken about

where they fall on the spectrum from conventional- to independent-minded.

Conventional-minded people don't like to think of themselves as

conventional-minded. And in any case, it genuinely feels to them

as if they make up their own minds about everything. It's just a

coincidence that their beliefs are identical to their peers'. And

the independent-minded, meanwhile, are often unaware how different

their ideas are from conventional ones, at least till they state

them publicly.

[1]By the time they reach adulthood, most people know roughly how smart

they are (in the narrow sense of ability to solve pre-set problems),

because they're constantly being tested and ranked according to it.

But schools generally ignore independent-mindedness, except to the

extent they try to suppress it. So we don't get anything like the

same kind of feedback about how independent-minded we are.There may even be a phenomenon like Dunning-Kruger at work, where

the most conventional-minded people are confident that they're

independent-minded, while the genuinely independent-minded worry

they might not be independent-minded enough.

\_\_\_\_\_\_\_\_\_\_\_

Can you make yourself more independent-minded? I think so. This

quality may be largely inborn, but there seem to be ways to magnify

it, or at least not to suppress it.One of the most effective techniques is one practiced unintentionally

by most nerds: simply to be less aware what conventional beliefs

are. It's hard to be a conformist if you don't know what you're

supposed to conform to. Though again, it may be that such people

already are independent-minded. A conventional-minded person would

probably feel anxious not knowing what other people thought, and

make more effort to find out.It matters a lot who you surround yourself with. If you're surrounded

by conventional-minded people, it will constrain which ideas you

can express, and that in turn will constrain which ideas you have.

But if you surround yourself with independent-minded people, you'll

have the opposite experience: hearing other people say surprising

things will encourage you to, and to think of more.Because the independent-minded find it uncomfortable to be surrounded

by conventional-minded people, they tend to self-segregate once

they have a chance to. The problem with high school is that they

haven't yet had a chance to. Plus high school tends to be an

inward-looking little world whose inhabitants lack confidence, both

of which magnify the forces of conformism. And so high school is

often a bad time for the

independent-minded. But there is some advantage even here: it

teaches you what to avoid. If you later find yourself in a situation

that makes you think "this is like high school," you know you should

get out.

[2]Another place where the independent- and conventional-minded are

thrown together is in successful startups. The founders and early

employees are almost always independent-minded; otherwise the startup

wouldn't be successful. But conventional-minded people greatly

outnumber independent-minded ones, so as the company grows, the

original spirit of independent-mindedness is inevitably diluted.

This causes all kinds of problems besides the obvious one that the

company starts to suck. One of the strangest is that the founders

find themselves able to speak more freely with founders of other

companies than with their own employees.

[3]Fortunately you don't have to spend all your time with independent-minded

people. It's enough to have one or two you can talk to regularly.

And once you find them, they're usually as eager to talk as you

are; they need you too. Although universities no longer have the

kind of monopoly they used to have on education, good universities

are still an excellent way to meet independent-minded people. Most

students will still be conventional-minded, but you'll at least

find clumps of independent-minded ones, rather than the near zero

you may have found in high school.It also works to go in the other direction: as well as cultivating

a small collection of independent-minded friends, to try to meet

as many different types of people as you can. It will decrease the

influence of your immediate peers if you have several other groups

of peers. Plus if you're part of several different worlds, you can

often import ideas from one to another.But by different types of people, I don't mean demographically

different. For this technique to work, they have to think differently.

So while it's an excellent idea to go and visit other countries,

you can probably find people who think differently right around the

corner. When I meet someone who knows a lot about something unusual

(which includes practically everyone, if you dig deep enough), I

try to learn what they know that other people don't. There are

almost always surprises here. It's a good way to make conversation

when you meet strangers, but I don't do it to make conversation.

I really want to know.You can expand the source of influences in time as well as space,

by reading history. When I read history I do it not just to learn

what happened, but to try to get inside the heads of people who

lived in the past. How did things look to them? This is hard to do,

but worth the effort for the same reason it's worth travelling far

to triangulate a point.You can also take more explicit measures to prevent yourself from

automatically adopting conventional opinions. The most general is

to cultivate an attitude of skepticism. When you hear someone say

something, stop and ask yourself "Is that true?" Don't say it out

loud. I'm not suggesting that you impose on everyone who talks to

you the burden of proving what they say, but rather that you take

upon yourself the burden of evaluating what they say.Treat it as a puzzle. You know that some accepted ideas will later

turn out to be wrong. See if you can guess which. The end goal is

not to find flaws in the things you're told, but to find the new

ideas that had been concealed by the broken ones. So this game

should be an exciting quest for novelty, not a boring protocol for

intellectual hygiene. And you'll be surprised, when you start asking

"Is this true?", how often the answer is not an immediate yes. If

you have any imagination, you're more likely to have too many leads

to follow than too few.More generally your goal should be not to let anything into your

head unexamined, and things don't always enter your head in the

form of statements. Some of the most powerful influences are implicit.

How do you even notice these? By standing back and watching how

other people get their ideas.When you stand back at a sufficient distance, you can see ideas

spreading through groups of people like waves. The most obvious are

in fashion: you notice a few people wearing a certain kind of shirt,

and then more and more, until half the people around you are wearing

the same shirt. You may not care much what you wear, but there are

intellectual fashions too, and you definitely don't want to participate

in those. Not just because you want sovereignty over your own

thoughts, but because unfashionable

ideas are disproportionately likely to lead somewhere interesting.

The best place to find undiscovered ideas is where no one else is

looking.

[4]

\_\_\_\_\_\_\_\_\_\_\_

To go beyond this general advice, we need to look at the internal

structure of independent-mindedness at the individual muscles

we need to exercise, as it were. It seems to me that it has three

components: fastidiousness about truth, resistance to being told

what to think, and curiosity.Fastidiousness about truth means more than just not believing things

that are false. It means being careful about degree of belief. For

most people, degree of belief rushes unexamined toward the extremes:

the unlikely becomes impossible, and the probable becomes certain.

[5]

To the independent-minded, this seems unpardonably sloppy.

They're willing to have anything in their heads, from highly

speculative hypotheses to (apparent) tautologies, but on subjects

they care about, everything has to be labelled with a carefully

considered degree of belief.

[6]The independent-minded thus have a horror of ideologies, which

require one to accept a whole collection of beliefs at once, and

to treat them as articles of faith. To an independent-minded person

that would seem revolting, just as it would seem to someone fastidious

about food to take a bite of a submarine sandwich filled with a

large variety of ingredients of indeterminate age and provenance.Without this fastidiousness about truth, you can't be truly

independent-minded. It's not enough just to have resistance to being

told what to think. Those kind of people reject conventional ideas

only to replace them with the most random conspiracy theories. And

since these conspiracy theories have often been manufactured to

capture them, they end up being less independent-minded than ordinary

people, because they're subject to a much more exacting master than

mere convention.

[7]Can you increase your fastidiousness about truth? I would think so.

In my experience, merely thinking about something you're fastidious

about causes that fastidiousness to grow. If so, this is one of

those rare virtues we can have more of merely by wanting it. And

if it's like other forms of fastidiousness, it should also be

possible to encourage in children. I certainly got a strong dose

of it from my father.

[8]The second component of independent-mindedness, resistance to being

told what to think, is the most visible of the three. But even this

is often misunderstood. The big mistake people make about it is to

think of it as a merely negative quality. The language we use

reinforces that idea. You're unconventional. You don't care

what other people think. But it's not just a kind of immunity. In

the most independent-minded people, the desire not to be told what

to think is a positive force. It's not mere skepticism, but an

active delight in ideas that subvert

the conventional wisdom, the more counterintuitive the better.Some of the most novel ideas seemed at the time almost like practical

jokes. Think how often your reaction to a novel idea is to laugh.

I don't think it's because novel ideas are funny per se, but because

novelty and humor share a certain kind of surprisingness. But while

not identical, the two are close enough that there is a definite

correlation between having a sense of humor and being independent-minded

just as there is between being humorless and being conventional-minded.

[9]I don't think we can significantly increase our resistance to being

told what to think. It seems the most innate of the three components

of independent-mindedness; people who have this quality as adults

usually showed all too visible signs of it as children. But if we

can't increase our resistance to being told what to think, we can

at least shore it up, by surrounding ourselves with other

independent-minded people.The third component of independent-mindedness, curiosity, may be

the most interesting. To the extent that we can give a brief answer

to the question of where novel ideas come from, it's curiosity. That's

what people are usually feeling before having them.In my experience, independent-mindedness and curiosity predict one

another perfectly. Everyone I know who's independent-minded is

deeply curious, and everyone I know who's conventional-minded isn't.

Except, curiously, children. All small children are curious. Perhaps

the reason is that even the conventional-minded have to be curious

in the beginning, in order to learn what the conventions are. Whereas

the independent-minded are the gluttons of curiosity, who keep

eating even after they're full.

[10]The three components of independent-mindedness work in concert:

fastidiousness about truth and resistance to being told what to

think leave space in your brain, and curiosity finds new ideas to

fill it.Interestingly, the three components can substitute for one another

in much the same way muscles can. If you're sufficiently fastidious

about truth, you don't need to be as resistant to being told what

to think, because fastidiousness alone will create sufficient gaps

in your knowledge. And either one can compensate for curiosity,

because if you create enough space in your brain, your discomfort

at the resulting vacuum will add force to your curiosity. Or curiosity

can compensate for them: if you're sufficiently curious, you don't

need to clear space in your brain, because the new ideas you discover

will push out the conventional ones you acquired by default.Because the components of independent-mindedness are so interchangeable,

you can have them to varying degrees and still get the same result.

So there is not just a single model of independent-mindedness. Some

independent-minded people are openly subversive, and others are

quietly curious. They all know the secret handshake though.Is there a way to cultivate curiosity? To start with, you want to

avoid situations that suppress it. How much does the work you're

currently doing engage your curiosity? If the answer is "not much,"

maybe you should change something.The most important active step you can take to cultivate your

curiosity is probably to seek out the topics that engage it. Few

adults are equally curious about everything, and it doesn't seem

as if you can choose which topics interest you. So it's up to you

to find them. Or invent them, if

necessary.Another way to increase your curiosity is to indulge it, by

investigating things you're interested in. Curiosity is unlike

most other appetites in this respect: indulging it tends to increase

rather than to sate it. Questions lead to more questions.Curiosity seems to be more individual than fastidiousness about

truth or resistance to being told what to think. To the degree

people have the latter two, they're usually pretty general, whereas

different people can be curious about very different things. So

perhaps curiosity is the compass here. Perhaps, if your goal is to

discover novel ideas, your motto should not be "do what you love"

so much as "do what you're curious about."Notes[1]

One convenient consequence of the fact that no one identifies

as conventional-minded is that you can say what you like about

conventional-minded people without getting in too much trouble.

When I wrote "The Four Quadrants of

Conformism" I expected a firestorm of rage from the

aggressively conventional-minded, but in fact it was quite muted.

They sensed that there was something about the essay that they

disliked intensely, but they had a hard time finding a specific

passage to pin it on.[2]

When I ask myself what in my life is like high school, the

answer is Twitter. It's not just full of conventional-minded people,

as anything its size will inevitably be, but subject to violent

storms of conventional-mindedness that remind me of descriptions

of Jupiter. But while it probably is a net loss to spend time there,

it has at least made me think more about the distinction between

independent- and conventional-mindedness, which I probably wouldn't

have done otherwise.[3]

The decrease in independent-mindedness in growing startups is

still an open problem, but there may be solutions.Founders can delay the problem by making a conscious effort only

to hire independent-minded people. Which of course also has the

ancillary benefit that they have better ideas.Another possible solution is to create policies that somehow disrupt

the force of conformism, much as control rods slow chain reactions,

so that the conventional-minded aren't as dangerous. The physical

separation of Lockheed's Skunk Works may have had this as a side

benefit. Recent examples suggest employee forums like Slack may not

be an unmitigated good.The most radical solution would be to grow revenues without growing

the company. You think hiring that junior PR person will be cheap,

compared to a programmer, but what will be the effect on the average

level of independent-mindedness in your company? (The growth in

staff relative to faculty seems to have had a similar effect on

universities.) Perhaps the rule about outsourcing work that's not

your "core competency" should be augmented by one about outsourcing

work done by people who'd ruin your culture as employees.Some investment firms already seem to be able to grow revenues

without growing the number of employees. Automation plus the ever

increasing articulation of the "tech stack" suggest this may one

day be possible for product companies.[4]

There are intellectual fashions in every field, but their

influence varies. One of the reasons politics, for example, tends

to be boring is that it's so extremely subject to them. The threshold

for having opinions about politics is much lower than the one for having

opinions about set theory. So while there are some ideas in politics,

in practice they tend to be swamped by waves of intellectual fashion.[5]

The conventional-minded are often fooled by the strength of

their opinions into believing that they're independent-minded. But

strong convictions are not a sign of independent-mindedness. Rather

the opposite.[6]

Fastidiousness about truth doesn't imply that an independent-minded

person won't be dishonest, but that he won't be deluded. It's sort

of like the definition of a gentleman as someone who is never

unintentionally rude.[7]

You see this especially among political extremists. They think

themselves nonconformists, but actually they're niche conformists.

Their opinions may be different from the average person's, but they

are often more influenced by their peers' opinions than the average

person's are.[8]

If we broaden the concept of fastidiousness about truth so that

it excludes pandering, bogusness, and pomposity as well as falsehood

in the strict sense, our model of independent-mindedness can expand

further into the arts.[9]

This correlation is far from perfect, though. Gödel and Dirac

don't seem to have been very strong in the humor department. But

someone who is both "neurotypical" and humorless is very likely to

be conventional-minded.[10]

Exception: gossip. Almost everyone is curious about gossip.

Thanks to Trevor Blackwell, Paul Buchheit, Patrick Collison, Jessica

Livingston, Robert Morris, Harj Taggar, and Peter Thiel for reading

drafts of this.

Early Work

October 2020One of the biggest things holding people back from doing great work

is the fear of making something lame. And this fear is not an

irrational one. Many great projects go through a stage early on

where they don't seem very impressive, even to their creators. You

have to push through this stage to reach the great work that lies

beyond. But many people don't. Most people don't even reach the

stage of making something they're embarrassed by, let alone continue

past it. They're too frightened even to start.Imagine if we could turn off the fear of making something lame.

Imagine how much more we'd do.Is there any hope of turning it off? I think so. I think the habits

at work here are not very deeply rooted.Making new things is itself a new thing for us as a species. It has

always happened, but till the last few centuries it happened so

slowly as to be invisible to individual humans. And since we didn't

need customs for dealing with new ideas, we didn't develop any.We just don't have enough experience with early versions of ambitious

projects to know how to respond to them. We judge them as we would

judge more finished work, or less ambitious projects. We don't

realize they're a special case.Or at least, most of us don't. One reason I'm confident we can do

better is that it's already starting to happen. There are already

a few places that are living in the future in this respect. Silicon

Valley is one of them: an unknown person working on a strange-sounding

idea won't automatically be dismissed the way they would back home.

In Silicon Valley, people have learned how dangerous that is.The right way to deal with new ideas is to treat them as a challenge

to your imagination not just to have lower standards, but to

switch polarity entirely, from listing

the reasons an idea won't

work to trying to think of ways it could. That's what I do when I

meet people with new ideas. I've become quite good at it, but I've

had a lot of practice. Being a partner at Y Combinator means being

practically immersed in strange-sounding ideas proposed by unknown

people. Every six months you get thousands of new ones thrown at

you and have to sort through them, knowing that in a world with a

power-law distribution of outcomes, it will be painfully obvious

if you miss the needle in this haystack. Optimism becomes

urgent.But I'm hopeful that, with time, this kind of optimism can become

widespread enough that it becomes a social custom, not just a trick

used by a few specialists. It is after all an extremely lucrative

trick, and those tend to spread quickly.Of course, inexperience is not the only reason people are too harsh

on early versions of ambitious projects. They also do it to seem

clever. And in a field where the new ideas are risky, like startups,

those who dismiss them are in fact more likely to be right. Just

not when their predictions are

weighted by outcome.But there is another more sinister reason people dismiss new ideas.

If you try something ambitious, many of those around you will hope,

consciously or unconsciously, that you'll fail. They worry that if

you try something ambitious and succeed, it will put you above them.

In some countries this is not just an individual failing but part

of the national culture.I wouldn't claim that people in Silicon Valley overcome these

impulses because they're morally better.

[1]

The reason many hope

you'll succeed is that they hope to rise with you. For investors

this incentive is particularly explicit. They want you to succeed

because they hope you'll make them rich in the process. But many

other people you meet can hope to benefit in some way from your

success. At the very least they'll be able to say, when you're

famous, that they've known you since way back.But even if Silicon Valley's encouraging attitude

is rooted in self-interest, it has over time actually grown into a

sort of benevolence. Encouraging startups has been practiced for

so long that it has become a custom. Now it just seems that that's

what one does with startups.Maybe Silicon Valley is too optimistic. Maybe it's too easily fooled

by impostors. Many less optimistic journalists want to believe that.

But the lists of impostors they cite are suspiciously short, and

plagued with asterisks.

[2] If you use revenue as the test, Silicon

Valley's optimism seems better tuned than the rest of the world's.

And because it works, it will spread.There's a lot more to new ideas than new startup ideas, of course.

The fear of making something lame holds people back in every field.

But Silicon Valley shows how quickly customs can evolve to support

new ideas. And that in turn proves that dismissing new ideas is not

so deeply rooted in human nature that it can't be unlearnt.

\_\_\_\_\_\_\_\_\_\_\_

Unfortunately, if you want to do new things, you'll face a force

more powerful than other people's skepticism: your own skepticism.

You too will judge your early work too harshly. How do you avoid

that?This is a difficult problem, because you don't want to completely

eliminate your horror of making something lame. That's what steers

you toward doing good work. You just want to turn it off temporarily,

the way a painkiller temporarily turns off pain.People have already discovered several techniques that work. Hardy

mentions two in A Mathematician's Apology:

Good work is not done by "humble" men. It is one of the first

duties of a professor, for example, in any subject, to exaggerate

a little both the importance of his subject and his importance

in it.

If you overestimate the importance of what you're working on, that

will compensate for your mistakenly harsh judgment of your initial

results. If you look at something that's 20% of the way to a goal

worth 100 and conclude that it's 10% of the way to a goal worth

200, your estimate of its expected value is correct even though

both components are wrong.It also helps, as Hardy suggests, to be slightly overconfident.

I've noticed in many fields that the most successful people are

slightly overconfident. On the face of it this seems implausible.

Surely it would be optimal to have exactly the right estimate of

one's abilities. How could it be an advantage to be mistaken?

Because this error compensates for other sources of error in the

opposite direction: being slightly overconfident armors you against

both other people's skepticism and your own.Ignorance has a similar effect. It's safe to make the mistake of

judging early work as finished work if you're a sufficiently lax

judge of finished work. I doubt it's possible to cultivate this

kind of ignorance, but empirically it's a real advantage, especially

for the young.Another way to get through the lame phase of ambitious projects is

to surround yourself with the right people to create an eddy in

the social headwind. But it's not enough to collect people who are

always encouraging. You'd learn to discount that. You need colleagues

who can actually tell an ugly duckling from a baby swan. The people

best able to do this are those working on similar projects of their

own, which is why university departments and research labs work so

well. You don't need institutions to collect colleagues. They

naturally coalesce, given the chance. But it's very much worth

accelerating this process by seeking out other people trying to do

new things.Teachers are in effect a special case of colleagues. It's a teacher's

job both to see the promise of early work and to encourage you to

continue. But teachers who are good at this are unfortunately quite

rare, so if you have the opportunity to learn from one, take it.

[3]For some it might work to rely on sheer discipline: to tell yourself

that you just have to press on through the initial crap phase and

not get discouraged. But like a lot of "just tell yourself" advice,

this is harder than it sounds. And it gets still harder as you get

older, because your standards rise. The old do have one compensating

advantage though: they've been through this before.It can help if you focus less on where you are and more on the rate

of change. You won't worry so much about doing bad work if you can

see it improving. Obviously the faster it improves, the easier this

is. So when you start something new, it's good if you can spend a

lot of time on it. That's another advantage of being young: you

tend to have bigger blocks of time.Another common trick is to start by considering new work to be of

a different, less exacting type. To start a painting saying that

it's just a sketch, or a new piece of software saying that it's

just a quick hack. Then you judge your initial results by a lower

standard. Once the project is rolling you can sneakily convert it

to something more.

[4]This will be easier if you use a medium that lets you work fast and

doesn't require too much commitment up front. It's easier to convince

yourself that something is just a sketch when you're drawing in a

notebook than when you're carving stone. Plus you get initial results

faster.

[5]

[6]It will be easier to try out a risky project if you think of it as

a way to learn and not just as a way to make something. Then even

if the project truly is a failure, you'll still have gained by it.

If the problem is sharply enough defined, failure itself is

knowledge: if the theorem you're trying to prove turns out to

be false, or you use a structural member of a certain size and

it fails under stress, you've learned something, even if it

isn't what you wanted to learn.

[7]One motivation that works particularly well for me is curiosity.

I like to try new things just to see how they'll turn out. We started

Y Combinator in this spirit, and it was one of main things that

kept me going while I was working on

Bel. Having worked for so long

with various dialects of Lisp, I was very curious to see what its

inherent shape was: what you'd end up with if you followed the

axiomatic approach all the way.But it's a bit strange that you have to play mind games with yourself

to avoid being discouraged by lame-looking early efforts. The thing

you're trying to trick yourself into believing is in fact the truth.

A lame-looking early version of an ambitious project truly is more

valuable than it seems. So the ultimate solution may be to teach

yourself that.One way to do it is to study the histories of people who've

done great work. What were they thinking early on? What was the

very first thing they did? It can sometimes be hard to get an

accurate answer to this question, because people are often embarrassed

by their earliest work and make little effort to publish it. (They

too misjudge it.) But when you can get an accurate picture of the

first steps someone made on the path to some great work, they're

often pretty feeble.

[8]Perhaps if you study enough such cases, you can teach yourself to

be a better judge of early work. Then you'll be immune both to other

people's skepticism and your own fear of making something lame.

You'll see early work for what it is.Curiously enough, the solution to the problem of judging early work

too harshly is to realize that our attitudes toward it are themselves

early work. Holding everything to the same standard is a crude

version 1. We're already evolving better customs, and we can already

see signs of how big the payoff will be.

Notes[1]

This assumption may be too conservative. There is some evidence

that historically the Bay Area has attracted a

different sort of person than,

say, New York City.[2]

One of their great favorites is Theranos. But the most conspicuous

feature of Theranos's cap table is the absence of Silicon Valley

firms. Journalists were fooled by Theranos, but Silicon Valley

investors weren't.[3]

I made two mistakes about teachers when I was younger. I

cared more about professors' research than their reputations as

teachers, and I was also wrong about what it meant to be a good

teacher. I thought it simply meant to be good at explaining things.[4]

Patrick Collison points out that you can go past treating

something as a hack in the sense of a prototype and onward to the

sense of the word that means something closer to a practical joke:

I think there may be something related to being a hack that can

be powerful the idea of making the tenuousness and implausibility

a feature. "Yes, it's a bit ridiculous, right? I'm just trying

to see how far such a naive approach can get." YC seemed to me

to have this characteristic.

[5]

Much of the advantage of switching from physical to digital

media is not the software per se but that it lets you start something

new with little upfront commitment.[6]

John Carmack adds:

The value of a medium without a vast gulf between the early work

and the final work is exemplified in game mods. The original

Quake game was a golden age for mods, because everything was very

flexible, but so crude due to technical limitations, that quick

hacks to try out a gameplay idea weren't all that far from the

official game. Many careers were born from that, but as the

commercial game quality improved over the years, it became almost

a full time job to make a successful mod that would be appreciated

by the community. This was dramatically reversed with Minecraft

and later Roblox, where the entire esthetic of the experience was

so explicitly crude that innovative gameplay concepts became the

overriding value. These "crude" game mods by single authors are

now often bigger deals than massive professional teams' work.

[7]

Lisa Randall suggests that we

treat new things as experiments. That way there's no such thing

as failing, since you learn something no matter what. You treat

it like an experiment in the sense that if it really rules something

out, you give up and move on, but if there's some way to vary it

to make it work better, go ahead and do that

[8]

Michael Nielsen points out that the internet has made this

easier, because you can see programmers' first commits, musicians'

first videos, and so on.Thanks to Trevor Blackwell, John Carmack, Patrick Collison, Jessica

Livingston, Michael Nielsen, and Lisa Randall for reading drafts

of this.

Modeling a Wealth Tax

August 2020Some politicians are proposing to introduce wealth taxes in addition

to income and capital gains taxes. Let's try modeling the effects of various levels

of wealth tax to see what they would mean in practice for a startup

founder.Suppose you start a successful startup in your twenties, and then

live for another 60 years. How much of your stock will a wealth tax

consume?If the wealth tax applies to all your assets, it's easy to

calculate its effect. A wealth tax of 1% means you get to keep

99% of your stock each year. After 60 years the proportion

of stock you'll have left will be .99^60, or .547. So a

straight 1% wealth tax means the government will over the

course of your life take 45% of your stock.(Losing shares does not, obviously, mean becoming net

poorer unless the value per share is increasing by less than the

wealth tax rate.)Here's how much stock the government would take over 60

years at various levels of wealth tax:

wealth taxgovernment takes

0.1%6%0.5%26%

1.0%45%

2.0%70%

3.0%84%

4.0%91%5.0%95%

A wealth tax will usually have a threshold at which it starts.

How much difference would a high threshold make? To model that,

we need to make some assumptions about the initial value of

your stock and the growth rate.Suppose your stock is initially

worth $2 million, and the company's trajectory is as follows:

the value of your stock grows 3x for 2 years, then 2x for 2 years,

then 50% for 2 years, after

which you just get a typical public company growth rate,

which we'll call 8%. Suppose the wealth tax threshold is

$50 million. How much stock does the government take now?

wealth taxgovernment takes

0.1%5%0.5%23%

1.0%41%

2.0%65%

3.0%79%

4.0%88%5.0%93%

It may at first seem surprising that such apparently small tax rates

produce such dramatic effects. A 2% wealth tax with a $50 million

threshold takes about two thirds of a successful founder's stock.The reason wealth taxes have such dramatic effects is that they're

applied over and over to the same money. Income tax

happens every year, but only to that year's income. Whereas if you

live for 60 years after acquiring some asset, a wealth tax will tax

that same asset 60 times. A wealth tax compounds.

The Four Quadrants of Conformism

July 2020One of the most revealing ways to classify people is by the degree

and aggressiveness of their conformism. Imagine a Cartesian coordinate

system whose horizontal axis runs from conventional-minded on the

left to independent-minded on the right, and whose vertical axis

runs from passive at the bottom to aggressive at the top. The

resulting four quadrants define four types of people. Starting in

the upper left and going counter-clockwise: aggressively

conventional-minded, passively conventional-minded, passively

independent-minded, and aggressively independent-minded.I think that you'll find all four types in most societies, and that

which quadrant people fall into depends more on their own personality

than the beliefs prevalent in their society.

[1]Young children offer some of the best evidence for both points.

Anyone who's been to primary school has seen the four types, and

the fact that school rules are so arbitrary is strong evidence that

the quadrant people fall into depends more on them than the rules.The kids in the upper left quadrant, the aggressively conventional-minded

ones, are the tattletales. They believe not only that rules must

be obeyed, but that those who disobey them must be punished.The kids in the lower left quadrant, the passively conventional-minded,

are the sheep. They're careful to obey the rules, but when other

kids break them, their impulse is to worry that those kids will be

punished, not to ensure that they will.The kids in the lower right quadrant, the passively independent-minded,

are the dreamy ones. They don't care much about rules and probably

aren't 100% sure what the rules even are.And the kids in the upper right quadrant, the aggressively

independent-minded, are the naughty ones. When they see a rule,

their first impulse is to question it. Merely being told what to

do makes them inclined to do the opposite.When measuring conformism, of course, you have to say with respect

to what, and this changes as kids get older. For younger kids it's

the rules set by adults. But as kids get older, the source of rules

becomes their peers. So a pack of teenagers who all flout school

rules in the same way are not independent-minded; rather the opposite.In adulthood we can recognize the four types by their distinctive

calls, much as you could recognize four species of birds. The call

of the aggressively conventional-minded is "Crush <outgroup>!" (It's

rather alarming to see an exclamation point after a variable, but

that's the whole problem with the aggressively conventional-minded.)

The call of the passively conventional-minded is "What will the

neighbors think?" The call of the passively independent-minded is

"To each his own." And the call of the aggressively independent-minded

is "Eppur si muove."The four types are not equally common. There are more passive people

than aggressive ones, and far more conventional-minded people than

independent-minded ones. So the passively conventional-minded are

the largest group, and the aggressively independent-minded the

smallest.Since one's quadrant depends more on one's personality than the

nature of the rules, most people would occupy the same quadrant

even if they'd grown up in a quite different society.Princeton professor Robert George recently wrote:

I sometimes ask students what their position on slavery would

have been had they been white and living in the South before

abolition. Guess what? They all would have been abolitionists!

They all would have bravely spoken out against slavery, and

worked tirelessly against it.

He's too polite to say so, but of course they wouldn't. And indeed,

our default assumption should not merely be that his students would,

on average, have behaved the same way people did at the time, but

that the ones who are aggressively conventional-minded today would

have been aggressively conventional-minded then too. In other words,

that they'd not only not have fought against slavery, but that

they'd have been among its staunchest defenders.I'm biased, I admit, but it seems to me that aggressively

conventional-minded people are responsible for a disproportionate

amount of the trouble in the world, and that a lot of the customs

we've evolved since the Enlightenment have been designed to protect

the rest of us from them. In particular, the retirement of the

concept of heresy and its replacement by the principle of freely

debating all sorts of different ideas, even ones that are currently

considered unacceptable, without any punishment for those who try

them out to see if they work.

[2]Why do the independent-minded need to be protected, though? Because

they have all the new ideas. To be a successful scientist, for

example, it's not enough just to be right. You have to be right

when everyone else is wrong. Conventional-minded people can't do

that. For similar reasons, all successful startup CEOs are not

merely independent-minded, but aggressively so. So it's no coincidence

that societies prosper only to the extent that they have customs

for keeping the conventional-minded at bay.

[3]In the last few years, many of us have noticed that the customs

protecting free inquiry have been weakened. Some say we're overreacting

that they haven't been weakened very much, or that they've been

weakened in the service of a greater good. The latter I'll dispose

of immediately. When the conventional-minded get the upper hand,

they always say it's in the service of a greater good. It just

happens to be a different, incompatible greater good each time.As for the former worry, that the independent-minded are being

oversensitive, and that free inquiry hasn't been shut down that

much, you can't judge that unless you are yourself independent-minded.

You can't know how much of the space of ideas is being lopped off

unless you have them, and only the independent-minded have the ones

at the edges. Precisely because of this, they tend to be very

sensitive to changes in how freely one can explore ideas. They're

the canaries in this coalmine.The conventional-minded say, as they always do, that they don't

want to shut down the discussion of all ideas, just the bad ones.You'd think it would be obvious just from that sentence what a

dangerous game they're playing. But I'll spell it out. There are

two reasons why we need to be able to discuss even "bad" ideas.The first is that any process for deciding which ideas to ban is

bound to make mistakes. All the more so because no one intelligent

wants to undertake that kind of work, so it ends up being done by

the stupid. And when a process makes a lot of mistakes, you need

to leave a margin for error. Which in this case means you need to

ban fewer ideas than you'd like to. But that's hard for the

aggressively conventional-minded to do, partly because they enjoy

seeing people punished, as they have since they were children, and

partly because they compete with one another. Enforcers of orthodoxy

can't allow a borderline idea to exist, because that gives other

enforcers an opportunity to one-up them in the moral purity department,

and perhaps even to turn enforcer upon them. So instead of getting

the margin for error we need, we get the opposite: a race to the

bottom in which any idea that seems at all bannable ends up being

banned.

[4]The second reason it's dangerous to ban the discussion of ideas is

that ideas are more closely related than they look. Which means if

you restrict the discussion of some topics, it doesn't only affect

those topics. The restrictions propagate back into any topic that

yields implications in the forbidden ones. And that is not an edge

case. The best ideas do exactly that: they have consequences

in fields far removed from their origins. Having ideas in a world

where some ideas are banned is like playing soccer on a pitch that

has a minefield in one corner. You don't just play the same game

you would have, but on a different shaped pitch. You play a much

more subdued game even on the ground that's safe.In the past, the way the independent-minded protected themselves

was to congregate in a handful of places first in courts, and

later in universities where they could to some extent make their

own rules. Places where people work with ideas tend to have customs

protecting free inquiry, for the same reason wafer fabs have powerful

air filters, or recording studios good sound insulation. For the

last couple centuries at least, when the aggressively conventional-minded

were on the rampage for whatever reason, universities were the

safest places to be.That may not work this time though, due to the unfortunate fact

that the latest wave of intolerance began in universities. It began

in the mid 1980s, and by 2000 seemed to have died down, but it has

recently flared up again with the arrival of social media. This

seems, unfortunately, to have been an own goal by Silicon Valley.

Though the people who run Silicon Valley are almost all independent-minded,

they've handed the aggressively conventional-minded a tool such as

they could only have dreamed of.On the other hand, perhaps the decline in the spirit of free inquiry

within universities is as much the symptom of the departure of the

independent-minded as the cause. People who would have become

professors 50 years ago have other options now. Now they can become

quants or start startups. You have to be independent-minded to

succeed at either of those. If these people had been professors,

they'd have put up a stiffer resistance on behalf of academic

freedom. So perhaps the picture of the independent-minded fleeing

declining universities is too gloomy. Perhaps the universities are

declining because so many have already left.

[5]Though I've spent a lot of time thinking about this situation, I

can't predict how it plays out. Could some universities reverse the

current trend and remain places where the independent-minded want

to congregate? Or will the independent-minded gradually abandon

them? I worry a lot about what we might lose if that happened.But I'm hopeful long term. The independent-minded are good at

protecting themselves. If existing institutions are compromised,

they'll create new ones. That may require some imagination. But

imagination is, after all, their specialty.

Notes[1]

I realize of course that if people's personalities vary in any

two ways, you can use them as axes and call the resulting four

quadrants personality types. So what I'm really claiming is that

the axes are orthogonal and that there's significant variation in

both.[2]

The aggressively conventional-minded aren't responsible for all

the trouble in the world. Another big source of trouble is the sort

of charismatic leader who gains power by appealing to them. They

become much more dangerous when such leaders emerge.[3]

I never worried about writing things that offended the

conventional-minded when I was running Y Combinator. If YC were a

cookie company, I'd have faced a difficult moral choice.

Conventional-minded people eat cookies too. But they don't start

successful startups. So if I deterred them from applying to YC, the

only effect was to save us work reading applications.[4]

There has been progress in one area: the punishments for talking

about banned ideas are less severe than in the past. There's little

danger of being killed, at least in richer countries. The aggressively

conventional-minded are mostly satisfied with getting people fired.[5]

Many professors are independent-minded especially in math,

the hard sciences, and engineering, where you have to be to succeed.

But students are more representative of the general population, and

thus mostly conventional-minded. So when professors and students

are in conflict, it's not just a conflict between generations but

also between different types of people.Thanks to Sam Altman, Trevor Blackwell, Nicholas Christakis, Patrick

Collison, Sam Gichuru, Jessica Livingston, Patrick McKenzie, Geoff

Ralston, and Harj Taggar for reading drafts of this.German TranslationKorean TranslationSerbian Translation

Orthodox Privilege

July 2020There has been a lot of talk about privilege lately. Although the

concept is overused, there is something to it, and in particular

to the idea that privilege makes you blind that you can't see

things that are visible to someone whose life is very different

from yours.But one of the most pervasive examples of this kind of blindness

is one that I haven't seen mentioned explicitly. I'm going to call

it orthodox privilege: The more conventional-minded someone is, the

more it seems to them that it's safe for everyone to express their

opinions.It's safe for them to express their opinions, because the source

of their opinions is whatever it's currently acceptable to believe.

So it seems to them that it must be safe for everyone. They literally

can't imagine a true statement that would get them in trouble.And yet at every point in history, there were true things that would

get you in terrible trouble to say. Is ours the first where this

isn't so? What an amazing coincidence that would be.Surely it should at least be the default assumption that our time

is not unique, and that there are true things you can't say now,

just as there have always been. You would think. But even in the

face of such overwhelming historical evidence, most people will go

with their gut on this one.The spectral signature of orthodox privilege is "Why don't you just

say it?" If you think there's something true that people can't say,

why don't you be brave, and own it? The more extreme will even

accuse you of specific heresies they imagine you must have in mind,

though if there's more than one heresy current in your time, these

accusations will tend to be nondeterministic: you must either be

an xist or a yist.Frustrating as it is to deal with these people, it's important to

realize that they're in earnest. They're not pretending they think

it's impossible for an idea to be both unorthodox and true. The

world really looks that way to them.How do you respond to orthodox privilege? Merely giving it a name

may help somewhat, because it will remind you, when you encounter it,

why the people you're talking to seem so strangely unreasonable.

Because this is a uniquely tenacious form of privilege. People can

overcome the blindness induced by most forms of privilege by learning

more about whatever they're not. But they can't overcome orthodox

privilege just by learning more. They'd have to become more

independent-minded. If that happens at all, it doesn't happen on

the time scale of one conversation.It may be possible to convince some people that orthodox privilege

must exist even though they can't sense it, just as one can with,

say, dark matter. There may be some who could be convinced, for

example, that it's very unlikely that this is the first point in

history at which there's nothing true you can't say, even if they

can't imagine specific examples.But except with these people, I don't think it will work to say

"check your privilege" about this type of privilege, because those

in its demographic don't realize they're in it. It doesn't seem to

conventional-minded people that they're conventional-minded. It

just seems to them that they're right. Indeed, they tend to be

particularly sure of it.Perhaps the solution is to appeal to politeness. If someone says

they can hear a high-pitched noise that you can't, it's only polite

to take them at their word, instead of demanding evidence that's

impossible to produce, or simply denying that they hear anything.

Imagine how rude that would seem. Similarly, if someone says they

can think of things that are true but that cannot be said, it's

only polite to take them at their word, even if you can't think of

any yourself.Once you realize that orthodox privilege exists, a lot of other

things become clearer. For example, how can it be that a large

number of reasonable, intelligent people worry about something they

call "cancel culture," while other reasonable, intelligent people

deny that it's a problem? Once you understand the concept of orthodox

privilege, it's easy to see the source of this disagreement. If

you believe there's nothing true that you can't say, then anyone

who gets in trouble for something they say must deserve it.Thanks to Sam Altman, Trevor Blackwell, Patrick Collison, Antonio Garcia-Martinez,

Jessica Livingston, Robert Morris, Michael Nielsen, Geoff Ralston, Max Roser, and

Harj Taggar for reading drafts of this.

Coronavirus and Credibility

April 2020I recently saw a

video

of TV journalists and politicians confidently

saying that the coronavirus would be no worse than the flu. What

struck me about it was not just how mistaken they seemed, but how

daring. How could they feel safe saying such things?The answer, I realized, is that they didn't think they could get

caught. They didn't realize there was any danger in making false

predictions. These people constantly make false predictions, and

get away with it, because the things they make predictions about

either have mushy enough outcomes that they can bluster their way

out of trouble, or happen so far in the future that few remember

what they said.An epidemic is different. It falsifies your predictions rapidly and

unequivocally.But epidemics are rare enough that these people clearly

didn't realize this was even a possibility. Instead they just

continued to use their ordinary m.o., which, as the epidemic has

made clear, is to talk confidently about things they don't

understand.An event like this is thus a uniquely powerful way of taking people's

measure. As Warren Buffett said, "It's only when the tide goes out

that you learn who's been swimming naked." And the tide has just

gone out like never before.Now that we've seen the results, let's remember what we saw, because

this is the most accurate test of credibility we're ever likely to have. I hope.Finnish TranslationGerman TranslationFrench Translation

How to Write Usefully

February 2020What should an essay be? Many people would say persuasive. That's

what a lot of us were taught essays should be. But I think we can

aim for something more ambitious: that an essay should be useful.To start with, that means it should be correct. But it's not enough

merely to be correct. It's easy to make a statement correct by

making it vague. That's a common flaw in academic writing, for

example. If you know nothing at all about an issue, you can't go

wrong by saying that the issue is a complex one, that there are

many factors to be considered, that it's a mistake to take too

simplistic a view of it, and so on.Though no doubt correct, such statements tell the reader nothing.

Useful writing makes claims that are as strong as they can be made

without becoming false.For example, it's more useful to say that Pike's Peak is near the

middle of Colorado than merely somewhere in Colorado. But if I say

it's in the exact middle of Colorado, I've now gone too far, because

it's a bit east of the middle.Precision and correctness are like opposing forces. It's easy to

satisfy one if you ignore the other. The converse of vaporous

academic writing is the bold, but false, rhetoric of demagogues.

Useful writing is bold, but true.It's also two other things: it tells people something important,

and that at least some of them didn't already know.Telling people something they didn't know doesn't always mean

surprising them. Sometimes it means telling them something they

knew unconsciously but had never put into words. In fact those may

be the more valuable insights, because they tend to be more

fundamental.Let's put them all together. Useful writing tells people something

true and important that they didn't already know, and tells them

as unequivocally as possible.Notice these are all a matter of degree. For example, you can't

expect an idea to be novel to everyone. Any insight that you have

will probably have already been had by at least one of the world's

7 billion people. But it's sufficient if an idea is novel to a lot

of readers.Ditto for correctness, importance, and strength. In effect the four

components are like numbers you can multiply together to get a score

for usefulness. Which I realize is almost awkwardly reductive, but

nonetheless true.\_\_\_\_\_

How can you ensure that the things you say are true and novel and

important? Believe it or not, there is a trick for doing this. I

learned it from my friend Robert Morris, who has a horror of saying

anything dumb. His trick is not to say anything unless he's sure

it's worth hearing. This makes it hard to get opinions out of him,

but when you do, they're usually right.Translated into essay writing, what this means is that if you write

a bad sentence, you don't publish it. You delete it and try again.

Often you abandon whole branches of four or five paragraphs. Sometimes

a whole essay.You can't ensure that every idea you have is good, but you can

ensure that every one you publish is, by simply not publishing the

ones that aren't.In the sciences, this is called publication bias, and is considered

bad. When some hypothesis you're exploring gets inconclusive results,

you're supposed to tell people about that too. But with essay

writing, publication bias is the way to go.My strategy is loose, then tight. I write the first draft of an

essay fast, trying out all kinds of ideas. Then I spend days rewriting

it very carefully.I've never tried to count how many times I proofread essays, but

I'm sure there are sentences I've read 100 times before publishing

them. When I proofread an essay, there are usually passages that

stick out in an annoying way, sometimes because they're clumsily

written, and sometimes because I'm not sure they're true. The

annoyance starts out unconscious, but after the tenth reading or

so I'm saying "Ugh, that part" each time I hit it. They become like

briars that catch your sleeve as you walk past. Usually I won't

publish an essay till they're all gone till I can read through

the whole thing without the feeling of anything catching.I'll sometimes let through a sentence that seems clumsy, if I can't

think of a way to rephrase it, but I will never knowingly let through

one that doesn't seem correct. You never have to. If a sentence

doesn't seem right, all you have to do is ask why it doesn't, and

you've usually got the replacement right there in your head.This is where essayists have an advantage over journalists. You

don't have a deadline. You can work for as long on an essay as you

need to get it right. You don't have to publish the essay at all,

if you can't get it right. Mistakes seem to lose courage in the

face of an enemy with unlimited resources. Or that's what it feels

like. What's really going on is that you have different expectations

for yourself. You're like a parent saying to a child "we can sit

here all night till you eat your vegetables." Except you're the

child too.I'm not saying no mistake gets through. For example, I added condition

(c) in "A Way to Detect Bias"

after readers pointed out that I'd

omitted it. But in practice you can catch nearly all of them.There's a trick for getting importance too. It's like the trick I

suggest to young founders for getting startup ideas: to make something

you yourself want. You can use yourself as a proxy for the reader.

The reader is not completely unlike you, so if you write about

topics that seem important to you, they'll probably seem important

to a significant number of readers as well.Importance has two factors. It's the number of people something

matters to, times how much it matters to them. Which means of course

that it's not a rectangle, but a sort of ragged comb, like a Riemann

sum.The way to get novelty is to write about topics you've thought about

a lot. Then you can use yourself as a proxy for the reader in this

department too. Anything you notice that surprises you, who've

thought about the topic a lot, will probably also surprise a

significant number of readers. And here, as with correctness and

importance, you can use the Morris technique to ensure that you

will. If you don't learn anything from writing an essay, don't

publish it.You need humility to measure novelty, because acknowledging the

novelty of an idea means acknowledging your previous ignorance of

it. Confidence and humility are often seen as opposites, but in

this case, as in many others, confidence helps you to be humble.

If you know you're an expert on some topic, you can freely admit

when you learn something you didn't know, because you can be confident

that most other people wouldn't know it either.The fourth component of useful writing, strength, comes from two

things: thinking well, and the skillful use of qualification. These

two counterbalance each other, like the accelerator and clutch in

a car with a manual transmission. As you try to refine the expression

of an idea, you adjust the qualification accordingly. Something

you're sure of, you can state baldly with no qualification at all,

as I did the four components of useful writing. Whereas points that

seem dubious have to be held at arm's length with perhapses.As you refine an idea, you're pushing in the direction of less

qualification. But you can rarely get it down to zero. Sometimes

you don't even want to, if it's a side point and a fully refined

version would be too long.Some say that qualifications weaken writing. For example, that you

should never begin a sentence in an essay with "I think," because

if you're saying it, then of course you think it. And it's true

that "I think x" is a weaker statement than simply "x." Which is

exactly why you need "I think." You need it to express your degree

of certainty.But qualifications are not scalars. They're not just experimental

error. There must be 50 things they can express: how broadly something

applies, how you know it, how happy you are it's so, even how it

could be falsified. I'm not going to try to explore the structure

of qualification here. It's probably more complex than the whole

topic of writing usefully. Instead I'll just give you a practical

tip: Don't underestimate qualification. It's an important skill in

its own right, not just a sort of tax you have to pay in order to

avoid saying things that are false. So learn and use its full range.

It may not be fully half of having good ideas, but it's part of

having them.There's one other quality I aim for in essays: to say things as

simply as possible. But I don't think this is a component of

usefulness. It's more a matter of consideration for the reader. And

it's a practical aid in getting things right; a mistake is more

obvious when expressed in simple language. But I'll admit that the

main reason I write simply is not for the reader's sake or because

it helps get things right, but because it bothers me to use more

or fancier words than I need to. It seems inelegant, like a program

that's too long.I realize florid writing works for some people. But unless you're

sure you're one of them, the best advice is to write as simply as

you can.\_\_\_\_\_

I believe the formula I've given you, importance + novelty +

correctness + strength, is the recipe for a good essay. But I should

warn you that it's also a recipe for making people mad.The root of the problem is novelty. When you tell people something

they didn't know, they don't always thank you for it. Sometimes the

reason people don't know something is because they don't want to

know it. Usually because it contradicts some cherished belief. And

indeed, if you're looking for novel ideas, popular but mistaken

beliefs are a good place to find them. Every popular mistaken belief

creates a dead zone of ideas around

it that are relatively unexplored because they contradict it.The strength component just makes things worse. If there's anything

that annoys people more than having their cherished assumptions

contradicted, it's having them flatly contradicted.Plus if you've used the Morris technique, your writing will seem

quite confident. Perhaps offensively confident, to people who

disagree with you. The reason you'll seem confident is that you are

confident: you've cheated, by only publishing the things you're

sure of. It will seem to people who try to disagree with you that

you never admit you're wrong. In fact you constantly admit you're

wrong. You just do it before publishing instead of after.And if your writing is as simple as possible, that just makes things

worse. Brevity is the diction of command. If you watch someone

delivering unwelcome news from a position of inferiority, you'll

notice they tend to use lots of words, to soften the blow. Whereas

to be short with someone is more or less to be rude to them.It can sometimes work to deliberately phrase statements more weakly

than you mean. To put "perhaps" in front of something you're actually

quite sure of. But you'll notice that when writers do this, they

usually do it with a wink.I don't like to do this too much. It's cheesy to adopt an ironic

tone for a whole essay. I think we just have to face the fact that

elegance and curtness are two names for the same thing.You might think that if you work sufficiently hard to ensure that

an essay is correct, it will be invulnerable to attack. That's sort

of true. It will be invulnerable to valid attacks. But in practice

that's little consolation.In fact, the strength component of useful writing will make you

particularly vulnerable to misrepresentation. If you've stated an

idea as strongly as you could without making it false, all anyone

has to do is to exaggerate slightly what you said, and now it is

false.Much of the time they're not even doing it deliberately. One of the

most surprising things you'll discover, if you start writing essays,

is that people who disagree with you rarely disagree with what

you've actually written. Instead they make up something you said

and disagree with that.For what it's worth, the countermove is to ask someone who does

this to quote a specific sentence or passage you wrote that they

believe is false, and explain why. I say "for what it's worth"

because they never do. So although it might seem that this could

get a broken discussion back on track, the truth is that it was

never on track in the first place.Should you explicitly forestall likely misinterpretations? Yes, if

they're misinterpretations a reasonably smart and well-intentioned

person might make. In fact it's sometimes better to say something

slightly misleading and then add the correction than to try to get

an idea right in one shot. That can be more efficient, and can also

model the way such an idea would be discovered.But I don't think you should explicitly forestall intentional

misinterpretations in the body of an essay. An essay is a place to

meet honest readers. You don't want to spoil your house by putting

bars on the windows to protect against dishonest ones. The place

to protect against intentional misinterpretations is in end-notes.

But don't think you can predict them all. People are as ingenious

at misrepresenting you when you say something they don't want to

hear as they are at coming up with rationalizations for things they

want to do but know they shouldn't. I suspect it's the same skill.\_\_\_\_\_

As with most other things, the way to get better at writing essays

is to practice. But how do you start? Now that we've examined the

structure of useful writing, we can rephrase that question more

precisely. Which constraint do you relax initially? The answer is,

the first component of importance: the number of people who care

about what you write.If you narrow the topic sufficiently, you can probably find something

you're an expert on. Write about that to start with. If you only

have ten readers who care, that's fine. You're helping them, and

you're writing. Later you can expand the breadth of topics you write

about.The other constraint you can relax is a little surprising: publication.

Writing essays doesn't have to mean publishing them. That may seem

strange now that the trend is to publish every random thought, but

it worked for me. I wrote what amounted to essays in notebooks for

about 15 years. I never published any of them and never expected

to. I wrote them as a way of figuring things out. But when the web

came along I'd had a lot of practice.Incidentally,

Steve

Wozniak did the same thing. In high school he

designed computers on paper for fun. He couldn't build them because

he couldn't afford the components. But when Intel launched 4K DRAMs

in 1975, he was ready.\_\_\_\_\_

How many essays are there left to write though? The answer to that

question is probably the most exciting thing I've learned about

essay writing. Nearly all of them are left to write.Although the essay

is an old form, it hasn't been assiduously

cultivated. In the print era, publication was expensive, and there

wasn't enough demand for essays to publish that many. You could

publish essays if you were already well known for writing something

else, like novels. Or you could write book reviews that you took

over to express your own ideas. But there was not really a direct

path to becoming an essayist. Which meant few essays got written,

and those that did tended to be about a narrow range of subjects.Now, thanks to the internet, there's a path. Anyone can publish

essays online. You start in obscurity, perhaps, but at least you

can start. You don't need anyone's permission.It sometimes happens that an area of knowledge sits quietly for

years, till some change makes it explode. Cryptography did this to

number theory. The internet is doing it to the essay.The exciting thing is not that there's a lot left to write, but

that there's a lot left to discover. There's a certain kind of idea

that's best discovered by writing essays. If most essays are still

unwritten, most such ideas are still undiscovered.Notes[1] Put railings on the balconies, but don't put bars on the windows.[2] Even now I sometimes write essays that are not meant for

publication. I wrote several to figure out what Y Combinator should

do, and they were really helpful.Thanks to Trevor Blackwell, Daniel Gackle, Jessica Livingston, and

Robert Morris for reading drafts of this.Spanish TranslationJapanese Translation

Being a Noob

January 2020When I was young, I thought old people had everything figured out.

Now that I'm old, I know this isn't true.I constantly feel like a noob. It seems like I'm always talking to

some startup working in a new field I know nothing about, or reading

a book about a topic I don't understand well enough, or visiting some new

country where I don't know how things work.It's not pleasant to feel like a noob. And the word "noob" is

certainly not a compliment. And yet today I realized something

encouraging about being a noob: the more of a noob you are locally,

the less of a noob you are globally.For example, if you stay in your home country, you'll feel less

of a noob than if you move to Farawavia, where everything works

differently. And yet you'll know more if you move.

So the feeling of being a noob is inversely correlated with actual

ignorance.But if the feeling of being a noob is good for us, why do we dislike

it? What evolutionary purpose could such an aversion serve?I think the answer is that there are two sources of feeling like a

noob: being stupid, and doing something novel. Our dislike of feeling

like a noob is our brain telling us "Come on, come on, figure this

out." Which was the right thing to be thinking for most of human

history. The life of hunter-gatherers was complex, but it didn't

change as much as life does now. They didn't suddenly have to figure

out what to do about cryptocurrency. So it made sense to be biased

toward competence at existing problems over the discovery of new

ones. It made sense for humans to dislike the feeling of being a

noob, just as, in a world where food was scarce, it made sense for

them to dislike the feeling of being hungry.Now that too much food is more of a problem than too little, our

dislike of feeling hungry leads us astray. And I think our dislike

of feeling like a noob does too.Though it feels unpleasant, and people will sometimes ridicule you

for it, the more you feel like a noob, the better.Japanese TranslationArabic TranslationFrench TranslationKorean TranslationPolish TranslationChinese TranslationSerbian TranslationFrench Translation

Haters

January 2020(I originally intended this for startup founders, who are often

surprised by the attention they get as their companies grow, but

it applies equally to anyone who becomes famous.)If you become sufficiently famous, you'll acquire some fans who

like you too much. These people are sometimes called "fanboys," and

though I dislike that term, I'm going to have to use it here. We

need some word for them, because this is a distinct phenomenon from

someone simply liking your work.A fanboy is obsessive and uncritical. Liking you becomes part of

their identity, and they create an image of you in their own head

that is much better than reality. Everything you do is good, because

you do it. If you do something bad, they find a way to see it as

good. And their love for you is not, usually, a quiet, private one.

They want everyone to know how great you are.Well, you may be thinking, I could do without this kind of obsessive

fan, but I know there are all kinds of people in the world, and if

this is the worst consequence of fame, that's not so bad.Unfortunately this is not the worst consequence of fame. As well

as fanboys, you'll have haters.A hater is obsessive and uncritical. Disliking you becomes part of

their identity, and they create an image of you in their own head

that is much worse than reality. Everything you do is bad, because

you do it. If you do something good, they find a way to see it as

bad. And their dislike for you is not, usually, a quiet, private

one. They want everyone to know how awful you are.If you're thinking of checking, I'll save you the trouble. The

second and fifth paragraphs are identical except for "good" being

switched to "bad" and so on.I spent years puzzling about haters. What are they, and where do

they come from? Then one day it dawned on me. Haters are just fanboys

with the sign switched.Note that by haters, I don't simply mean trolls. I'm not talking about

people who say bad things about you and then move on. I'm talking

about the much smaller group of people for whom this becomes a

kind of obsession and who do it repeatedly over a long period.Like fans, haters seem to be an automatic consequence of fame.

Anyone sufficiently famous will have them. And like fans, haters

are energized by the fame of whoever they hate. They hear a song

by some pop singer. They don't like it much. If the singer were an

obscure one, they'd just forget about it. But instead they keep

hearing her name, and this seems to drive some people crazy.

Everyone's always going on about this singer, but she's no good!

She's a fraud!That word "fraud" is an important one. It's the spectral signature

of a hater to regard the object of their hatred as a

fraud. They

can't deny their fame. Indeed, their fame is if anything exaggerated

in the hater's mind. They notice every mention of the singer's name,

because every mention makes them angrier. In their own minds they

exaggerate both the singer's fame and her lack of talent, and the

only way to reconcile those two ideas is to conclude that she has

tricked everyone.What sort of people become haters? Can anyone become one? I'm not

sure about this, but I've noticed some patterns. Haters are generally

losers in a very specific sense: although they are occasionally

talented, they have never achieved much. And indeed, anyone

successful enough to have achieved significant fame would be unlikely

to regard another famous person as a fraud on that account, because

anyone famous knows how random fame is.But haters are not always complete losers. They are not always the

proverbial guy living in his mom's basement. Many are, but some

have some amount of talent. In fact I suspect that a sense of

frustrated talent is what drives some people to become haters.

They're not just saying "It's unfair that so-and-so is famous," but

"It's unfair that so-and-so is famous, and not me."Could a hater be cured if they achieved something impressive? My

guess is that's a moot point, because they

never will. I've been

able to observe for long enough that I'm fairly confident the pattern

works both ways: not only do people who do great work never become

haters, haters never do great work. Although I dislike the word

"fanboy," it's evocative of something important about both haters

and fanboys. It implies that the fanboy is so slavishly predictable in his admiration

that he's diminished as a result, that he's less than a man.Haters seem even more diminished. I can imagine being a fanboy.

I can think of people whose work I admire so much that I could abase

myself before them out of sheer gratitude. If P. G. Wodehouse were

still alive, I could see myself being a Wodehouse fanboy. But I

could not imagine being a hater.Knowing that haters are just fanboys with the sign bit flipped makes

it much easier to deal with them. We don't need a separate theory

of haters. We can just use existing techniques for dealing with

obsessive fans.The most important of which is simply not to think much about them.

If you're like most people who become famous enough to acquire

haters, your initial reaction will be one of mystification. Why

does this guy seem to have it in for me? Where does his obsessive

energy come from, and what makes him so appallingly nasty? What did

I do to set him off? Is it something I can fix?The mistake here is to think of the hater as someone you have a

dispute with. When you have a dispute with someone, it's usually a

good idea to try to understand why they're upset and then fix things

if you can. Disputes are distracting. But it's a false analogy to

think of a hater as someone you have a dispute with. It's an

understandable mistake, if you've never encountered haters before.

But when you realize that you're dealing with a hater, and what a

hater is, it's clear that it's a waste of time even to think about

them. If you have obsessive fans, do you spend any time wondering

what makes them love you so excessively? No, you just think "some

people are kind of crazy," and that's the end of it.Since haters are equivalent to fanboys, that's the way to deal with

them too. There may have been something that set them off. But it's

not something that would have set off a normal person, so there's

no reason to spend any time thinking about it. It's not you, it's

them.Notes[1] There are of course some people who are genuine frauds. How can

you distinguish between x calling y a fraud because x is a hater,

and because y is a fraud? Look at neutral opinion. Actual frauds

are usually pretty conspicuous. Thoughtful people are rarely taken

in by them. So if there are some thoughtful people who like y, you

can usually assume y is not a fraud.[2] I would make an exception for teenagers, who sometimes act in

such extreme ways that they are literally not themselves. I can

imagine a teenage kid being a hater and then growing out of it. But

not anyone over 25.[3] I have a much worse memory for misdeeds than my wife Jessica,

who is a connoisseur of character, but I don't wish it were better.

Most disputes are a waste of time even if you're in the right, and

it's easy to bury the hatchet with someone if you can't remember

why you were mad at them.[4] A competent hater will not merely attack you individually but

will try to get mobs after you. In some cases you may want to refute

whatever bogus claim they made in order to do so. But err on the

side of not, because ultimately it probably won't matter.Thanks to Austen Allred, Trevor Blackwell, Patrick Collison,

Christine Ford, Daniel Gackle, Jessica Livingston, Robert Morris,

Elon Musk, Harj Taggar, and Peter Thiel for reading drafts of this.Japanese TranslationArabic TranslationPolish Translation

The Two Kinds of Moderate

December 2019There are two distinct ways to be politically moderate: on purpose

and by accident. Intentional moderates are trimmers, deliberately

choosing a position mid-way between the extremes of right and left.

Accidental moderates end up in the middle, on average, because they

make up their own minds about each question, and the far right and

far left are roughly equally wrong.You can distinguish intentional from accidental moderates by the

distribution of their opinions. If the far left opinion on some

matter is 0 and the far right opinion 100, an intentional moderate's

opinion on every question will be near 50. Whereas an accidental

moderate's opinions will be scattered over a broad range, but will,

like those of the intentional moderate, average to about 50.Intentional moderates are similar to those on the far left and the

far right in that their opinions are, in a sense, not their own.

The defining quality of an ideologue, whether on the left or the

right, is to acquire one's opinions in bulk. You don't get to pick

and choose. Your opinions about taxation can be predicted from your

opinions about same-sex marriage. And although intentional moderates

might seem to be the opposite of ideologues, their beliefs (though

in their case the word "positions" might be more accurate) are also

acquired in bulk. If the median opinion shifts to the right or left,

the intentional moderate must shift with it. Otherwise they stop

being moderate.Accidental moderates, on the other hand, not only choose their own

answers, but choose their own questions. They may not care at all

about questions that the left and right both think are terribly

important. So you can only even measure the politics of an accidental

moderate from the intersection of the questions they care about and

those the left and right care about, and this can

sometimes be vanishingly small.It is not merely a manipulative rhetorical trick to say "if you're

not with us, you're against us," but often simply false.Moderates are sometimes derided as cowards, particularly by

the extreme left. But while it may be accurate to call intentional

moderates cowards, openly being an accidental moderate requires the

most courage of all, because you get attacked from both right and

left, and you don't have the comfort of being an orthodox member

of a large group to sustain you.Nearly all the most impressive people I know are accidental moderates.

If I knew a lot of professional athletes, or people in the entertainment

business, that might be different. Being on the far left or far

right doesn't affect how fast you run or how well you sing. But

someone who works with ideas has to be independent-minded to do it

well.Or more precisely, you have to be independent-minded about the ideas

you work with. You could be mindlessly doctrinaire in your politics

and still be a good mathematician. In the 20th century, a lot of

very smart people were Marxists just no one who was smart about

the subjects Marxism involves. But if the ideas you use in your

work intersect with the politics of your time, you have two choices:

be an accidental moderate, or be mediocre.Notes[1] It's possible in theory for one side to be entirely right and

the other to be entirely wrong. Indeed, ideologues must always

believe this is the case. But historically it rarely has been.[2] For some reason the far right tend to ignore moderates rather

than despise them as backsliders. I'm not sure why. Perhaps it

means that the far right is less ideological than the far left. Or

perhaps that they are more confident, or more resigned, or simply

more disorganized. I just don't know.[3] Having heretical opinions doesn't mean you have to express

them openly. It may be

easier to have them if you don't.

Thanks to Austen Allred, Trevor Blackwell, Patrick Collison, Jessica Livingston,

Amjad Masad, Ryan Petersen, and Harj Taggar for reading drafts of this.Japanese Translation

Fashionable Problems

December 2019I've seen the same pattern in many different fields: even though

lots of people have worked hard in the field, only a small fraction

of the space of possibilities has been explored, because they've

all worked on similar things.Even the smartest, most imaginative people are surprisingly

conservative when deciding what to work on. People who would never

dream of being fashionable in any other way get sucked into working

on fashionable problems.If you want to try working on unfashionable problems, one of the

best places to look is in fields that people think have already been

fully explored: essays, Lisp, venture funding you may notice a

pattern here. If you can find a new approach into a big but apparently

played out field, the value of whatever you discover will be

multiplied by its enormous surface area.The best protection against getting drawn into working on the same

things as everyone else may be to genuinely

love what you're doing.

Then you'll continue to work on it even if you make the same mistake

as other people and think that it's too marginal to matter.Japanese TranslationArabic TranslationFrench Translation

Having Kids

December 2019Before I had kids, I was afraid of having kids. Up to that point I

felt about kids the way the young Augustine felt about living

virtuously. I'd have been sad to think I'd never have children.

But did I want them now? No.If I had kids, I'd become a parent, and parents, as I'd known since

I was a kid, were uncool. They were dull and responsible and had

no fun. And while it's not surprising that kids would believe that,

to be honest I hadn't seen much as an adult to change my mind.

Whenever I'd noticed parents with kids, the kids seemed to be

terrors, and the parents pathetic harried creatures, even when they

prevailed.When people had babies, I congratulated them enthusiastically,

because that seemed to be what one did. But I didn't feel it at

all. "Better you than me," I was thinking.Now when people have babies I congratulate them enthusiastically and

I mean it. Especially the first one. I feel like they just got the best gift in the world.What changed, of course, is that I had kids. Something I dreaded

turned out to be wonderful.Partly, and I won't deny it, this is because of serious chemical

changes that happened almost instantly when our first child was

born. It was like someone flipped a switch. I suddenly felt

protective not just toward our child, but toward all children. As I was

driving my wife and new son home from the hospital, I approached a

crosswalk full of pedestrians, and I found myself thinking "I have

to be really careful of all these people. Every one of them is

someone's child!"So to some extent you can't trust me when I say having kids is

great. To some extent I'm like a religious cultist telling you

that you'll be happy if you join the cult too but only because

joining the cult will alter your mind in a way that will make you

happy to be a cult member.But not entirely. There were some things

about having kids that I clearly got wrong before I had them.For example, there was a huge amount of selection bias in my

observations of parents and children. Some parents may have noticed

that I wrote "Whenever I'd noticed parents with kids." Of course

the times I noticed kids were when things were going wrong. I only

noticed them when they made noise. And where was I when I noticed

them? Ordinarily I never went to places with kids, so the only

times I encountered them were in shared bottlenecks like airplanes.

Which is not exactly a representative sample. Flying with a toddler

is something very few parents enjoy.What I didn't notice, because they tend to be much quieter, were

all the great moments parents had with kids. People don't talk about

these much the magic is hard to put into words, and all other

parents know about them anyway but one of the great things about

having kids is that there are so many times when you feel there is

nowhere else you'd rather be, and nothing else you'd rather be

doing. You don't have to be doing anything special. You could just

be going somewhere together, or putting them to bed, or pushing

them on the swings at the park. But you wouldn't trade these moments

for anything. One doesn't tend to associate kids with peace, but

that's what you feel. You don't need to look any

further than where you are right now.Before I had kids, I had moments of this kind of peace, but they

were rarer. With kids it can happen several times a day.My other source of data about kids was my own childhood, and that

was similarly misleading. I was pretty bad, and was always in trouble

for something or other. So it seemed to me that parenthood was

essentially law enforcement. I didn't realize there were good times

too.I remember my mother telling me once when I was about 30 that she'd

really enjoyed having me and my sister. My god, I thought, this

woman is a saint. She not only endured all the pain we subjected

her to, but actually enjoyed it? Now I realize she was simply telling

the truth.She said that one reason she liked having us was that we'd been

interesting to talk to. That took me by surprise when I had kids.

You don't just love them. They become your friends too. They're

really interesting. And while I admit small children are disastrously

fond of repetition (anything worth doing once is worth doing fifty

times) it's often genuinely fun to play with them. That surprised

me too. Playing with a 2 year old was fun when I was 2 and definitely

not fun when I was 6. Why would it become fun again later? But it

does.There are of course times that are pure drudgery. Or worse still,

terror. Having kids is one of those intense types of experience

that are hard to imagine unless you've had them. But it is not, as I

implicitly believed before having kids, simply your DNA heading for

the lifeboats.Some of my worries about having kids were right, though. They

definitely make you less productive. I know having kids makes some

people get their act together, but if your act was already together,

you're going to have less time to do it in. In particular, you're

going to have to work to a schedule. Kids have schedules. I'm not

sure if it's because that's how kids are, or because it's the only

way to integrate their lives with adults', but once you have kids,

you tend to have to work on their schedule.You will have chunks of time to work. But you can't let work spill

promiscuously through your whole life, like I used to before I had

kids. You're going to have to work at the same time every day,

whether inspiration is flowing or not, and there are going to be

times when you have to stop, even if it is.I've been able to adapt to working this way. Work, like love, finds

a way. If there are only certain times it can happen, it happens

at those times. So while I don't get as much done as before I had

kids, I get enough done.I hate to say this, because being ambitious has always been a part

of my identity, but having kids may make one less ambitious. It

hurts to see that sentence written down. I squirm to avoid it. But

if there weren't something real there, why would I squirm? The

fact is, once you have kids, you're probably going to care more

about them than you do about yourself. And attention is a zero-sum

game. Only one idea at a time can be the

top idea in your mind.

Once you have kids, it will often be your kids, and that means it

will less often be some project you're working on.I have some hacks for sailing close to this wind. For example, when

I write essays, I think about what I'd want my kids to know. That

drives me to get things right. And when I was writing

Bel, I told

my kids that once I finished it I'd take them to Africa. When you

say that sort of thing to a little kid, they treat it as a promise.

Which meant I had to finish or I'd be taking away their trip to

Africa. Maybe if I'm really lucky such tricks could put me net

ahead. But the wind is there, no question.On the other hand, what kind of wimpy ambition do you have if it

won't survive having kids? Do you have so little to spare?And while having kids may be warping my present judgement, it hasn't

overwritten my memory. I remember perfectly well what life was like

before. Well enough to miss some things a lot, like the

ability to take off for some other country at a moment's notice.

That was so great. Why did I never do that?See what I did there? The fact is, most of the freedom I had before

kids, I never used. I paid for it in loneliness, but I never used

it.I had plenty of happy times before I had kids. But if I count up

happy moments, not just potential happiness but actual happy moments,

there are more after kids than before. Now I practically have it

on tap, almost any bedtime.People's experiences as parents

vary a lot, and I know I've been lucky. But I think the worries I

had before having kids must be pretty common, and judging by other

parents' faces when they see their kids, so must the happiness that

kids bring.

Note[1] Adults are sophisticated enough to see 2 year olds for the

fascinatingly complex characters they are, whereas to most 6 year

olds, 2 year olds are just defective 6 year olds.Thanks to Trevor Blackwell, Jessica Livingston, and Robert Morris

for reading drafts of this.Arabic TranslationSlovak Translation

The Lesson to Unlearn

December 2019

The most damaging thing you learned in school wasn't something you

learned in any specific class. It was learning to get good grades.When I was in college, a particularly earnest philosophy grad student

once told me that he never cared what grade he got in a class, only

what he learned in it. This stuck in my mind because it was the

only time I ever heard anyone say such a thing.For me, as for most students, the measurement of what I was learning

completely dominated actual learning in college. I was fairly

earnest; I was genuinely interested in most of the classes I took,

and I worked hard. And yet I worked by far the hardest when I was

studying for a test.In theory, tests are merely what their name implies: tests of what

you've learned in the class. In theory you shouldn't have to prepare

for a test in a class any more than you have to prepare for a blood

test. In theory you learn from taking the class, from going to the

lectures and doing the reading and/or assignments, and the test

that comes afterward merely measures how well you learned.In practice, as almost everyone reading this will know, things are

so different that hearing this explanation of how classes and tests

are meant to work is like hearing the etymology of a word whose

meaning has changed completely. In practice, the phrase "studying

for a test" was almost redundant, because that was when one really

studied. The difference between diligent and slack students was

that the former studied hard for tests and the latter didn't. No

one was pulling all-nighters two weeks into the semester.Even though I was a diligent student, almost all the work I did in

school was aimed at getting a good grade on something.To many people, it would seem strange that the preceding sentence

has a "though" in it. Aren't I merely stating a tautology? Isn't

that what a diligent student is, a straight-A student? That's how

deeply the conflation of learning with grades has infused our

culture.Is it so bad if learning is conflated with grades? Yes, it is bad.

And it wasn't till decades after college, when I was running Y Combinator, that I realized how bad it is.I knew of course when I was a student that studying for a test is

far from identical with actual learning. At the very least, you

don't retain knowledge you cram into your head the night before an

exam. But the problem is worse than that. The real problem is that

most tests don't come close to measuring what they're supposed to.If tests truly were tests of learning, things wouldn't be so bad.

Getting good grades and learning would converge, just a little late.

The problem is that nearly all tests given to students are terribly

hackable. Most people who've gotten good grades know this, and know

it so well they've ceased even to question it. You'll see when you

realize how naive it sounds to act otherwise.Suppose you're taking a class on medieval history and the final

exam is coming up. The final exam is supposed to be a test of your

knowledge of medieval history, right? So if you have a couple days

between now and the exam, surely the best way to spend the time,

if you want to do well on the exam, is to read the best books you

can find about medieval history. Then you'll know a lot about it,

and do well on the exam.No, no, no, experienced students are saying to themselves. If you

merely read good books on medieval history, most of the stuff you

learned wouldn't be on the test. It's not good books you want to

read, but the lecture notes and assigned reading in this class.

And even most of that you can ignore, because you only have to worry

about the sort of thing that could turn up as a test question.

You're looking for sharply-defined chunks of information. If one

of the assigned readings has an interesting digression on some

subtle point, you can safely ignore that, because it's not the sort

of thing that could be turned into a test question. But if the

professor tells you that there were three underlying causes of the

Schism of 1378, or three main consequences of the Black Death, you'd

better know them. And whether they were in fact the causes or

consequences is beside the point. For the purposes of this class

they are.At a university there are often copies of old exams floating around,

and these narrow still further what you have to learn. As well as

learning what kind of questions this professor asks, you'll often

get actual exam questions. Many professors re-use them. After

teaching a class for 10 years, it would be hard not to, at least

inadvertently.In some classes, your professor will have had some sort of political

axe to grind, and if so you'll have to grind it too. The need for

this varies. In classes in math or the hard sciences or engineering

it's rarely necessary, but at the other end of the spectrum there

are classes where you couldn't get a good grade without it.Getting a good grade in a class on x is so different from learning

a lot about x that you have to choose one or the other, and you

can't blame students if they choose grades. Everyone judges them

by their grades graduate programs, employers, scholarships, even

their own parents.I liked learning, and I really enjoyed some of the papers and

programs I wrote in college. But did I ever, after turning in a

paper in some class, sit down and write another just for fun? Of

course not. I had things due in other classes. If it ever came to

a choice of learning or grades, I chose grades. I hadn't come to

college to do badly.Anyone who cares about getting good grades has to play this game,

or they'll be surpassed by those who do. And at elite universities,

that means nearly everyone, since someone who didn't care about

getting good grades probably wouldn't be there in the first place.

The result is that students compete to maximize the difference

between learning and getting good grades.Why are tests so bad? More precisely, why are they so hackable?

Any experienced programmer could answer that. How hackable is

software whose author hasn't paid any attention to preventing it

from being hacked? Usually it's as porous as a colander.Hackable is the default for any test imposed by an authority. The

reason the tests you're given are so consistently bad so consistently

far from measuring what they're supposed to measure is simply

that the people creating them haven't made much effort to prevent

them from being hacked.But you can't blame teachers if their tests are hackable. Their job

is to teach, not to create unhackable tests. The real problem is

grades, or more precisely, that grades have been overloaded. If

grades were merely a way for teachers to tell students what they

were doing right and wrong, like a coach giving advice to an athlete,

students wouldn't be tempted to hack tests. But unfortunately after

a certain age grades become more than advice. After a certain age,

whenever you're being taught, you're usually also being judged.I've used college tests as an example, but those are actually the

least hackable. All the tests most students take their whole lives

are at least as bad, including, most spectacularly of all, the test

that gets them into college. If getting into college were merely a

matter of having the quality of one's mind measured by admissions

officers the way scientists measure the mass of an object, we could

tell teenage kids "learn a lot" and leave it at that. You can tell

how bad college admissions are, as a test, from how unlike high

school that sounds. In practice, the freakishly specific nature of

the stuff ambitious kids have to do in high school is directly

proportionate to the hackability of college admissions. The classes

you don't care about that are mostly memorization, the random

"extracurricular activities" you have to participate in to show

you're "well-rounded," the standardized tests as artificial as

chess, the "essay" you have to write that's presumably meant to hit

some very specific target, but you're not told what.As well as being bad in what it does to kids, this test is also bad

in the sense of being very hackable. So hackable that whole industries

have grown up to hack it. This is the explicit purpose of test-prep

companies and admissions counsellors, but it's also a significant

part of the function of private schools.Why is this particular test so hackable? I think because of what

it's measuring. Although the popular story is that the way to get

into a good college is to be really smart, admissions officers at

elite colleges neither are, nor claim to be, looking only for that.

What are they looking for? They're looking for people who are not

simply smart, but admirable in some more general sense. And how

is this more general admirableness measured? The admissions officers

feel it. In other words, they accept who they like.So what college admissions is a test of is whether you suit the

taste of some group of people. Well, of course a test like that is

going to be hackable. And because it's both very hackable and there's

(thought to be) a lot at stake, it's hacked like nothing else.

That's why it distorts your life so much for so long.It's no wonder high school students often feel alienated. The shape

of their lives is completely artificial.But wasting your time is not the worst thing the educational system

does to you. The worst thing it does is to train you that the way

to win is by hacking bad tests. This is a much subtler problem

that I didn't recognize until I saw it happening to other people.When I started advising startup founders at Y Combinator, especially

young ones, I was puzzled by the way they always seemed to make

things overcomplicated. How, they would ask, do you raise money?

What's the trick for making venture capitalists want to invest in

you? The best way to make VCs want to invest in you, I would explain,

is to actually be a good investment. Even if you could trick VCs

into investing in a bad startup, you'd be tricking yourselves too.

You're investing time in the same company you're asking them to

invest money in. If it's not a good investment, why are you even

doing it?Oh, they'd say, and then after a pause to digest this revelation,

they'd ask: What makes a startup a good investment?So I would explain that what makes a startup promising, not just

in the eyes of investors but in fact, is

growth.

Ideally in revenue,

but failing that in usage. What they needed to do was get lots of

users.How does one get lots of users? They had all kinds of ideas about

that. They needed to do a big launch that would get them "exposure."

They needed influential people to talk about them. They even knew

they needed to launch on a tuesday, because that's when one gets

the most attention.No, I would explain, that is not how to get lots of users. The way

you get lots of users is to make the product really great. Then

people will not only use it but recommend it to their friends, so

your growth will be exponential once you

get it started.At this point I've told the founders something you'd think would

be completely obvious: that they should make a good company by

making a good product. And yet their reaction would be something

like the reaction many physicists must have had when they first

heard about the theory of relativity: a mixture of astonishment at

its apparent genius, combined with a suspicion that anything so

weird couldn't possibly be right. Ok, they would say, dutifully.

And could you introduce us to such-and-such influential person? And

remember, we want to launch on Tuesday.It would sometimes take founders years to grasp these simple lessons.

And not because they were lazy or stupid. They just seemed blind

to what was right in front of them.Why, I would ask myself, do they always make things so complicated?

And then one day I realized this was not a rhetorical question.Why did founders tie themselves in knots doing the wrong things

when the answer was right in front of them? Because that was what

they'd been trained to do. Their education had taught them that the

way to win was to hack the test. And without even telling them they

were being trained to do this. The younger ones, the recent graduates,

had never faced a non-artificial test. They thought this was just

how the world worked: that the first thing you did, when facing any

kind of challenge, was to figure out what the trick was for hacking

the test. That's why the conversation would always start with how

to raise money, because that read as the test. It came at the end

of YC. It had numbers attached to it, and higher numbers seemed to

be better. It must be the test.There are certainly big chunks of the world where the way to win

is to hack the test. This phenomenon isn't limited to schools. And

some people, either due to ideology or ignorance, claim that this

is true of startups too. But it isn't. In fact, one of the most

striking things about startups is the degree to which you win by

simply doing good work. There are edge cases, as there are in

anything, but in general you win by getting users, and what users

care about is whether the product does what they want.Why did it take me so long to understand why founders made startups

overcomplicated? Because I hadn't realized explicitly that schools

train us to win by hacking bad tests. And not just them, but me!

I'd been trained to hack bad tests too, and hadn't realized it till

decades later.I had lived as if I realized it, but without knowing why. For

example, I had avoided working for big companies. But if you'd asked

why, I'd have said it was because they were bogus, or bureaucratic.

Or just yuck. I never understood how much of my dislike of big

companies was due to the fact that you win by hacking bad tests.Similarly, the fact that the tests were unhackable was a lot of

what attracted me to startups. But again, I hadn't realized that

explicitly.I had in effect achieved by successive approximations something

that may have a closed-form solution. I had gradually undone my

training in hacking bad tests without knowing I was doing it. Could

someone coming out of school banish this demon just by knowing its

name, and saying begone? It seems worth trying.Merely talking explicitly about this phenomenon is likely to make

things better, because much of its power comes from the fact that

we take it for granted. After you've noticed it, it seems the

elephant in the room, but it's a pretty well camouflaged elephant.

The phenomenon is so old, and so pervasive. And it's simply the

result of neglect. No one meant things to be this way. This is just

what happens when you combine learning with grades, competition,

and the naive assumption of unhackability.It was mind-blowing to realize that two of the things I'd puzzled

about the most the bogusness of high school, and the difficulty

of getting founders to see the obvious both had the same cause.

It's rare for such a big block to slide into place so late.Usually when that happens it has implications in a lot of different

areas, and this case seems no exception. For example, it suggests

both that education could be done better, and how you might fix it.

But it also suggests a potential answer to the question all big

companies seem to have: how can we be more like a startup? I'm not

going to chase down all the implications now. What I want to focus

on here is what it means for individuals.To start with, it means that most ambitious kids graduating from

college have something they may want to unlearn. But it also changes

how you look at the world. Instead of looking at all the different

kinds of work people do and thinking of them vaguely as more or

less appealing, you can now ask a very specific question that will

sort them in an interesting way: to what extent do you win at this

kind of work by hacking bad tests?It would help if there was a way to recognize bad tests quickly.

Is there a pattern here? It turns out there is.Tests can be divided into two kinds: those that are imposed by

authorities, and those that aren't. Tests that aren't imposed by

authorities are inherently unhackable, in the sense that no one is

claiming they're tests of anything more than they actually test. A

football match, for example, is simply a test of who wins, not which

team is better. You can tell that from the fact that commentators

sometimes say afterward that the better team won. Whereas tests

imposed by authorities are usually proxies for something else. A

test in a class is supposed to measure not just how well you did

on that particular test, but how much you learned in the class.

While tests that aren't imposed by authorities are inherently

unhackable, those imposed by authorities have to be made unhackable.

Usually they aren't. So as a first approximation, bad tests are

roughly equivalent to tests imposed by authorities.You might actually like to win by hacking bad tests. Presumably

some people do. But I bet most people who find themselves doing

this kind of work don't like it. They just take it for granted that

this is how the world works, unless you want to drop out and be

some kind of hippie artisan.I suspect many people implicitly assume that working in a

field with bad tests is the price of making lots of money. But that,

I can tell you, is false. It used to be true. In the mid-twentieth

century, when the economy was

composed of oligopolies,

the only way

to the top was by playing their game. But it's not true now. There

are now ways to get rich by doing good work, and that's part of the

reason people are so much more excited about getting rich than they

used to be. When I was a kid, you could either become an engineer

and make cool things, or make lots of money by becoming an "executive."

Now you can make lots of money by making cool things.Hacking bad tests is becoming less important as the link between

work and authority erodes. The erosion of that link is one of the

most important trends happening now, and we see its effects in

almost every kind of work people do. Startups are one of the most

visible examples, but we see much the same thing in writing. Writers

no longer have to submit to publishers and editors to reach readers;

now they can go direct.The more I think about this question, the more optimistic I get.

This seems one of those situations where we don't realize how much

something was holding us back until it's eliminated. And I can

foresee the whole bogus edifice crumbling. Imagine what happens as

more and more people start to ask themselves if they want to win

by hacking bad tests, and decide that they don't. The kinds of

work where you win by hacking bad tests will be starved of talent,

and the kinds where you win by doing good work will see an influx

of the most ambitious people. And as hacking bad tests shrinks in

importance, education will evolve to stop training us to do it.

Imagine what the world could look like if that happened.This is not just a lesson for individuals to unlearn, but one for

society to unlearn, and we'll be amazed at the energy that's liberated

when we do.

Notes[1] If using tests only to measure learning sounds impossibly

utopian, that is already the way things work at Lambda School.

Lambda School doesn't have grades. You either graduate or you don't.

The only purpose of tests is to decide at each stage of the curriculum

whether you can continue to the next. So in effect the whole school

is pass/fail.[2] If the final exam consisted of a long conversation with the

professor, you could prepare for it by reading good books on medieval

history. A lot of the hackability of tests in schools is due to the

fact that the same test has to be given to large numbers of students.[3] Learning is the naive algorithm for getting good grades.[4] Hacking has

multiple senses. There's a narrow sense in which

it means to compromise something. That's the sense in which one

hacks a bad test. But there's another, more general sense, meaning

to find a surprising solution to a problem, often by thinking

differently about it. Hacking in this sense is a wonderful thing.

And indeed, some of the hacks people use on bad tests are impressively

ingenious; the problem is not so much the hacking as that, because

the tests are hackable, they don't test what they're meant to.[5] The people who pick startups at Y Combinator are similar to

admissions officers, except that instead of being arbitrary, their

acceptance criteria are trained by a very tight feedback loop. If

you accept a bad startup or reject a good one, you will usually know it

within a year or two at the latest, and often within a month.[6] I'm sure admissions officers are tired of reading applications

from kids who seem to have no personality beyond being willing to

seem however they're supposed to seem to get accepted. What they

don't realize is that they are, in a sense, looking in a mirror.

The lack of authenticity in the applicants is a reflection of the

arbitrariness of the application process. A dictator might just as

well complain about the lack of authenticity in the people around

him.[7] By good work, I don't mean morally good, but good in the sense

in which a good craftsman does good work.[8] There are borderline cases where it's hard to say which category

a test falls in. For example, is raising venture capital like college

admissions, or is it like selling to a customer?[9] Note that a good test is merely one that's unhackable. Good

here doesn't mean morally good, but good in the sense of working

well. The difference between fields with bad tests and good ones

is not that the former are bad and the latter are good, but that

the former are bogus and the latter aren't. But those two measures

are not unrelated. As Tara Ploughman said, the path from good to

evil goes through bogus.[10] People who think the recent increase in

economic inequality is

due to changes in tax policy seem very naive to anyone with experience

in startups. Different people are getting rich now than used to,

and they're getting much richer than mere tax savings could make

them.[11] Note to tiger parents: you may think you're training your kids

to win, but if you're training them to win by hacking bad tests,

you are, as parents so often do, training them to fight the last

war.Thanks to Austen Allred, Trevor Blackwell, Patrick Collison,

Jessica Livingston, Robert Morris, and Harj Taggar for reading

drafts of this.Russian TranslationArabic TranslationSwedish Translation

Novelty and Heresy

November 2019If you discover something new, there's a significant chance you'll be

accused of some form of heresy.To discover new things, you have

to work on ideas that are good but non-obvious; if an idea is

obviously good, other people are probably already working on it.

One common way for a good idea to be non-obvious is for it to be hidden in the

shadow of some mistaken assumption that people are very attached to.

But anything you discover from working on such an idea will tend to

contradict the mistaken assumption that was concealing it. And you

will thus get a lot of heat from people attached to the mistaken

assumption. Galileo and Darwin are famous examples of this phenomenon,

but it's probably always an ingredient in the resistance to new

ideas.So it's particularly dangerous for an organization or society to

have a culture of pouncing on heresy. When you suppress heresies,

you don't just prevent people from contradicting the mistaken

assumption you're trying to protect. You also suppress any idea

that implies indirectly that it's false.

Every cherished mistaken assumption has

a dead zone of unexplored ideas around it. And the more preposterous

the assumption, the bigger the dead zone it creates.There is a positive side to this phenomenon though. If you're

looking for new ideas, one way to find them is by looking for

heresies. When you look at the question this way, the depressingly

large dead zones around mistaken assumptions become excitingly large

mines of new ideas.Japanese TranslationRussian TranslationSimplified Chinese Translation

The Bus Ticket Theory of Genius

November 2019Everyone knows that to do great work you need both natural ability

and determination. But there's a third ingredient that's not as

well understood: an obsessive interest in a particular topic.To explain this point I need to burn my reputation with some group

of people, and I'm going to choose bus ticket collectors. There

are people who collect old bus tickets. Like many collectors, they

have an obsessive interest in the minutiae of what they collect.

They can keep track of distinctions between different types of bus

tickets that would be hard for the rest of us to remember. Because

we don't care enough. What's the point of spending so much time

thinking about old bus tickets?Which leads us to the second feature of this kind of obsession:

there is no point. A bus ticket collector's love is disinterested.

They're not doing it to impress us or to make themselves rich, but

for its own sake.When you look at the lives of people who've done great work, you

see a consistent pattern. They often begin with a bus ticket

collector's obsessive interest in something that would have seemed

pointless to most of their contemporaries. One of the most striking

features of Darwin's book about his voyage on the Beagle is the

sheer depth of his interest in natural history. His curiosity seems

infinite. Ditto for Ramanujan, sitting by the hour working out on

his slate what happens to series.It's a mistake to think they were "laying the groundwork" for the

discoveries they made later. There's too much intention in that

metaphor. Like bus ticket collectors, they were doing it

because they liked it.But there is a difference between Ramanujan and a bus ticket

collector. Series matter, and bus tickets don't.If I had to put the recipe for genius into one sentence, that might

be it: to have a disinterested obsession with something that matters.Aren't I forgetting about the other two ingredients? Less than you

might think. An obsessive interest in a topic is both a proxy for

ability and a substitute for determination. Unless you have

sufficient mathematical aptitude, you won't find series interesting.

And when you're obsessively interested in something, you don't need

as much determination: you don't need to push yourself as hard when

curiosity is pulling you.An obsessive interest will even bring you luck, to the extent

anything can. Chance, as Pasteur said, favors the prepared mind,

and if there's one thing an obsessed mind is, it's prepared.The disinterestedness of this kind of obsession is its most important

feature. Not just because it's a filter for earnestness, but because

it helps you discover new ideas.The paths that lead to new ideas tend to look unpromising. If they

looked promising, other people would already have explored them.

How do the people who do great work discover these paths that others

overlook? The popular story is that they simply have better vision:

because they're so talented, they see paths that others miss. But

if you look at the way great discoveries are made, that's not what

happens. Darwin didn't pay closer attention to individual species

than other people because he saw that this would lead to great

discoveries, and they didn't. He was just really, really interested

in such things.Darwin couldn't turn it off. Neither could Ramanujan. They didn't

discover the hidden paths that they did because they seemed promising,

but because they couldn't help it. That's what allowed them to

follow paths that someone who was merely ambitious would have

ignored.What rational person would decide that the way to write great novels

was to begin by spending several years creating an imaginary elvish

language, like Tolkien, or visiting every household in southwestern

Britain, like Trollope? No one, including Tolkien and Trollope.The bus ticket theory is similar to Carlyle's famous definition of

genius as an infinite capacity for taking pains. But there are two

differences. The bus ticket theory makes it clear that the source

of this infinite capacity for taking pains is not infinite diligence,

as Carlyle seems to have meant, but the sort of infinite interest

that collectors have. It also adds an important qualification: an

infinite capacity for taking pains about something that matters.So what matters? You can never be sure. It's precisely because no

one can tell in advance which paths are promising that you can

discover new ideas by working on what you're interested in.But there are some heuristics you can use to guess whether an

obsession might be one that matters. For example, it's more promising

if you're creating something, rather than just consuming something

someone else creates. It's more promising if something you're

interested in is difficult, especially if it's more difficult for

other people than it is for you. And the obsessions of talented

people are more likely to be promising. When talented people become

interested in random things, they're not truly random.But you can never be sure. In fact, here's an interesting idea

that's also rather alarming if it's true: it may be that to do great

work, you also have to waste a lot of time.In many different areas, reward is proportionate to risk. If that

rule holds here, then the way to find paths that lead to truly great

work is to be willing to expend a lot of effort on things that turn

out to be every bit as unpromising as they seem.I'm not sure if this is true. On one hand, it seems surprisingly

difficult to waste your time so long as you're working hard on

something interesting. So much of what you do ends up being useful.

But on the other hand, the rule about the relationship between risk

and reward is so powerful that it seems to hold wherever risk occurs.

Newton's case, at least, suggests that the risk/reward rule holds

here. He's famous for one particular obsession of his that turned

out to be unprecedentedly fruitful: using math to describe the

world. But he had two other obsessions, alchemy and theology, that

seem to have been complete wastes of time. He ended up net ahead.

His bet on what we now call physics paid off so well that it more

than compensated for the other two. But were the other two necessary,

in the sense that he had to take big risks to make such big

discoveries? I don't know.Here's an even more alarming idea: might one make all bad bets? It

probably happens quite often. But we don't know how often, because

these people don't become famous.It's not merely that the returns from following a path are hard to

predict. They change dramatically over time. 1830 was a really good

time to be obsessively interested in natural history. If Darwin had

been born in 1709 instead of 1809, we might never have heard of

him.What can one do in the face of such uncertainty? One solution is

to hedge your bets, which in this case means to follow the obviously

promising paths instead of your own private obsessions. But as with

any hedge, you're decreasing reward when you decrease risk. If you

forgo working on what you like in order to follow some more

conventionally ambitious path, you might miss something wonderful

that you'd otherwise have discovered. That too must happen all the

time, perhaps even more often than the genius whose bets all fail.The other solution is to let yourself be interested in lots of

different things. You don't decrease your upside if you switch

between equally genuine interests based on which seems to be working

so far. But there is a danger here too: if you work on too many

different projects, you might not get deeply enough into any of

them.One interesting thing about the bus ticket theory is that it may

help explain why different types of people excel at different kinds

of work. Interest is much more unevenly distributed than ability.

If natural ability is all you need to do great work, and natural

ability is evenly distributed, you have to invent elaborate theories

to explain the skewed distributions we see among those who actually

do great work in various fields. But it may be that much of the

skew has a simpler explanation: different people are interested in

different things.The bus ticket theory also explains why people are less likely to

do great work after they have children. Here interest has to compete

not just with external obstacles, but with another interest, and

one that for most people is extremely powerful. It's harder to find

time for work after you have kids, but that's the easy part. The

real change is that you don't want to.But the most exciting implication of the bus ticket theory is that

it suggests ways to encourage great work. If the recipe for genius

is simply natural ability plus hard work, all we can do is hope we

have a lot of ability, and work as hard as we can. But if interest

is a critical ingredient in genius, we may be able, by cultivating

interest, to cultivate genius.For example, for the very ambitious, the bus ticket theory suggests

that the way to do great work is to relax a little. Instead of

gritting your teeth and diligently pursuing what all your peers

agree is the most promising line of research, maybe you should try

doing something just for fun. And if you're stuck, that may be the

vector along which to break out.I've always liked Hamming's famous double-barrelled question: what

are the most important problems in your field, and why aren't you

working on one of them? It's a great way to shake yourself up. But

it may be overfitting a bit. It might be at least as useful to ask

yourself: if you could take a year off to work on something that

probably wouldn't be important but would be really interesting,

what would it be?The bus ticket theory also suggests a way to avoid slowing down as

you get older. Perhaps the reason people have fewer new ideas as

they get older is not simply that they're losing their edge. It may

also be because once you become established, you can no longer mess

about with irresponsible side projects the way you could when you

were young and no one cared what you did.The solution to that is obvious: remain irresponsible. It will be

hard, though, because the apparently random projects you take up

to stave off decline will read to outsiders as evidence of it. And

you yourself won't know for sure that they're wrong. But it will

at least be more fun to work on what you want.It may even be that we can cultivate a habit of intellectual bus

ticket collecting in kids. The usual plan in education is to start

with a broad, shallow focus, then gradually become more specialized.

But I've done the opposite with my kids. I know I can count on their

school to handle the broad, shallow part, so I take them deep.When they get interested in something, however random, I encourage

them to go preposterously, bus ticket collectorly, deep. I don't

do this because of the bus ticket theory. I do it because I want

them to feel the joy of learning, and they're never going to feel

that about something I'm making them learn. It has to be something

they're interested in. I'm just following the path of least resistance;

depth is a byproduct. But if in trying to show them the joy of

learning I also end up training them to go deep, so much the better.Will it have any effect? I have no idea. But that uncertainty may

be the most interesting point of all. There is so much more to learn

about how to do great work. As old as human civilization feels,

it's really still very young if we haven't nailed something so

basic. It's exciting to think there are still discoveries to make

about discovery. If that's the sort of thing you're interested in.

Notes[1] There are other types of collecting that illustrate this point

better than bus tickets, but they're also more popular. It seemed

just as well to use an inferior example rather than offend more

people by telling them their hobby doesn't matter.[2] I worried a little about using the word "disinterested," since

some people mistakenly believe it means not interested. But anyone

who expects to be a genius will have to know the meaning of such a

basic word, so I figure they may as well start now.[3] Think how often genius must have been nipped in the bud by

people being told, or telling themselves, to stop messing about and

be responsible. Ramanujan's mother was a huge enabler. Imagine if

she hadn't been. Imagine if his parents had made him go out and get

a job instead of sitting around at home doing math.On the other hand, anyone quoting the preceding paragraph to justify

not getting a job is probably mistaken.[4] 1709 Darwin is to time what the Milanese Leonardo is to space.[5] "An infinite capacity for taking pains" is a paraphrase of what

Carlyle wrote. What he wrote, in his History of Frederick the Great,

was "... it is the fruit of 'genius' (which means transcendent

capacity of taking trouble, first of all)...." Since the paraphrase

seems the name of the idea at this point, I kept it.Carlyle's History was published in 1858. In 1785 Hérault de Séchelles

quoted Buffon as saying "Le génie n'est qu'une plus grande aptitude

à la patience." (Genius is only a greater aptitude for patience.)[6] Trollope was establishing the system of postal routes. He himself

sensed the obsessiveness with which he pursued this goal.

It is amusing to watch how a passion will grow upon a man. During

those two years it was the ambition of my life to cover the

country with rural letter-carriers.

Even Newton occasionally sensed the degree of his obsessiveness.

After computing pi to 15 digits, he wrote in a letter to a friend:

I am ashamed to tell you to how many figures I carried these

computations, having no other business at the time.

Incidentally, Ramanujan was also a compulsive calculator. As Kanigel

writes in his excellent biography:

One Ramanujan scholar, B. M. Wilson, later told how Ramanujan's

research into number theory was often "preceded by a table of

numerical results, carried usually to a length from which most

of us would shrink."

[7] Working to understand the natural world counts as creating

rather than consuming.Newton tripped over this distinction when he chose

to work on theology. His beliefs did not allow him to see it, but

chasing down paradoxes in nature is fruitful in a way that chasing

down paradoxes in sacred texts is not.[8] How much of people's propensity to become interested in a topic

is inborn? My experience so far suggests the answer is: most of

it. Different kids get interested in different things, and it's

hard to make a child interested in something they wouldn't otherwise

be. Not in a way that sticks. The most you can do on behalf of a

topic is to make sure it gets a fair showing to make it clear to

them, for example, that there's more to math than the dull drills

they do in school. After that it's up to the child.Thanks to Marc Andreessen, Trevor Blackwell, Patrick Collison, Kevin

Lacker, Jessica Livingston, Jackie McDonough, Robert Morris, Lisa

Randall, Zak Stone, and my 7 year old for reading drafts of this.Spanish TranslationRussian TranslationKorean Translation

General and Surprising

September 2017The most valuable insights are both general and surprising.

F = ma for example. But general and surprising is a hard

combination to achieve. That territory tends to be picked

clean, precisely because those insights are so valuable.Ordinarily, the best that people can do is one without the

other: either surprising without being general (e.g.

gossip), or general without being surprising (e.g.

platitudes).Where things get interesting is the moderately valuable

insights. You get those from small additions of whichever

quality was missing. The more common case is a small

addition of generality: a piece of gossip that's more than

just gossip, because it teaches something interesting about

the world. But another less common approach is to focus on

the most general ideas and see if you can find something new

to say about them. Because these start out so general, you

only need a small delta of novelty to produce a useful

insight.A small delta of novelty is all you'll be able to get most

of the time. Which means if you take this route, your ideas

will seem a lot like ones that already exist. Sometimes

you'll find you've merely rediscovered an idea that did

already exist. But don't be discouraged. Remember the huge

multiplier that kicks in when you do manage to think of

something even a little new.Corollary: the more general the ideas you're talking about,

the less you should worry about repeating yourself. If you

write enough, it's inevitable you will. Your brain is much

the same from year to year and so are the stimuli that hit

it. I feel slightly bad when I find I've said something

close to what I've said before, as if I were plagiarizing

myself. But rationally one shouldn't. You won't say

something exactly the same way the second time, and that

variation increases the chance you'll get that tiny but

critical delta of novelty.And of course, ideas beget ideas. (That sounds

familiar.)

An idea with a small amount of novelty could lead to one

with more. But only if you keep going. So it's doubly

important not to let yourself be discouraged by people who

say there's not much new about something you've discovered.

"Not much new" is a real achievement when you're talking

about the most general ideas. It's not true that there's nothing new under the sun. There

are some domains where there's almost nothing new. But

there's a big difference between nothing and almost nothing,

when it's multiplied by the area under the sun.

Thanks to Sam Altman, Patrick Collison, and Jessica

Livingston for reading drafts of this.Japanese Translation

Charisma / Power

January 2017People who are powerful but uncharismatic will tend to be disliked.

Their power makes them a target for criticism that they don't have

the charisma to disarm. That was Hillary Clinton's problem. It also

tends to be a problem for any CEO who is more of a builder than a

schmoozer. And yet the builder-type CEO is (like Hillary) probably

the best person for the job.I don't think there is any solution to this problem. It's human

nature. The best we can do is to recognize that it's happening, and

to understand that being a magnet for criticism is sometimes a sign

not that someone is the wrong person for a job, but that they're

the right one.

The Risk of Discovery

January 2017Because biographies of famous scientists tend to

edit out their mistakes, we underestimate the

degree of risk they were willing to take.

And because anything a famous scientist did that

wasn't a mistake has probably now become the

conventional wisdom, those choices don't

seem risky either.Biographies of Newton, for example, understandably focus

more on physics than alchemy or theology.

The impression we get is that his unerring judgment

led him straight to truths no one else had noticed.

How to explain all the time he spent on alchemy

and theology? Well, smart people are often kind of

crazy.But maybe there is a simpler explanation. Maybe

the smartness and the craziness were not as separate

as we think. Physics seems to us a promising thing

to work on, and alchemy and theology obvious wastes

of time. But that's because we know how things

turned out. In Newton's day the three problems

seemed roughly equally promising. No one knew yet

what the payoff would be for inventing what we

now call physics; if they had, more people would

have been working on it. And alchemy and theology

were still then in the category Marc Andreessen would

describe as "huge, if true."Newton made three bets. One of them worked. But

they were all risky.Japanese Translation

How to Make Pittsburgh a Startup Hub

April 2016(This is a talk I gave at an event called Opt412 in Pittsburgh.

Much of it will apply to other towns. But not all, because

as I say in the talk, Pittsburgh has some important advantages over

most would-be startup hubs.)What would it take to make Pittsburgh into a startup hub, like

Silicon Valley? I understand Pittsburgh pretty well,

because I grew up here, in Monroeville. And I understand Silicon

Valley pretty well because that's where I live now. Could you get

that kind of startup ecosystem going here?When I agreed to speak here, I didn't think I'd be able to give a

very optimistic talk. I thought I'd be talking about what Pittsburgh

could do to become a startup hub, very much in the subjunctive.

Instead I'm going to talk about what Pittsburgh can do.What changed my mind was an article I read in, of all places, the New

York Times food section. The title was "Pittsburgh's Youth-Driven

Food Boom." To most people that might not even sound interesting,

let alone something related to startups. But it was electrifying

to me to read that title. I don't think I could pick a more promising

one if I tried. And when I read the article I got even more excited.

It said "people ages 25 to 29 now make up 7.6 percent of all

residents, up from 7 percent about a decade ago." Wow, I thought,

Pittsburgh could be the next Portland. It could become the cool

place all the people in their twenties want to go live.When I got here a couple days ago, I could feel the difference. I

lived here from 1968 to 1984. I didn't realize it at the time, but

during that whole period the city was in free fall. On top of the

flight to the suburbs that happened everywhere, the steel and nuclear

businesses were both dying. Boy are things different now. It's not

just that downtown seems a lot more prosperous. There is an energy

here that was not here when I was a kid.When I was a kid, this was a place young people left. Now it's a

place that attracts them.What does that have to do with startups? Startups are made

of people, and the average age of the people in a typical startup

is right in that 25 to 29 bracket.I've seen how powerful it is for a city to have those people. Five

years ago they shifted the center of gravity of Silicon Valley from

the peninsula to San Francisco. Google and Facebook are on the

peninsula, but the next generation of big winners are all in SF.

The reason the center of gravity shifted was the talent war, for

programmers especially. Most 25 to 29 year olds want to live in

the city, not down in the boring suburbs. So whether they like it

or not, founders know they have to be in the city. I know multiple

founders who would have preferred to live down in the Valley proper,

but who made themselves move to SF because they knew otherwise

they'd lose the talent war.So being a magnet for people in their twenties is a very promising

thing to be. It's hard to imagine a place becoming a startup hub

without also being that. When I read that statistic about the

increasing percentage of 25 to 29 year olds, I had exactly the same

feeling of excitement I get when I see a startup's graphs start to

creep upward off the x axis.Nationally the percentage of 25 to 29 year olds is 6.8%. That means

you're .8% ahead. The population is 306,000, so we're talking about

a surplus of about 2500 people. That's the population of a small

town, and that's just the surplus. So you have a toehold. Now you

just have to expand it.And though "youth-driven food boom" may sound frivolous, it is

anything but. Restaurants and cafes are a big part of the personality

of a city. Imagine walking down a street in Paris. What are you

walking past? Little restaurants and cafes. Imagine driving through

some depressing random exurb. What are you driving past? Starbucks

and McDonalds and Pizza Hut. As Gertrude Stein said, there is no

there there. You could be anywhere.These independent restaurants and cafes are not just feeding people.

They're making there be a there here.So here is my first concrete recommendation for turning Pittsburgh

into the next Silicon Valley: do everything you can to encourage

this youth-driven food boom. What could the city do? Treat the

people starting these little restaurants and cafes as your users,

and go ask them what they want. I can guess at least one thing

they might want: a fast permit process. San Francisco has left you

a huge amount of room to beat them in that department.I know restaurants aren't the prime mover though. The prime mover,

as the Times article said, is cheap housing. That's a big advantage.

But that phrase "cheap housing" is a bit misleading. There are

plenty of places that are cheaper. What's special about Pittsburgh

is not that it's cheap, but that it's a cheap place you'd actually

want to live.Part of that is the buildings themselves. I realized a long time

ago, back when I was a poor twenty-something myself, that the best

deals were places that had once been rich, and then became poor.

If a place has always been rich, it's nice but too expensive. If

a place has always been poor, it's cheap but grim. But if a place

was once rich and then got poor, you can find palaces for cheap.

And that's what's bringing people here. When Pittsburgh was rich,

a hundred years ago, the people who lived here built big solid

buildings. Not always in the best taste, but definitely solid. So

here is another piece of advice for becoming a startup hub: don't

destroy the buildings that are bringing people here. When cities

are on the way back up, like Pittsburgh is now, developers race to

tear down the old buildings. Don't let that happen. Focus on

historic preservation. Big real estate development projects are

not what's bringing the twenty-somethings here. They're the opposite

of the new restaurants and cafes; they subtract personality from

the city.The empirical evidence suggests you cannot be too strict about

historic preservation. The tougher cities are about it, the better

they seem to do.But the appeal of Pittsburgh is not just the buildings themselves.

It's the neighborhoods they're in. Like San Francisco and New York,

Pittsburgh is fortunate in being a pre-car city. It's not too

spread out. Because those 25 to 29 year olds do not like driving.

They prefer walking, or bicycling, or taking public transport. If

you've been to San Francisco recently you can't help noticing the

huge number of bicyclists. And this is not just a fad that the

twenty-somethings have adopted. In this respect they have discovered

a better way to live. The beards will go, but not the bikes. Cities

where you can get around without driving are just better period.

So I would suggest you do everything you can to capitalize on this.

As with historic preservation, it seems impossible to go too far.Why not make Pittsburgh the most bicycle and pedestrian friendly

city in the country? See if you can go so far that you make San

Francisco seem backward by comparison. If you do, it's very unlikely

you'll regret it. The city will seem like a paradise to the young

people you want to attract. If they do leave to get jobs elsewhere,

it will be with regret at leaving behind such a place. And what's

the downside? Can you imagine a headline "City ruined by becoming

too bicycle-friendly?" It just doesn't happen.So suppose cool old neighborhoods and cool little restaurants make

this the next Portland. Will that be enough? It will put you in

a way better position than Portland itself, because Pittsburgh has

something Portland lacks: a first-rate research university. CMU

plus little cafes means you have more than hipsters drinking lattes.

It means you have hipsters drinking lattes while talking about

distributed systems. Now you're getting really close to San

Francisco.In fact you're better off than San Francisco in one way, because

CMU is downtown, but Stanford and Berkeley are out in the suburbs.What can CMU do to help Pittsburgh become a startup hub? Be an

even better research university. CMU is one of the best universities

in the world, but imagine what things would be like if it were the

very best, and everyone knew it. There are a lot of ambitious

people who must go to the best place, wherever it is. If CMU were it, they would all come here. There would be

kids in Kazakhstan dreaming of one day living in Pittsburgh.Being that kind of talent magnet is the most important contribution

universities can make toward making their city a startup hub. In

fact it is practically the only contribution they can make.But wait, shouldn't universities be setting up programs with words

like "innovation" and "entrepreneurship" in their names? No, they

should not. These kind of things almost always turn out to be

disappointments. They're pursuing the wrong targets. The way to

get innovation is not to aim for innovation but to aim for something

more specific, like better batteries or better 3D printing. And

the way to learn about entrepreneurship is to do it, which you

can't

in school.I know it may disappoint some administrators to hear that the best

thing a university can do to encourage startups is to be a great

university. It's like telling people who want to lose weight that

the way to do it is to eat less.But if you want to know where startups come from, look at the

empirical evidence. Look at the histories of the most successful

startups, and you'll find they grow organically out of a couple of

founders building something that starts as an interesting side

project. Universities are great at bringing together founders, but

beyond that the best thing they can do is get out of the way. For

example, by not claiming ownership of "intellectual property" that

students and faculty develop, and by having liberal rules about

deferred admission and leaves of absence.In fact, one of the most effective things a university could do to

encourage startups is an elaborate form of getting out of the way

invented by Harvard. Harvard used to have exams for the fall

semester after Christmas. At the beginning of January they had

something called "Reading Period" when you were supposed to be

studying for exams. And Microsoft and Facebook have something in

common that few people realize: they were both started during Reading

Period. It's the perfect situation for producing the sort of side

projects that turn into startups. The students are all on campus,

but they don't have to do anything because they're supposed to be

studying for exams.Harvard may have closed this window, because a few years ago they

moved exams before Christmas and shortened reading period from 11

days to 7. But if a university really wanted to help its students

start startups, the empirical evidence, weighted by market cap,

suggests the best thing they can do is literally nothing.The culture of Pittsburgh is another of its strengths. It seems

like a city has to be socially liberal to be a startup hub,

and it's pretty clear why. A city has to tolerate strangeness to

be a home for startups, because startups are so strange. And you

can't choose to allow just the forms of strangeness that will turn

into big startups, because they're all intermingled. You have to

tolerate all strangeness.That immediately rules out big chunks of the US. I'm optimistic

it doesn't rule out Pittsburgh. One of the things I remember from

growing up here, though I didn't realize at the time that there was

anything unusual about it, is how well people got along. I'm still

not sure why. Maybe one reason was that everyone felt like an

immigrant. When I was a kid in Monroeville, people didn't call

themselves American. They called themselves Italian or Serbian or

Ukranian. Just imagine what it must have been like here a hundred

years ago, when people were pouring in from twenty different

countries. Tolerance was the only option.What I remember about the culture of Pittsburgh is that it was

both tolerant and pragmatic. That's how I'd describe the culture

of Silicon Valley too. And it's not a coincidence, because Pittsburgh

was the Silicon Valley of its time. This was a city where people

built new things. And while the things people build have changed,

the spirit you need to do that kind of work is the same.So although an influx of latte-swilling hipsters may be annoying

in some ways, I would go out of my way to encourage them. And more

generally to tolerate strangeness, even unto the degree wacko

Californians do. For Pittsburgh that is a conservative choice:

it's a return to the city's roots.Unfortunately I saved the toughest part for last. There is one more

thing you need to be a startup hub, and Pittsburgh hasn't got it:

investors. Silicon Valley has a big investor community because

it's had 50 years to grow one. New York has a big investor community

because it's full of people who like money a lot and are quick to

notice new ways to get it. But Pittsburgh has neither of these.

And the cheap housing that draws other people here has no effect

on investors.If an investor community grows up here, it will happen the same way

it did in Silicon Valley: slowly and organically. So I would not

bet on having a big investor community in the short term. But

fortunately there are three trends that make that less necessary

than it used to be. One is that startups are increasingly cheap

to start, so you just don't need as much outside money as you used

to. The second is that thanks to things like Kickstarter, a startup

can get to revenue faster. You can put something on Kickstarter

from anywhere. The third is programs like Y Combinator. A startup

from anywhere in the world can go to YC for 3 months, pick up

funding, and then return home if they want.My advice is to make Pittsburgh a great place for startups, and

gradually more of them will stick. Some of those will succeed;

some of their founders will become investors; and still more startups

will stick.This is not a fast path to becoming a startup hub. But it is at

least a path, which is something few other cities have. And it's

not as if you have to make painful sacrifices in the meantime.

Think about what I've suggested you should do. Encourage local

restaurants, save old buildings, take advantage of density, make

CMU the best, promote tolerance. These are the things that make

Pittsburgh good to live in now. All I'm saying is that you should

do even more of them.And that's an encouraging thought. If Pittsburgh's path to becoming

a startup hub is to be even more itself, then it has a good chance

of succeeding. In fact it probably has the best chance of any city

its size. It will take some effort, and a lot of time, but if any

city can do it, Pittsburgh can.Thanks to Charlie Cheever and Jessica Livingston for reading

drafts of this, and to Meg Cheever for organizing Opt412 and inviting

me to speak.

Life is Short

January 2016Life is short, as everyone knows. When I was a kid I used to wonder

about this. Is life actually short, or are we really complaining

about its finiteness? Would we be just as likely to feel life was

short if we lived 10 times as long?Since there didn't seem any way to answer this question, I stopped

wondering about it. Then I had kids. That gave me a way to answer

the question, and the answer is that life actually is short.Having kids showed me how to convert a continuous quantity, time,

into discrete quantities. You only get 52 weekends with your 2 year

old. If Christmas-as-magic lasts from say ages 3 to 10, you only

get to watch your child experience it 8 times. And while it's

impossible to say what is a lot or a little of a continuous quantity

like time, 8 is not a lot of something. If you had a handful of 8

peanuts, or a shelf of 8 books to choose from, the quantity would

definitely seem limited, no matter what your lifespan was.Ok, so life actually is short. Does it make any difference to know

that?It has for me. It means arguments of the form "Life is too short

for x" have great force. It's not just a figure of speech to say

that life is too short for something. It's not just a synonym for

annoying. If you find yourself thinking that life is too short for

something, you should try to eliminate it if you can.When I ask myself what I've found life is too short for, the word

that pops into my head is "bullshit." I realize that answer is

somewhat tautological. It's almost the definition of bullshit that

it's the stuff that life is too short for. And yet bullshit does

have a distinctive character. There's something fake about it.

It's the junk food of experience.

[1]If you ask yourself what you spend your time on that's bullshit,

you probably already know the answer. Unnecessary meetings, pointless

disputes, bureaucracy, posturing, dealing with other people's

mistakes, traffic jams, addictive but unrewarding pastimes.There are two ways this kind of thing gets into your life: it's

either forced on you, or it tricks you. To some extent you have to

put up with the bullshit forced on you by circumstances. You need

to make money, and making money consists mostly of errands. Indeed,

the law of supply and demand insures that: the more rewarding some

kind of work is, the cheaper people will do it. It may be that

less bullshit is forced on you than you think, though. There has

always been a stream of people who opt out of the default grind and

go live somewhere where opportunities are fewer in the conventional

sense, but life feels more authentic. This could become more common.You can do it on a smaller scale without moving. The amount of

time you have to spend on bullshit varies between employers. Most

large organizations (and many small ones) are steeped in it. But

if you consciously prioritize bullshit avoidance over other factors

like money and prestige, you can probably find employers that will

waste less of your time.If you're a freelancer or a small company, you can do this at the

level of individual customers. If you fire or avoid toxic customers,

you can decrease the amount of bullshit in your life by more than

you decrease your income.But while some amount of bullshit is inevitably forced on you, the

bullshit that sneaks into your life by tricking you is no one's

fault but your own. And yet the bullshit you choose may be harder

to eliminate than the bullshit that's forced on you. Things that

lure you into wasting your time have to be really good at

tricking you. An example that will be familiar to a lot of people

is arguing online. When someone

contradicts you, they're in a sense attacking you. Sometimes pretty

overtly. Your instinct when attacked is to defend yourself. But

like a lot of instincts, this one wasn't designed for the world we

now live in. Counterintuitive as it feels, it's better most of

the time not to defend yourself. Otherwise these people are literally

taking your life.

[2]Arguing online is only incidentally addictive. There are more

dangerous things than that. As I've written before, one byproduct

of technical progress is that things we like tend to become more

addictive. Which means we will increasingly have to make a conscious

effort to avoid addictions to stand outside ourselves and ask "is

this how I want to be spending my time?"As well as avoiding bullshit, one should actively seek out things

that matter. But different things matter to different people, and

most have to learn what matters to them. A few are lucky and realize

early on that they love math or taking care of animals or writing,

and then figure out a way to spend a lot of time doing it. But

most people start out with a life that's a mix of things that

matter and things that don't, and only gradually learn to distinguish

between them.For the young especially, much of this confusion is induced by the

artificial situations they find themselves in. In middle school and

high school, what the other kids think of you seems the most important

thing in the world. But when you ask adults what they got wrong

at that age, nearly all say they cared too much what other kids

thought of them.One heuristic for distinguishing stuff that matters is to ask

yourself whether you'll care about it in the future. Fake stuff

that matters usually has a sharp peak of seeming to matter. That's

how it tricks you. The area under the curve is small, but its shape

jabs into your consciousness like a pin.The things that matter aren't necessarily the ones people would

call "important." Having coffee with a friend matters. You won't

feel later like that was a waste of time.One great thing about having small children is that they make you

spend time on things that matter: them. They grab your sleeve as

you're staring at your phone and say "will you play with me?" And

odds are that is in fact the bullshit-minimizing option.If life is short, we should expect its shortness to take us by

surprise. And that is just what tends to happen. You take things

for granted, and then they're gone. You think you can always write

that book, or climb that mountain, or whatever, and then you realize

the window has closed. The saddest windows close when other people

die. Their lives are short too. After my mother died, I wished I'd

spent more time with her. I lived as if she'd always be there.

And in her typical quiet way she encouraged that illusion. But an

illusion it was. I think a lot of people make the same mistake I

did.The usual way to avoid being taken by surprise by something is to

be consciously aware of it. Back when life was more precarious,

people used to be aware of death to a degree that would now seem a

bit morbid. I'm not sure why, but it doesn't seem the right answer

to be constantly reminding oneself of the grim reaper hovering at

everyone's shoulder. Perhaps a better solution is to look at the

problem from the other end. Cultivate a habit of impatience about

the things you most want to do. Don't wait before climbing that

mountain or writing that book or visiting your mother. You don't

need to be constantly reminding yourself why you shouldn't wait.

Just don't wait.I can think of two more things one does when one doesn't have much

of something: try to get more of it, and savor what one has. Both

make sense here.How you live affects how long you live. Most people could do better.

Me among them.But you can probably get even more effect by paying closer attention

to the time you have. It's easy to let the days rush by. The

"flow" that imaginative people love so much has a darker cousin

that prevents you from pausing to savor life amid the daily slurry

of errands and alarms. One of the most striking things I've read

was not in a book, but the title of one: James Salter's Burning

the Days.It is possible to slow time somewhat. I've gotten better at it.

Kids help. When you have small children, there are a lot of moments

so perfect that you can't help noticing.It does help too to feel that you've squeezed everything out of

some experience. The reason I'm sad about my mother is not just

that I miss her but that I think of all the things we could have

done that we didn't. My oldest son will be 7 soon. And while I

miss the 3 year old version of him, I at least don't have any regrets

over what might have been. We had the best time a daddy and a 3

year old ever had.Relentlessly prune bullshit, don't wait to do things that matter,

and savor the time you have. That's what you do when life is short.Notes[1]

At first I didn't like it that the word that came to mind was

one that had other meanings. But then I realized the other meanings

are fairly closely related. Bullshit in the sense of things you

waste your time on is a lot like intellectual bullshit.[2]

I chose this example deliberately as a note to self. I get

attacked a lot online. People tell the craziest lies about me.

And I have so far done a pretty mediocre job of suppressing the

natural human inclination to say "Hey, that's not true!"Thanks to Jessica Livingston and Geoff Ralston for reading drafts

of this.Korean TranslationJapanese TranslationChinese Translation

Economic Inequality

January 2016Since the 1970s, economic inequality in the US has increased

dramatically. And in particular, the rich have gotten a lot richer.

Nearly everyone who writes about the topic says that economic inequality

should be decreased.I'm interested in this question because I was one of the founders of

a company called Y Combinator that helps people start startups.

Almost by definition, if a startup succeeds, its founders become

rich. Which means by helping startup founders I've been helping to

increase economic inequality. If economic inequality should be

decreased, I shouldn't be helping founders. No one should

be.But that doesn't sound right. What's going on here? What's going

on is that while economic inequality is a single measure (or more

precisely, two: variation in income, and variation in wealth), it

has multiple causes. Many of these causes are bad, like tax loopholes

and drug addiction. But some are good, like Larry Page and

Sergey Brin starting the company you use to find things online.If you want to understand economic inequality — and more importantly,

if you actually want to fix the bad aspects of it — you have to

tease apart the components. And yet the trend in nearly everything

written about the subject is to do the opposite: to squash together

all the aspects of economic inequality as if it were a single

phenomenon.Sometimes this is done for ideological reasons. Sometimes it's

because the writer only has very high-level data and so draws

conclusions from that, like the proverbial drunk who looks for his

keys under the lamppost, instead of where he dropped them, because the

light is better there. Sometimes it's because the writer doesn't

understand critical aspects of inequality, like the role of technology

in wealth creation. Much of the time, perhaps most of the time,

writing about economic inequality combines all three.\_\_\_The most common mistake people make about economic inequality is

to treat it as a single phenomenon. The most naive version of which

is the one based on the pie fallacy: that the rich get rich by

taking money from the poor.Usually this is an assumption people start from rather than a

conclusion they arrive at by examining the evidence. Sometimes the

pie fallacy is stated explicitly:

...those at the top are grabbing an increasing fraction of the

nation's income — so much of a larger share that what's left over

for the rest is diminished....

[1]

Other times it's more unconscious. But the unconscious form is very

widespread. I think because we grow up in a world where the pie

fallacy is actually true. To kids, wealth is a fixed pie

that's shared out, and if one person gets more, it's at the expense

of another. It takes a conscious effort to remind oneself that the

real world doesn't work that way.In the real world you can create wealth as well as taking it from

others. A woodworker creates wealth. He makes a chair, and you

willingly give him money in return for it. A high-frequency trader

does not. He makes a dollar only when someone on the other end of

a trade loses a dollar.If the rich people in a society got that way by taking wealth from

the poor, then you have the degenerate case of economic inequality,

where the cause of poverty is the same as the cause of wealth. But

instances of inequality don't have to be instances of the degenerate

case. If one woodworker makes 5 chairs and another makes none, the

second woodworker will have less money, but not because anyone took

anything from him.Even people sophisticated enough to know about the pie fallacy are

led toward it by the custom of describing economic inequality as a

ratio of one quantile's income or wealth to another's. It's so

easy to slip from talking about income shifting from one quantile

to another, as a figure of speech, into believing that is literally

what's happening.Except in the degenerate case, economic inequality can't be described

by a ratio or even a curve. In the general case it consists of

multiple ways people become poor, and multiple ways people become

rich. Which means to understand economic inequality in a country,

you have to go find individual people who are poor or rich and

figure out why.

[2]If you want to understand change in economic inequality, you

should ask what those people would have done when it was different.

This is one way I know the rich aren't all getting richer simply

from some new system for transferring wealth to them from

everyone else. When you use the would-have method with startup

founders, you find what most would have done

back in 1960, when

economic inequality was lower, was to join big companies or become

professors. Before Mark Zuckerberg started Facebook, his default

expectation was that he'd end up working at Microsoft. The reason

he and most other startup founders are richer than they would have

been in the mid 20th century is not because of some right turn the

country took during the Reagan administration, but because progress

in technology has made it much easier to start a new company that

grows fast.Traditional economists seem strangely averse to studying individual

humans. It seems to be a rule with them that everything has to start

with statistics. So they give you very precise numbers about

variation in wealth and income, then follow it with the most naive

speculation about the underlying causes.But while there are a lot of people who get rich through rent-seeking

of various forms, and a lot who get rich by playing zero-sum games,

there are also a significant number

who get rich by creating wealth. And creating wealth, as a source

of economic inequality, is different from taking it — not just

morally, but also practically, in the sense that it is harder to

eradicate. One reason is that variation in productivity is

accelerating. The rate at which individuals can create wealth

depends on the technology available to them, and that grows

exponentially. The other reason creating wealth is such a tenacious

source of inequality is that it can expand to accommodate a lot of

people.\_\_\_I'm all for shutting down the crooked ways to get rich. But that

won't eliminate great variations in wealth, because as long as you leave

open the option of getting rich by creating wealth, people who want

to get rich will do that instead.Most people who get rich tend to be fairly driven. Whatever their

other flaws, laziness is usually not one of them. Suppose new

policies make it hard to make a fortune in finance. Does it seem

plausible that the people who currently go into finance to make

their fortunes will continue to do so, but be content to work for

ordinary salaries? The reason they go into finance is not because

they love finance but because they want to get rich. If the only

way left to get rich is to start startups, they'll start startups.

They'll do well at it too, because determination is the main factor

in the success of a startup.

[3]

And while it would probably be

a good thing for the world if people who wanted to get rich switched

from playing zero-sum games to creating wealth, that would not only

not eliminate great variations in wealth, but might even

exacerbate them.

In a zero-sum game there is at least a limit to the upside. Plus

a lot of the new startups would create new technology that further

accelerated variation in productivity.Variation in productivity is far from the only source of economic

inequality, but it is the irreducible core of it, in the sense that

you'll have that left when you eliminate all other sources. And if

you do, that core will be big, because it will have expanded to

include the efforts of all the refugees. Plus it will have a large

Baumol penumbra around it: anyone who could get rich by creating

wealth on their own account will have to be paid enough to prevent

them from doing it.You can't prevent great variations in wealth without preventing people

from getting rich, and you can't do that without preventing them

from starting startups.So let's be clear about that. Eliminating great variations in wealth would

mean eliminating startups. And that doesn't seem a wise move.

Especially since it would only mean you eliminated

startups in your own country. Ambitious people already move halfway

around the world to further their careers, and startups can operate

from anywhere nowadays. So if you made it impossible to get rich

by creating wealth in your country, people who wanted to do that

would just leave and do it somewhere else. Which would

certainly get you a lower Gini coefficient, along with a lesson in

being careful what you ask for.

[4]I think rising economic inequality is the inevitable fate of countries

that don't choose something worse. We had a 40 year stretch in the

middle of the 20th century that convinced some people otherwise.

But as I explained in The Refragmentation,

that was an anomaly — a

unique combination of circumstances that compressed American society

not just economically but culturally too.

[5]And while some of the growth in economic inequality we've seen since

then has been due to bad behavior of various kinds, there has

simultaneously been a huge increase in individuals' ability to

create wealth. Startups are almost entirely a product of this

period. And even within the startup world, there has been a qualitative

change in the last 10 years. Technology has decreased the cost of

starting a startup so much that founders now have the upper hand

over investors. Founders get less diluted, and it is now common

for them to retain

board control as well. Both further increase

economic inequality, the former because founders own more stock,

and the latter because, as investors have learned, founders tend

to be better at running their companies than investors.While the surface manifestations change, the underlying forces are

very, very old. The acceleration of productivity we see in Silicon

Valley has been happening for thousands of years. If you look at

the history of stone tools, technology was already accelerating in

the Mesolithic. The acceleration would have been too slow to

perceive in one lifetime. Such is the nature of the leftmost part

of an exponential curve. But it was the same curve.You do not want to design your society in a way that's incompatible

with this curve. The evolution of technology is one of the most

powerful forces in history.Louis Brandeis said "We may have democracy, or we may have wealth

concentrated in the hands of a few, but we can't have both." That

sounds plausible. But if I have to choose between ignoring him and

ignoring an exponential curve that has been operating for thousands

of years, I'll bet on the curve. Ignoring any trend that has been

operating for thousands of years is dangerous. But exponential

growth, especially, tends to bite you.\_\_\_If accelerating variation in productivity is always going to produce

some baseline growth in economic inequality, it would be a good

idea to spend some time thinking about that future. Can you have

a healthy society with great variation in wealth? What would it

look like?Notice how novel it feels to think about that. The public conversation

so far has been exclusively about the need to decrease economic

inequality. We've barely given a thought to how to live with it.I'm hopeful we'll be able to. Brandeis was a product of the Gilded

Age, and things have changed since then. It's harder to hide

wrongdoing now. And to get rich now you don't have to buy politicians

the way railroad or oil magnates did.

[6]

The great concentrations

of wealth I see around me in Silicon Valley don't seem to be

destroying democracy.There are lots of things wrong with the US that have economic

inequality as a symptom. We should fix those things. In the process

we may decrease economic inequality. But we can't start from the

symptom and hope to fix the underlying causes.

[7]The most obvious is poverty. I'm sure most of those who want to

decrease economic inequality want to do it mainly to help the poor,

not to hurt the rich.

[8]

Indeed, a good number are merely being

sloppy by speaking of decreasing economic inequality when what they

mean is decreasing poverty. But this is a situation where it would

be good to be precise about what we want. Poverty and economic

inequality are not identical. When the city is turning off your

water

because you can't pay the bill, it doesn't make any difference

what Larry Page's net worth is compared to yours. He might only

be a few times richer than you, and it would still be just as much

of a problem that your water was getting turned off.Closely related to poverty is lack of social mobility. I've seen

this myself: you don't have to grow up rich or even upper middle

class to get rich as a startup founder, but few successful founders

grew up desperately poor. But again, the problem here is not simply

economic inequality. There is an enormous difference in wealth

between the household Larry Page grew up in and that of a successful

startup founder, but that didn't prevent him from joining their

ranks. It's not economic inequality per se that's blocking social

mobility, but some specific combination of things that go wrong

when kids grow up sufficiently poor.One of the most important principles in Silicon Valley is that "you

make what you measure." It means that if you pick some number to

focus on, it will tend to improve, but that you have to choose the

right number, because only the one you choose will improve; another

that seems conceptually adjacent might not. For example, if you're

a university president and you decide to focus on graduation rates,

then you'll improve graduation rates. But only graduation rates,

not how much students learn. Students could learn less, if to

improve graduation rates you made classes easier.Economic inequality is sufficiently far from identical with the

various problems that have it as a symptom that we'll probably only

hit whichever of the two we aim at. If we aim at economic inequality,

we won't fix these problems. So I say let's aim at the problems.For example, let's attack poverty, and if necessary damage wealth

in the process. That's much more likely to work than attacking

wealth in the hope that you will thereby fix poverty.

[9]

And if

there are people getting rich by tricking consumers or lobbying the

government for anti-competitive regulations or tax loopholes, then

let's stop them. Not because it's causing economic inequality, but

because it's stealing.

[10]If all you have is statistics, it seems like that's what you need

to fix. But behind a broad statistical measure like economic

inequality there are some things that are good and some that are

bad, some that are historical trends with immense momentum and

others that are random accidents. If we want to fix the world

behind the statistics, we have to understand it, and focus our

efforts where they'll do the most good.Notes[1]

Stiglitz, Joseph. The Price of Inequality. Norton, 2012. p.

32.[2]

Particularly since economic inequality is a matter of outliers,

and outliers are disproportionately likely to have gotten where

they are by ways that have little do with the sort of things

economists usually think about, like wages and productivity, but

rather by, say, ending up on the wrong side of the "War on Drugs."[3]

Determination is the most important factor in deciding between

success and failure, which in startups tend to be sharply differentiated.

But it takes more than determination to create one of the hugely

successful startups. Though most founders start out excited about

the idea of getting rich, purely mercenary founders will usually

take one of the big acquisition offers most successful startups get

on the way up. The founders who go on to the next stage tend to

be driven by a sense of mission. They have the same attachment to

their companies that an artist or writer has to their work. But

it is very hard to predict at the outset which founders will do

that. It's not simply a function of their initial attitude. Starting

a company changes people.[4]

After reading a draft of this essay, Richard Florida told me

how he had once talked to a group of Europeans "who said

they wanted to make Europe more entrepreneurial and more

like Silicon Valley. I said by definition this will give you more

inequality. They thought I was insane — they could not process

it."[5]

Economic inequality has been decreasing globally. But this

is mainly due to the erosion of the kleptocracies that formerly

dominated all the poorer countries. Once the playing field is

leveler politically, we'll see economic inequality start to rise

again. The US is the bellwether. The situation we face here, the

rest of the world will sooner or later.[6]

Some people still get rich by buying politicians. My point is that

it's no longer a precondition.[7]

As well as problems that have economic inequality as a symptom,

there are those that have it as a cause. But in most if not all,

economic inequality is not the primary cause. There is usually

some injustice that is allowing economic inequality to turn into

other forms of inequality, and that injustice is what we need to

fix. For example, the police in the US treat the poor worse than

the rich. But the solution is not to make people richer. It's to

make the police treat people more equitably. Otherwise they'll

continue to maltreat people who are weak in other ways.[8]

Some who read this essay will say that I'm clueless or even

being deliberately misleading by focusing so much on the richer end

of economic inequality — that economic inequality is really about

poverty. But that is exactly the point I'm making, though sloppier

language than I'd use to make it. The real problem is poverty, not

economic inequality. And if you conflate them you're aiming at the

wrong target.Others will say I'm clueless or being misleading by focusing on

people who get rich by creating wealth — that startups aren't the

problem, but corrupt practices in finance, healthcare, and so on.

Once again, that is exactly my point. The problem is not economic

inequality, but those specific abuses.It's a strange task to write an essay about why something isn't the

problem, but that's the situation you find yourself in when so many

people mistakenly think it is.[9]

Particularly since many causes of poverty are only partially

driven by people trying to make money from them. For example,

America's abnormally high incarceration rate is a major cause of

poverty. But although for-profit prison companies and

prison guard unions both spend

a lot lobbying for harsh sentencing laws, they

are not the original source of them.[10]

Incidentally, tax loopholes are definitely not a product

of some power shift due to recent increases in economic inequality.

The golden age of economic equality in the mid 20th century was

also the golden age of tax avoidance. Indeed, it was so widespread

and so effective that I'm skeptical whether economic inequality was

really so low then as we think. In a period when people are trying

to hide wealth from the government, it will tend to be hidden from

statistics too. One sign of the potential magnitude of the problem

is the discrepancy between government receipts as a percentage of

GDP, which have remained more or less constant during the entire

period from the end of World War II to the present, and tax rates,

which have varied dramatically.

Thanks to Sam Altman, Tiffani Ashley Bell, Patrick Collison, Ron

Conway, Richard Florida, Ben Horowitz, Jessica Livingston, Robert

Morris, Tim O'Reilly, Max Roser, and Alexia Tsotsis for reading

drafts of this.Note: This is a new version from which I

removed a pair of metaphors that made a lot of people mad,

essentially by macroexpanding them. If anyone wants to see

the old version, I put it here.Related:The Short VersionA Reply to Ezra KleinA Reply to Russell OkungFrench Translation

The Refragmentation

January 2016One advantage of being old is that you can see change happen in

your lifetime. A lot of the change I've seen is fragmentation. US

politics is much more polarized than it used to be. Culturally we

have ever less common ground. The creative class flocks to a handful

of happy cities, abandoning the rest. And increasing economic

inequality means the spread between rich and poor is growing too.

I'd like to propose a hypothesis: that all these trends are instances

of the same phenomenon. And moreover, that the cause is not some

force that's pulling us apart, but rather the erosion of forces

that had been pushing us together.Worse still, for those who worry about these trends, the forces

that were pushing us together were an anomaly, a one-time combination

of circumstances that's unlikely to be repeated — and indeed, that

we would not want to repeat.The two forces were war (above all World War II), and the rise of

large corporations.The effects of World War II were both economic and social.

Economically, it decreased variation in income. Like all modern

armed forces, America's were socialist economically. From each

according to his ability, to each according to his need. More or

less. Higher ranking members of the military got more (as higher

ranking members of socialist societies always do), but what they

got was fixed according to their rank. And the flattening effect

wasn't limited to those under arms, because the US economy was

conscripted too. Between 1942 and 1945 all wages were set by the

National War Labor Board. Like the military, they defaulted to

flatness. And this national standardization of wages was so pervasive

that its effects could still be seen years after the war ended.

[1]Business owners weren't supposed to be making money either. FDR

said "not a single war millionaire" would be permitted. To ensure

that, any increase in a company's profits over prewar levels was

taxed at 85%. And when what was left after corporate taxes reached

individuals, it was taxed again at a marginal rate of 93%.

[2]Socially too the war tended to decrease variation. Over 16 million

men and women from all sorts of different backgrounds were brought

together in a way of life that was literally uniform. Service rates

for men born in the early 1920s approached 80%. And working toward

a common goal, often under stress, brought them still closer together.Though strictly speaking World War II lasted less than 4 years for

the US, its effects lasted longer. Wars make central governments

more powerful, and World War II was an extreme case of this. In

the US, as in all the other Allied countries, the federal government

was slow to give up the new powers it had acquired. Indeed, in

some respects the war didn't end in 1945; the enemy just switched

to the Soviet Union. In tax rates, federal power, defense spending,

conscription, and nationalism, the decades after the war looked more

like wartime than prewar peacetime.

[3]

And the social effects

lasted too. The kid pulled into the army from behind a mule team

in West Virginia didn't simply go back to the farm afterward.

Something else was waiting for him, something that looked a lot

like the army.If total war was the big political story of the 20th century, the

big economic story was the rise of a new kind of company. And this

too tended to produce both social and economic cohesion.

[4]The 20th century was the century of the big, national corporation.

General Electric, General Foods, General Motors. Developments in

finance, communications, transportation, and manufacturing enabled

a new type of company whose goal was above all scale. Version 1

of this world was low-res: a Duplo world of a few giant companies

dominating each big market.

[5]The late 19th and early 20th centuries had been a time of consolidation,

led especially by J. P. Morgan. Thousands of companies run by their

founders were merged into a couple hundred giant ones run by

professional managers. Economies of scale ruled the day. It seemed

to people at the time that this was the final state of things. John

D. Rockefeller said in 1880

The day of combination is here to stay. Individualism has gone,

never to return.

He turned out to be mistaken, but he seemed right for the next

hundred years.The consolidation that began in the late 19th century continued for

most of the 20th. By the end of World War II, as Michael Lind

writes, "the major sectors of the economy were either organized

as government-backed cartels or dominated by a few oligopolistic

corporations."For consumers this new world meant the same choices everywhere, but

only a few of them. When I grew up there were only 2 or 3 of most

things, and since they were all aiming at the middle of the market

there wasn't much to differentiate them.One of the most important instances of this phenomenon was in TV.

Here there were 3 choices: NBC, CBS, and ABC. Plus public TV for

eggheads and communists. The programs that the 3 networks offered were

indistinguishable. In fact, here there was a triple pressure toward

the center. If one show did try something daring, local affiliates

in conservative markets would make them stop. Plus since TVs were

expensive, whole families watched the same shows together, so they

had to be suitable for everyone.And not only did everyone get the same thing, they got it at the

same time. It's difficult to imagine now, but every night tens of

millions of families would sit down together in front of their TV

set watching the same show, at the same time, as their next door

neighbors. What happens now with the Super Bowl used to happen

every night. We were literally in sync.

[6]In a way mid-century TV culture was good. The view it gave of the

world was like you'd find in a children's book, and it probably had

something of the effect that (parents hope) children's books have

in making people behave better. But, like children's books, TV was

also misleading. Dangerously misleading, for adults. In his

autobiography, Robert MacNeil talks of seeing gruesome images that

had just come in from Vietnam and thinking, we can't show these to

families while they're having dinner.I know how pervasive the common culture was, because I tried to opt

out of it, and it was practically impossible to find alternatives.

When I was 13 I realized, more from internal evidence than any

outside source, that the ideas we were being fed on TV were crap,

and I stopped watching it.

[7]

But it wasn't just TV. It seemed

like everything around me was crap. The politicians all saying the

same things, the consumer brands making almost identical products

with different labels stuck on to indicate how prestigious they

were meant to be, the balloon-frame houses with fake "colonial"

skins, the cars with several feet of gratuitous metal on each end

that started to fall apart after a couple years, the "red delicious"

apples that were red but only nominally

apples. And in retrospect, it was crap.

[8]But when I went looking for alternatives to fill this void, I found

practically nothing. There was no Internet then. The only place

to look was in the chain bookstore in our local shopping mall.

[9]

There I found a copy of The Atlantic. I wish I could say it became

a gateway into a wider world, but in fact I found it boring and

incomprehensible. Like a kid tasting whisky for the first time and

pretending to like it, I preserved that magazine as carefully as

if it had been a book. I'm sure I still have it somewhere. But

though it was evidence that there was, somewhere, a world that

wasn't red delicious, I didn't find it till college.It wasn't just as consumers that the big companies made us similar.

They did as employers too. Within companies there were powerful

forces pushing people toward a single model of how to look and act.

IBM was particularly notorious for this, but they were only a little

more extreme than other big companies. And the models of how to

look and act varied little between companies. Meaning everyone

within this world was expected to seem more or less the same. And

not just those in the corporate world, but also everyone who aspired

to it — which in the middle of the 20th century meant most people

who weren't already in it. For most of the 20th century, working-class

people tried hard to look middle class. You can see it in old

photos. Few adults aspired to look dangerous in 1950.But the rise of national corporations didn't just compress us

culturally. It compressed us economically too, and on both ends.Along with giant national corporations, we got giant national labor

unions. And in the mid 20th century the corporations cut deals

with the unions where they paid over market price for labor. Partly

because the unions were monopolies.

[10]

Partly because, as

components of oligopolies themselves, the corporations knew they

could safely pass the cost on to their customers, because their

competitors would have to as well. And partly because in mid-century

most of the giant companies were still focused on finding new ways

to milk economies of scale. Just as startups rightly pay AWS a

premium over the cost of running their own servers so they can focus

on growth, many of the big national corporations were willing to

pay a premium for labor.

[11]As well as pushing incomes up from the bottom, by overpaying unions,

the big companies of the 20th century also pushed incomes down at

the top, by underpaying their top management. Economist J. K.

Galbraith wrote in 1967 that "There are few corporations in which

it would be suggested that executive salaries are at a maximum."

[12]To some extent this was an illusion. Much of the de facto pay of

executives never showed up on their income tax returns, because it

took the form of perks. The higher the rate of income tax, the

more pressure there was to pay employees upstream of it. (In the

UK, where taxes were even higher than in the US, companies would

even pay their kids' private school tuitions.) One of the most

valuable things the big companies of the mid 20th century gave their

employees was job security, and this too didn't show up in tax

returns or income statistics. So the nature of employment in these

organizations tended to yield falsely low numbers about economic

inequality. But even accounting for that, the big companies paid

their best people less than market price. There was no market; the

expectation was that you'd work for the same company for decades

if not your whole career.

[13]Your work was so illiquid there was little chance of getting market

price. But that same illiquidity also encouraged you not to seek

it. If the company promised to employ you till you retired and

give you a pension afterward, you didn't want to extract as much

from it this year as you could. You needed to take care of the

company so it could take care of you. Especially when you'd been

working with the same group of people for decades. If you tried

to squeeze the company for more money, you were squeezing the

organization that was going to take care of them. Plus if

you didn't put the company first you wouldn't be promoted, and if

you couldn't switch ladders, promotion on this one was the only way

up.

[14]To someone who'd spent several formative years in the armed forces,

this situation didn't seem as strange as it does to us now. From

their point of view, as big company executives, they were high-ranking

officers. They got paid a lot more than privates. They got to

have expense account lunches at the best restaurants and fly around

on the company's Gulfstreams. It probably didn't occur to most of

them to ask if they were being paid market price.The ultimate way to get market price is to work for yourself, by

starting your own company. That seems obvious to any ambitious

person now. But in the mid 20th century it was an alien concept.

Not because starting one's own company seemed too ambitious, but

because it didn't seem ambitious enough. Even as late as the 1970s,

when I grew up, the ambitious plan was to get lots of education at

prestigious institutions, and then join some other prestigious

institution and work one's way up the hierarchy. Your prestige was

the prestige of the institution you belonged to. People did start

their own businesses of course, but educated people rarely did,

because in those days there was practically zero concept of starting

what we now call a startup:

a business that starts small and grows

big. That was much harder to do in the mid 20th century. Starting

one's own business meant starting a business that would start small

and stay small. Which in those days of big companies often meant

scurrying around trying to avoid being trampled by elephants. It

was more prestigious to be one of the executive class riding the

elephant.By the 1970s, no one stopped to wonder where the big prestigious

companies had come from in the first place. It seemed like they'd

always been there, like the chemical elements. And indeed, there

was a double wall between ambitious kids in the 20th century and

the origins of the big companies. Many of the big companies were

roll-ups that didn't have clear founders. And when they did, the

founders didn't seem like us. Nearly all of them had been uneducated,

in the sense of not having been to college. They were what Shakespeare

called rude mechanicals. College trained one to be a member of the

professional classes. Its graduates didn't expect to do the sort

of grubby menial work that Andrew Carnegie or Henry Ford started

out doing.

[15]And in the 20th century there were more and more college graduates.

They increased from about 2% of the population in 1900 to about 25%

in 2000. In the middle of the century our two big forces intersect,

in the form of the GI Bill, which sent 2.2 million World War II

veterans to college. Few thought of it in these terms, but the

result of making college the canonical path for the ambitious was

a world in which it was socially acceptable to work for Henry Ford,

but not to be Henry Ford.

[16]I remember this world well. I came of age just as it was starting

to break up. In my childhood it was still dominant. Not quite so

dominant as it had been. We could see from old TV shows and yearbooks

and the way adults acted that people in the 1950s and 60s had been

even more conformist than us. The mid-century model was already

starting to get old. But that was not how we saw it at the time.

We would at most have said that one could be a bit more daring in

1975 than 1965. And indeed, things hadn't changed much yet.But change was coming soon. And when the Duplo economy started to

disintegrate, it disintegrated in several different ways at once.

Vertically integrated companies literally dis-integrated because

it was more efficient to. Incumbents faced new competitors as (a)

markets went global and (b) technical innovation started to trump

economies of scale, turning size from an asset into a liability.

Smaller companies were increasingly able to survive as formerly

narrow channels to consumers broadened. Markets themselves started

to change faster, as whole new categories of products appeared. And

last but not least, the federal government, which had previously

smiled upon J. P. Morgan's world as the natural state of things,

began to realize it wasn't the last word after all.What J. P. Morgan was to the horizontal axis, Henry Ford was to the

vertical. He wanted to do everything himself. The giant plant he

built at River Rouge between 1917 and 1928 literally took in iron

ore at one end and sent cars out the other. 100,000 people worked

there. At the time it seemed the future. But that is not how car

companies operate today. Now much of the design and manufacturing

happens in a long supply chain, whose products the car companies

ultimately assemble and sell. The reason car companies operate

this way is that it works better. Each company in the supply chain

focuses on what they know best. And they each have to do it well

or they can be swapped out for another supplier.Why didn't Henry Ford realize that networks of cooperating companies

work better than a single big company? One reason is that supplier

networks take a while to evolve. In 1917, doing everything himself

seemed to Ford the only way to get the scale he needed. And the

second reason is that if you want to solve a problem using a network

of cooperating companies, you have to be able to coordinate their

efforts, and you can do that much better with computers. Computers

reduce the transaction costs that Coase argued are the raison d'etre

of corporations. That is a fundamental change.In the early 20th century, big companies were synonymous with

efficiency. In the late 20th century they were synonymous with

inefficiency. To some extent this was because the companies

themselves had become sclerotic. But it was also because our

standards were higher.It wasn't just within existing industries that change occurred.

The industries themselves changed. It became possible to make lots

of new things, and sometimes the existing companies weren't the

ones who did it best.Microcomputers are a classic example. The market was pioneered by

upstarts like Apple. When it got big enough, IBM decided it was

worth paying attention to. At the time IBM completely dominated

the computer industry. They assumed that all they had to do, now

that this market was ripe, was to reach out and pick it. Most

people at the time would have agreed with them. But what happened

next illustrated how much more complicated the world had become.

IBM did launch a microcomputer. Though quite successful, it did

not crush Apple. But even more importantly, IBM itself ended up

being supplanted by a supplier coming in from the side — from

software, which didn't even seem to be the same business. IBM's

big mistake was to accept a non-exclusive license for DOS. It must

have seemed a safe move at the time. No other computer manufacturer

had ever been able to outsell them. What difference did it make if

other manufacturers could offer DOS too? The result of that

miscalculation was an explosion of inexpensive PC clones. Microsoft

now owned the PC standard, and the customer. And the microcomputer

business ended up being Apple vs Microsoft.Basically, Apple bumped IBM and then Microsoft stole its wallet.

That sort of thing did not happen to big companies in mid-century.

But it was going to happen increasingly often in the future.Change happened mostly by itself in the computer business. In other

industries, legal obstacles had to be removed first. Many of the

mid-century oligopolies had been anointed by the federal government

with policies (and in wartime, large orders) that kept out competitors.

This didn't seem as dubious to government officials at the time as

it sounds to us. They felt a two-party system ensured sufficient

competition in politics. It ought to work for business too.Gradually the government realized that anti-competitive policies

were doing more harm than good, and during the Carter administration

it started to remove them. The word used for this process was

misleadingly narrow: deregulation. What was really happening was

de-oligopolization. It happened to one industry after another.

Two of the most visible to consumers were air travel and long-distance

phone service, which both became dramatically cheaper after

deregulation.Deregulation also contributed to the wave of hostile takeovers in

the 1980s. In the old days the only limit on the inefficiency of

companies, short of actual bankruptcy, was the inefficiency of their

competitors. Now companies had to face absolute rather than relative

standards. Any public company that didn't generate sufficient

returns on its assets risked having its management replaced with

one that would. Often the new managers did this by breaking companies

up into components that were more valuable separately.

[17]Version 1 of the national economy consisted of a few big blocks

whose relationships were negotiated in back rooms by a handful of

executives, politicians, regulators, and labor leaders. Version 2

was higher resolution: there were more companies, of more different

sizes, making more different things, and their relationships changed

faster. In this world there were still plenty of back room negotiations,

but more was left to market forces. Which further accelerated the

fragmentation.It's a little misleading to talk of versions when describing a

gradual process, but not as misleading as it might seem. There was

a lot of change in a few decades, and what we ended up with was

qualitatively different. The companies in the S&P 500 in 1958 had

been there an average of 61 years. By 2012 that number was 18 years.

[18]The breakup of the Duplo economy happened simultaneously with the

spread of computing power. To what extent were computers a precondition?

It would take a book to answer that. Obviously the spread of computing

power was a precondition for the rise of startups. I suspect it

was for most of what happened in finance too. But was it a

precondition for globalization or the LBO wave? I don't know, but

I wouldn't discount the possibility. It may be that the refragmentation

was driven by computers in the way the industrial revolution was

driven by steam engines. Whether or not computers were a precondition,

they have certainly accelerated it.The new fluidity of companies changed people's relationships with

their employers. Why climb a corporate ladder that might be yanked

out from under you? Ambitious people started to think of a career

less as climbing a single ladder than as a series of jobs that might

be at different companies. More movement (or even potential movement)

between companies introduced more competition in salaries. Plus

as companies became smaller it became easier to estimate how much

an employee contributed to the company's revenue. Both changes

drove salaries toward market price. And since people vary dramatically

in productivity, paying market price meant salaries started to

diverge.By no coincidence it was in the early 1980s that the term "yuppie"

was coined. That word is not much used now, because the phenomenon

it describes is so taken for granted, but at the time it was a label

for something novel. Yuppies were young professionals who made lots

of money. To someone in their twenties today, this wouldn't seem

worth naming. Why wouldn't young professionals make lots of money?

But until the 1980s, being underpaid early in your career was part

of what it meant to be a professional. Young professionals were

paying their dues, working their way up the ladder. The rewards

would come later. What was novel about yuppies was that they wanted

market price for the work they were doing now.The first yuppies did not work for startups. That was still in the

future. Nor did they work for big companies. They were professionals

working in fields like law, finance, and consulting. But their example

rapidly inspired their peers. Once they saw that new BMW 325i, they

wanted one too.Underpaying people at the beginning of their career only works if

everyone does it. Once some employer breaks ranks, everyone else

has to, or they can't get good people. And once started this process

spreads through the whole economy, because at the beginnings of

people's careers they can easily switch not merely employers but

industries.But not all young professionals benefitted. You had to produce to

get paid a lot. It was no coincidence that the first yuppies worked

in fields where it was easy to measure that.More generally, an idea was returning whose name sounds old-fashioned

precisely because it was so rare for so long: that you could make

your fortune. As in the past there were multiple ways to do it.

Some made their fortunes by creating wealth, and others by playing

zero-sum games. But once it became possible to make one's fortune,

the ambitious had to decide whether or not to. A physicist who

chose physics over Wall Street in 1990 was making a sacrifice that

a physicist in 1960 didn't have to think about.The idea even flowed back into big companies. CEOs of big companies

make more now than they used to, and I think much of the reason is

prestige. In 1960, corporate CEOs had immense prestige. They were

the winners of the only economic game in town. But if they made as

little now as they did then, in real dollar terms, they'd seem like

small fry compared to professional athletes and whiz kids making

millions from startups and hedge funds. They don't like that idea,

so now they try to get as much as they can, which is more than they

had been getting.

[19]Meanwhile a similar fragmentation was happening at the other end

of the economic scale. As big companies' oligopolies became less

secure, they were less able to pass costs on to customers and thus

less willing to overpay for labor. And as the Duplo world of a few

big blocks fragmented into many companies of different sizes — some

of them overseas — it became harder for unions to enforce their

monopolies. As a result workers' wages also tended toward market

price. Which (inevitably, if unions had been doing their job) tended

to be lower. Perhaps dramatically so, if automation had decreased

the need for some kind of work.And just as the mid-century model induced social as well as economic

cohesion, its breakup brought social as well as economic fragmentation.

People started to dress and act differently. Those who would later

be called the "creative class" became more mobile. People who didn't

care much for religion felt less pressure to go to church for

appearances' sake, while those who liked it a lot opted for

increasingly colorful forms. Some switched from meat loaf to tofu,

and others to Hot Pockets. Some switched from driving Ford sedans

to driving small imported cars, and others to driving SUVs. Kids

who went to private schools or wished they did started to dress

"preppy," and kids who wanted to seem rebellious made a conscious

effort to look disreputable. In a hundred ways people spread apart.

[20]Almost four decades later, fragmentation is still increasing. Has

it been net good or bad? I don't know; the question may be

unanswerable. Not entirely bad though. We take for granted the

forms of fragmentation we like, and worry only about the ones we

don't. But as someone who caught the tail end of mid-century

conformism,

I can tell you it was no utopia.

[21]My goal here is not to say whether fragmentation has been good or

bad, just to explain why it's happening. With the centripetal

forces of total war and 20th century oligopoly mostly gone, what

will happen next? And more specifically, is it possible to reverse

some of the fragmentation we've seen?If it is, it will have to happen piecemeal. You can't reproduce

mid-century cohesion the way it was originally produced. It would

be insane to go to war just to induce more national unity. And

once you understand the degree to which the economic history of the

20th century was a low-res version 1, it's clear you can't reproduce

that either.20th century cohesion was something that happened at least in a

sense naturally. The war was due mostly to external forces, and

the Duplo economy was an evolutionary phase. If you want cohesion

now, you'd have to induce it deliberately. And it's not obvious

how. I suspect the best we'll be able to do is address the symptoms

of fragmentation. But that may be enough.The form of fragmentation people worry most about lately is economic inequality, and if you want to eliminate

that you're up against a truly formidable headwind — one that has

been in operation since the stone age: technology.Technology is

a lever. It magnifies work. And the lever not only grows increasingly

long, but the rate at which it grows is itself increasing.Which in turn means the variation in the amount of wealth people

can create has not only been increasing, but accelerating. The

unusual conditions that prevailed in the mid 20th century masked

this underlying trend. The ambitious had little choice but to join

large organizations that made them march in step with lots of other

people — literally in the case of the armed forces, figuratively

in the case of big corporations. Even if the big corporations had

wanted to pay people proportionate to their value, they couldn't

have figured out how. But that constraint has gone now. Ever since

it started to erode in the 1970s, we've seen the underlying forces

at work again.

[22]Not everyone who gets rich now does it by creating wealth, certainly.

But a significant number do, and the Baumol Effect means all their

peers get dragged along too.

[23]

And as long as it's possible to

get rich by creating wealth, the default tendency will be for

economic inequality to increase. Even if you eliminate all the

other ways to get rich. You can mitigate this with subsidies at

the bottom and taxes at the top, but unless taxes are high enough

to discourage people from creating wealth, you're always going to

be fighting a losing battle against increasing variation in

productivity.

[24]That form of fragmentation, like the others, is here to stay. Or

rather, back to stay. Nothing is forever, but the tendency toward

fragmentation should be more forever than most things, precisely

because it's not due to any particular cause. It's simply a reversion

to the mean. When Rockefeller said individualism was gone, he was

right for a hundred years. It's back now, and that's likely to be

true for longer.I worry that if we don't acknowledge this, we're headed for trouble.

If we think 20th century cohesion disappeared because of few policy

tweaks, we'll be deluded into thinking we can get it back (minus

the bad parts, somehow) with a few countertweaks. And then we'll

waste our time trying to eliminate fragmentation, when we'd be

better off thinking about how to mitigate its consequences.

Notes[1]

Lester Thurow, writing in 1975, said the wage differentials

prevailing at the end of World War II had become so embedded that

they "were regarded as 'just' even after the egalitarian pressures

of World War II had disappeared. Basically, the same differentials

exist to this day, thirty years later." But Goldin and Margo think

market forces in the postwar period also helped preserve the wartime

compression of wages — specifically increased demand for unskilled

workers, and oversupply of educated ones.(Oddly enough, the American custom of having employers pay for

health insurance derives from efforts by businesses to circumvent

NWLB wage controls in order to attract workers.)[2]

As always, tax rates don't tell the whole story. There were

lots of exemptions, especially for individuals. And in World War

II the tax codes were so new that the government had little acquired

immunity to tax avoidance. If the rich paid high taxes during the

war it was more because they wanted to than because they had to.After the war, federal tax receipts as a percentage of GDP were

about the same as they are now. In fact, for the entire period since

the war, tax receipts have stayed close to 18% of GDP, despite

dramatic changes in tax rates. The lowest point occurred when

marginal income tax rates were highest: 14.1% in 1950. Looking at

the data, it's hard to avoid the conclusion that tax rates have had

little effect on what people actually paid.[3]

Though in fact the decade preceding the war had been a time

of unprecedented federal power, in response to the Depression.

Which is not entirely a coincidence, because the Depression was one

of the causes of the war. In many ways the New Deal was a sort of

dress rehearsal for the measures the federal government took during

wartime. The wartime versions were much more drastic and more

pervasive though. As Anthony Badger wrote, "for many Americans the

decisive change in their experiences came not with the New Deal but

with World War II."[4]

I don't know enough about the origins of the world wars to

say, but it's not inconceivable they were connected to the rise of

big corporations. If that were the case, 20th century cohesion would

have a single cause.[5]

More precisely, there was a bimodal economy consisting, in

Galbraith's words, of "the world of the technically dynamic, massively

capitalized and highly organized corporations on the one hand and

the hundreds of thousands of small and traditional proprietors on

the other." Money, prestige, and power were concentrated in the

former, and there was near zero crossover.[6]

I wonder how much of the decline in families eating together

was due to the decline in families watching TV together afterward.[7]

I know when this happened because it was the season Dallas

premiered. Everyone else was talking about what was happening on

Dallas, and I had no idea what they meant.[8]

I didn't realize it till I started doing research for this

essay, but the meretriciousness of the products I grew up with is

a well-known byproduct of oligopoly. When companies can't compete

on price, they compete on tailfins.[9]

Monroeville Mall was at the time of its completion in 1969

the largest in the country. In the late 1970s the movie Dawn of

the Dead was shot there. Apparently the mall was not just the

location of the movie, but its inspiration; the crowds of shoppers

drifting through this huge mall reminded George Romero of zombies.

My first job was scooping ice cream in the Baskin-Robbins.[10]

Labor unions were exempted from antitrust laws by the Clayton

Antitrust Act in 1914 on the grounds that a person's work is not

"a commodity or article of commerce." I wonder if that means service

companies are also exempt.[11]

The relationships between unions and unionized companies can

even be symbiotic, because unions will exert political pressure to

protect their hosts. According to Michael Lind, when politicians

tried to attack the A&P supermarket chain because it was putting

local grocery stores out of business, "A&P successfully defended

itself by allowing the unionization of its workforce in 1938, thereby

gaining organized labor as a constituency." I've seen this phenomenon

myself: hotel unions are responsible for more of the political

pressure against Airbnb than hotel companies.[12]

Galbraith was clearly puzzled that corporate executives would

work so hard to make money for other people (the shareholders)

instead of themselves. He devoted much of The New Industrial

State to trying to figure this out.His theory was that professionalism had replaced money as a motive,

and that modern corporate executives were, like (good) scientists,

motivated less by financial rewards than by the desire to do good

work and thereby earn the respect of their peers. There is something

in this, though I think lack of movement between companies combined

with self-interest explains much of observed behavior.[13]

Galbraith (p. 94) says a 1952 study of the 800 highest paid

executives at 300 big corporations found that three quarters of

them had been with their company for more than 20 years.[14]

It seems likely that in the first third of the 20th century

executive salaries were low partly because companies then were more

dependent on banks, who would have disapproved if executives got

too much. This was certainly true in the beginning. The first big

company CEOs were J. P. Morgan's hired hands.Companies didn't start to finance themselves with retained earnings

till the 1920s. Till then they had to pay out their earnings in

dividends, and so depended on banks for capital for expansion.

Bankers continued to sit on corporate boards till the Glass-Steagall

act in 1933.By mid-century big companies funded 3/4 of their growth from earnings.

But the early years of bank dependence, reinforced by the financial

controls of World War II, must have had a big effect on social

conventions about executive salaries. So it may be that the lack

of movement between companies was as much the effect of low salaries

as the cause.Incidentally, the switch in the 1920s to financing growth with

retained earnings was one cause of the 1929 crash. The banks now

had to find someone else to lend to, so they made more margin loans.[15]

Even now it's hard to get them to. One of the things I find

hardest to get into the heads of would-be startup founders is how

important it is to do certain kinds of menial work early in the

life of a company. Doing things that don't

scale is to how Henry Ford got started as a high-fiber diet is

to the traditional peasant's diet: they had no choice but to do the

right thing, while we have to make a conscious effort.[16]

Founders weren't celebrated in the press when I was a kid.

"Our founder" meant a photograph of a severe-looking man with a

walrus mustache and a wing collar who had died decades ago. The

thing to be when I was a kid was an executive. If you weren't

around then it's hard to grasp the cachet that term had. The fancy

version of everything was called the "executive" model.[17]

The wave of hostile takeovers in the 1980s was enabled by a

combination of circumstances: court decisions striking down state

anti-takeover laws, starting with the Supreme Court's 1982 decision

in Edgar v. MITE Corp.; the Reagan administration's comparatively

sympathetic attitude toward takeovers; the Depository Institutions

Act of 1982, which allowed banks and savings and loans to buy

corporate bonds; a new SEC rule issued in 1982 (rule 415) that made

it possible to bring corporate bonds to market faster; the creation

of the junk bond business by Michael Milken; a vogue for conglomerates

in the preceding period that caused many companies to be combined

that never should have been; a decade of inflation that left many

public companies trading below the value of their assets; and not

least, the increasing complacency of managements.[18]

Foster, Richard. "Creative Destruction Whips through Corporate

America." Innosight, February 2012.[19]

CEOs of big companies may be overpaid. I don't know enough

about big companies to say. But it is certainly not impossible for

a CEO to make 200x as much difference to a company's revenues as

the average employee. Look at what Steve Jobs did for Apple when

he came back as CEO. It would have been a good deal for the board

to give him 95% of the company. Apple's market cap the day Steve

came back in July 1997 was 1.73 billion. 5% of Apple now (January

2016) would be worth about 30 billion. And it would not be if Steve

hadn't come back; Apple probably wouldn't even exist anymore.Merely including Steve in the sample might be enough to answer the

question of whether public company CEOs in the aggregate are overpaid.

And that is not as facile a trick as it might seem, because the

broader your holdings, the more the aggregate is what you care

about.[20]

The late 1960s were famous for social upheaval. But that was

more rebellion (which can happen in any era if people are provoked

sufficiently) than fragmentation. You're not seeing fragmentation

unless you see people breaking off to both left and right.[21]

Globally the trend has been in the other direction. While

the US is becoming more fragmented, the world as a whole is becoming

less fragmented, and mostly in good ways.[22]

There were a handful of ways to make a fortune in the mid

20th century. The main one was drilling for oil, which was open

to newcomers because it was not something big companies could

dominate through economies of scale. How did individuals accumulate

large fortunes in an era of such high taxes? Giant tax loopholes

defended by two of the most powerful men in Congress, Sam Rayburn

and Lyndon Johnson.But becoming a Texas oilman was not in 1950 something one could

aspire to the way starting a startup or going to work on Wall Street

were in 2000, because (a) there was a strong local component and

(b) success depended so much on luck.[23]

The Baumol Effect induced by startups is very visible in

Silicon Valley. Google will pay people millions of dollars a year

to keep them from leaving to start or join startups.[24]

I'm not claiming variation in productivity is the only cause

of economic inequality in the US. But it's a significant cause, and

it will become as big a cause as it needs to, in the sense that if

you ban other ways to get rich, people who want to get rich will

use this route instead.Thanks to Sam Altman, Trevor Blackwell, Paul Buchheit, Patrick

Collison, Ron Conway, Chris Dixon, Benedict Evans, Richard Florida,

Ben Horowitz, Jessica Livingston, Robert Morris, Tim O'Reilly, Geoff

Ralston, Max Roser, Alexia Tsotsis, and Qasar Younis for reading

drafts of this. Max also told me about several valuable sources.BibliographyAllen, Frederick Lewis. The Big Change. Harper, 1952.Averitt, Robert. The Dual Economy. Norton, 1968.Badger, Anthony. The New Deal. Hill and Wang, 1989.Bainbridge, John. The Super-Americans. Doubleday, 1961.Beatty, Jack. Collossus. Broadway, 2001.Brinkley, Douglas. Wheels for the World. Viking, 2003.Brownleee, W. Elliot. Federal Taxation in America. Cambridge, 1996.Chandler, Alfred. The Visible Hand. Harvard, 1977.Chernow, Ron. The House of Morgan. Simon & Schuster, 1990.Chernow, Ron. Titan: The Life of John D. Rockefeller. Random House,

1998.Galbraith, John. The New Industrial State. Houghton Mifflin, 1967.Goldin, Claudia and Robert A. Margo. "The Great Compression: The

Wage Structure in the United States at Mid-Century." NBER Working

Paper 3817, 1991.Gordon, John. An Empire of Wealth. HarperCollins, 2004.Klein, Maury. The Genesis of Industrial America, 1870-1920. Cambridge,

2007.Lind, Michael. Land of Promise. HarperCollins, 2012.Mickelthwaite, John, and Adrian Wooldridge. The Company. Modern

Library, 2003.Nasaw, David. Andrew Carnegie. Penguin, 2006.Sobel, Robert. The Age of Giant Corporations. Praeger, 1993.Thurow, Lester. Generating Inequality: Mechanisms of Distribution.

Basic Books, 1975.Witte, John. The Politics and Development of the Federal Income

Tax. Wisconsin, 1985.Related:Too Many Elite American Men Are Obsessed With Work and Wealth

Jessica Livingston

November 2015A few months ago an article about Y Combinator said that early on

it had been a "one-man show." It's sadly common to read that sort

of thing. But the problem with that description is not just that

it's unfair. It's also misleading. Much of what's most novel about

YC is due to Jessica Livingston. If you don't understand her, you

don't understand YC. So let me tell you a little about Jessica.YC had 4 founders. Jessica and I decided one night to start it,

and the next day we recruited my friends Robert Morris and Trevor

Blackwell. Jessica and I ran YC day to day, and Robert and Trevor

read applications and did interviews with us.Jessica and I were already dating when we started YC. At first we

tried to act "professional" about this, meaning we tried to conceal

it. In retrospect that seems ridiculous, and we soon dropped the

pretense. And the fact that Jessica and I were a couple is a big

part of what made YC what it was. YC felt like a family. The

founders early on were mostly young. We all had dinner together

once a week, cooked for the first couple years by me. Our first

building had been a private home. The overall atmosphere was

shockingly different from a VC's office on Sand Hill Road, in a way

that was entirely for the better. There was an authenticity that

everyone who walked in could sense. And that didn't just mean that

people trusted us. It was the perfect quality to instill in startups.

Authenticity is one of the most important things YC looks for in

founders, not just because fakers and opportunists are annoying,

but because authenticity is one of the main things that separates

the most successful startups from the rest.Early YC was a family, and Jessica was its mom. And the culture

she defined was one of YC's most important innovations. Culture

is important in any organization, but at YC culture wasn't just how

we behaved when we built the product. At YC, the culture was the

product.Jessica was also the mom in another sense: she had the last word.

Everything we did as an organization went through her first — who

to fund, what to say to the public, how to deal with other companies,

who to hire, everything.Before we had kids, YC was more or less our life. There was no real

distinction between working hours and not. We talked about YC all

the time. And while there might be some businesses that it would

be tedious to let infect your private life, we liked it. We'd started

YC because it was something we were interested in. And some of the

problems we were trying to solve were endlessly difficult. How do

you recognize good founders? You could talk about that for years,

and we did; we still do.I'm better at some things than Jessica, and she's better at some

things than me. One of the things she's best at is judging people.

She's one of those rare individuals with x-ray vision for character.

She can see through any kind of faker almost immediately. Her

nickname within YC was the Social Radar, and this special power of

hers was critical in making YC what it is. The earlier you pick

startups, the more you're picking the founders. Later stage investors

get to try products and look at growth numbers. At the stage where

YC invests, there is often neither a product nor any numbers.Others thought YC had some special insight about the future of

technology. Mostly we had the same sort of insight Socrates claimed:

we at least knew we knew nothing. What made YC successful was being

able to pick good founders. We thought Airbnb was a bad idea. We

funded it because we liked the founders.During interviews, Robert and Trevor and I would pepper the applicants

with technical questions. Jessica would mostly watch. A lot of

the applicants probably read her as some kind of secretary, especially

early on, because she was the one who'd go out and get each new

group and she didn't ask many questions. She was ok with that. It

was easier for her to watch people if they didn't notice her. But

after the interview, the three of us would turn to Jessica and ask

"What does the Social Radar say?"

[1]Having the Social Radar at interviews wasn't just how we picked

founders who'd be successful. It was also how we picked founders

who were good people. At first we did this because we couldn't

help it. Imagine what it would feel like to have x-ray vision for

character. Being around bad people would be intolerable. So we'd

refuse to fund founders whose characters we had doubts about even

if we thought they'd be successful.Though we initially did this out of self-indulgence, it turned out

to be very valuable to YC. We didn't realize it in the beginning,

but the people we were picking would become the YC alumni network.

And once we picked them, unless they did something really egregious,

they were going to be part of it for life. Some now think YC's

alumni network is its most valuable feature. I personally think

YC's advice is pretty good too, but the alumni network is certainly

among the most valuable features. The level of trust and helpfulness

is remarkable for a group of such size. And Jessica is the main

reason why.(As we later learned, it probably cost us little to reject people

whose characters we had doubts about, because how good founders are

and how well they do are not orthogonal. If bad founders succeed

at all, they tend to sell early. The most successful founders are

almost all good.)If Jessica was so important to YC, why don't more people realize

it? Partly because I'm a writer, and writers always get disproportionate

attention. YC's brand was initially my brand, and our applicants

were people who'd read my essays. But there is another reason:

Jessica hates attention. Talking to reporters makes her nervous.

The thought of giving a talk paralyzes her. She was even uncomfortable

at our wedding, because the bride is always the center of attention.

[2]It's not just because she's shy that she hates attention, but because

it throws off the Social Radar. She can't be herself. You can't

watch people when everyone is watching you.Another reason attention worries her is that she hates bragging.

In anything she does that's publicly visible, her biggest fear

(after the obvious fear that it will be bad) is that it will seem

ostentatious. She says being too modest is a common problem for

women. But in her case it goes beyond that. She has a horror of

ostentation so visceral it's almost a phobia.She also hates fighting. She can't do it; she just shuts down. And

unfortunately there is a good deal of fighting in being the public

face of an organization.So although Jessica more than anyone made YC unique, the very

qualities that enabled her to do it mean she tends to get written

out of YC's history. Everyone buys this story that PG started YC

and his wife just kind of helped. Even YC's haters buy it. A

couple years ago when people were attacking us for not funding more

female founders (than exist), they all treated YC as identical with

PG. It would have spoiled the narrative to acknowledge Jessica's

central role at YC.Jessica was boiling mad that people were accusing her company of

sexism. I've never seen her angrier about anything. But she did

not contradict them. Not publicly. In private there was a great

deal of profanity. And she wrote three separate essays about the

question of female founders. But she could never bring herself to

publish any of them. She'd seen the level of vitriol in this debate,

and she shrank from engaging.

[3]It wasn't just because she disliked fighting. She's so sensitive

to character that it repels her even to fight with dishonest people.

The idea of mixing it up with linkbait journalists or Twitter trolls

would seem to her not merely frightening, but disgusting.But Jessica knew her example as a successful female founder would

encourage more women to start companies, so last year she did

something YC had never done before and hired a PR firm to get her

some interviews. At one of the first she did, the reporter brushed

aside her insights about startups and turned it into a sensationalistic

story about how some guy had tried to chat her up as she was waiting

outside the bar where they had arranged to meet. Jessica was

mortified, partly because the guy had done nothing wrong, but more

because the story treated her as a victim significant only for being

a woman, rather than one of the most knowledgeable investors in the

Valley.After that she told the PR firm to stop.You're not going to be hearing in the press about what Jessica has

achieved. So let me tell you what Jessica has achieved. Y Combinator

is fundamentally a nexus of people, like a university. It doesn't

make a product. What defines it is the people. Jessica more than

anyone curated and nurtured that collection of people. In that

sense she literally made YC.Jessica knows more about the qualities of startup founders than

anyone else ever has. Her immense data set and x-ray vision are the

perfect storm in that respect. The qualities of the founders are

the best predictor of how a startup will do. And startups are in

turn the most important source of growth in mature economies.The person who knows the most about the most important factor in

the growth of mature economies — that is who Jessica Livingston is.

Doesn't that sound like someone who should be better known?Notes[1]

Harj Taggar reminded me that while Jessica didn't ask many

questions, they tended to be important ones:"She was always good at sniffing out any red flags about the team

or their determination and disarmingly asking the right question,

which usually revealed more than the founders realized."[2]

Or more precisely, while she likes getting attention in the

sense of getting credit for what she has done, she doesn't like

getting attention in the sense of being watched in real time.

Unfortunately, not just for her but for a lot of people, how much

you get of the former depends a lot on how much you get of the

latter.Incidentally, if you saw Jessica at a public event, you would never

guess she

hates attention, because (a) she is very polite and (b) when she's

nervous, she expresses it by smiling more.[3]

The existence of people like Jessica is not just something

the mainstream media needs to learn to acknowledge, but something

feminists need to learn to acknowledge as well. There are successful

women who don't like to fight. Which means if the public conversation

about women consists of fighting, their voices will be silenced.There's a sort of Gresham's Law of conversations. If a conversation

reaches a certain level of incivility, the more thoughtful people

start to leave. No one understands female founders better than

Jessica. But it's unlikely anyone will ever hear her speak candidly

about the topic. She ventured a toe in that water a while ago, and

the reaction was so violent that she decided "never again."

Thanks to Sam Altman, Paul Buchheit, Patrick Collison,

Daniel Gackle, Carolynn

Levy, Jon Levy, Kirsty Nathoo, Robert Morris, Geoff Ralston, and

Harj Taggar for reading drafts of this. And yes, Jessica Livingston,

who made me cut surprisingly little.

A Way to Detect Bias

October 2015This will come as a surprise to a lot of people, but in some cases

it's possible to detect bias in a selection process without knowing

anything about the applicant pool. Which is exciting because among

other things it means third parties can use this technique to detect

bias whether those doing the selecting want them to or not.You can use this technique whenever (a) you have at least

a random sample of the applicants that were selected, (b) their

subsequent performance is measured, and (c) the groups of

applicants you're comparing have roughly equal distribution of ability.How does it work? Think about what it means to be biased. What

it means for a selection process to be biased against applicants

of type x is that it's harder for them to make it through. Which

means applicants of type x have to be better to get selected than

applicants not of type x.

[1]

Which means applicants of type x

who do make it through the selection process will outperform other

successful applicants. And if the performance of all the successful

applicants is measured, you'll know if they do.Of course, the test you use to measure performance must be a valid

one. And in particular it must not be invalidated by the bias you're

trying to measure.

But there are some domains where performance can be measured, and

in those detecting bias is straightforward. Want to know if the

selection process was biased against some type of applicant? Check

whether they outperform the others. This is not just a heuristic

for detecting bias. It's what bias means.For example, many suspect that venture capital firms are biased

against female founders. This would be easy to detect: among their

portfolio companies, do startups with female founders outperform

those without? A couple months ago, one VC firm (almost certainly

unintentionally) published a study showing bias of this type. First

Round Capital found that among its portfolio companies, startups

with female founders outperformed

those without by 63%.

[2]The reason I began by saying that this technique would come as a

surprise to many people is that we so rarely see analyses of this

type. I'm sure it will come as a surprise to First Round that they

performed one. I doubt anyone there realized that by limiting their

sample to their own portfolio, they were producing a study not of

startup trends but of their own biases when selecting companies.I predict we'll see this technique used more in the future. The

information needed to conduct such studies is increasingly available.

Data about who applies for things is usually closely guarded by the

organizations selecting them, but nowadays data about who gets

selected is often publicly available to anyone who takes the trouble

to aggregate it.

Notes[1]

This technique wouldn't work if the selection process looked

for different things from different types of applicants—for

example, if an employer hired men based on their ability but women

based on their appearance.[2]

As Paul Buchheit points out, First Round excluded their most

successful investment, Uber, from the study. And while it

makes sense to exclude outliers from some types of studies,

studies of returns from startup investing, which is all about

hitting outliers, are not one of them.

Thanks to Sam Altman, Jessica Livingston, and Geoff Ralston for reading

drafts of this.Arabic TranslationSwedish Translation

Write Like You Talk

October 2015Here's a simple trick for getting more people to read what you

write: write in spoken language.Something comes over most people when they start writing. They write

in a different language than they'd use if they were talking to a

friend. The sentence structure and even the words are different.

No one uses "pen" as a verb in spoken English. You'd feel like an

idiot using "pen" instead of "write" in a conversation with a friend.The last straw for me was a sentence I read a couple days ago:

The mercurial Spaniard himself declared: "After Altamira, all is

decadence."

It's from Neil Oliver's A History of Ancient Britain. I feel bad

making an example of this book, because it's no worse than lots of

others. But just imagine calling Picasso "the mercurial Spaniard" when

talking to a friend. Even one

sentence of this would raise eyebrows in conversation. And yet

people write whole books of it.Ok, so written and spoken language are different. Does that make

written language worse?If you want people to read and understand what you write, yes.

Written language is more complex, which makes it more work to read.

It's also more formal and distant, which gives the reader's attention

permission to drift. But perhaps worst of all, the complex sentences

and fancy words give you, the writer, the false impression that

you're saying more than you actually are.You don't need complex sentences to express complex ideas. When

specialists in some abstruse topic talk to one another about ideas

in their field, they don't use sentences any more complex than they

do when talking about what to have for lunch. They use different

words, certainly. But even those they use no more than necessary.

And in my experience, the harder the subject, the more informally

experts speak. Partly, I think, because they have less to prove,

and partly because the harder the ideas you're talking about, the

less you can afford to let language get in the way.Informal language is the athletic clothing of ideas.I'm not saying spoken language always works best. Poetry is as much

music as text, so you can say things you wouldn't say in conversation.

And there are a handful of writers who can get away with using fancy

language in prose. And then of course there are cases where writers

don't want to make it easy to understand what they're saying—in

corporate announcements of bad news, for example, or at the more

bogus end of the humanities. But for nearly everyone else, spoken

language is better.It seems to be hard for most people to write in spoken language.

So perhaps the best solution is to write your first draft the way

you usually would, then afterward look at each sentence and ask "Is

this the way I'd say this if I were talking to a friend?" If it

isn't, imagine what you would say, and use that instead. After a

while this filter will start to operate as you write. When you write

something you wouldn't say, you'll hear the clank as it hits the

page.Before I publish a new essay, I read it out loud and fix everything

that doesn't sound like conversation. I even fix bits that are

phonetically awkward; I don't know if that's necessary, but it

doesn't cost much.This trick may not always be enough. I've seen writing so far

removed from spoken language that it couldn't be fixed sentence by

sentence. For cases like that there's a more drastic solution.

After writing the first draft, try explaining to a friend what you

just wrote. Then replace the draft with what you said to your friend.People often tell me how much my essays sound like me talking.

The fact that this seems worthy of comment shows how rarely people

manage to write in spoken language. Otherwise everyone's writing

would sound like them talking.If you simply manage to write in spoken language, you'll be ahead

of 95% of writers. And it's so easy to do: just don't let a sentence

through unless it's the way you'd say it to a friend.Thanks to Patrick Collison and Jessica Livingston for reading drafts of this.Japanese TranslationArabic Translation

Default Alive or Default Dead?

October 2015When I talk to a startup that's been operating for more than 8 or

9 months, the first thing I want to know is almost always the same.

Assuming their expenses remain constant and their revenue growth

is what it has been over the last several months, do they make it to

profitability on the money they have left? Or to put it more

dramatically, by default do they live or die?The startling thing is how often the founders themselves don't know.

Half the founders I talk to don't know whether they're default alive

or default dead.If you're among that number, Trevor Blackwell has made a handy

calculator you can use to find out.The reason I want to know first whether a startup is default alive

or default dead is that the rest of the conversation depends on the

answer. If the company is default alive, we can talk about ambitious

new things they could do. If it's default dead, we probably need

to talk about how to save it. We know the current trajectory ends

badly. How can they get off that trajectory?Why do so few founders know whether they're default alive or default

dead? Mainly, I think, because they're not used to asking that.

It's not a question that makes sense to ask early on, any more than

it makes sense to ask a 3 year old how he plans to support

himself. But as the company grows older, the question switches from

meaningless to critical. That kind of switch often takes people

by surprise.I propose the following solution: instead of starting to ask too

late whether you're default alive or default dead, start asking too

early. It's hard to say precisely when the question switches

polarity. But it's probably not that dangerous to start worrying

too early that you're default dead, whereas it's very dangerous to

start worrying too late.The reason is a phenomenon I wrote about earlier: the

fatal pinch.

The fatal pinch is default dead + slow growth + not enough

time to fix it. And the way founders end up in it is by not realizing

that's where they're headed.There is another reason founders don't ask themselves whether they're

default alive or default dead: they assume it will be easy to raise

more money. But that assumption is often false, and worse still, the

more you depend on it, the falser it becomes.Maybe it will help to separate facts from hopes. Instead of thinking

of the future with vague optimism, explicitly separate the components.

Say "We're default dead, but we're counting on investors to save

us." Maybe as you say that, it will set off the same alarms in your

head that it does in mine. And if you set off the alarms sufficiently

early, you may be able to avoid the fatal pinch.It would be safe to be default dead if you could count on investors

saving you. As a rule their interest is a function of

growth. If you have steep revenue growth, say over 5x a year, you

can start to count on investors being interested even if you're not

profitable.

[1]

But investors are so fickle that you can never

do more than start to count on them. Sometimes something about your

business will spook investors even if your growth is great. So no

matter how good your growth is, you can never safely treat fundraising

as more than a plan A. You should always have a plan B as well: you

should know (as in write down) precisely what you'll need to do to

survive if you can't raise more money, and precisely when you'll

have to switch to plan B if plan A isn't working.In any case, growing fast versus operating cheaply is far from the

sharp dichotomy many founders assume it to be. In practice there

is surprisingly little connection between how much a startup spends

and how fast it grows. When a startup grows fast, it's usually

because the product hits a nerve, in the sense of hitting some big

need straight on. When a startup spends a lot, it's usually because

the product is expensive to develop or sell, or simply because

they're wasteful.If you're paying attention, you'll be asking at this point not just

how to avoid the fatal pinch, but how to avoid being default dead.

That one is easy: don't hire too fast. Hiring too fast is by far

the biggest killer of startups that raise money.

[2]Founders tell themselves they need to hire in order to grow. But

most err on the side of overestimating this need rather than

underestimating it. Why? Partly because there's so much work to

do. Naive founders think that if they can just hire enough

people, it will all get done. Partly because successful startups have

lots of employees, so it seems like that's what one does in order

to be successful. In fact the large staffs of successful startups

are probably more the effect of growth than the cause. And

partly because when founders have slow growth they don't want to

face what is usually the real reason: the product is not appealing

enough.Plus founders who've just raised money are often encouraged to

overhire by the VCs who funded them. Kill-or-cure strategies are

optimal for VCs because they're protected by the portfolio effect.

VCs want to blow you up, in one sense of the phrase or the other.

But as a founder your incentives are different. You want above all

to survive.

[3]Here's a common way startups die. They make something moderately

appealing and have decent initial growth. They raise their first

round fairly easily, because the founders seem smart and the idea

sounds plausible. But because the product is only moderately

appealing, growth is ok but not great. The founders convince

themselves that hiring a bunch of people is the way to boost growth.

Their investors agree. But (because the product is only moderately

appealing) the growth never comes. Now they're rapidly running out

of runway. They hope further investment will save them. But because

they have high expenses and slow growth, they're now unappealing

to investors. They're unable to raise more, and the company dies.What the company should have done is address the fundamental problem:

that the product is only moderately appealing. Hiring people is

rarely the way to fix that. More often than not it makes it harder.

At this early stage, the product needs to evolve more than to be

"built out," and that's usually easier with fewer people.

[4]Asking whether you're default alive or default dead may save you

from this. Maybe the alarm bells it sets off will counteract the

forces that push you to overhire. Instead you'll be compelled to

seek growth in other ways. For example, by doing

things that don't scale, or by redesigning the product in the

way only founders can.

And for many if not most startups, these paths to growth will be

the ones that actually work.Airbnb waited 4 months after raising money at the end of Y Combinator

before they hired their first employee. In the meantime the founders

were terribly overworked. But they were overworked evolving Airbnb

into the astonishingly successful organism it is now.Notes[1]

Steep usage growth will also interest investors. Revenue

will ultimately be a constant multiple of usage, so x% usage growth

predicts x% revenue growth. But in practice investors discount

merely predicted revenue, so if you're measuring usage you need a

higher growth rate to impress investors.[2]

Startups that don't raise money are saved from hiring too

fast because they can't afford to. But that doesn't mean you should

avoid raising money in order to avoid this problem, any more than

that total abstinence is the only way to avoid becoming an alcoholic.[3]

I would not be surprised if VCs' tendency to push founders

to overhire is not even in their own interest. They don't know how

many of the companies that get killed by overspending might have

done well if they'd survived. My guess is a significant number.[4]

After reading a draft, Sam Altman wrote:"I think you should make the hiring point more strongly. I think

it's roughly correct to say that YC's most successful companies

have never been the fastest to hire, and one of the marks of a great

founder is being able to resist this urge."Paul Buchheit adds:"A related problem that I see a lot is premature scaling—founders

take a small business that isn't really working (bad unit economics,

typically) and then scale it up because they want impressive growth

numbers. This is similar to over-hiring in that it makes the business

much harder to fix once it's big, plus they are bleeding cash really

fast."

Thanks to Sam Altman, Paul Buchheit, Joe Gebbia, Jessica Livingston,

and Geoff Ralston for reading drafts of this.

Why It's Safe for Founders to Be Nice

August 2015I recently got an email from a founder that helped me understand

something important: why it's safe for startup founders to be nice

people.I grew up with a cartoon idea of a very successful businessman (in

the cartoon it was always a man): a rapacious, cigar-smoking,

table-thumping guy in his fifties who wins by exercising power, and

isn't too fussy about how. As I've written before, one of

the things that has surprised me most about startups is

how few of

the most successful founders are like that. Maybe successful people

in other industries are; I don't know; but not startup founders.

[1]I knew this empirically, but I never saw the math of why till I got

this founder's email. In it he said he worried that he was

fundamentally soft-hearted and tended to give away too much for

free. He thought perhaps he needed "a little dose of sociopath-ness."I told him not to worry about it, because so long as he built

something good enough to spread by word of mouth, he'd have a

superlinear growth curve. If he was bad at extracting money from

people, at worst this curve would be some constant multiple less

than 1 of what it might have been. But a constant multiple of any

curve is exactly the same shape. The numbers on the Y axis are

smaller, but the curve is just as steep, and when anything grows

at the rate of a successful startup, the Y axis will take care of

itself.Some examples will make this clear. Suppose your company is making

$1000 a month now, and you've made something so great that it's

growing at 5% a week. Two years from now, you'll be making about

$160k a month.Now suppose you're so un-rapacious that you only extract half as

much from your users as you could. That means two years later

you'll be making $80k a month instead of $160k. How far behind are

you? How long will it take to catch up with where you'd have been

if you were extracting every penny? A mere 15 weeks. After two

years, the un-rapacious founder is only 3.5 months behind the

rapacious one.

[2]If you're going to optimize a number, the one to choose is your

growth rate. Suppose as before that you only extract half as much

from users as you could, but that you're able to grow 6% a week

instead of 5%. Now how are you doing compared to the rapacious

founder after two years? You're already ahead—$214k a month

versus $160k—and pulling away fast. In another year you'll be

making $4.4 million a month to the rapacious founder's $2 million.Obviously one case where it would help to be rapacious is when

growth depends on that. What makes startups different is that

usually it doesn't. Startups usually win by making something so

great that people recommend it to their friends. And being rapacious

not only doesn't help you do that, but probably hurts.

[3]The reason startup founders can safely be nice is that making great

things is compounded, and rapacity isn't.So if you're a founder, here's a deal you can make with yourself

that will both make you happy and make your company successful.

Tell yourself you can be as nice as you want, so long as you work

hard on your growth rate to compensate. Most successful startups

make that tradeoff unconsciously. Maybe if you do it consciously

you'll do it even better.Notes[1]

Many think successful startup founders are driven by money.

In fact the secret weapon of the most successful founders is that

they aren't. If they were, they'd have taken one of the acquisition

offers that every fast-growing startup gets on the way up. What

drives the most successful founders is the same thing that drives

most people who make things: the company is their project.[2]

In fact since 2 ≈ 1.05 ^ 15, the un-rapacious founder is

always 15 weeks behind the rapacious one.[3]

The other reason it might help to be good at squeezing money

out of customers is that startups usually lose money at first, and

making more per customer makes it easier to get to profitability

before your initial funding runs out. But while it is very common

for startups to die

from running through their initial funding and then being unable

to raise more, the underlying cause is usually slow growth or

excessive spending rather than insufficient effort to extract money

from existing customers.Thanks to Sam Altman, Harj Taggar, Jessica Livingston, and

Geoff Ralston for reading drafts of this, and to Randall Bennett

for being such a nice guy.

Change Your Name

August 2015If you have a US startup called X and you don't have x.com, you

should probably change your name.The reason is not just that people can't find you. For companies

with mobile apps, especially, having the right domain name is not

as critical as it used to be for getting users. The problem with

not having the .com of your name is that it signals weakness. Unless

you're so big that your reputation precedes you, a marginal domain

suggests you're a marginal company. Whereas

(as Stripe shows)

having x.com signals strength even if it has no relation to what you

do.Even good founders can be in denial about this. Their denial derives

from two very powerful forces: identity, and lack of imagination.X is what we are, founders think. There's no other name as good.

Both of which are false.You can fix the first by stepping back from the problem. Imagine

you'd called your company something else. If you had, surely you'd

be just as attached to that name as you are to your current one.

The idea of switching to your current name would seem repellent.

[1]There's nothing intrinsically great about your current name. Nearly

all your attachment to it comes from it being attached to you.

[2]The way to neutralize the second source of denial, your inability

to think of other potential names, is to acknowledge that you're

bad at naming. Naming is a completely separate skill from those

you need to be a good founder. You can be a great startup founder

but hopeless at thinking of names for your company.Once you acknowledge that, you stop believing there is nothing else

you could be called. There are lots of other potential names that

are as good or better; you just can't think of them.How do you find them? One answer is the default way to solve

problems you're bad at: find someone else who can think of names.

But with company names there is another possible

approach. It turns out almost any word or word pair that is not

an obviously bad name is a sufficiently good one, and the number

of such domains is so large that you can find plenty that are cheap

or even untaken. So make a list and try to buy some. That's what

Stripe

did. (Their search also turned up parse.com, which their

friends at Parse took.)The reason I know that naming companies is a distinct skill orthogonal

to the others you need in a startup is that I happen to have it.

Back when I was running YC and did more office hours with startups,

I would often help them find new names. 80% of the time we could

find at least one good name in a 20 minute office hour slot.Now when I do office hours I have to focus on more important

questions, like what the company is doing. I tell them when they

need to change their name. But I know the power of the forces that

have them in their grip, so I know most won't listen.

[3]There are of course examples of startups that have succeeded without

having the .com of their name. There are startups that have succeeded despite any

number of different mistakes. But this mistake is less excusable

than most. It's something that can be fixed in a couple days if

you have sufficient discipline to acknowledge the problem.100% of the top 20 YC companies by valuation have the .com of their

name. 94% of the top 50 do. But only 66% of companies in the current

batch have the .com of their name. Which suggests there are lessons

ahead for most of the rest, one way or another.

Notes[1]

Incidentally, this thought experiment works for

nationality and religion too.[2]

The liking you have for a name that has become part of your

identity manifests itself not directly, which would be easy to

discount, but as a collection of specious beliefs about its intrinsic

qualities. (This too is true of nationality and religion as well.)[3]

Sometimes founders know it's a problem that they don't have

the .com of their name, but delusion strikes a step later in the belief that they'll

be able to buy it despite having no evidence it's for sale. Don't

believe a domain is for sale unless the owner has already told you

an asking price.

Thanks to Sam Altman, Jessica Livingston, and Geoff Ralston

for reading drafts of this.

What Microsoft Is this the Altair Basic of?

February 2015One of the most valuable exercises you can try if you want to

understand startups is to look at the most successful companies and

explain why they were not as lame as they seemed when they first

launched. Because they practically all seemed lame at first. Not

just small, lame. Not just the first step up a big mountain. More

like the first step into a swamp.A Basic interpreter for the Altair? How could that ever grow into

a giant company? People sleeping on airbeds in strangers' apartments?

A web site for college students to stalk one another? A wimpy

little single-board computer for hobbyists that used a TV as a

monitor? A new search engine, when there were already about 10,

and they were all trying to de-emphasize search? These ideas didn't

just seem small. They seemed wrong. They were the kind of ideas

you could not merely ignore, but ridicule.Often the founders themselves didn't know why their ideas were

promising. They were attracted to these ideas by instinct, because

they were living in the future and

they sensed that something was missing. But they could not have

put into words exactly how their ugly ducklings were going to grow

into big, beautiful swans.Most people's first impulse when they hear about a lame-sounding

new startup idea is to make fun of it. Even a lot of people who

should know better.When I encounter a startup with a lame-sounding idea, I ask "What

Microsoft is this the Altair Basic of?" Now it's a puzzle, and the

burden is on me to solve it. Sometimes I can't think of an answer,

especially when the idea is a made-up one. But it's remarkable how

often there does turn out to be an answer. Often it's one the

founders themselves hadn't seen yet.Intriguingly, there are sometimes multiple answers. I talked to a

startup a few days ago that could grow into 3 distinct Microsofts.

They'd probably vary in size by orders of magnitude. But you can

never predict how big a Microsoft is going to be, so in cases like

that I encourage founders to follow whichever path is most immediately

exciting to them. Their instincts got them this far. Why stop now?

The Ronco Principle

January 2015No one, VC or angel, has invested in more of the top startups than

Ron Conway. He knows what happened in every deal in the Valley,

half the time because he arranged it.And yet he's a super nice guy. In fact, nice is not the word.

Ronco is good. I know of zero instances in which he has behaved

badly. It's hard even to imagine.When I first came to Silicon Valley I thought "How lucky that someone

so powerful is so benevolent." But gradually I realized it wasn't

luck. It was by being benevolent that Ronco became so powerful.

All the deals he gets to invest in come to him through referrals.

Google did. Facebook did. Twitter was a referral from Evan Williams

himself. And the reason so many people refer deals to him is that

he's proven himself to be a good guy.Good does not mean being a pushover. I would not want to face an

angry Ronco. But if Ron's angry at you, it's because you did

something wrong. Ron is so old school he's Old Testament. He will

smite you in his just wrath, but there's no malice in it.In almost every domain there are advantages to seeming good. It

makes people trust you. But actually being good is an expensive

way to seem good. To an amoral person it might seem to be overkill.In some fields it might be, but apparently not in the startup world.

Though plenty of investors are jerks, there is a clear trend among

them: the most successful investors are also the most upstanding.

[1]It was not always this way. I would not feel confident saying that

about investors twenty years ago.What changed? The startup world became more transparent and more

unpredictable. Both make it harder to seem good without actually

being good.It's obvious why transparency has that effect. When an investor

maltreats a founder now, it gets out. Maybe not all the way to the

press, but other founders hear about it, and that investor

starts to lose deals.

[2]The effect of unpredictability is more subtle. It increases the

work of being inconsistent. If you're going to be two-faced, you

have to know who you should be nice to and who you can get away

with being nasty to. In the startup world, things change so rapidly

that you can't tell. The random college kid you talk to today might

in a couple years be the CEO of the hottest startup in the Valley.

If you can't tell who to be nice to, you have to be nice to everyone.

And probably the only people who can manage that are the people who

are genuinely good.In a sufficiently connected and unpredictable world, you can't seem

good without being good.As often happens, Ron discovered how to be the investor of the

future by accident. He didn't foresee the future of startup

investing, realize it would pay to be upstanding, and force himself

to behave that way. It would feel unnatural to him to behave any

other way. He was already

living in the future.Fortunately that future is not limited to the startup world. The

startup world is more transparent and unpredictable than most, but

almost everywhere the trend is in that direction.Notes[1]

I'm not saying that if you sort investors by benevolence

you've also sorted them by returns, but rather that if you do a

scatterplot with benevolence on the x axis and returns on the y,

you'd see a clear upward trend.[2]

Y Combinator in particular, because it aggregates data

from so many startups, has a pretty comprehensive view of

investor behavior.

Thanks to Sam Altman and Jessica Livingston for reading drafts of

this.Japanese Translation

What Doesn't Seem Like Work?

January 2015My father is a mathematician. For most of my childhood he worked

for Westinghouse, modelling nuclear reactors.He was one of those lucky people who know early on what they want to

do. When you talk to him about his childhood, there's a clear

watershed at about age 12, when he "got interested in maths."He

grew up in the small Welsh seacoast town of Pwllheli. As we retraced

his walk to school on Google Street View, he said that it had been

nice growing up in the country."Didn't it get boring when you got to be about 15?" I asked."No," he said, "by then I was interested in maths."In another conversation he told me that what he really liked was

solving problems. To me the exercises at the end of each chapter

in a math textbook represent work, or at best a way to reinforce

what you learned in that chapter. To him the problems were the

reward. The text of each chapter was just some advice about solving

them. He said that as soon as he got a new textbook he'd immediately

work out all the problems — to the slight annoyance of his teacher,

since the class was supposed to work through the book gradually.Few people know so early or so certainly what they want to work on.

But talking to my father reminded me of a heuristic the rest of us

can use. If something that seems like work to other people doesn't

seem like work to you, that's something you're well suited for.

For example, a lot of programmers I know, including me, actually

like debugging. It's not something people tend to volunteer; one

likes it the way one likes popping zits. But you may have to like

debugging to like programming, considering the degree to which

programming consists of it.The stranger your tastes seem to other people, the stronger evidence

they probably are of what you should do. When I was in college I

used to write papers for my friends. It was quite interesting to

write a paper for a class I wasn't taking. Plus they were always

so relieved.It seemed curious that the same task could be painful to one person

and pleasant to another, but I didn't realize at the time what this

imbalance implied, because I wasn't looking for it. I didn't realize

how hard it can be to decide what you should work on, and that you

sometimes have to figure it out from subtle clues, like a detective

solving a case in a mystery novel. So I bet it would help a lot

of people to ask themselves about this explicitly. What seems like

work to other people that doesn't seem like work to you?

Thanks to Sam Altman, Trevor Blackwell, Jessica Livingston,

Robert Morris, and my father for reading drafts of this.Robert Morris: All About ProgrammingFrench Translation

Don't Talk to Corp Dev

January 2015Corporate Development, aka corp dev, is the group within companies

that buys other companies. If you're talking to someone from corp

dev, that's why, whether you realize it yet or not.It's usually a mistake to talk to corp dev unless (a) you want to

sell your company right now and (b) you're sufficiently likely to

get an offer at an acceptable price. In practice that means startups

should only talk to corp dev when they're either doing really well

or really badly. If you're doing really badly, meaning the company

is about to die, you may as well talk to them, because you have

nothing to lose. And if you're doing really well, you can safely

talk to them, because you both know the price will have to be high,

and if they show the slightest sign of wasting your time, you'll

be confident enough to tell them to get lost.The danger is to companies in the middle. Particularly to young

companies that are growing fast, but haven't been doing it for long

enough to have grown big yet. It's usually a mistake for a promising

company less than a year old even to talk to corp dev.But it's a mistake founders constantly make. When someone from

corp dev wants to meet, the founders tell themselves they should

at least find out what they want. Besides, they don't want to

offend Big Company by refusing to meet.Well, I'll tell you what they want. They want to talk about buying

you. That's what the title "corp dev" means. So before agreeing

to meet with someone from corp dev, ask yourselves, "Do we want to

sell the company right now?" And if the answer is no, tell them

"Sorry, but we're focusing on growing the company." They won't be

offended. And certainly the founders of Big Company won't be

offended. If anything they'll think more highly of you. You'll

remind them of themselves. They didn't sell either; that's why

they're in a position now to buy other companies.

[1]Most founders who get contacted by corp dev already know what it

means. And yet even when they know what corp dev does and know

they don't want to sell, they take the meeting. Why do they do it?

The same mix of denial and wishful thinking that underlies most

mistakes founders make. It's flattering to talk to someone who wants

to buy you. And who knows, maybe their offer will be surprisingly

high. You should at least see what it is, right?No. If they were going to send you an offer immediately by email,

sure, you might as well open it. But that is not how conversations

with corp dev work. If you get an offer at all, it will be at the

end of a long and unbelievably distracting process. And if the

offer is surprising, it will be surprisingly low.Distractions are the thing you can least afford in a startup. And

conversations with corp dev are the worst sort of distraction,

because as well as consuming your attention they undermine your

morale. One of the tricks to surviving a grueling process is not

to stop and think how tired you are. Instead you get into a sort

of flow.

[2]

Imagine what it would do to you if at mile 20 of a

marathon, someone ran up beside you and said "You must feel really

tired. Would you like to stop and take a rest?" Conversations

with corp dev are like that but worse, because the suggestion of

stopping gets combined in your mind with the imaginary high price

you think they'll offer.And then you're really in trouble. If they can, corp dev people

like to turn the tables on you. They like to get you to the point

where you're trying to convince them to buy instead of them trying

to convince you to sell. And surprisingly often they succeed.This is a very slippery slope, greased with some of the most powerful

forces that can work on founders' minds, and attended by an experienced

professional whose full time job is to push you down it.Their tactics in pushing you down that slope are usually fairly

brutal. Corp dev people's whole job is to buy companies, and they

don't even get to choose which. The only way their performance is

measured is by how cheaply they can buy you, and the more ambitious

ones will stop at nothing to achieve that. For example, they'll

almost always start with a lowball offer, just to see if you'll

take it. Even if you don't, a low initial offer will demoralize you

and make you easier to manipulate.And that is the most innocent of their tactics. Just wait till

you've agreed on a price and think you have a done deal, and then

they come back and say their boss has vetoed the deal and won't do

it for more than half the agreed upon price. Happens all the time.

If you think investors can behave badly, it's nothing compared to

what corp dev people can do. Even corp dev people at companies

that are otherwise benevolent.I remember once complaining to a

friend at Google about some nasty trick their corp dev people had

pulled on a YC startup."What happened to Don't be Evil?" I asked."I don't think corp dev got the memo," he replied.The tactics you encounter in M&A conversations can be like nothing

you've experienced in the otherwise comparatively

upstanding world

of Silicon Valley. It's as if a chunk of genetic material from the

old-fashioned robber baron business world got incorporated into the

startup world.

[3]The simplest way to protect yourself is to use the trick that John

D. Rockefeller, whose grandfather was an alcoholic, used to protect

himself from becoming one. He once told a Sunday school class

Boys, do you know why I never became a drunkard? Because I never

took the first drink.

Do you want to sell your company right now? Not eventually, right

now. If not, just don't take the first meeting. They won't be

offended. And you in turn will be guaranteed to be spared one of

the worst experiences that can happen to a startup.If you do want to sell, there's another set of

techniques

for doing

that. But the biggest mistake founders make in dealing with corp

dev is not doing a bad job of talking to them when they're ready

to, but talking to them before they are. So if you remember only

the title of this essay, you already know most of what you need to

know about M&A in the first year.Notes[1]

I'm not saying you should never sell. I'm saying you should

be clear in your own mind about whether you want to sell or not,

and not be led by manipulation or wishful thinking into trying to

sell earlier than you otherwise would have.[2]

In a startup, as in most competitive sports, the task at hand

almost does this for you; you're too busy to feel tired. But when

you lose that protection, e.g. at the final whistle, the fatigue

hits you like a wave. To talk to corp dev is to let yourself feel

it mid-game.[3]

To be fair, the apparent misdeeds of corp dev people are magnified

by the fact that they function as the face of a large organization

that often doesn't know its own mind. Acquirers can be surprisingly

indecisive about acquisitions, and their flakiness is indistinguishable

from dishonesty by the time it filters down to you.Thanks to Marc Andreessen, Jessica Livingston, Geoff

Ralston, and Qasar Younis for reading drafts of this.

Let the Other 95% of Great Programmers In

December 2014American technology companies want the government to make immigration

easier because they say they can't find enough programmers in the

US. Anti-immigration people say that instead of letting foreigners

take these jobs, we should train more Americans to be programmers.

Who's right?The technology companies are right. What the anti-immigration people

don't understand is that there is a huge variation in ability between

competent programmers and exceptional ones, and while you can train

people to be competent, you can't train them to be exceptional.

Exceptional programmers have an aptitude for and

interest in

programming that is not merely the product of training.

[1]The US has less than 5% of the world's population. Which means if

the qualities that make someone a great programmer are evenly

distributed, 95% of great programmers are born outside the US.The anti-immigration people have to invent some explanation to

account for all the effort technology companies have expended trying

to make immigration easier. So they claim it's because they want

to drive down salaries. But if you talk to startups, you find

practically every one over a certain size has gone through legal

contortions to get programmers into the US, where they then

paid them the same as they'd have paid an American. Why would they

go to extra trouble to get programmers for the same price? The

only explanation is that they're telling the truth: there are just

not enough great programmers to go around.

[2]I asked the CEO of a startup with about 70 programmers how many

more he'd hire if he could get all the great programmers he wanted.

He said "We'd hire 30 tomorrow morning." And this is one of the

hot startups that always win recruiting battles. It's the same all

over Silicon Valley. Startups are that constrained for talent.It would be great if more Americans were trained as programmers,

but no amount of training can flip a ratio as overwhelming as 95

to 5. Especially since programmers are being trained in other

countries too. Barring some cataclysm, it will always be true that

most great programmers are born outside the US. It will always be

true that most people who are great at anything are born outside

the US.

[3]Exceptional performance implies immigration. A country with only

a few percent of the world's population will be exceptional in some

field only if there are a lot of immigrants working in it.But this whole discussion has taken something for granted: that if

we let more great programmers into the US, they'll want to come.

That's true now, and we don't realize how lucky we are that it is.

If we want to keep this option open, the best way to do it is to

take advantage of it: the more of the world's great programmers are

here, the more the rest will want to come here.And if we don't, the US could be seriously fucked. I realize that's

strong language, but the people dithering about this don't seem to

realize the power of the forces at work here. Technology gives the

best programmers huge leverage. The world market in programmers

seems to be becoming dramatically more liquid. And since good

people like good colleagues, that means the best programmers could

collect in just a few hubs. Maybe mostly in one hub.What if most of the great programmers collected in one hub, and it

wasn't here? That scenario may seem unlikely now, but it won't be

if things change as much in the next 50 years as they did in the

last 50.We have the potential to ensure that the US remains a technology

superpower just by letting in a few thousand great programmers a

year. What a colossal mistake it would be to let that opportunity

slip. It could easily be the defining mistake this generation of

American politicians later become famous for. And unlike other

potential mistakes on that scale, it costs nothing to fix.So please, get on with it.

Notes[1]

How much better is a great programmer than an ordinary one?

So much better that you can't even measure the difference directly.

A great programmer doesn't merely do the same work faster. A great

programmer will invent things an ordinary programmer would never

even think of. This doesn't mean a great programmer is infinitely

more valuable, because any invention has a finite market value.

But it's easy to imagine cases where a great programmer might invent

things worth 100x or even 1000x an average programmer's salary.[2]

There are a handful of consulting firms that rent out big

pools of foreign programmers they bring in on H1-B visas. By all

means crack down on these. It should be easy to write legislation

that distinguishes them, because they are so different from technology

companies. But it is dishonest of the anti-immigration people to

claim that companies like Google and Facebook are driven by the

same motives. An influx of inexpensive but mediocre programmers

is the last thing they'd want; it would destroy them.[3]

Though this essay talks about programmers, the group of people

we need to import is broader, ranging from designers to programmers

to electrical engineers. The best one could do as a general term

might be "digital talent." It seemed better to make the argument a

little too narrow than to confuse everyone with a neologism.

Thanks to Sam Altman, John Collison, Patrick Collison, Jessica

Livingston, Geoff Ralston, Fred Wilson, and Qasar Younis for reading

drafts of this.Spanish Translation

How to Be an Expert in a Changing World

December 2014If the world were static, we could have monotonically increasing

confidence in our beliefs. The more (and more varied) experience

a belief survived, the less likely it would be false. Most people

implicitly believe something like this about their opinions. And

they're justified in doing so with opinions about things that don't

change much, like human nature. But you can't trust your opinions

in the same way about things that change, which could include

practically everything else.When experts are wrong, it's often because they're experts on an

earlier version of the world.Is it possible to avoid that? Can you protect yourself against

obsolete beliefs? To some extent, yes. I spent almost a decade

investing in early stage startups, and curiously enough protecting

yourself against obsolete beliefs is exactly what you have to do

to succeed as a startup investor. Most really good startup ideas

look like bad ideas at first, and many of those look bad specifically

because some change in the world just switched them from bad to

good. I spent a lot of time learning to recognize such ideas, and

the techniques I used may be applicable to ideas in general.The first step is to have an explicit belief in change. People who

fall victim to a monotonically increasing confidence in their

opinions are implicitly concluding the world is static. If you

consciously remind yourself it isn't, you start to look for change.Where should one look for it? Beyond the moderately useful

generalization that human nature doesn't change much, the unfortunate

fact is that change is hard to predict. This is largely a tautology

but worth remembering all the same: change that matters usually

comes from an unforeseen quarter.So I don't even try to predict it. When I get asked in interviews

to predict the future, I always have to struggle to come up with

something plausible-sounding on the fly, like a student who hasn't

prepared for an exam.

[1]

But it's not out of laziness that I haven't

prepared. It seems to me that beliefs about the future are so

rarely correct that they usually aren't worth the extra rigidity

they impose, and that the best strategy is simply to be aggressively

open-minded. Instead of trying to point yourself in the right

direction, admit you have no idea what the right direction is, and

try instead to be super sensitive to the winds of change.It's ok to have working hypotheses, even though they may constrain

you a bit, because they also motivate you. It's exciting to chase

things and exciting to try to guess answers. But you have to be

disciplined about not letting your hypotheses harden into anything

more.

[2]I believe this passive m.o. works not just for evaluating new ideas

but also for having them. The way to come up with new ideas is not

to try explicitly to, but to try to solve problems and simply not

discount weird hunches you have in the process.The winds of change originate in the unconscious minds of domain

experts. If you're sufficiently expert in a field, any weird idea

or apparently irrelevant question that occurs to you is ipso facto

worth exploring.

[3]

Within Y Combinator, when an idea is described

as crazy, it's a compliment—in fact, on average probably a

higher compliment than when an idea is described as good.Startup investors have extraordinary incentives for correcting

obsolete beliefs. If they can realize before other investors that

some apparently unpromising startup isn't, they can make a huge

amount of money. But the incentives are more than just financial.

Investors' opinions are explicitly tested: startups come to them

and they have to say yes or no, and then, fairly quickly, they learn

whether they guessed right. The investors who say no to a Google

(and there were several) will remember it for the rest of their

lives.Anyone who must in some sense bet on ideas rather than merely

commenting on them has similar incentives. Which means anyone who

wants such incentives can have them, by turning their comments into

bets: if you write about a topic in some fairly durable and public

form, you'll find you worry much more about getting things right

than most people would in a casual conversation.

[4]Another trick I've found to protect myself against obsolete beliefs

is to focus initially on people rather than ideas. Though the nature

of future discoveries is hard to predict, I've found I can predict

quite well what sort of people will make them. Good new ideas come

from earnest, energetic, independent-minded people.Betting on people over ideas saved me countless times as an investor.

We thought Airbnb was a bad idea, for example. But we could tell

the founders were earnest, energetic, and independent-minded.

(Indeed, almost pathologically so.) So we suspended disbelief and

funded them.This too seems a technique that should be generally applicable.

Surround yourself with the sort of people new ideas come from. If

you want to notice quickly when your beliefs become obsolete, you

can't do better than to be friends with the people whose discoveries

will make them so.It's hard enough already not to become the prisoner of your own

expertise, but it will only get harder, because change is accelerating.

That's not a recent trend; change has been accelerating since the

paleolithic era. Ideas beget ideas. I don't expect that to change.

But I could be wrong.

Notes[1]

My usual trick is to talk about aspects of the present that

most people haven't noticed yet.[2]

Especially if they become well enough known that people start

to identify them with you. You have to be extra skeptical about

things you want to believe, and once a hypothesis starts to be

identified with you, it will almost certainly start to be in that

category.[3]

In practice "sufficiently expert" doesn't require one to be

recognized as an expert—which is a trailing indicator in any

case. In many fields a year of focused work plus caring a lot would

be enough.[4]

Though they are public and persist indefinitely, comments on

e.g. forums and places like Twitter seem empirically to work like

casual conversation. The threshold may be whether what you write

has a title.

Thanks to Sam Altman, Patrick Collison, and Robert Morris

for reading drafts of this.Spanish TranslationArabic Translation

How You Know

December 2014I've read Villehardouin's chronicle of the Fourth Crusade at least

two times, maybe three. And yet if I had to write down everything

I remember from it, I doubt it would amount to much more than a

page. Multiply this times several hundred, and I get an uneasy

feeling when I look at my bookshelves. What use is it to read all

these books if I remember so little from them?A few months ago, as I was reading Constance Reid's excellent

biography of Hilbert, I figured out if not the answer to this

question, at least something that made me feel better about it.

She writes:

Hilbert had no patience with mathematical lectures which filled

the students with facts but did not teach them how to frame a

problem and solve it. He often used to tell them that "a perfect

formulation of a problem is already half its solution."

That has always seemed to me an important point, and I was even

more convinced of it after hearing it confirmed by Hilbert.But how had I come to believe in this idea in the first place? A

combination of my own experience and other things I'd read. None

of which I could at that moment remember! And eventually I'd forget

that Hilbert had confirmed it too. But my increased belief in the

importance of this idea would remain something I'd learned from

this book, even after I'd forgotten I'd learned it.Reading and experience train your model of the world. And even if

you forget the experience or what you read, its effect on your model

of the world persists. Your mind is like a compiled program you've

lost the source of. It works, but you don't know why.The place to look for what I learned from Villehardouin's chronicle

is not what I remember from it, but my mental models of the crusades,

Venice, medieval culture, siege warfare, and so on. Which doesn't

mean I couldn't have read more attentively, but at least the harvest

of reading is not so miserably small as it might seem.This is one of those things that seem obvious in retrospect. But

it was a surprise to me and presumably would be to anyone else who

felt uneasy about (apparently) forgetting so much they'd read.Realizing it does more than make you feel a little better about

forgetting, though. There are specific implications.For example, reading and experience are usually "compiled" at the

time they happen, using the state of your brain at that time. The

same book would get compiled differently at different points in

your life. Which means it is very much worth reading important

books multiple times. I always used to feel some misgivings about

rereading books. I unconsciously lumped reading together with work

like carpentry, where having to do something again is a sign you

did it wrong the first time. Whereas now the phrase "already read"

seems almost ill-formed.Intriguingly, this implication isn't limited to books. Technology

will increasingly make it possible to relive our experiences. When

people do that today it's usually to enjoy them again (e.g. when

looking at pictures of a trip) or to find the origin of some bug in

their compiled code (e.g. when Stephen Fry succeeded in remembering

the childhood trauma that prevented him from singing). But as

technologies for recording and playing back your life improve, it

may become common for people to relive experiences without any goal

in mind, simply to learn from them again as one might when rereading

a book.Eventually we may be able not just to play back experiences but

also to index and even edit them. So although not knowing how you

know things may seem part of being human, it may not be.

Thanks to Sam Altman, Jessica Livingston, and Robert Morris for reading

drafts of this.Japanese Translation

The Fatal Pinch

December 2014Many startups go through a point a few months before they die where

although they have a significant amount of money in the bank, they're

also losing a lot each month, and revenue growth is either nonexistent

or mediocre. The company has, say, 6 months of runway. Or to put

it more brutally, 6 months before they're out of business. They

expect to avoid that by raising more from investors.

[1]That last sentence is the fatal one.There may be nothing founders are so prone to delude themselves

about as how interested investors will be in giving them additional

funding. It's hard to convince investors the first time too, but

founders expect that. What bites them the second time is a confluence

of three forces:

The company is spending more now than it did the first time

it raised money. Investors have much higher standards for companies that have

already raised money. The company is now starting to read as a failure. The first

time it raised money, it was neither a success nor a failure; it

was too early to ask. Now it's possible to ask that question, and

the default answer is failure, because at this point that is the

default outcome.

I'm going to call the situation I described in the first paragraph "the fatal pinch." I try to resist

coining phrases, but making up a name for this situation may snap

founders into realizing when they're in it.One of the things that makes the fatal pinch so dangerous is

that it's self-reinforcing. Founders overestimate their chances

of raising more money, and so are slack about reaching

profitability, which further decreases their chances of raising

money.Now that you know about the fatal pinch, how do you avoid it? Y Combinator tells

founders who raise money to act as if it's the last they'll ever

get. Because the self-reinforcing nature of this situation works

the other way too: the less you need further investment, the easier

it is to get.What do you do if you're already in the fatal pinch? The

first step is to re-evaluate the probability of raising more money.

I will now, by an amazing feat of clairvoyance, do this for you:

the probability is zero.

[2]Three options remain: you can shut down the company, you can increase

how much you make, and you can decrease how much you spend.You should shut down the company if you're certain it will

fail no matter what you do. Then at least you can give back the

money you have left, and save yourself however many months you would

have spent riding it down.Companies rarely have to fail though. What I'm really doing

here is giving you the option of admitting you've already given up.If you don't want to shut down the company, that leaves increasing

revenues and decreasing expenses. In most startups, expenses =

people, and decreasing expenses = firing people.

[3]

Deciding to

fire people is usually hard, but there's one case in which it

shouldn't be: when there are people you already know you should

fire but you're in denial about it. If so, now's the time.If that makes you profitable, or will enable you to make it to

profitability on the money you have left, you've avoided the immediate

danger.Otherwise you have three options: you either have to fire good

people, get some or all of the employees to take less salary for a

while, or increase revenues.Getting people to take less salary is a weak solution that will

only work when the problem isn't too bad. If your current trajectory

won't quite get you to profitability but you can get over the threshold

by cutting salaries a little,

you might be able to make the case to everyone for doing it.

Otherwise you're probably just postponing the problem, and that

will be obvious to the people whose salaries you're proposing to

cut.

[4]Which leaves two options, firing good people and making more money.

While trying to balance them, keep in mind the eventual goal: to be

a successful product company in the sense of having a single thing

lots of people use.You should lean more toward firing people if the source of your

trouble is overhiring. If you went out and hired 15 people before

you even knew what you were building, you've created a broken

company. You need to figure out what you're building, and it will

probably be easier to do that with a handful of people than 15.

Plus those 15 people might not even be the ones you need for whatever

you end up building. So the solution may be to shrink and then

figure out what direction to grow in. After all, you're not doing

those 15 people any favors if you fly the company into ground with

them aboard. They'll all lose their jobs eventually, along with

all the time they expended on this doomed company.Whereas if you only have a handful of people, it may be better to

focus on trying to make more money. It may seem facile to suggest

a startup make more money, as if that could be done for the asking.

Usually a startup is already trying as hard as it can to sell

whatever it sells. What I'm suggesting here is not so much to try

harder to make money but to try to make money in a different way.

For example, if you have only one person selling while the rest are

writing code, consider having everyone work on selling. What good

will more code do you when you're out of business?

If you have to write code to close a certain deal, go ahead;

that follows from everyone working on selling. But only work on

whatever will get you the most revenue the soonest.Another way to make money differently is to sell different things,

and in particular to do more consultingish work. I say consultingish

because there is a long slippery slope from making products to pure

consulting, and you don't have to go far down it before you start

to offer something really attractive to customers. Although your

product may not be very appealing yet, if you're a startup your

programmers will often be way better than the ones your customers

have. Or you may have expertise in some new field they

don't understand. So if you change your sales conversations

just a little from "do you want to buy our product?" to "what do

you need that you'd pay a lot for?" you may find it's suddenly a

lot easier to extract money from customers.Be ruthlessly mercenary when you start doing this, though. You're

trying to save your company from death here, so make customers pay

a lot, quickly. And to the extent you can, try to avoid the

worst pitfalls of consulting. The ideal thing might be if you built

a precisely defined derivative version of your product for the

customer, and it was otherwise a straight product sale. You keep

the IP and no billing by the hour.In the best case, this consultingish work may not be just something

you do to survive, but may turn out to be the thing-that-doesn't-scale that defines your

company. Don't expect it to be, but as you dive into individual

users' needs, keep your eyes open for narrow openings that have

wide vistas beyond.There is usually so much demand for custom work that unless you're

really incompetent there has to be some point down the slope of

consulting at which you can survive. But I didn't use the term

slippery slope by accident; customers' insatiable demand for custom

work will always be pushing you toward the bottom. So while you'll

probably survive, the problem now becomes to survive with the least

damage and distraction.The good news is, plenty of successful startups have passed through

near-death experiences and gone on to flourish. You just have to

realize in time that you're near death. And if you're in the fatal pinch,

you are.

Notes[1]

There are a handful of companies that can't reasonably expect

to make money for the first year or two, because what they're

building takes so long. For these companies substitute "progress"

for "revenue growth." You're not one of these companies unless

your initial investors agreed in advance that you were. And frankly

even these companies wish they weren't, because the illiquidity of

"progress" puts them at the mercy of investors.[2]

There's a variant of the fatal pinch where your existing

investors help you along by promising to invest more. Or rather,

where you read them as promising to invest more, while they think

they're just mentioning the possibility. The way to solve this

problem, if you have 8 months of runway or less, is to try to get

the money right now. Then you'll either get the money, in which

case (immediate) problem solved, or at least prevent your investors

from helping you to remain in denial about your fundraising prospects.[3]

Obviously, if you have significant expenses other than salaries

that you can eliminate, do it now.[4]

Unless of course the source of the problem is that you're paying

yourselves high salaries. If by cutting the founders' salaries to

the minimum you need, you can make it to profitability, you should.

But it's a bad sign if you needed to read this to realize that.

Thanks to Sam Altman, Paul Buchheit, Jessica Livingston, and

Geoff Ralston for reading drafts of this.Arabic Translation

Mean People Fail

November 2014It struck me recently how few of the most successful people I know

are mean. There are exceptions, but remarkably few.Meanness isn't rare. In fact, one of the things the internet has

shown us is how mean people can be. A few decades ago, only famous

people and professional writers got to publish their opinions. Now

everyone can, and we can all see the long tail of

meanness that had previously been hidden.And yet while there are clearly a lot of mean people out there,

there are next to none among the most successful people I know.

What's going on here? Are meanness and success inversely correlated?Part of what's going on, of course, is selection bias. I only know

people who work in certain fields: startup founders, programmers,

professors. I'm willing to believe that successful people in other

fields are mean. Maybe successful hedge fund managers are mean; I

don't know enough to say. It seems quite likely that most successful

drug lords are mean. But there are at least big chunks of the world

that mean people don't rule, and that territory seems to be growing.My wife and Y Combinator cofounder Jessica is one of those rare

people who have x-ray vision for character. Being married to her

is like standing next to an airport baggage scanner. She came to

the startup world from investment banking, and she has always been

struck both by how consistently successful startup founders turn

out to be good people, and how consistently bad people fail as

startup founders.Why? I think there are several reasons. One is that being mean

makes you stupid. That's why I hate fights. You never do your best

work in a fight, because fights are not sufficiently general.

Winning is always a function of the situation and the people involved.

You don't win fights by thinking of big ideas but by thinking of

tricks that work in one particular case. And yet fighting is just

as much work as thinking about real problems. Which is particularly

painful to someone who cares how their brain is used: your brain

goes fast but you get nowhere, like a car spinning its wheels.Startups don't win by attacking. They win by transcending. There

are exceptions of course, but usually the way to win is to race

ahead, not to stop and fight.Another reason mean founders lose is that they can't get the best

people to work for them. They can hire people who will put up with

them because they need a job. But the best people have other options.

A mean person can't convince the best people to work for him unless

he is super convincing. And while having the best people helps any

organization, it's critical for startups.There is also a complementary force at work: if you want to build

great things, it helps to be driven by a spirit of benevolence. The startup founders who end up

richest are not the ones driven by money. The ones driven by money

take the big acquisition offer that nearly every successful startup

gets en route.

[1]

The ones who keep going are driven by something

else. They may not say so explicitly, but they're usually trying

to improve the world. Which means people with a desire to improve

the world have a natural advantage.

[2]The exciting thing is that startups are not just one random type

of work in which meanness and success are inversely correlated.

This kind of work is the future.For most of history success meant control of scarce resources. One

got that by fighting, whether literally in the case of pastoral

nomads driving hunter-gatherers into marginal lands, or metaphorically

in the case of Gilded Age financiers contending with one another

to assemble railroad monopolies. For most of history, success meant

success at zero-sum games. And in most of them meanness was not a

handicap but probably an advantage.That is changing. Increasingly the games that matter are not zero-sum.

Increasingly you win not by fighting to get control of a scarce

resource, but by having new ideas and building new things.

[3]There have long been games where you won by having new ideas. In

the third century BC, Archimedes won by doing that. At least until

an invading Roman army killed him. Which illustrates why

this change is happening: for new ideas to matter, you need a certain

degree of civil order. And not just not being at war. You also

need to prevent the sort of economic violence that nineteenth century

magnates practiced against one another and communist countries

practiced against their citizens. People need to feel that what

they create can't be stolen.

[4]That has always been the case for thinkers, which is why this trend

began with them. When you think of successful people from history

who weren't ruthless, you get mathematicians and writers and artists.

The exciting thing is that their m.o. seems to be spreading. The

games played by intellectuals are leaking into the real world, and

this is reversing the historical polarity of the relationship between

meanness and success.So I'm really glad I stopped to think about this. Jessica and I

have always worked hard to teach our kids not to be mean. We

tolerate noise and mess and junk food, but not meanness. And now

I have both an additional reason to crack down on it, and an

additional argument to use when I do: that being mean makes you

fail.

Notes[1]

I'm not saying all founders who take big acquisition offers

are driven only by money, but rather that those who don't aren't.

Plus one can have benevolent motives for being driven by money — for

example, to take care of one's family, or to be free to work

on projects that improve the world.[2]

It's unlikely that every successful startup improves the

world. But their founders, like parents, truly believe they do.

Successful founders are in love with their companies. And while

this sort of love is as blind as the love people have for one

another, it is genuine.[3]

Peter

Thiel would point out that successful founders still

get rich from controlling monopolies, just monopolies they create

rather than ones they capture. And while this is largely true, it

means a big change in the sort of person who wins.[4]

To be fair, the Romans didn't mean to kill Archimedes. The

Roman commander specifically ordered that he be spared. But he got

killed in the chaos anyway.In sufficiently disordered times, even thinking requires

control of scarce resources, because living at all is a scarce

resource.Thanks to Sam Altman, Ron Conway, Daniel Gackle, Jessica Livingston, Robert Morris,

Geoff Ralston, and Fred Wilson for reading drafts of this.Portuguese TranslationJapanese TranslationArabic Translation

Before the Startup

Want to start a startup? Get funded by

Y Combinator.

October 2014(This essay is derived from a guest lecture in Sam Altman's startup class at

Stanford. It's intended for college students, but much of it is

applicable to potential founders at other ages.)One of the advantages of having kids is that when you have to give

advice, you can ask yourself "what would I tell my own kids?" My

kids are little, but I can imagine what I'd tell them about startups

if they were in college, and that's what I'm going to tell you.Startups are very counterintuitive. I'm not sure why. Maybe it's

just because knowledge about them hasn't permeated our culture yet.

But whatever the reason, starting a startup is a task where you

can't always trust your instincts.It's like skiing in that way. When you first try skiing and you

want to slow down, your instinct is to lean back. But if you lean

back on skis you fly down the hill out of control. So part of

learning to ski is learning to suppress that impulse. Eventually

you get new habits, but at first it takes a conscious effort. At

first there's a list of things you're trying to remember as you

start down the hill.Startups are as unnatural as skiing, so there's a similar list for

startups. Here I'm going to give you the first part of it — the things

to remember if you want to prepare yourself to start a startup.

CounterintuitiveThe first item on it is the fact I already mentioned: that startups

are so weird that if you trust your instincts, you'll make a lot

of mistakes. If you know nothing more than this, you may at least

pause before making them.When I was running Y Combinator I used to joke that our function

was to tell founders things they would ignore. It's really true.

Batch after batch, the YC partners warn founders about mistakes

they're about to make, and the founders ignore them, and then come

back a year later and say "I wish we'd listened."Why do the founders ignore the partners' advice? Well, that's the

thing about counterintuitive ideas: they contradict your intuitions.

They seem wrong. So of course your first impulse is to disregard

them. And in fact my joking description is not merely the curse

of Y Combinator but part of its raison d'etre. If founders' instincts

already gave them the right answers, they wouldn't need us. You

only need other people to give you advice that surprises you. That's

why there are a lot of ski instructors and not many running

instructors.

[1]You can, however, trust your instincts about people. And in fact

one of the most common mistakes young founders make is not to

do that enough. They get involved with people who seem impressive,

but about whom they feel some misgivings personally. Later when

things blow up they say "I knew there was something off about him,

but I ignored it because he seemed so impressive."If you're thinking about getting involved with someone — as a

cofounder, an employee, an investor, or an acquirer — and you

have misgivings about them, trust your gut. If someone seems

slippery, or bogus, or a jerk, don't ignore it.This is one case where it pays to be self-indulgent. Work with

people you genuinely like, and you've known long enough to be sure.

ExpertiseThe second counterintuitive point is that it's not that important

to know a lot about startups. The way to succeed in a startup is

not to be an expert on startups, but to be an expert on your users

and the problem you're solving for them.

Mark Zuckerberg didn't succeed because he was an expert on startups.

He succeeded despite being a complete noob at startups, because he

understood his users really well.If you don't know anything about, say, how to raise an angel round,

don't feel bad on that account. That sort of thing you can learn

when you need to, and forget after you've done it.In fact, I worry it's not merely unnecessary to learn in great

detail about the mechanics of startups, but possibly somewhat

dangerous. If I met an undergrad who knew all about convertible

notes and employee agreements and (God forbid) class FF stock, I

wouldn't think "here is someone who is way ahead of their peers."

It would set off alarms. Because another of the characteristic

mistakes of young founders is to go through the motions of starting

a startup. They make up some plausible-sounding idea, raise money

at a good valuation, rent a cool office, hire a bunch of people.

From the outside that seems like what startups do. But the next

step after rent a cool office and hire a bunch of people is: gradually

realize how completely fucked they are, because while imitating all

the outward forms of a startup they have neglected the one thing

that's actually essential: making something people want.

GameWe saw this happen so often that we made up a name for it: playing

house. Eventually I realized why it was happening. The reason

young founders go through the motions of starting a startup is

because that's what they've been trained to do for their whole lives

up to that point. Think about what you have to do to get into

college, for example. Extracurricular activities, check. Even in

college classes most of the work is as artificial as running laps.I'm not attacking the educational system for being this way. There

will always be a certain amount of fakeness in the work you do when

you're being taught something, and if you measure their performance

it's inevitable that people will exploit the difference to the point

where much of what you're measuring is artifacts of the fakeness.I confess I did it myself in college. I found that in a lot of

classes there might only be 20 or 30 ideas that were the right shape

to make good exam questions. The way I studied for exams in these

classes was not (except incidentally) to master the material taught

in the class, but to make a list of potential exam questions and

work out the answers in advance. When I walked into the final, the

main thing I'd be feeling was curiosity about which of my questions

would turn up on the exam. It was like a game.It's not surprising that after being trained for their whole lives

to play such games, young founders' first impulse on starting a

startup is to try to figure out the tricks for winning at this new

game. Since fundraising appears to be the measure of success for

startups (another classic noob mistake), they always want to know what the

tricks are for convincing investors. We tell them the best way to

convince investors is to make a startup

that's actually doing well, meaning growing fast, and then simply

tell investors so. Then they want to know what the tricks are for

growing fast. And we have to tell them the best way to do that is

simply to make something people want.So many of the conversations YC partners have with young founders

begin with the founder asking "How do we..." and the partner replying

"Just..."Why do the founders always make things so complicated? The reason,

I realized, is that they're looking for the trick.So this is the third counterintuitive thing to remember about

startups: starting a startup is where gaming the system stops

working. Gaming the system may continue to work if you go to work

for a big company. Depending on how broken the company is, you can

succeed by sucking up to the right people, giving the impression

of productivity, and so on.

[2]

But that doesn't work with startups.

There is no boss to trick, only users, and all users care about is

whether your product does what they want. Startups are as impersonal

as physics. You have to make something people want, and you prosper

only to the extent you do.The dangerous thing is, faking does work to some degree on investors.

If you're super good at sounding like you know what you're talking

about, you can fool investors for at least one and perhaps even two

rounds of funding. But it's not in your interest to. The company

is ultimately doomed. All you're doing is wasting your own time

riding it down.So stop looking for the trick. There are tricks in startups, as

there are in any domain, but they are an order of magnitude less

important than solving the real problem. A founder who knows nothing

about fundraising but has made something users love will have an

easier time raising money than one who knows every trick in the

book but has a flat usage graph. And more importantly, the founder

who has made something users love is the one who will go on to

succeed after raising the money.Though in a sense it's bad news in that you're deprived of one of

your most powerful weapons, I think it's exciting that gaming the

system stops working when you start a startup. It's exciting that

there even exist parts of the world where you win by doing good

work. Imagine how depressing the world would be if it were all

like school and big companies, where you either have to spend a lot

of time on bullshit things or lose to people who do.

[3]

I would

have been delighted if I'd realized in college that there were parts

of the real world where gaming the system mattered less than others,

and a few where it hardly mattered at all. But there are, and this

variation is one of the most important things to consider when

you're thinking about your future. How do you win in each type of

work, and what would you like to win by doing?

[4]

All-ConsumingThat brings us to our fourth counterintuitive point: startups are

all-consuming. If you start a startup, it will take over your life

to a degree you cannot imagine. And if your startup succeeds, it

will take over your life for a long time: for several years at the

very least, maybe for a decade, maybe for the rest of your working

life. So there is a real opportunity cost here.Larry Page may seem to have an enviable life, but there are aspects

of it that are unenviable. Basically at 25 he started running as

fast as he could and it must seem to him that he hasn't stopped to

catch his breath since. Every day new shit happens in the Google

empire that only the CEO can deal with, and he, as CEO, has to deal

with it. If he goes on vacation for even a week, a whole week's

backlog of shit accumulates. And he has to bear this uncomplainingly,

partly because as the company's daddy he can never show fear or

weakness, and partly because billionaires get less than zero sympathy

if they talk about having difficult lives. Which has the strange

side effect that the difficulty of being a successful startup founder

is concealed from almost everyone except those who've done it.Y Combinator has now funded several companies that can be called

big successes, and in every single case the founders say the same

thing. It never gets any easier. The nature of the problems change.

You're worrying about construction delays at your London office

instead of the broken air conditioner in your studio apartment.

But the total volume of worry never decreases; if anything it

increases.Starting a successful startup is similar to having kids in that

it's like a button you push that changes your life irrevocably.

And while it's truly wonderful having kids, there are a lot of

things that are easier to do before you have them than after. Many

of which will make you a better parent when you do have kids. And

since you can delay pushing the button for a while, most people in

rich countries do.Yet when it comes to startups, a lot of people seem to think they're

supposed to start them while they're still in college. Are you

crazy? And what are the universities thinking? They go out of

their way to ensure their students are well supplied with contraceptives,

and yet they're setting up entrepreneurship programs and startup

incubators left and right.To be fair, the universities have their hand forced here. A lot

of incoming students are interested in startups. Universities are,

at least de facto, expected to prepare them for their careers. So

students who want to start startups hope universities can teach

them about startups. And whether universities can do this or not,

there's some pressure to claim they can, lest they lose applicants

to other universities that do.Can universities teach students about startups? Yes and no. They

can teach students about startups, but as I explained before, this

is not what you need to know. What you need to learn about are the

needs of your own users, and you can't do that until you actually

start the company.

[5]

So starting a startup is intrinsically

something you can only really learn by doing it. And it's impossible

to do that in college, for the reason I just explained: startups

take over your life. You can't start a startup for real as a

student, because if you start a startup for real you're not a student

anymore. You may be nominally a student for a bit, but you won't even

be that for long.

[6]Given this dichotomy, which of the two paths should you take? Be

a real student and not start a startup, or start a real startup and

not be a student? I can answer that one for you. Do not start a

startup in college. How to start a startup is just a subset of a

bigger problem you're trying to solve: how to have a good life.

And though starting a startup can be part of a good life for a lot

of ambitious people, age 20 is not the optimal time to do it.

Starting a startup is like a brutally fast depth-first search. Most

people should still be searching breadth-first at 20.You can do things in your early 20s that you can't do as well before

or after, like plunge deeply into projects on a whim and travel

super cheaply with no sense of a deadline. For unambitious people,

this sort of thing is the dreaded "failure to launch," but for the

ambitious ones it can be an incomparably valuable sort of exploration.

If you start a startup at 20 and you're sufficiently successful,

you'll never get to do it.

[7]Mark Zuckerberg will never get to bum around a foreign country. He

can do other things most people can't, like charter jets to fly him

to foreign countries. But success has taken a lot of the serendipity

out of his life. Facebook is running him as much as he's running

Facebook. And while it can be very cool to be in the grip of a

project you consider your life's work, there are advantages to

serendipity too, especially early in life. Among other things it

gives you more options to choose your life's work from.There's not even a tradeoff here. You're not sacrificing anything

if you forgo starting a startup at 20, because you're more likely

to succeed if you wait. In the unlikely case that you're 20 and

one of your side projects takes off like Facebook did, you'll face

a choice of running with it or not, and it may be reasonable to run

with it. But the usual way startups take off is for the founders

to make them take off, and it's gratuitously

stupid to do that at 20.

TryShould you do it at any age? I realize I've made startups sound

pretty hard. If I haven't, let me try again: starting a startup

is really hard. What if it's too hard? How can you tell if you're

up to this challenge?The answer is the fifth counterintuitive point: you can't tell. Your

life so far may have given you some idea what your prospects might

be if you tried to become a mathematician, or a professional football

player. But unless you've had a very strange life you haven't done

much that was like being a startup founder.

Starting a startup will change you a lot. So what you're trying

to estimate is not just what you are, but what you could grow into,

and who can do that?For the past 9 years it was my job to predict whether people would

have what it took to start successful startups. It was easy to

tell how smart they were, and most people reading this will be over

that threshold. The hard part was predicting how tough and ambitious they would become. There

may be no one who has more experience at trying to predict that,

so I can tell you how much an expert can know about it, and the

answer is: not much. I learned to keep a completely open mind about

which of the startups in each batch would turn out to be the stars.The founders sometimes think they know. Some arrive feeling sure

they will ace Y Combinator just as they've aced every one of the (few,

artificial, easy) tests they've faced in life so far. Others arrive

wondering how they got in, and hoping YC doesn't discover whatever

mistake caused it to accept them. But there is little correlation

between founders' initial attitudes and how well their companies

do.I've read that the same is true in the military — that the

swaggering recruits are no more likely to turn out to be really

tough than the quiet ones. And probably for the same reason: that

the tests involved are so different from the ones in their previous

lives.If you're absolutely terrified of starting a startup, you probably

shouldn't do it. But if you're merely unsure whether you're up to

it, the only way to find out is to try. Just not now.

IdeasSo if you want to start a startup one day, what should you do in

college? There are only two things you need initially: an idea and

cofounders. And the m.o. for getting both is the same. Which leads

to our sixth and last counterintuitive point: that the way to get

startup ideas is not to try to think of startup ideas.I've written a whole essay on this,

so I won't repeat it all here. But the short version is that if

you make a conscious effort to think of startup ideas, the ideas

you come up with will not merely be bad, but bad and plausible-sounding,

meaning you'll waste a lot of time on them before realizing they're

bad.The way to come up with good startup ideas is to take a step back.

Instead of making a conscious effort to think of startup ideas,

turn your mind into the type that startup ideas form in without any

conscious effort. In fact, so unconsciously that you don't even

realize at first that they're startup ideas.This is not only possible, it's how Apple, Yahoo, Google, and

Facebook all got started. None of these companies were even meant

to be companies at first. They were all just side projects. The

best startups almost have to start as side projects, because great

ideas tend to be such outliers that your conscious mind would reject

them as ideas for companies.Ok, so how do you turn your mind into the type that startup ideas

form in unconsciously? (1) Learn a lot about things that matter,

then (2) work on problems that interest you (3) with people you

like and respect. The third part, incidentally, is how you get

cofounders at the same time as the idea.The first time I wrote that paragraph, instead of "learn a lot about

things that matter," I wrote "become good at some technology." But

that prescription, though sufficient, is too narrow. What was

special about Brian Chesky and Joe Gebbia was not that they were

experts in technology. They were good at design, and perhaps even

more importantly, they were good at organizing groups and making

projects happen. So you don't have to work on technology per se,

so long as you work on problems demanding enough to stretch you.What kind of problems are those? That is very hard to answer in

the general case. History is full of examples of young people who

were working on important problems that no

one else at the time thought were important, and in particular

that their parents didn't think were important. On the other hand,

history is even fuller of examples of parents who thought their

kids were wasting their time and who were right. So how do you

know when you're working on real stuff?

[8]I know how I know. Real problems are interesting, and I am

self-indulgent in the sense that I always want to work on interesting

things, even if no one else cares about them (in fact, especially

if no one else cares about them), and find it very hard to make

myself work on boring things, even if they're supposed to be

important.My life is full of case after case where I worked on something just

because it seemed interesting, and it turned out later to be useful

in some worldly way. Y

Combinator itself was something I only did because it seemed

interesting. So I seem to have some sort of internal compass that

helps me out. But I don't know what other people have in their

heads. Maybe if I think more about this I can come up with heuristics

for recognizing genuinely interesting problems, but for the moment

the best I can offer is the hopelessly question-begging advice that

if you have a taste for genuinely interesting problems, indulging

it energetically is the best way to prepare yourself for a startup.

And indeed, probably also the best way to live.

[9]But although I can't explain in the general case what counts as an

interesting problem, I can tell you about a large subset of them.

If you think of technology as something that's spreading like a

sort of fractal stain, every moving point on the edge represents

an interesting problem. So one guaranteed way to turn your mind

into the type that has good startup ideas is to get yourself to the

leading edge of some technology — to cause yourself, as Paul

Buchheit put it, to "live in the future." When you reach that point,

ideas that will seem to other people uncannily prescient will seem

obvious to you. You may not realize they're startup ideas, but

you'll know they're something that ought to exist.For example, back at Harvard in the mid 90s a fellow grad student

of my friends Robert and Trevor wrote his own voice over IP software.

He didn't mean it to be a startup, and he never tried to turn it

into one. He just wanted to talk to his girlfriend in Taiwan without

paying for long distance calls, and since he was an expert on

networks it seemed obvious to him that the way to do it was turn

the sound into packets and ship it over the Internet. He never did

any more with his software than talk to his girlfriend, but this

is exactly the way the best startups get started.So strangely enough the optimal thing to do in college if you want

to be a successful startup founder is not some sort of new, vocational

version of college focused on "entrepreneurship." It's the classic

version of college as education for its own sake. If you want to

start a startup after college, what you should do in college is

learn powerful things. And if you have genuine intellectual

curiosity, that's what you'll naturally tend to do if you just

follow your own inclinations.

[10]The component of entrepreneurship that really matters is domain

expertise. The way to become Larry Page was to become an expert

on search. And the way to become an expert on search was to be

driven by genuine curiosity, not some ulterior motive.At its best, starting a startup is merely an ulterior motive for

curiosity. And you'll do it best if you introduce the ulterior

motive toward the end of the process.So here is the ultimate advice for young would-be startup founders,

boiled down to two words: just learn.

Notes[1]

Some founders listen more than others, and this tends to be a

predictor of success. One of the things I

remember about the Airbnbs during YC is how intently they listened.[2]

In fact, this is one of the reasons startups are possible. If

big companies weren't plagued by internal inefficiencies, they'd

be proportionately more effective, leaving less room for startups.[3]

In a startup you have to spend a lot of time on schleps, but this sort of work is merely

unglamorous, not bogus.[4]

What should you do if your true calling is gaming the system?

Management consulting.[5]

The company may not be incorporated, but if you start to get

significant numbers of users, you've started it, whether you realize

it yet or not.[6]

It shouldn't be that surprising that colleges can't teach

students how to be good startup founders, because they can't teach

them how to be good employees either.The way universities "teach" students how to be employees is to

hand off the task to companies via internship programs. But you

couldn't do the equivalent thing for startups, because by definition

if the students did well they would never come back.[7]

Charles Darwin was 22 when he received an invitation to travel

aboard the HMS Beagle as a naturalist. It was only because he was

otherwise unoccupied, to a degree that alarmed his family, that he

could accept it. And yet if he hadn't we probably would not know

his name.[8]

Parents can sometimes be especially conservative in this

department. There are some whose definition of important problems

includes only those on the critical path to med school.[9]

I did manage to think of a heuristic for detecting whether you

have a taste for interesting ideas: whether you find known boring

ideas intolerable. Could you endure studying literary theory, or

working in middle management at a large company?[10]

In fact, if your goal is to start a startup, you can stick

even more closely to the ideal of a liberal education than past

generations have. Back when students focused mainly on getting a

job after college, they thought at least a little about how the

courses they took might look to an employer. And perhaps even

worse, they might shy away from taking a difficult class lest they

get a low grade, which would harm their all-important GPA. Good

news: users don't care what your GPA

was. And I've never heard of investors caring either. Y Combinator

certainly never asks what classes you took in college or what grades

you got in them.

Thanks to Sam Altman, Paul Buchheit, John Collison, Patrick

Collison, Jessica Livingston, Robert Morris, Geoff Ralston, and

Fred Wilson for reading drafts of this.

How to Raise Money

Want to start a startup? Get funded by

Y Combinator.

September 2013Most startups that raise money do it more than once. A typical

trajectory might be (1) to get started with a few tens of thousands

from something like Y Combinator or individual angels, then

(2) raise a few hundred thousand to a few million to build the company,

and then (3) once the company is clearly succeeding, raise one or

more later rounds to accelerate growth.Reality can be messier. Some companies raise money twice in phase

2. Others skip phase 1 and go straight to phase 2. And at Y Combinator

we get an increasing number of companies that have already

raised amounts in the hundreds of thousands. But the three phase

path is at least the one about which individual startups' paths

oscillate.This essay focuses on phase 2 fundraising. That's the type the

startups we fund are doing on Demo Day, and this essay is the advice

we give them.

ForcesFundraising is hard in both senses: hard like lifting a heavy weight,

and hard like solving a puzzle. It's hard like lifting a weight

because it's intrinsically hard to convince people to part with

large sums of money. That problem is irreducible; it should be

hard. But much of the other kind of difficulty can be eliminated.

Fundraising only seems a puzzle because it's an alien world to most

founders, and I hope to fix that by supplying a map through it.To founders, the behavior of investors is often opaque — partly

because their motivations are obscure, but partly because they

deliberately mislead you. And the misleading ways of investors

combine horribly with the wishful thinking of inexperienced founders.

At YC we're always warning founders about this danger, and investors

are probably more circumspect with YC startups than with other

companies they talk to, and even so we witness a constant series

of explosions as these two volatile components combine.

[1]If you're an inexperienced founder, the only way to survive is by

imposing external constraints on yourself. You can't trust your

intuitions. I'm going to give you a set of rules here that will

get you through this process if anything will. At certain moments

you'll be tempted to ignore them. So rule number zero is: these

rules exist for a reason. You wouldn't need a rule to keep you

going in one direction if there weren't powerful forces pushing you

in another.The ultimate source of the forces acting on you are the forces

acting on investors. Investors are pinched between two kinds of

fear: fear of investing in startups that fizzle, and fear of missing

out on startups that take off. The cause of all this fear is the

very thing that makes startups such attractive investments: the

successful ones grow very fast. But that fast growth means investors

can't wait around. If you wait till a startup is obviously a

success, it's too late. To get the really high returns, you have

to invest in startups when it's still unclear how they'll do. But

that in turn makes investors nervous they're about to invest in a

flop. As indeed they often are.What investors would like to do, if they could, is wait. When a

startup is only a few months old, every week that passes gives you

significantly more information about them. But if you wait too

long, other investors might take the deal away from you. And of

course the other investors are all subject to the same forces. So

what tends to happen is that they all wait as long as they can,

then when some act the rest have to.

Don't raise money unless you want it and it wants you.Such a high proportion of successful startups raise money that it

might seem fundraising is one of the defining qualities of a startup.

Actually it isn't. Rapid growth is what

makes a company a startup. Most companies in a position to grow

rapidly find that (a) taking outside money helps them grow faster,

and (b) their growth potential makes it easy to attract such money.

It's so common for both (a) and (b) to be true of a successful

startup that practically all do raise outside money. But there may

be cases where a startup either wouldn't want to grow faster, or

outside money wouldn't help them to, and if you're one of them,

don't raise money.The other time not to raise money is when you won't be able to. If

you try to raise money before you can convince

investors, you'll not only waste your time, but also burn your

reputation with those investors.

Be in fundraising mode or not.One of the things that surprises founders most about fundraising

is how distracting it is. When you start fundraising, everything

else grinds to a halt. The problem is not the time fundraising

consumes but that it becomes the top idea in

your mind. A startup can't endure that level of distraction

for long. An early stage startup grows mostly because the founders

make it grow, and if the founders look away,

growth usually drops sharply.Because fundraising is so distracting, a startup should either be

in fundraising mode or not. And when you do decide to raise money,

you should focus your whole attention on it so you can get it done

quickly and get back to work.

[2]You can take money from investors when you're not in fundraising

mode. You just can't expend any attention on it. There are two

things that take attention: convincing investors, and negotiating

with them. So when you're not in fundraising mode, you should take

money from investors only if they require no convincing, and are

willing to invest on terms you'll take without negotiation. For

example, if a reputable investor is willing to invest on a convertible

note, using standard paperwork, that is either uncapped or capped

at a good valuation, you can take that without having to think.

[3]

The terms will be whatever they turn out to be in your next

equity round. And "no convincing" means just that: zero time spent

meeting with investors or preparing materials for them. If an

investor says they're ready to invest, but they need you to come

in for one meeting to meet some of the partners, tell them no, if

you're not in fundraising mode, because that's fundraising.

[4]

Tell them politely; tell them you're focusing on the company right

now, and that you'll get back to them when you're fundraising; but

do not get sucked down the slippery slope.Investors will try to lure you into fundraising when you're not.

It's great for them if they can, because they can thereby get a

shot at you before everyone else. They'll send you emails saying

they want to meet to learn more about you. If you get cold-emailed

by an associate at a VC firm, you shouldn't meet even if you are

in fundraising mode. Deals don't happen that way.

[5]

But even

if you get an email from a partner you should try to delay meeting

till you're in fundraising mode. They may say they just want to

meet and chat, but investors never just want to meet and chat. What

if they like you? What if they start to talk about giving you

money? Will you be able to resist having that conversation? Unless

you're experienced enough at fundraising to have a casual conversation

with investors that stays casual, it's safer to tell them that you'd

be happy to later, when you're fundraising, but that right now you

need to focus on the company.

[6]Companies that are successful at raising money in phase 2 sometimes

tack on a few investors after leaving fundraising mode. This is

fine; if fundraising went well, you'll be able to do it without

spending time convincing them or negotiating about terms.

Get introductions to investors.Before you can talk to investors, you have to be introduced to them.

If you're presenting at a Demo Day, you'll be introduced to a whole

bunch simultaneously. But even if you are, you should supplement

these with intros you collect yourself.Do you have to be introduced? In phase 2, yes. Some investors

will let you email them a business plan, but you can tell from the

way their sites are organized that they don't really want startups

to approach them directly.Intros vary greatly in effectiveness. The best type of intro is

from a well-known investor who has just invested in you. So when

you get an investor to commit, ask them to introduce you to other

investors they respect.

[7]

The next best type of intro is from a

founder of a company they've funded. You can also get intros from

other people in the startup community, like lawyers and reporters.There are now sites like AngelList, FundersClub, and WeFunder that

can introduce you to investors. We recommend startups treat them

as auxiliary sources of money. Raise money first from leads you

get yourself. Those will on average be better investors. Plus

you'll have an easier time raising money on these sites once you

can say you've already raised some from well-known investors.

Hear no till you hear yes.Treat investors as saying no till they unequivocally say yes, in

the form of a definite offer with no contingencies.I mentioned earlier that investors prefer to wait if they can.

What's particularly dangerous for founders is the way they wait.

Essentially, they lead you on. They seem like they're about to

invest right up till the moment they say no. If they even say no.

Some of the worse ones never actually do say no; they just stop

replying to your emails. They hope that way to get a free option

on investing. If they decide later that they want to invest — usually

because they've heard you're a hot deal — they can pretend they

just got distracted and then restart the conversation as if they'd

been about to.

[8]That's not the worst thing investors will do. Some will use language

that makes it sound as if they're committing, but which doesn't

actually commit them. And wishful thinking founders are happy to

meet them half way.

[9]Fortunately, the next rule is a tactic for neutralizing this behavior.

But to work it depends on you not being tricked by the no that

sounds like yes. It's so common for founders to be misled/mistaken

about this that we designed a protocol to fix the

problem. If you believe an investor has committed, get them to

confirm it. If you and they have different views of reality, whether

the source of the discrepancy is their sketchiness or your wishful

thinking, the prospect of confirming a commitment in writing will

flush it out. And till they confirm, regard them as saying no.

Do breadth-first search weighted by expected value.When you talk to investors your m.o. should be breadth-first search,

weighted by expected value. You should always talk to investors

in parallel rather than serially. You can't afford the time it

takes to talk to investors serially, plus if you only talk to one

investor at a time, they don't have the pressure of other investors

to make them act. But you shouldn't pay the same attention to every

investor, because some are more promising prospects than others.

The optimal solution is to talk to all potential investors in

parallel, but give higher priority to the more promising ones.

[10]Expected value = how likely an investor is to say yes, multiplied

by how good it would be if they did. So for example, an eminent

investor who would invest a lot, but will be hard to convince, might

have the same expected value as an obscure angel who won't invest

much, but will be easy to convince. Whereas an obscure angel who

will only invest a small amount, and yet needs to meet multiple

times before making up his mind, has very low expected value. Meet

such investors last, if at all.

[11]Doing breadth-first search weighted by expected value will save you

from investors who never explicitly say no but merely drift away,

because you'll drift away from them at the same rate. It protects

you from investors who flake in much the same way that a distributed

algorithm protects you from processors that fail. If some investor

isn't returning your emails, or wants to have lots of meetings but

isn't progressing toward making you an offer, you automatically

focus less on them. But you have to be disciplined about assigning

probabilities. You can't let how much you want an investor influence

your estimate of how much they want you.

Know where you stand.How do you judge how well you're doing with an investor, when

investors habitually seem more positive than they are? By looking

at their actions rather than their words. Every investor has some

track they need to move along from the first conversation to wiring

the money, and you should always know what that track consists of,

where you are on it, and how fast you're moving forward.Never leave a meeting with an investor without asking what happens

next. What more do they need in order to decide? Do they need

another meeting with you? To talk about what? And how soon? Do

they need to do something internally, like talk to their partners,

or investigate some issue? How long do they expect it to take?

Don't be too pushy, but know where you stand. If investors are

vague or resist answering such questions, assume the worst; investors

who are seriously interested in you will usually be happy to talk

about what has to happen between now and wiring the money, because

they're already running through that in their heads.

[12]If you're experienced at negotiations, you already know how to ask

such questions.

[13]

If you're not, there's a trick you can use

in this situation. Investors know you're inexperienced at raising

money. Inexperience there doesn't make you unattractive. Being a

noob at technology would, if you're starting a technology startup,

but not being a noob at fundraising. Larry and Sergey were noobs

at fundraising. So you can just confess that you're inexperienced

at this and ask how their process works and where you are in it.

[14]

Get the first commitment.The biggest factor in most investors' opinions of you is the opinion

of other investors. Once you start getting

investors to commit, it becomes increasingly easy to get more to.

But the other side of this coin is that it's often hard to get the

first commitment.Getting the first substantial offer can be half the total difficulty

of fundraising. What counts as a substantial offer depends on who

it's from and how much it is. Money from friends and family doesn't

usually count, no matter how much. But if you get $50k from a well

known VC firm or angel investor, that will usually be enough to set

things rolling.

[15]

Close committed money.It's not a deal till the money's in the bank. I often hear

inexperienced founders say things like "We've raised $800,000,"

only to discover that zero of it is in the bank so far. Remember

the twin fears that torment investors? The fear of missing out

that makes them jump early, and the fear of jumping onto a turd

that results? This is a market where people are exceptionally prone

to buyer's remorse. And it's also one that furnishes them plenty

of excuses to gratify it. The public markets snap startup investing

around like a whip. If the Chinese economy blows up tomorrow, all

bets are off. But there are lots of surprises for individual

startups too, and they tend to be concentrated around fundraising.

Tomorrow a big competitor could appear, or you could get C&Ded, or

your cofounder could quit.

[16]Even a day's delay can bring news that causes an investor to change

their mind. So when someone commits, get the money. Knowing where

you stand doesn't end when they say they'll invest. After they say

yes, know what the timetable is for getting the money, and then

babysit that process till it happens. Institutional investors have

people in charge of wiring money, but you may have to hunt angels

down in person to collect a check.Inexperienced investors are the ones most likely to get buyer's

remorse. Established ones have learned to treat saying yes as like

diving off a diving board, and they also have more brand to preserve.

But I've heard of cases of even top-tier VC firms welching on deals.

Avoid investors who don't "lead."Since getting the first offer is most of the difficulty of fundraising,

that should be part of your calculation of expected value when you

start. You have to estimate not just the probability that an

investor will say yes, but the probability that they'd be the first

to say yes, and the latter is not simply a constant fraction of the

former. Some investors are known for deciding quickly, and those

are extra valuable early on.Conversely, an investor who will only invest once other investors

have is worthless initially. And while most investors are influenced

by how interested other investors are in you, there are some who

have an explicit policy of only investing after other investors

have. You can recognize this contemptible subspecies of investor

because they often talk about "leads." They say that they don't

lead, or that they'll invest once you have a lead. Sometimes they

even claim to be willing to lead themselves, by which they mean

they won't invest till you get $x from other investors. (It's great

if by "lead" they mean they'll invest unilaterally, and in addition

will help you raise more. What's lame is when they use the term

to mean they won't invest unless you can raise more elsewhere.)

[17]Where does this term "lead" come from? Up till a few years ago,

startups raising money in phase 2 would usually raise equity rounds

in which several investors invested at the same time using the same

paperwork. You'd negotiate the terms with one "lead" investor, and

then all the others would sign the same documents and all the money

change hands at the closing.Series A rounds still work that way, but things now work differently

for most fundraising prior to the series A. Now there are rarely

actual rounds before the A round, or leads for them. Now startups

simply raise money from investors one at a time till they feel they

have enough.Since there are no longer leads, why do investors use that term?

Because it's a more legitimate-sounding way of saying what they

really mean. All they really mean is that their interest in you

is a function of other investors' interest in you. I.e. the spectral

signature of all mediocre investors. But when phrased in terms of

leads, it sounds like there is something structural and therefore

legitimate about their behavior.When an investor tells you "I want to invest in you, but I don't

lead," translate that in your mind to "No, except yes if you turn

out to be a hot deal." And since that's the default opinion of any

investor about any startup, they've essentially just told you

nothing.When you first start fundraising, the expected value of an investor

who won't "lead" is zero, so talk to such investors last if at all.

Have multiple plans.Many investors will ask how much you're planning to raise. This

question makes founders feel they should be planning to raise a

specific amount. But in fact you shouldn't. It's a mistake to

have fixed plans in an undertaking as unpredictable as fundraising.So why do investors ask how much you plan to raise? For much the

same reasons a salesperson in a store will ask "How much were you

planning to spend?" if you walk in looking for a gift for a friend.

You probably didn't have a precise amount in mind; you just want

to find something good, and if it's inexpensive, so much the better.

The salesperson asks you this not because you're supposed to have

a plan to spend a specific amount, but so they can show you only

things that cost the most you'll pay.Similarly, when investors ask how much you plan to raise, it's not

because you're supposed to have a plan. It's to see whether you'd

be a suitable recipient for the size of investment they like to

make, and also to judge your ambition, reasonableness, and how far

you are along with fundraising.If you're a wizard at fundraising, you can say "We plan to raise

a $7 million series A round, and we'll be accepting termsheets next

tuesday." I've known a handful of founders who could pull that off

without having VCs laugh in their faces. But if you're in the

inexperienced but earnest majority, the solution is analogous to

the solution I recommend for pitching

your startup: do the right thing and then just tell investors what

you're doing.And the right strategy, in fundraising, is to have multiple plans

depending on how much you can raise. Ideally you should be able

to tell investors something like: we can make it to profitability

without raising any more money, but if we raise a few hundred

thousand we can hire one or two smart friends, and if we raise a

couple million, we can hire a whole engineering team, etc.Different plans match different investors. If you're talking to a

VC firm that only does series A rounds (though there are few of

those left), it would be a waste of time talking about any but your

most expensive plan. Whereas if you're talking to an angel who

invests $20k at a time and you haven't raised any money yet, you

probably want to focus on your least expensive plan.If you're so fortunate as to have to think about the upper limit

on what you should raise, a good rule of thumb is to multiply the

number of people you want to hire times $15k times 18 months. In

most startups, nearly all the costs are a function of the number

of people, and $15k per month is the conventional total cost

(including benefits and even office space) per person. $15k per

month is high, so don't actually spend that much. But it's ok to

use a high estimate when fundraising to add a margin for error. If

you have additional expenses, like manufacturing, add in those at

the end. Assuming you have none and you think you might hire 20

people, the most you'd want to raise is 20 x $15k x 18 = $5.4

million.

[18]

Underestimate how much you want.Though you can focus on different plans when talking to different

types of investors, you should on the whole err on the side of

underestimating the amount you hope to raise.For example, if you'd like to raise $500k, it's better to say

initially that you're trying to raise $250k. Then when you reach

$150k you're more than half done. That sends two useful signals

to investors: that you're doing well, and that they have to decide

quickly because you're running out of room. Whereas if you'd said

you were raising $500k, you'd be less than a third done at $150k.

If fundraising stalled there for an appreciable time, you'd start

to read as a failure.Saying initially that you're raising $250k doesn't limit you to

raising that much. When you reach your initial target and you still

have investor interest, you can just decide to raise more. Startups

do that all the time. In fact, most startups that are very successful

at fundraising end up raising more than they originally intended.I'm not saying you should lie, but that you should lower your

expectations initially. There is almost no downside in starting

with a low number. It not only won't cap the amount you raise, but

will on the whole tend to increase it.A good metaphor here is angle of attack. If you try to fly at too

steep an angle of attack, you just stall. If you say right out of

the gate that you want to raise a $5 million series A round, unless

you're in a very strong position, you not only won't get that but

won't get anything. Better to start at a low angle of attack, build

up speed, and then gradually increase the angle if you want.

Be profitable if you can.You will be in a much stronger position if your collection of plans

includes one for raising zero dollars — i.e. if you can make

it to profitability without raising any additional money. Ideally

you want to be able to say to investors "We'll succeed no matter

what, but raising money will help us do it faster."There are many analogies between fundraising and dating, and this

is one of the strongest. No one wants you if you seem desperate.

And the best way not to seem desperate is not to be desperate.

That's one reason we urge startups during YC to keep expenses low

and to try to make it to ramen

profitability before Demo Day. Though it sounds slightly

paradoxical, if you want to raise money, the best thing you can do

is get yourself to the point where you don't need to.There are almost two distinct modes of fundraising: one in which

founders who need money knock on doors seeking it, knowing that

otherwise the company will die or at the very least people will

have to be fired, and one in which founders who don't need money

take some to grow faster than they could merely on their own revenues.

To emphasize the distinction I'm going to name them: type A fundraising

is when you don't need money, and type B fundraising is when you

do.Inexperienced founders read about famous startups doing what was

type A fundraising, and decide they should raise money too, since

that seems to be how startups work. Except when they raise money

they don't have a clear path to profitability and are thus doing

type B fundraising. And they are then surprised how difficult and

unpleasant it is.Of course not all startups can make it to ramen profitability in a

few months. And some that don't still manage to have the upper

hand over investors, if they have some other advantage like

extraordinary growth numbers or exceptionally formidable founders.

But as time passes it gets increasingly difficult to fundraise from

a position of strength without being profitable.

[19]

Don't optimize for valuation.When you raise money, what should your valuation be? The most

important thing to understand about valuation is that it's not that

important.Founders who raise money at high valuations tend to be unduly proud

of it. Founders are often competitive people, and since valuation

is usually the only visible number attached to a startup, they end

up competing to raise money at the highest valuation. This is

stupid, because fundraising is not the test that matters. The real

test is revenue. Fundraising is just a means to that end. Being

proud of how well you did at fundraising is like being proud of

your college grades.Not only is fundraising not the test that matters, valuation is not

even the thing to optimize about fundraising. The number one thing

you want from phase 2 fundraising is to get the money you need, so

you can get back to focusing on the real test, the success of your

company. Number two is good investors. Valuation is at best third.The empirical evidence shows just how unimportant it is. Dropbox

and Airbnb are the most successful companies we've funded so far,

and they raised money after Y Combinator at premoney valuations of

$4 million and $2.6 million respectively. Prices are so much higher

now that if you can raise money at all you'll probably raise it at

higher valuations than Dropbox and Airbnb. So let that satisfy

your competitiveness. You're doing better than Dropbox and Airbnb!

At a test that doesn't matter.When you start fundraising, your initial valuation (or valuation

cap) will be set by the deal you make with the first investor who

commits. You can increase the price for later investors, if you

get a lot of interest, but by default the valuation you got from

the first investor becomes your asking price.So if you're raising money from multiple investors, as most companies

do in phase 2, you have to be careful to avoid raising the first

from an over-eager investor at a price you won't be able to

sustain. You can of course lower your price if you need to (in

which case you should give the same terms to investors who invested

earlier at a higher price), but you may lose a bunch of leads in

the process of realizing you need to do this.What you can do if you have eager first investors is raise money

from them on an uncapped convertible note with an MFN clause. This

is essentially a way of saying that the valuation cap of the note

will be determined by the next investors you raise money from.It will be easier to raise money at a lower valuation. It shouldn't

be, but it is. Since phase 2 prices vary at most 10x and the big

successes generate returns of at least 100x, investors should pick

startups entirely based on their estimate of the probability that

the company will be a big success and hardly at all on price. But

although it's a mistake for investors to care about price, a

significant number do. A startup that investors seem to like but

won't invest in at a cap of $x will have an easier time at $x/2.

[20]

Yes/no before valuation.Some investors want to know what your valuation is before they even

talk to you about investing. If your valuation has already been

set by a prior investment at a specific valuation or cap, you can

tell them that number. But if it isn't set because you haven't

closed anyone yet, and they try to push you to name a price, resist

doing so. If this would be the first investor you've closed, then

this could be the tipping point of fundraising. That means closing

this investor is the first priority, and you need to get the

conversation onto that instead of being dragged sideways into a

discussion of price.Fortunately there is a way to avoid naming a price in this situation.

And it is not just a negotiating trick; it's how you (both) should

be operating. Tell them that valuation is not the most important

thing to you and that you haven't thought much about it, that you

are looking for investors you want to partner with and who want to

partner with you, and that you should talk first about whether they

want to invest at all. Then if they decide they do want to invest,

you can figure out a price. But first things first.Since valuation isn't that important and getting fundraising rolling

is, we usually tell founders to give the first investor who commits

as low a price as they need to. This is a safe technique so long

as you combine it with the next one.

[21]

Beware "valuation sensitive" investors.Occasionally you'll encounter investors who describe themselves as

"valuation sensitive." What this means in practice is that they

are compulsive negotiators who will suck up a lot of your time

trying to push your price down. You should therefore never approach

such investors first. While you shouldn't chase high valuations,

you also don't want your valuation to be set artificially low because

the first investor who committed happened to be a compulsive

negotiator. Some such investors have value, but the time to approach

them is near the end of fundraising, when you're in a position to

say "this is the price everyone else has paid; take it or leave it"

and not mind if they leave it. This way, you'll not only get market

price, but it will also take less time.Ideally you know which investors have a reputation for being

"valuation sensitive" and can postpone dealing with them till last,

but occasionally one you didn't know about will pop up early on.

The rule of doing breadth first search weighted by expected value

already tells you what to do in this case: slow down your interactions

with them.There are a handful of investors who will try to invest at a lower

valuation even when your price has already been set. Lowering your

price is a backup plan you resort to when you discover you've let

the price get set too high to close all the money you need. So

you'd only want to talk to this sort of investor if you were about

to do that anyway. But since investor meetings have to be arranged

at least a few days in advance and you can't predict when you'll

need to resort to lowering your price, this means in practice that

you should approach this type of investor last if at all.If you're surprised by a lowball offer, treat it as a backup offer

and delay responding to it. When someone makes an offer in good

faith, you have a moral obligation to respond in a reasonable time.

But lowballing you is a dick move that should be met with the

corresponding countermove.

Accept offers greedily.I'm a little leery of using the term "greedily" when writing about

fundraising lest non-programmers misunderstand me, but a greedy

algorithm is simply one that doesn't try to look into the future.

A greedy algorithm takes the best of the options in front of it

right now. And that is how startups should approach fundraising

in phases 2 and later. Don't try to look into the future because

(a) the future is unpredictable, and indeed in this business you're

often being deliberately misled about it and (b) your first priority

in fundraising should be to get it finished and get back to work

anyway.If someone makes you an acceptable offer, take it. If you have

multiple incompatible offers, take the best. Don't reject an

acceptable offer in the hope of getting a better one in the future.These simple rules cover a wide variety of cases. If you're raising

money from many investors, roll them up as they say yes. As you

start to feel you've raised enough, the threshold for acceptable

will start to get higher.In practice offers exist for stretches of time, not points. So

when you get an acceptable offer that would be incompatible with

others (e.g. an offer to invest most of the money you need), you

can tell the other investors you're talking to that you have an

offer good enough to accept, and give them a few days to make their

own. This could lose you some that might have made an offer if

they had more time. But by definition you don't care; the initial

offer was acceptable.Some investors will try to prevent others from having time to decide

by giving you an "exploding" offer, meaning one that's only valid

for a few days. Offers from the very best investors explode less

frequently and less rapidly — Fred Wilson never gives exploding

offers, for example — because they're confident you'll pick

them. But lower-tier investors sometimes give offers with very

short fuses, because they believe no one who had other options would

choose them. A deadline of three working days is acceptable. You

shouldn't need more than that if you've been talking to investors

in parallel. But a deadline any shorter is a sign you're dealing

with a sketchy investor. You can usually call their bluff, and you

may need to.

[22]It might seem that instead of accepting offers greedily, your goal

should be to get the best investors as partners. That is certainly

a good goal, but in phase 2 "get the best investors" only rarely

conflicts with "accept offers greedily," because the best investors

don't usually take any longer to decide than the others. The only

case where the two strategies give conflicting advice is when you

have to forgo an offer from an acceptable investor to see if you'll

get an offer from a better one. If you talk to investors in parallel

and push back on exploding offers with excessively short deadlines,

that will almost never happen. But if it does, "get the best

investors" is in the average case bad advice. The best investors

are also the most selective, because they get their pick of all the

startups. They reject nearly everyone they talk to, which means

in the average case it's a bad trade to exchange a definite offer

from an acceptable investor for a potential offer from a better

one.(The situation is different in phase 1. You can't apply to all the

incubators in parallel, because some offset their schedules to

prevent this. In phase 1, "accept offers greedily" and "get the

best investors" do conflict, so if you want to apply to multiple

incubators, you should do it in such a way that the ones you want

most decide first.)Sometimes when you're raising money from multiple investors, a

series A will emerge out of those conversations, and these rules

even cover what to do in that case. When an investor starts to

talk to you about a series A, keep taking smaller investments till

they actually give you a termsheet. There's no practical difficulty.

If the smaller investments are on convertible notes, they'll just

convert into the series A round. The series A investor won't like

having all these other random investors as bedfellows, but if it

bothers them so much they should get on with giving you a termsheet.

Till they do, you don't know for sure they will, and the greedy

algorithm tells you what to do.

[23]

Don't sell more than 25% in phase 2.If you do well, you will probably raise a series A round eventually.

I say probably because things are changing with series A rounds.

Startups may start to skip them. But only one company we've funded

has so far, so tentatively assume the path to huge passes through

an A round.

[24]Which means you should avoid doing things in earlier rounds that

will mess up raising an A round. For example, if you've sold more

than about 40% of your company total, it starts to get harder to

raise an A round, because VCs worry there will not be enough stock

left to keep the founders motivated.Our rule of thumb is not to sell more than 25% in phase 2, on top

of whatever you sold in phase 1, which should be less than 15%. If

you're raising money on uncapped notes, you'll have to guess what

the eventual equity round valuation might be. Guess conservatively.(Since the goal of this rule is to avoid messing up the series A,

there's obviously an exception if you end up raising a series A in

phase 2, as a handful of startups do.)

Have one person handle fundraising.If you have multiple founders, pick one to handle fundraising so

the other(s) can keep working on the company. And since the danger

of fundraising is not the time taken up by the actual meetings but

that it becomes the top idea in your mind, the founder who handles

fundraising should make a conscious effort to insulate the other

founder(s) from the details of the process.

[25](If the founders mistrust one another, this could cause some friction.

But if the founders mistrust one another, you have worse problems

to worry about than how to organize fundraising.)The founder who handles fundraising should be the CEO, who should

in turn be the most formidable of the founders. Even if the CEO

is a programmer and another founder is a salesperson? Yes. If you

happen to be that type of founding team, you're effectively a single

founder when it comes to fundraising.It's ok to bring all the founders to meet an investor who will

invest a lot, and who needs this meeting as the final step before

deciding. But wait till that point. Introducing an investor to

your cofounder(s) should be like introducing a girl/boyfriend to

your parents — something you do only when things reach a certain

stage of seriousness.Even if there are still one or more founders focusing on the company

during fundraising, growth will slow. But try to get as much growth

as you can, because fundraising is a segment of time, not a point,

and what happens to the company during that time affects the outcome.

If your numbers grow significantly between two investor meetings,

investors will be hot to close, and if your numbers are flat or

down they'll start to get cold feet.

You'll need an executive summary and (maybe) a deck.Traditionally phase 2 fundraising consists of presenting a slide

deck in person to investors. Sequoia describes what such a deck

should contain, and

since they're the customer you can take their word for it.I say "traditionally" because I'm ambivalent about decks, and (though

perhaps this is wishful thinking) they seem to be on the way out.

A lot of the most successful startups we fund never make decks in

phase 2. They just talk to investors and explain what they plan

to do. Fundraising usually takes off fast for the startups that

are most successful at it, and they're thus able to excuse themselves

by saying that they haven't had time to make a deck.You'll also want an executive summary, which should be no more than

a page long and describe in the most matter of fact language what

you plan to do, why it's a good idea, and what progress you've made

so far. The point of the summary is to remind the investor (who

may have met many startups that day) what you talked about.Assume that if you give someone a copy of your deck or executive

summary, it will be passed on to whoever you'd least like to have

it. But don't refuse on that account to give copies to investors

you meet. You just have to treat such leaks as a cost of doing

business. In practice it's not that high a cost. Though founders

are rightly indignant when their plans get leaked to competitors,

I can't think of a startup whose outcome has been affected by it.Sometimes an investor will ask you to send them your deck and/or

executive summary before they decide whether to meet with you. I

wouldn't do that. It's a sign they're not really interested.

Stop fundraising when it stops working.When do you stop fundraising? Ideally when you've raised enough.

But what if you haven't raised as much as you'd like? When do you

give up?It's hard to give general advice about this, because there have

been cases of startups that kept trying to raise money even when

it seemed hopeless, and miraculously succeeded. But what I usually

tell founders is to stop fundraising when you start to get a lot

of air in the straw. When you're drinking through a straw, you can

tell when you get to the end of the liquid because you start to get

a lot of air in the straw. When your fundraising options run out,

they usually run out in the same way. Don't keep sucking on the

straw if you're just getting air. It's not going to get better.

Don't get addicted to fundraising.Fundraising is a chore for most founders, but some find it more

interesting than working on their startup. The work at an early

stage startup often consists of unglamorous schleps. Whereas fundraising, when it's

going well, can be quite the opposite. Instead of sitting in your

grubby apartment listening to users complain about bugs in your

software, you're being offered millions of dollars by famous investors

over lunch at a nice restaurant.

[26]The danger of fundraising is particularly acute for people who are

good at it. It's always fun to work on something you're good at.

If you're one of these people, beware. Fundraising is not what

will make your company successful. Listening to users complain

about bugs in your software is what will make you successful. And

the big danger of getting addicted to fundraising is not merely

that you'll spend too long on it or raise too much money. It's

that you'll start to think of yourself as being already successful,

and lose your taste for the schleps you need to undertake to actually

be successful. Startups can be destroyed by this.When I see a startup with young founders that is fabulously successful

at fundraising, I mentally decrease my estimate of the probability

that they'll succeed. The press may be writing about them as if

they'd been anointed as the next Google, but I'm thinking "this is

going to end badly."

Don't raise too much.Though only a handful of startups have to worry about this, it is

possible to raise too much. The dangers of raising too much are

subtle but insidious. One is that it will set impossibly high

expectations. If you raise an excessive amount of money, it will

be at a high valuation, and the danger of raising money at too high

a valuation is that you won't be able to increase it sufficiently

the next time you raise money.A company's valuation is expected to rise each time it raises money.

If not it's a sign of a company in trouble, which makes you

unattractive to investors. So if you raise money in phase 2 at a

post-money valuation of $30 million, the pre-money valuation of

your next round, if you want to raise one, is going to have to be

at least $50 million. And you have to be doing really, really well

to raise money at $50 million.It's very dangerous to let the competitiveness of your current round

set the performance threshold you have to meet to raise your next

one, because the two are only loosely coupled.But the money itself may be more dangerous than the valuation. The

more you raise, the more you spend, and spending a lot of money can

be disastrous for an early stage startup. Spending a lot makes it

harder to become profitable, and perhaps even worse, it makes you

more rigid, because the main way to spend money is people, and the

more people you have, the harder it is to change directions. So

if you do raise a huge amount of money, don't spend it. (You will

find that advice almost impossible to follow, so hot will be the

money burning a hole in your pocket, but I feel obliged at least

to try.)

Be nice.Startups raising money occasionally alienate investors by seeming

arrogant. Sometimes because they are arrogant, and sometimes because

they're noobs clumsily attempting to mimic the toughness they've

observed in experienced founders.It's a mistake to behave arrogantly to investors. While there are

certain situations in which certain investors like certain kinds

of arrogance, investors vary greatly in this respect, and a flick

of the whip that will bring one to heel will make another roar with

indignation. The only safe strategy is never to seem arrogant at

all.That will require some diplomacy if you follow the advice I've given

here, because the advice I've given is essentially how to play

hardball back. When you refuse to meet an investor because you're

not in fundraising mode, or slow down your interactions with an

investor who moves too slow, or treat a contingent offer as the no

it actually is and then, by accepting offers greedily, end up leaving

that investor out, you're going to be doing things investors don't

like. So you must cushion the blow with soft words. At YC we tell

startups they can blame us. And now that I've written this, everyone

else can blame me if they want. That plus the inexperience card

should work in most situations: sorry, we think you're great, but

PG said startups shouldn't \_\_\_, and since we're new to fundraising,

we feel like we have to play it safe.The danger of behaving arrogantly is greatest when you're doing

well. When everyone wants you, it's hard not to let it go to your

head. Especially if till recently no one wanted you. But restrain

yourself. The startup world is a small place, and startups have

lots of ups and downs. This is a domain where it's more true than

usual that pride goeth before a fall.

[27]Be nice when investors reject you as well. The best investors are

not wedded to their initial opinion of you. If they reject you in

phase 2 and you end up doing well, they'll often invest in phase

3. In fact investors who reject you are some of your warmest leads

for future fundraising. Any investor who spent significant time

deciding probably came close to saying yes. Often you have some

internal champion who only needs a little more evidence to convince

the skeptics. So it's wise not merely to be nice to investors who

reject you, but (unless they behaved badly) to treat it as the

beginning of a relationship.

The bar will be higher next time.Assume the money you raise in phase 2 will be the last you ever

raise. You must make it to profitability on this money if you can.Over the past several years, the investment community has evolved

from a strategy of anointing a small number of winners early and

then supporting them for years to a strategy of spraying money at

early stage startups and then ruthlessly culling them at the next

stage. This is probably the optimal strategy for investors. It's

too hard to pick winners early on. Better to let the market do it

for you. But it often comes as a surprise to startups how much

harder it is to raise money in phase 3.When your company is only a couple months old, all it has to be is

a promising experiment that's worth funding to see how it turns

out. The next time you raise money, the experiment has to have

worked. You have to be on a trajectory that leads to going public.

And while there are some ideas where the proof that the experiment

worked might consist of e.g. query response times, usually the proof

is profitability. Usually phase 3 fundraising has to be type A

fundraising.In practice there are two ways startups hose themselves between

phases 2 and 3. Some are just too slow to become profitable. They

raise enough money to last for two years. There doesn't seem any

particular urgency to be profitable. So they don't make any effort

to make money for a year. But by that time, not making money has

become habitual. When they finally decide to try, they find they

can't.The other way companies hose themselves is by letting their expenses

grow too fast. Which almost always means hiring too many people.

You usually shouldn't go out and hire 8 people as soon as you raise

money at phase 2. Usually you want to wait till you have growth

(and thus usually revenues) to justify them. A lot of VCs will

encourage you to hire aggressively. VCs generally tell you to spend

too much, partly because as money people they err on the side of

solving problems by spending money, and partly because they want

you to sell them more of your company in subsequent rounds. Don't

listen to them.

Don't make things complicated.I realize it may seem odd to sum up this huge treatise by saying

that my overall advice is not to make fundraising too complicated,

but if you go back and look at this list you'll see it's basically

a simple recipe with a lot of implications and edge cases. Avoid

investors till you decide to raise money, and then when you do,

talk to them all in parallel, prioritized by expected value, and

accept offers greedily. That's fundraising in one sentence. Don't

introduce complicated optimizations, and don't let investors introduce

complications either.Fundraising is not what will make you successful. It's just a means

to an end. Your primary goal should be to get it over with and get

back to what will make you successful — making things and talking

to users — and the path I've described will for most startups

be the surest way to that destination.Be good, take care of yourselves, and don't leave the path.

Notes[1]

The worst explosions happen when unpromising-seeming startups

encounter mediocre investors. Good investors don't lead startups

on; their reputations are too valuable. And startups that seem

promising can usually get enough money from good investors that

they don't have to talk to mediocre ones. It is the unpromising-seeming

startups that have to resort to raising money from mediocre investors.

And it's particularly damaging when these investors flake, because

unpromising-seeming startups are usually more desperate for money.(Not all unpromising-seeming startups do badly. Some are merely

ugly ducklings in the sense that they violate current startup

fashions.)[2]

One YC founder told me:

I think in general we've done ok at fundraising, but I managed

to screw up twice at the exact same thing — trying to focus

on building the company and fundraising at the same time.

[3]

There is one subtle danger you have to watch out for here, which

I warn about later: beware of getting too high a valuation from an

eager investor, lest that set an impossibly high target when raising

additional money.[4]

If they really need a meeting, then they're not ready to invest,

regardless of what they say. They're still deciding, which means

you're being asked to come in and convince them. Which is fundraising.[5]

Associates at VC firms regularly cold email startups. Naive

founders think "Wow, a VC is interested in us!" But an associate

is not a VC. They have no decision-making power. And while they

may introduce startups they like to partners at their firm, the

partners discriminate against deals that come to them this way. I

don't know of a single VC investment that began with an associate

cold-emailing a startup. If you want to approach a specific firm,

get an intro to a partner from someone they respect.It's ok to talk to an associate if you get an intro to a VC firm

or they see you at a Demo Day and they begin by having an associate

vet you. That's not a promising lead and should therefore get low

priority, but it's not as completely worthless as a cold email.Because the title "associate" has gotten a bad reputation, a few

VC firms have started to give their associates the title "partner,"

which can make things very confusing. If you're a YC startup you

can ask us who's who; otherwise you may have to do some research

online. There may be a special title for actual partners. If

someone speaks for the firm in the press or a blog on the firm's

site, they're probably a real partner. If they're on boards of

directors they're probably a real partner.There are titles between "associate" and "partner," including

"principal" and "venture partner." The meanings of these titles

vary too much to generalize.[6]

For similar reasons, avoid casual conversations with potential

acquirers. They can lead to distractions even more dangerous than

fundraising. Don't even take a meeting with a potential acquirer

unless you want to sell your company right now.[7]

Joshua Reeves specifically suggests asking each investor to

intro you to two more investors.Don't ask investors who say no for introductions to other investors.

That will in many cases be an anti-recommendation.[8]

This is not always as deliberate as its sounds. A lot of the

delays and disconnects between founders and investors are induced

by the customs of the venture business, which have evolved the way

they have because they suit investors' interests.[9]

One YC founder who read a draft of this essay wrote:

This is the most important section. I think it might bear stating

even more clearly. "Investors will deliberately affect more

interest than they have to preserve optionality. If an investor

seems very interested in you, they still probably won't invest.

The solution for this is to assume the worst — that an investor

is just feigning interest — until you get a definite commitment."

[10]

Though you should probably pack investor meetings as closely

as you can, Jeff Byun mentions one reason not to: if you pack

investor meetings too closely, you'll have less time for your pitch

to evolve.Some founders deliberately schedule a handful of lame investors

first, to get the bugs out of their pitch.[11]

There is not an efficient market in this respect. Some of the

most useless investors are also the highest maintenance.[12]

Incidentally, this paragraph is sales 101. If you want to see

it in action, go talk to a car dealer.[13]

I know one very smooth founder who used to end investor meetings

with "So, can I count you in?" delivered as if it were "Can you

pass the salt?" Unless you're very smooth (if you're not sure...),

do not do this yourself. There is nothing more unconvincing, for

an investor, than a nerdy founder trying to deliver the lines meant

for a smooth one.Investors are fine with funding nerds. So if you're a nerd, just

try to be a good nerd, rather than doing a bad imitation of a smooth

salesman.[14]

Ian Hogarth suggests a good way to tell how serious potential

investors are: the resources they expend on you after the first

meeting. An investor who's seriously interested will already be

working to help you even before they've committed.[15]

In principle you might have to think about so-called "signalling

risk." If a prestigious VC makes a small seed investment in you,

what if they don't want to invest the next time you raise money?

Other investors might assume that the VC knows you well, since

they're an existing investor, and if they don't want to invest in

your next round, that must mean you suck. The reason I say "in

principle" is that in practice signalling hasn't been much of a

problem so far. It rarely arises, and in the few cases where it

does, the startup in question usually is doing badly and is doomed

anyway.If you have the luxury of choosing among seed investors, you can

play it safe by excluding VC firms. But it isn't critical to.[16]

Sometimes a competitor will deliberately threaten you with a

lawsuit just as you start fundraising, because they know you'll

have to disclose the threat to potential investors and they hope

this will make it harder for you to raise money. If this happens

it will probably frighten you more than investors. Experienced

investors know about this trick, and know the actual lawsuits rarely

happen. So if you're attacked in this way, be forthright with

investors. They'll be more alarmed if you seem evasive than if you

tell them everything.[17]

A related trick is to claim that they'll only invest contingently

on other investors doing so because otherwise you'd be "undercapitalized."

This is almost always bullshit. They can't estimate your minimum

capital needs that precisely.[18]

You won't hire all those 20 people at once, and you'll probably

have some revenues before 18 months are out. But those too are

acceptable or at least accepted additions to the margin for error.[19]

Type A fundraising is so much better that it might even be

worth doing something different if it gets you there sooner. One

YC founder told me that if he were a first-time founder again he'd

"leave ideas that are up-front capital intensive to founders with

established reputations."[20]

I don't know whether this happens because they're innumerate,

or because they believe they have zero ability to predict startup

outcomes (in which case this behavior at least wouldn't be irrational).

In either case the implications are similar.[21]

If you're a YC startup and you have an investor who for some

reason insists that you decide the price, any YC partner can estimate

a market price for you.[22]

You should respond in kind when investors behave upstandingly

too. When an investor makes you a clean offer with no deadline,

you have a moral obligation to respond promptly.[23]

Tell the investors talking to you about an A round about the

smaller investments you raise as you raise them. You owe them such

updates on your cap table, and this is also a good way to pressure

them to act. They won't like you raising other money and may

pressure you to stop, but they can't legitimately ask you to commit

to them till they also commit to you. If they want you to stop

raising money, the way to do it is to give you a series A termsheet

with a no-shop clause.You can relent a little if the potential series A investor has a

great reputation and they're clearly working fast to get you a

termsheet, particularly if a third party like YC is involved to

ensure there are no misunderstandings. But be careful.[24]

The company is Weebly, which made it to profitability on a

seed investment of $650k. They did try to raise a series A in the

fall of 2008 but (no doubt partly because it was the fall of 2008)

the terms they were offered were so bad that they decided to skip

raising an A round.[25]

Another advantage of having one founder take fundraising

meetings is that you never have to negotiate in real time, which

is something inexperienced founders should avoid. One YC founder

told me:

Investors are professional negotiators and can negotiate on the

spot very easily. If only one founder is in the room, you can

say "I need to circle back with my co-founder" before making any

commitments. I used to do this all the time.

[26]

You'll be lucky if fundraising feels pleasant enough to become

addictive. More often you have to worry about the other

extreme — becoming demoralized when investors reject you. As

one (very successful) YC founder wrote after reading a draft of

this:

It's hard to mentally deal with the sheer scale of rejection in

fundraising and if you are not in the right mindset you will fail.

Users may love you but these supposedly smart investors may not

understand you at all. At this point for me, rejection still

rankles but I've come to accept that investors are just not super

thoughtful for the most part and you need to play the game according

to certain somewhat depressing rules (many of which you are

listing) in order to win.

[27]

The actual sentence in the King James Bible is "Pride goeth

before destruction, and an haughty spirit before a fall."Thanks to Slava Akhmechet, Sam Altman, Nate Blecharczyk,

Adora Cheung, Bill Clerico, John Collison, Patrick Collison, Parker

Conrad, Ron Conway, Travis Deyle, Jason Freedman, Joe Gebbia, Mattan

Griffel, Kevin Hale, Jacob Heller, Ian Hogarth, Justin Kan, Professor

Moriarty, Nikhil Nirmel, David Petersen, Geoff Ralston, Joshua

Reeves, Yuri Sagalov, Emmett Shear, Rajat Suri, Garry Tan, and Nick

Tomarello for reading drafts of this.Russian Translation

Investor Herd Dynamics

Want to start a startup? Get funded by

Y Combinator.

August 2013The biggest component in most investors' opinion of you is the

opinion of other investors. Which is of course a recipe for

exponential growth. When one investor wants to invest in you, that

makes other investors want to, which makes others want to, and so

on.Sometimes inexperienced founders mistakenly conclude that manipulating

these forces is the essence of fundraising. They hear stories about

stampedes to invest in successful startups, and think it's therefore

the mark of a successful startup to have this happen. But actually

the two are not that highly correlated. Lots of startups that cause

stampedes end up flaming out (in extreme cases, partly as a result

of the stampede), and lots of very successful startups were only

moderately popular with investors the first time they raised money.So the point of this essay is not to explain how to create a stampede,

but merely to explain the forces that generate them. These forces

are always at work to some degree in fundraising, and they can cause

surprising situations. If you understand them, you can at least

avoid being surprised.One reason investors like you more when other investors like you

is that you actually become a better investment. Raising money

decreases the risk of failure. Indeed, although investors hate it,

you are for this reason justified in raising your valuation for

later investors. The investors who invested when you had no money

were taking more risk, and are entitled to higher returns. Plus a

company that has raised money is literally more valuable. After

you raise the first million dollars, the company is at least a

million dollars more valuable, because it's the same company as

before, plus it has a million dollars in the bank.

[1]Beware, though, because later investors so hate to have the price

raised on them that they resist even this self-evident reasoning.

Only raise the price on an investor you're comfortable with losing,

because some will angrily refuse.

[2]The second reason investors like you more when you've had some

success at fundraising is that it makes you more confident, and an

investors' opinion of you is the foundation

of their opinion of your company. Founders are often surprised how

quickly investors seem to know when they start to succeed at raising

money. And while there are in fact lots of ways for such information

to spread among investors, the main vector is probably the founders

themselves. Though they're often clueless about technology, most

investors are pretty good at reading people. When fundraising is

going well, investors are quick to sense it in your increased

confidence. (This is one case where the average founder's inability

to remain poker-faced works to your advantage.)But frankly the most important reason investors like you more when

you've started to raise money is that they're bad at judging startups.

Judging startups is hard even for the best investors. The mediocre

ones might as well be flipping coins. So when mediocre investors

see that lots of other people want to invest in you, they assume

there must be a reason. This leads to the phenomenon known in the

Valley as the "hot deal," where you have more interest from investors

than you can handle.The best investors aren't influenced much by the opinion of other

investors. It would only dilute their own judgment to average it

together with other people's. But they are indirectly influenced

in the practical sense that interest from other investors imposes

a deadline. This is the fourth way in which offers beget offers.

If you start to get far along the track toward an offer with one

firm, it will sometimes provoke other firms, even good ones, to

make up their minds, lest they lose the deal.Unless you're a wizard at negotiation (and if you're not sure,

you're not) be very careful about exaggerating this to push a good

investor to decide. Founders try this sort of thing all the time,

and investors are very sensitive to it. If anything oversensitive.

But you're safe so long as you're telling the truth. If you're

getting far along with investor B, but you'd rather raise money

from investor A, you can tell investor A that this is happening.

There's no manipulation in that. You're genuinely in a bind, because

you really would rather raise money from A, but you can't safely

reject an offer from B when it's still uncertain what A will decide.Do not, however, tell A who B is. VCs will sometimes ask which

other VCs you're talking to, but you should never tell them. Angels

you can sometimes tell about other angels, because angels cooperate

more with one another. But if VCs ask, just point out that they

wouldn't want you telling other firms about your conversations, and

you feel obliged to do the same for any firm you talk to. If they

push you, point out that you're inexperienced at fundraising — which

is always a safe card to play — and you feel you have to be

extra cautious.

[3]While few startups will experience a stampede of interest, almost

all will at least initially experience the other side of this

phenomenon, where the herd remains clumped together at a distance.

The fact that investors are so much influenced by other investors'

opinions means you always start out in something of a hole. So

don't be demoralized by how hard it is to get the first commitment,

because much of the difficulty comes from this external force. The

second will be easier.Notes[1]

An accountant might say that a company that has raised a million

dollars is no richer if it's convertible debt, but in practice money

raised as convertible debt is little different from money raised

in an equity round.[2]

Founders are often surprised by this, but investors can get

very emotional. Or rather indignant; that's the main emotion I've

observed; but it is very common, to the point where it sometimes

causes investors to act against their own interests. I know of one

investor who invested in a startup at a $15 million valuation cap.

Earlier he'd had an opportunity to invest at a $5 million cap, but

he refused because a friend who invested earlier had been able to

invest at a $3 million cap.[3]

If an investor pushes you hard to tell them about your conversations

with other investors, is this someone you want as an investor?

Thanks to Paul Buchheit, Jessica Livingston, Geoff Ralston, and Garry Tan

for reading drafts of this.Russian Translation

How to Convince Investors

Want to start a startup? Get funded by

Y Combinator.

August 2013When people hurt themselves lifting heavy things, it's usually

because they try to lift with their back. The right way to lift

heavy things is to let your legs do the work. Inexperienced founders

make the same mistake when trying to convince investors. They try

to convince with their pitch. Most would be better off if they let

their startup do the work — if they started by understanding why

their startup is worth investing in, then simply explained this

well to investors.Investors are looking for startups that will be very successful.

But that test is not as simple as it sounds. In startups, as in a

lot of other domains, the distribution of outcomes follows a power

law, but in startups the curve is startlingly steep. The big

successes are so big they

dwarf the rest. And since there are only

a handful each year (the conventional wisdom is 15), investors treat

"big success" as if it were binary. Most are interested in you if

you seem like you have a chance, however small, of being one of the

15 big successes, and otherwise not.

[1](There are a handful of angels who'd be interested in a company

with a high probability of being moderately successful. But angel

investors like big successes too.)How do you seem like you'll be one of the big successes? You need

three things: formidable founders, a promising market, and (usually)

some evidence of success so far.FormidableThe most important ingredient is formidable founders. Most investors

decide in the first few minutes whether you seem like a winner or

a loser, and once their opinion is set it's hard to change. [2]

Every startup has reasons both to invest and not to invest. If

investors think you're a winner they focus on the former, and if

not they focus on the latter. For example, it might be a rich

market, but with a slow sales cycle. If investors are impressed

with you as founders, they say they want to invest because it's a

rich market, and if not, they say they can't invest because of the

slow sales cycle.They're not necessarily trying to mislead you. Most investors are

genuinely unclear in their own minds why they like or dislike

startups. If you seem like a winner, they'll like your idea more.

But don't be too smug about this weakness of theirs, because you

have it too; almost everyone does.There is a role for ideas of course. They're fuel for the fire

that starts with liking the founders. Once investors like you,

you'll see them reaching for ideas: they'll be saying "yes, and you

could also do x." (Whereas when they don't like you, they'll be

saying "but what about y?")But the foundation of convincing investors is to seem formidable,

and since this isn't a word most people use in conversation much,

I should explain what it means. A formidable person is one who

seems like they'll get what they want, regardless of whatever

obstacles are in the way. Formidable is close to confident, except

that someone could be confident and mistaken. Formidable is roughly

justifiably confident.There are a handful of people who are really good at seeming

formidable — some because they actually are very formidable and

just let it show, and others because they are more or less con

artists.

[3]

But most founders, including many who will go on

to start very successful companies, are not that good at seeming

formidable the first time they try fundraising. What should they

do?

[4]What they should not do is try to imitate the swagger of more

experienced founders. Investors are not always that good at judging

technology, but they're good at judging confidence. If you try to

act like something you're not, you'll just end up in an uncanny

valley. You'll depart from sincere, but never arrive at convincing.TruthThe way to seem most formidable as an inexperienced founder is to

stick to the truth. How formidable you seem isn't a constant. It

varies depending on what you're saying. Most people can seem

confident when they're saying "one plus one is two," because they

know it's true. The most diffident person would be puzzled and

even slightly contemptuous if they told a VC "one plus one is two"

and the VC reacted with skepticism. The magic ability of people

who are good at seeming formidable is that they can do this with

the sentence "we're going to make a billion dollars a year." But

you can do the same, if not with that sentence with some fairly

impressive ones, so long as you convince yourself first.That's the secret. Convince yourself that your startup is worth

investing in, and then when you explain this to investors they'll

believe you. And by convince yourself, I don't mean play mind games

with yourself to boost your confidence. I mean truly evaluate

whether your startup is worth investing in. If it isn't, don't try

to raise money.

[5]

But if it is, you'll be telling the truth

when you tell investors it's worth investing in, and they'll sense

that. You don't have to be a smooth presenter if you understand

something well and tell the truth about it.To evaluate whether your startup is worth investing in, you have

to be a domain expert. If you're not a domain expert, you can be

as convinced as you like about your idea, and it will seem to

investors no more than an instance of the Dunning-Kruger effect.

Which in fact it will usually be. And investors can tell fairly

quickly whether you're a domain expert by how well you answer their

questions. Know everything about your market.

[6]Why do founders persist in trying to convince investors of things

they're not convinced of themselves? Partly because we've all been

trained to.When my friends Robert Morris and Trevor Blackwell were in grad

school, one of their fellow students was on the receiving end of a

question from their faculty advisor that we still quote today. When

the unfortunate fellow got to his last slide, the professor burst

out:

Which one of these conclusions do you actually believe?

One of the artifacts of the way schools are organized is that we

all get trained to talk even when we have nothing to say. If you

have a ten page paper due, then ten pages you must write, even if

you only have one page of ideas. Even if you have no ideas. You

have to produce something. And all too many startups go into

fundraising in the same spirit. When they think it's time to raise

money, they try gamely to make the best case they can for their

startup. Most never think of pausing beforehand to ask whether

what they're saying is actually convincing, because they've all

been trained to treat the need to present as a given — as an area

of fixed size, over which however much truth they have must needs

be spread, however thinly.The time to raise money is not when you need it, or when you reach

some artificial deadline like a Demo Day. It's when you can convince

investors, and not before.

[7]And unless you're a good con artist, you'll never convince investors

if you're not convinced yourself. They're far better at detecting

bullshit than you are at producing it, even if you're producing it

unknowingly. If you try to convince investors before you've convinced

yourself, you'll be wasting both your time.But pausing first to convince yourself will do more than save you

from wasting your time. It will force you to organize your thoughts.

To convince yourself that your startup is worth investing in, you'll

have to figure out why it's worth investing in. And if you can

do that you'll end up with more than added confidence. You'll also

have a provisional roadmap of how to succeed.MarketNotice I've been careful to talk about whether a startup is worth

investing in, rather than whether it's going to succeed. No one

knows whether a startup is going to succeed. And it's a good thing

for investors that this is so, because if you could know in advance

whether a startup would succeed, the stock price would already be

the future price, and there would be no room for investors to make

money. Startup investors know that every investment is a bet, and

against pretty long odds.So to prove you're worth investing in, you don't have to prove

you're going to succeed, just that you're a sufficiently good bet.

What makes a startup a sufficiently good bet? In addition to

formidable founders, you need a plausible path to owning a big piece

of a big market. Founders think of startups as ideas, but investors

think of them as markets. If there are x number of customers who'd

pay an average of $y per year for what you're making, then the total

addressable market, or TAM, of your company is $xy. Investors don't

expect you to collect all that money, but it's an upper bound on

how big you can get.Your target market has to be big, and it also has to be capturable

by you. But the market doesn't have to be big yet, nor do you

necessarily have to be in it yet. Indeed, it's often better to

start in a small market that will either turn into a big one or

from which you can move into a big one. There just has to be some

plausible sequence of hops that leads to dominating a big market a

few years down the line.The standard of plausibility varies dramatically depending on the

age of the startup. A three month old company at Demo Day only

needs to be a promising experiment that's worth funding to see how

it turns out. Whereas a two year old company raising a series A

round needs to be able to show the experiment worked.

[8]But every company that gets really big is "lucky" in the sense that

their growth is due mostly to some external wave they're riding,

so to make a convincing case for becoming huge, you have to identify

some specific trend you'll benefit from. Usually you can find this

by asking "why now?" If this is such a great idea, why hasn't

someone else already done it? Ideally the answer is that it only

recently became a good idea, because something changed, and no one

else has noticed yet.Microsoft for example was not going to grow huge selling Basic

interpreters. But by starting there they were perfectly poised to

expand up the stack of microcomputer software as microcomputers

grew powerful enough to support one. And microcomputers turned out

to be a really huge wave, bigger than even the most optimistic

observers would have predicted in 1975.But while Microsoft did really well and there is thus a temptation

to think they would have seemed a great bet a few months in, they

probably didn't. Good, but not great. No company, however successful,

ever looks more than a pretty good bet a few months in. Microcomputers

turned out to be a big deal, and Microsoft both executed well and

got lucky. But it was by no means obvious that this was how things

would play out. Plenty of companies seem as good a bet a few months

in. I don't know about startups in general, but at least half the

startups we fund could make as good a case as Microsoft could have

for being on a path to dominating a large market. And who can

reasonably expect more of a startup than that?RejectionIf you can make as good a case as Microsoft could have, will you

convince investors? Not always. A lot of VCs would have rejected

Microsoft.

[9]

Certainly some rejected Google. And getting

rejected will put you in a slightly awkward position, because as

you'll see when you start fundraising, the most common question

you'll get from investors will be "who else is investing?" What do

you say if you've been fundraising for a while and no one has

committed yet?

[10]The people who are really good at acting formidable often solve

this problem by giving investors the impression that while no

investors have committed yet, several are about to. This is arguably

a permissible tactic. It's slightly dickish of investors to care

more about who else is investing than any other aspect of your

startup, and misleading them about how far along you are with other

investors seems the complementary countermove. It's arguably an

instance of scamming a scammer. But I don't recommend this approach

to most founders, because most founders wouldn't be able to carry

it off. This is the single most common lie told to investors, and

you have to be really good at lying to tell members of some profession

the most common lie they're told.If you're not a master of negotiation (and perhaps even if you are)

the best solution is to tackle the problem head-on, and to explain

why investors have turned you down and why they're mistaken. If

you know you're on the right track, then you also know why investors

were wrong to reject you. Experienced investors are well aware that

the best ideas are also the scariest. They all know about the VCs

who rejected Google. If instead of seeming evasive and ashamed

about having been turned down (and thereby implicitly agreeing with

the verdict) you talk candidly about what scared investors about

you, you'll seem more confident, which they like, and you'll probably

also do a better job of presenting that aspect of your startup. At

the very least, that worry will now be out in the open instead of

being a gotcha left to be discovered by the investors you're currently

talking to, who will be proud of and thus attached to their discovery.

[11]This strategy will work best with the best investors, who are both

hard to bluff and who already believe most other investors are

conventional-minded drones doomed always to miss the big outliers.

Raising money is not like applying to college, where you can assume

that if you can get into MIT, you can also get into Foobar State.

Because the best investors are much smarter than the rest, and the

best startup ideas look initially like

bad ideas, it's not uncommon

for a startup to be rejected by all the VCs except the best ones.

That's what happened to Dropbox. Y Combinator started in Boston,

and for the first 3 years we ran alternating batches in Boston and

Silicon Valley. Because Boston investors were so few and so timid,

we used to ship Boston batches out for a second Demo Day in Silicon

Valley. Dropbox was part of a Boston batch, which means all those

Boston investors got the first look at Dropbox, and none of them

closed the deal. Yet another backup and syncing thing, they all

thought. A couple weeks later, Dropbox raised a series A round

from Sequoia.

[12]DifferentNot understanding that investors view investments as bets combines

with the ten page paper mentality to prevent founders from even

considering the possibility of being certain of what they're saying.

They think they're trying to convince investors of something very

uncertain — that their startup will be huge — and convincing anyone

of something like that must obviously entail some wild feat of

salesmanship. But in fact when you raise money you're trying to

convince investors of something so much less speculative — whether

the company has all the elements of a good bet — that you can

approach the problem in a qualitatively different way. You can

convince yourself, then convince them.And when you convince them, use the same matter-of-fact language

you used to convince yourself. You wouldn't use vague, grandiose

marketing-speak among yourselves. Don't use it with investors

either. It not only doesn't work on them, but seems a mark of

incompetence. Just be concise. Many investors explicitly use that

as a test, reasoning (correctly) that if you can't explain your

plans concisely, you don't really understand them. But even investors

who don't have a rule about this will be bored and frustrated by

unclear explanations.

[13]So here's the recipe for impressing investors when you're not already

good at seeming formidable:

Make something worth investing in. Understand why it's worth investing in. Explain that clearly to investors.

If you're saying something you know is true, you'll seem confident

when you're saying it. Conversely, never let pitching draw you

into bullshitting. As long as you stay on the territory of truth,

you're strong. Make the truth good, then just tell it.Notes[1]

There's no reason to believe this number is a constant. In

fact it's our explicit goal at Y Combinator to increase it, by

encouraging people to start startups who otherwise wouldn't have.[2]

Or more precisely, investors decide whether you're a loser

or possibly a winner. If you seem like a winner, they may then,

depending on how much you're raising, have several more meetings

with you to test whether that initial impression holds up.But if you seem like a loser they're done, at least for the next

year or so. And when they decide you're a loser they usually decide

in way less than the 50 minutes they may have allotted for the first

meeting. Which explains the astonished stories one always hears

about VC inattentiveness. How could these people make investment

decisions well when they're checking their messages during startups'

presentations? The solution to that mystery is that they've already

made the decision.[3]

The two are not mutually exclusive. There are people who are

both genuinely formidable, and also really good at acting that way.[4]

How can people who will go on to create giant companies not

seem formidable early on? I think the main reason is that their

experience so far has trained them to keep their wings folded, as

it were. Family, school, and jobs encourage cooperation, not

conquest. And it's just as well they do, because even being Genghis

Khan is probably 99% cooperation. But the result is that most

people emerge from the tube of their upbringing in their early

twenties compressed into the shape of the tube. Some find they

have wings and start to spread them. But this takes a few years.

In the beginning even they don't know yet what they're capable of.[5]

In fact, change what you're doing. You're investing your own

time in your startup. If you're not convinced that what you're

working on is a sufficiently good bet, why are you even working on

that?[6]

When investors ask you a question you don't know the answer

to, the best response is neither to bluff nor give up, but instead

to explain how you'd figure out the answer. If you can work out a

preliminary answer on the spot, so much the better, but explain

that's what you're doing.[7]

At YC we try to ensure startups are ready to raise money on

Demo Day by encouraging them to ignore investors and instead focus

on their companies till about a week before. That way most reach

the stage where they're sufficiently convincing well before Demo

Day. But not all do, so we also give any startup that wants to the

option of deferring to a later Demo Day.[8]

Founders are often surprised by how much harder it is to raise

the next round. There is a qualitative difference in investors'

attitudes. It's like the difference between being judged as a kid

and as an adult. The next time you raise money, it's not enough

to be promising. You have to be delivering results.So although it works well to show growth graphs at either stage,

investors treat them differently. At three months, a growth graph

is mostly evidence that the founders are effective. At two years,

it has to be evidence of a promising market and a company tuned to

exploit it.[9]

By this I mean that if the present day equivalent of the 3

month old Microsoft presented at a Demo Day, there would be investors

who turned them down. Microsoft itself didn't raise outside money,

and indeed the venture business barely existed when they got started

in 1975.[10]

The best investors rarely care who else is investing, but

mediocre investors almost all do. So you can use this question as

a test of investor quality.[11]

To use this technique, you'll have to find out why investors

who rejected you did so, or at least what they claim was the reason.

That may require asking, because investors don't always volunteer

a lot of detail. Make it clear when you ask that you're not trying

to dispute their decision — just that if there is some weakness in

your plans, you need to know about it. You won't always get a real

reason out of them, but you should at least try.[12]

Dropbox wasn't rejected by all the East Coast VCs. There was

one firm that wanted to invest but tried to lowball them.[13]

Alfred Lin points out that it's doubly important for the

explanation of a startup to be clear and concise, because it has

to convince at one remove: it has to work not just on the partner

you talk to, but when that partner re-tells it to colleagues.We consciously optimize for this at YC. When we work with founders

create a Demo Day pitch, the last step is to imagine how an investor

would sell it to colleagues.

Thanks to Marc Andreessen, Sam Altman, Patrick Collison, Ron Conway,

Chris Dixon, Alfred Lin, Ben Horowitz, Steve Huffman, Jessica

Livingston, Greg Mcadoo, Andrew Mason, Geoff Ralston, Yuri Sagalov,

Emmett Shear, Rajat Suri, Garry Tan, Albert Wenger, Fred Wilson,

and Qasar Younis for reading drafts of this.

Do Things that Don't Scale

Want to start a startup? Get funded by

Y Combinator.

July 2013One of the most common types of advice we give at Y Combinator is

to do things that don't scale. A lot of would-be founders believe

that startups either take off or don't. You build something, make

it available, and if you've made a better mousetrap, people beat a

path to your door as promised. Or they don't, in which case the

market must not exist.

[1]Actually startups take off because the founders make them take off.

There may be a handful that just grew by themselves, but usually

it takes some sort of push to get them going. A good metaphor would

be the cranks that car engines had before they got electric starters.

Once the engine was going, it would keep going, but there was a

separate and laborious process to get it going.RecruitThe most common unscalable thing founders have to do at the start

is to recruit users manually. Nearly all startups have to. You

can't wait for users to come to you. You have to go out and get

them.Stripe is one of the most successful startups we've funded, and the

problem they solved was an urgent one. If anyone could have sat

back and waited for users, it was Stripe. But in fact they're

famous within YC for aggressive early user acquisition.Startups building things for other startups have a big pool of

potential users in the other companies we've funded, and none took

better advantage of it than Stripe. At YC we use the term "Collison

installation" for the technique they invented. More diffident

founders ask "Will you try our beta?" and if the answer is yes,

they say "Great, we'll send you a link." But the Collison brothers

weren't going to wait. When anyone agreed to try Stripe they'd say

"Right then, give me your laptop" and set them up on the spot.There are two reasons founders resist going out and recruiting users

individually. One is a combination of shyness and laziness. They'd

rather sit at home writing code than go out and talk to a bunch of

strangers and probably be rejected by most of them. But for a

startup to succeed, at least one founder (usually the CEO) will

have to spend a lot of time on sales and marketing.

[2]The other reason founders ignore this path is that the absolute

numbers seem so small at first. This can't be how the big, famous

startups got started, they think. The mistake they make is to

underestimate the power of compound growth. We encourage every

startup to measure their progress by weekly growth

rate. If you have 100 users, you need to get 10 more next week

to grow 10% a week. And while 110 may not seem much better than

100, if you keep growing at 10% a week you'll be surprised how big

the numbers get. After a year you'll have 14,000 users, and after

2 years you'll have 2 million.You'll be doing different things when you're acquiring users a

thousand at a time, and growth has to slow down eventually. But

if the market exists you can usually start by recruiting users

manually and then gradually switch to less manual methods.

[3]Airbnb is a classic example of this technique. Marketplaces are

so hard to get rolling that you should expect to take heroic measures

at first. In Airbnb's case, these consisted of going door to door

in New York, recruiting new users and helping existing ones improve

their listings. When I remember the Airbnbs during YC, I picture

them with rolly bags, because when they showed up for tuesday dinners

they'd always just flown back from somewhere.FragileAirbnb now seems like an unstoppable juggernaut, but early on it

was so fragile that about 30 days of going out and engaging in

person with users made the difference between success and failure.That initial fragility was not a unique feature of Airbnb. Almost

all startups are fragile initially. And that's one of the biggest

things inexperienced founders and investors (and reporters and

know-it-alls on forums) get wrong about them. They unconsciously

judge larval startups by the standards of established ones. They're

like someone looking at a newborn baby and concluding "there's no

way this tiny creature could ever accomplish anything."It's harmless if reporters and know-it-alls dismiss your startup.

They always get things wrong. It's even ok if investors dismiss

your startup; they'll change their minds when they see growth. The

big danger is that you'll dismiss your startup yourself. I've seen

it happen. I often have to encourage founders who don't see the

full potential of what they're building. Even Bill Gates made that

mistake. He returned to Harvard for the fall semester after starting

Microsoft. He didn't stay long, but he wouldn't have returned at

all if he'd realized Microsoft was going to be even a fraction of

the size it turned out to be.

[4]The question to ask about an early stage startup is not "is this

company taking over the world?" but "how big could this company

get if the founders did the right things?" And the right things

often seem both laborious and inconsequential at the time. Microsoft

can't have seemed very impressive when it was just a couple guys

in Albuquerque writing Basic interpreters for a market of a few

thousand hobbyists (as they were then called), but in retrospect

that was the optimal path to dominating microcomputer software.

And I know Brian Chesky and Joe Gebbia didn't feel like they were

en route to the big time as they were taking "professional" photos

of their first hosts' apartments. They were just trying to survive.

But in retrospect that too was the optimal path to dominating a big

market.How do you find users to recruit manually? If you build something

to solve your own problems, then

you only have to find your peers, which is usually straightforward.

Otherwise you'll have to make a more deliberate effort to locate

the most promising vein of users. The usual way to do that is to

get some initial set of users by doing a comparatively untargeted

launch, and then to observe which kind seem most enthusiastic, and

seek out more like them. For example, Ben Silbermann noticed that

a lot of the earliest Pinterest users were interested in design,

so he went to a conference of design bloggers to recruit users, and

that worked well.

[5]DelightYou should take extraordinary measures not just to acquire users,

but also to make them happy. For as long as they could (which

turned out to be surprisingly long), Wufoo sent each new user a

hand-written thank you note. Your first users should feel that

signing up with you was one of the best choices they ever made.

And you in turn should be racking your brains to think of new ways

to delight them.Why do we have to teach startups this? Why is it counterintuitive

for founders? Three reasons, I think.One is that a lot of startup founders are trained as engineers,

and customer service is not part of the training of engineers.

You're supposed to build things that are robust and elegant, not

be slavishly attentive to individual users like some kind of

salesperson. Ironically, part of the reason engineering is

traditionally averse to handholding is that its traditions date

from a time when engineers were less powerful — when they were

only in charge of their narrow domain of building things, rather

than running the whole show. You can be ornery when you're Scotty,

but not when you're Kirk.Another reason founders don't focus enough on individual customers

is that they worry it won't scale. But when founders of larval

startups worry about this, I point out that in their current state

they have nothing to lose. Maybe if they go out of their way to

make existing users super happy, they'll one day have too many to

do so much for. That would be a great problem to have. See if you

can make it happen. And incidentally, when it does, you'll find

that delighting customers scales better than you expected. Partly

because you can usually find ways to make anything scale more than

you would have predicted, and partly because delighting customers

will by then have permeated your culture.I have never once seen a startup lured down a blind alley by trying

too hard to make their initial users happy.But perhaps the biggest thing preventing founders from realizing

how attentive they could be to their users is that they've never

experienced such attention themselves. Their standards for customer

service have been set by the companies they've been customers of,

which are mostly big ones. Tim Cook doesn't send you a hand-written

note after you buy a laptop. He can't. But you can. That's one

advantage of being small: you can provide a level of service no big

company can.

[6]Once you realize that existing conventions are not the upper bound

on user experience, it's interesting in a very pleasant way to think

about how far you could go to delight your users.ExperienceI was trying to think of a phrase to convey how extreme your attention

to users should be, and I realized Steve Jobs had already done it:

insanely great. Steve wasn't just using "insanely" as a synonym

for "very." He meant it more literally — that one should focus

on quality of execution to a degree that in everyday life would be

considered pathological.All the most successful startups we've funded have, and that probably

doesn't surprise would-be founders. What novice founders don't get

is what insanely great translates to in a larval startup. When

Steve Jobs started using that phrase, Apple was already an established

company. He meant the Mac (and its documentation and even

packaging — such is the nature of obsession) should be insanely

well designed and manufactured. That's not hard for engineers to

grasp. It's just a more extreme version of designing a robust and

elegant product.What founders have a hard time grasping (and Steve himself might

have had a hard time grasping) is what insanely great morphs into

as you roll the time slider back to the first couple months of a

startup's life. It's not the product that should be insanely great,

but the experience of being your user. The product is just one

component of that. For a big company it's necessarily the dominant

one. But you can and should give users an insanely great experience

with an early, incomplete, buggy product, if you make up the

difference with attentiveness.Can, perhaps, but should? Yes. Over-engaging with early users is

not just a permissible technique for getting growth rolling. For

most successful startups it's a necessary part of the feedback loop

that makes the product good. Making a better mousetrap is not an

atomic operation. Even if you start the way most successful startups

have, by building something you yourself need, the first thing you

build is never quite right. And except in domains with big penalties

for making mistakes, it's often better not to aim for perfection

initially. In software, especially, it usually works best to get

something in front of users as soon as it has a quantum of utility,

and then see what they do with it. Perfectionism is often an excuse

for procrastination, and in any case your initial model of users

is always inaccurate, even if you're one of them.

[7]The feedback you get from engaging directly with your earliest users

will be the best you ever get. When you're so big you have to

resort to focus groups, you'll wish you could go over to your users'

homes and offices and watch them use your stuff like you did when

there were only a handful of them.FireSometimes the right unscalable trick is to focus on a deliberately

narrow market. It's like keeping a fire contained at first to get

it really hot before adding more logs.That's what Facebook did. At first it was just for Harvard students.

In that form it only had a potential market of a few thousand people,

but because they felt it was really for them, a critical mass of

them signed up. After Facebook stopped being for Harvard students,

it remained for students at specific colleges for quite a while.

When I interviewed Mark Zuckerberg at Startup School, he said that

while it was a lot of work creating course lists for each school,

doing that made students feel the site was their natural home.Any startup that could be described as a marketplace usually has

to start in a subset of the market, but this can work for other

startups as well. It's always worth asking if there's a subset of

the market in which you can get a critical mass of users quickly.

[8]Most startups that use the contained fire strategy do it unconsciously.

They build something for themselves and their friends, who happen

to be the early adopters, and only realize later that they could

offer it to a broader market. The strategy works just as well if

you do it unconsciously. The biggest danger of not being consciously

aware of this pattern is for those who naively discard part of it.

E.g. if you don't build something for yourself and your friends,

or even if you do, but you come from the corporate world and your

friends are not early adopters, you'll no longer have a perfect

initial market handed to you on a platter.Among companies, the best early adopters are usually other startups.

They're more open to new things both by nature and because, having

just been started, they haven't made all their choices yet. Plus

when they succeed they grow fast, and you with them. It was one

of many unforeseen advantages of the YC model (and specifically of

making YC big) that B2B startups now have an instant market of

hundreds of other startups ready at hand.MerakiFor hardware startups there's a variant of

doing things that don't scale that we call "pulling a Meraki."

Although we didn't fund Meraki, the founders were Robert Morris's

grad students, so we know their history. They got started by doing

something that really doesn't scale: assembling their routers

themselves.Hardware startups face an obstacle that software startups don't.

The minimum order for a factory production run is usually several

hundred thousand dollars. Which can put you in a catch-22: without

a product you can't generate the growth you need to raise the money

to manufacture your product. Back when hardware startups had to

rely on investors for money, you had to be pretty convincing to

overcome this. The arrival of crowdfunding (or more precisely,

preorders) has helped a lot. But even so I'd advise startups to

pull a Meraki initially if they can. That's what Pebble did. The

Pebbles

assembled

the first several hundred watches themselves. If

they hadn't gone through that phase, they probably wouldn't have

sold $10 million worth of watches when they did go on Kickstarter.Like paying excessive attention to early customers, fabricating

things yourself turns out to be valuable for hardware startups.

You can tweak the design faster when you're the factory, and you

learn things you'd never have known otherwise. Eric Migicovsky of

Pebble said one of the things he learned was "how valuable it was to

source good screws." Who knew?ConsultSometimes we advise founders of B2B startups to take over-engagement

to an extreme, and to pick a single user and act as if they were

consultants building something just for that one user. The initial

user serves as the form for your mold; keep tweaking till you fit

their needs perfectly, and you'll usually find you've made something

other users want too. Even if there aren't many of them, there are

probably adjacent territories that have more. As long as you can

find just one user who really needs something and can act on that

need, you've got a toehold in making something people want, and

that's as much as any startup needs initially.

[9]Consulting is the canonical example of work that doesn't scale.

But (like other ways of bestowing one's favors liberally) it's safe

to do it so long as you're not being paid to. That's where companies

cross the line. So long as you're a product company that's merely

being extra attentive to a customer, they're very grateful even if

you don't solve all their problems. But when they start paying you

specifically for that attentiveness — when they start paying

you by the hour — they expect you to do everything.Another consulting-like technique for recruiting initially lukewarm

users is to use your software yourselves on their behalf. We

did that at Viaweb. When we approached merchants asking if they

wanted to use our software to make online stores, some said no, but

they'd let us make one for them. Since we would do anything to get

users, we did. We felt pretty lame at the time. Instead of

organizing big strategic e-commerce partnerships, we were trying

to sell luggage and pens and men's shirts. But in retrospect it

was exactly the right thing to do, because it taught us how it would

feel to merchants to use our software. Sometimes the feedback loop

was near instantaneous: in the middle of building some merchant's

site I'd find I needed a feature we didn't have, so I'd spend a

couple hours implementing it and then resume building the site.ManualThere's a more extreme variant where you don't just use your software,

but are your software. When you only have a small number of users,

you can sometimes get away with doing by hand things that you plan

to automate later. This lets you launch faster, and when you do

finally automate yourself out of the loop, you'll know exactly what

to build because you'll have muscle memory from doing it yourself.When manual components look to the user like software, this technique

starts to have aspects of a practical joke. For example, the way

Stripe delivered "instant" merchant accounts to its first users was

that the founders manually signed them up for traditional merchant

accounts behind the scenes.Some startups could be entirely manual at first. If you can find

someone with a problem that needs solving and you can solve it

manually, go ahead and do that for as long as you can, and then

gradually automate the bottlenecks. It would be a little frightening

to be solving users' problems in a way that wasn't yet automatic,

but less frightening than the far more common case of having something

automatic that doesn't yet solve anyone's problems.BigI should mention one sort of initial tactic that usually doesn't

work: the Big Launch. I occasionally meet founders who seem to

believe startups are projectiles rather than powered aircraft, and

that they'll make it big if and only if they're launched with

sufficient initial velocity. They want to launch simultaneously

in 8 different publications, with embargoes. And on a tuesday, of

course, since they read somewhere that's the optimum day to launch

something.It's easy to see how little launches matter. Think of some successful

startups. How many of their launches do you remember?

All you need from a launch is some initial core of users. How well

you're doing a few months later will depend more on how happy you

made those users than how many there were of them.

[10]So why do founders think launches matter? A combination of solipsism

and laziness. They think what they're building is so great that

everyone who hears about it will immediately sign up. Plus it would

be so much less work if you could get users merely by broadcasting

your existence, rather than recruiting them one at a time. But

even if what you're building really is great, getting users will

always be a gradual process — partly because great things

are usually also novel, but mainly because users have other things

to think about.Partnerships too usually don't work. They don't work for startups

in general, but they especially don't work as a way to get growth

started. It's a common mistake among inexperienced founders to

believe that a partnership with a big company will be their big

break. Six months later they're all saying the same thing: that

was way more work than we expected, and we ended up getting practically

nothing out of it.

[11]It's not enough just to do something extraordinary initially. You

have to make an extraordinary effort initially. Any strategy

that omits the effort — whether it's expecting a big launch to

get you users, or a big partner — is ipso facto suspect.VectorThe need to do something unscalably laborious to get started is so

nearly universal that it might be a good idea to stop thinking of

startup ideas as scalars. Instead we should try thinking of them

as pairs of what you're going to build, plus the unscalable thing(s)

you're going to do initially to get the company going.It could be interesting to start viewing startup ideas this way,

because now that there are two components you can try to be imaginative

about the second as well as the first. But in most cases the second

component will be what it usually is — recruit users manually

and give them an overwhelmingly good experience — and the main

benefit of treating startups as vectors will be to remind founders

they need to work hard in two dimensions.

[12]In the best case, both components of the vector contribute to your

company's DNA: the unscalable things you have to do to get started

are not merely a necessary evil, but change the company permanently

for the better. If you have to be aggressive about user acquisition

when you're small, you'll probably still be aggressive when you're

big. If you have to manufacture your own hardware, or use your

software on users's behalf, you'll learn things you couldn't have

learned otherwise. And most importantly, if you have to work hard

to delight users when you only have a handful of them, you'll keep

doing it when you have a lot.Notes[1]

Actually Emerson never mentioned mousetraps specifically. He

wrote "If a man has good corn or wood, or boards, or pigs, to sell,

or can make better chairs or knives, crucibles or church organs,

than anybody else, you will find a broad hard-beaten road to his

house, though it be in the woods."[2]

Thanks to Sam Altman for suggesting I make this explicit.

And no, you can't avoid doing sales by hiring someone to do it for

you. You have to do sales yourself initially. Later you can hire

a real salesperson to replace you.[3]

The reason this works is that as you get bigger, your size

helps you grow. Patrick Collison wrote "At some point, there was

a very noticeable change in how Stripe felt. It tipped from being

this boulder we had to push to being a train car that in fact had

its own momentum."[4]

One of the more subtle ways in which YC can help founders

is by calibrating their ambitions, because we know exactly how a

lot of successful startups looked when they were just getting

started.[5]

If you're building something for which you can't easily get

a small set of users to observe — e.g. enterprise software — and

in a domain where you have no connections, you'll have to rely on

cold calls and introductions. But should you even be working on

such an idea?[6]

Garry Tan pointed out an interesting trap founders fall into

in the beginning. They want so much to seem big that they imitate

even the flaws of big companies, like indifference to individual

users. This seems to them more "professional." Actually it's

better to embrace the fact that you're small and use whatever

advantages that brings.[7]

Your user model almost couldn't be perfectly accurate, because

users' needs often change in response to what you build for them.

Build them a microcomputer, and suddenly they need to run spreadsheets

on it, because the arrival of your new microcomputer causes someone

to invent the spreadsheet.[8]

If you have to choose between the subset that will sign up

quickest and those that will pay the most, it's usually best to

pick the former, because those are probably the early adopters.

They'll have a better influence on your product, and they won't

make you expend as much effort on sales. And though they have less

money, you don't need that much to maintain your target growth rate

early on.[9]

Yes, I can imagine cases where you could end up making

something that was really only useful for one user. But those are

usually obvious, even to inexperienced founders. So if it's not

obvious you'd be making something for a market of one, don't worry

about that danger.[10]

There may even be an inverse correlation between launch

magnitude and success. The only launches I remember are famous

flops like the Segway and Google Wave. Wave is a particularly

alarming example, because I think it was actually a great idea that

was killed partly by its overdone launch.[11]

Google grew big on the back of Yahoo, but that wasn't a

partnership. Yahoo was their customer.[12]

It will also remind founders that an idea where the second

component is empty — an idea where there is nothing you can do

to get going, e.g. because you have no way to find users to recruit

manually — is probably a bad idea, at least for those founders.Thanks to Sam Altman, Paul Buchheit, Patrick Collison, Kevin

Hale, Steven Levy, Jessica Livingston, Geoff Ralston, and Garry Tan for reading

drafts of this.Japanese TranslationRussian TranslationFrench TranslationArabic TranslationItalian Translation

Startup Investing Trends

June 2013(This talk was written for an audience of investors.)Y Combinator has now funded 564 startups including the current

batch, which has 53. The total valuation of the 287 that have

valuations (either by raising an equity round, getting acquired,

or dying) is about $11.7 billion, and the 511 prior to the current

batch have collectively raised about $1.7 billion.

[1]As usual those numbers are dominated by a few big winners. The top

10 startups account for 8.6 of that 11.7 billion. But there is a

peloton of younger startups behind them. There are about 40 more

that have a shot at being really big.Things got a little out of hand last summer when we had 84 companies

in the batch, so we tightened up our filter to decrease the batch

size.

[2]

Several journalists have tried to interpret that as

evidence for some macro story they were telling, but the reason had

nothing to do with any external trend. The reason was that we

discovered we were using an n² algorithm, and we needed to buy

time to fix it. Fortunately we've come up with several techniques

for sharding YC, and the problem now seems to be fixed. With a new

more scaleable model and only 53 companies, the current batch feels

like a walk in the park. I'd guess we can grow another 2 or 3x

before hitting the next bottleneck.

[3]One consequence of funding such a large number of startups is that

we see trends early. And since fundraising is one of the main

things we help startups with, we're in a good position to notice

trends in investing.I'm going to take a shot at describing where these trends are

leading. Let's start with the most basic question: will the future

be better or worse than the past? Will investors, in the aggregate,

make more money or less?I think more. There are multiple forces at work, some of which

will decrease returns, and some of which will increase them. I

can't predict for sure which forces will prevail, but I'll describe

them and you can decide for yourself.There are two big forces driving change in startup funding: it's

becoming cheaper to start a startup, and startups are becoming a

more normal thing to do.When I graduated from college in 1986, there were essentially two

options: get a job or go to grad school. Now there's a third: start

your own company.

That's a big change. In principle it was possible to start your

own company in 1986 too, but it didn't seem like a real possibility.

It seemed possible to start a consulting company, or a niche product

company, but it didn't seem possible to start a company that would

become big.

[4]That kind of change, from 2 paths to 3, is the sort of big social

shift that only happens once every few generations. I think we're

still at the beginning of this one. It's hard to predict how big

a deal it will be. As big a deal as the Industrial Revolution?

Maybe. Probably not. But it will be a big enough deal that it

takes almost everyone by surprise, because those big social shifts

always do.One thing we can say for sure is that there will be a lot more

startups. The monolithic, hierarchical companies of the mid 20th

century are being replaced by networks

of smaller companies. This process is not just something happening

now in Silicon Valley. It started decades ago, and it's happening

as far afield as the car industry. It has a long way to run.

[5]

The other big driver of change is that startups are becoming cheaper

to start. And in fact the two forces are related: the decreasing

cost of starting a startup is one of the reasons startups are

becoming a more normal thing to do.The fact that startups need less money means founders will increasingly

have the upper hand over investors. You still need just as much

of their energy and imagination, but they don't need as much of

your money. Because founders have the upper hand, they'll retain

an increasingly large share of the stock in, and control of, their

companies. Which means investors will get less stock and less

control.Does that mean investors will make less money? Not necessarily,

because there will be more good startups. The total amount of

desirable startup stock available to investors will probably increase,

because the number of desirable startups will probably grow faster

than the percentage they sell to investors shrinks.There's a rule of thumb in the VC business that there are about 15

companies a year that will be really successful. Although a lot

of investors unconsciously treat this number as if it were some

sort of cosmological constant, I'm certain it isn't. There are

probably limits on the rate at which technology can develop, but

that's not the limiting factor now. If it were, each successful

startup would be founded the month it became possible, and that is

not the case. Right now the limiting factor on the number of big

hits is the number of sufficiently good founders starting companies,

and that number can and will increase. There are still a lot of

people who'd make great founders who never end up starting a company.

You can see that from how randomly some of the most successful

startups got started. So many of the biggest startups almost didn't

happen that there must be a lot of equally good startups that

actually didn't happen.There might be 10x or even 50x more good founders out there. As

more of them go ahead and start startups, those 15 big hits a year

could easily become 50 or even 100.

[6]What about returns, though? Are we heading for a world in which

returns will be pinched by increasingly high valuations? I think

the top firms will actually make more money than they have in the

past. High returns don't come from investing at low valuations.

They come from investing in the companies that do really well. So

if there are more of those to be had each year, the best pickers

should have more hits.This means there should be more variability in the VC business.

The firms that can recognize and attract the best startups will do

even better, because there will be more of them to recognize and

attract. Whereas the bad firms will get the leftovers, as they do

now, and yet pay a higher price for them.Nor do I think it will be a problem that founders keep control of

their companies for longer. The empirical evidence on that is

already clear: investors make more money as founders' bitches than

their bosses. Though somewhat humiliating, this is actually good

news for investors, because it takes less time to serve founders

than to micromanage them.What about angels? I think there is a lot of opportunity there.

It used to suck to be an angel investor. You couldn't get access

to the best deals, unless you got lucky like Andy Bechtolsheim, and

when you did invest in a startup, VCs might try to strip you of

your stock when they arrived later. Now an angel can go to something

like Demo Day or AngelList and have access to the same deals VCs

do. And the days when VCs could wash angels out of the cap table

are long gone.I think one of the biggest unexploited opportunities in startup

investing right now is angel-sized investments made quickly. Few

investors understand the cost that raising money from them imposes

on startups. When the company consists only of the founders,

everything grinds to a halt during fundraising, which can easily

take 6 weeks. The current high cost of fundraising means there is

room for low-cost investors to undercut the rest. And in this

context, low-cost means deciding quickly. If there were a reputable

investor who invested $100k on good terms and promised to decide

yes or no within 24 hours, they'd get access to almost all the best

deals, because every good startup would approach them first. It

would be up to them to pick, because every bad startup would approach

them first too, but at least they'd see everything. Whereas if an

investor is notorious for taking a long time to make up their mind

or negotiating a lot about valuation, founders will save them for

last. And in the case of the most promising startups, which tend

to have an easy time raising money, last can easily become never.Will the number of big hits grow linearly with the total number of

new startups? Probably not, for two reasons. One is that the

scariness of starting a startup in the old days was a pretty effective

filter. Now that the cost of failing is becoming lower, we should

expect founders to do it more. That's not a bad thing. It's common

in technology for an innovation that decreases the cost of failure

to increase the number of failures and yet leave you net ahead.The other reason the number of big hits won't grow proportionately

to the number of startups is that there will start to be an increasing

number of idea clashes. Although the finiteness of the number of

good ideas is not the reason there are only 15 big hits a year, the

number has to be finite, and the more startups there are, the more

we'll see multiple companies doing the same thing at the same time.

It will be interesting, in a bad way, if idea clashes become a lot

more common.

[7]Mostly because of the increasing number of early failures, the startup

business of the future won't simply be the same shape, scaled up.

What used to be an obelisk will become a pyramid. It will be a

little wider at the top, but a lot wider at the bottom.What does that mean for investors? One thing it means is that there

will be more opportunities for investors at the earliest stage,

because that's where the volume of our imaginary solid is growing

fastest. Imagine the obelisk of investors that corresponds to

the obelisk of startups. As it widens out into a pyramid to match

the startup pyramid, all the contents are adhering to the top,

leaving a vacuum at the bottom.That opportunity for investors mostly means an opportunity for new

investors, because the degree of risk an existing investor or firm

is comfortable taking is one of the hardest things for them to

change. Different types of investors are adapted to different

degrees of risk, but each has its specific degree of risk deeply

imprinted on it, not just in the procedures they follow but in the

personalities of the people who work there.I think the biggest danger for VCs, and also the biggest opportunity,

is at the series A stage. Or rather, what used to be the series A

stage before series As turned into de facto series B rounds.Right now, VCs often knowingly invest too much money at the series

A stage. They do it because they feel they need to get a big chunk

of each series A company to compensate for the opportunity cost of

the board seat it consumes. Which means when there is a lot of

competition for a deal, the number that moves is the valuation (and

thus amount invested) rather than the percentage of the company

being sold. Which means, especially in the case of more promising

startups, that series A investors often make companies take more

money than they want.Some VCs lie and claim the company really needs that much. Others

are more candid, and admit their financial models require them to

own a certain percentage of each company. But we all know the

amounts being raised in series A rounds are not determined by asking

what would be best for the companies. They're determined by VCs

starting from the amount of the company they want to own, and the

market setting the valuation and thus the amount invested.Like a lot of bad things, this didn't happen intentionally. The

VC business backed into it as their initial assumptions gradually

became obsolete. The traditions and financial models of the VC

business were established when founders needed investors more. In

those days it was natural for founders to sell VCs a big chunk of

their company in the series A round. Now founders would prefer to

sell less, and VCs are digging in their heels because they're not

sure if they can make money buying less than 20% of each series A

company.The reason I describe this as a danger is that series A investors

are increasingly at odds with the startups they supposedly serve,

and that tends to come back to bite you eventually. The reason I

describe it as an opportunity is that there is now a lot of potential

energy built up, as the market has moved away from VCs' traditional

business model. Which means the first VC to break ranks and start

to do series A rounds for as much equity as founders want to sell

(and with no "option pool" that comes only from the founders' shares)

stands to reap huge benefits.What will happen to the VC business when that happens? Hell if I

know. But I bet that particular firm will end up ahead. If one

top-tier VC firm started to do series A rounds that started from

the amount the company needed to raise and let the percentage

acquired vary with the market, instead of the other way around,

they'd instantly get almost all the best startups. And that's where

the money is.You can't fight market forces forever. Over the last decade we've

seen the percentage of the company sold in series A rounds creep

inexorably downward. 40% used to be common. Now VCs are fighting

to hold the line at 20%. But I am daily waiting for the line to

collapse. It's going to happen. You may as well anticipate it,

and look bold.Who knows, maybe VCs will make more money by doing the right thing.

It wouldn't be the first time that happened. Venture capital is a

business where occasional big successes generate hundredfold returns.

How much confidence can you really have in financial models for

something like that anyway? The

big successes only have to get a tiny bit less occasional to

compensate for a 2x decrease in the stock sold in series A rounds.If you want to find new opportunities for investing, look for things

founders complain about. Founders are your customers, and the

things they complain about are unsatisfied demand. I've given two

examples of things founders complain about most—investors who

take too long to make up their minds, and excessive dilution in

series A rounds—so those are good places to look now. But

the more general recipe is: do something founders want.

Notes[1]

I realize revenue and not fundraising is the proper test of

success for a startup. The reason we quote statistics about

fundraising is because those are the numbers we have. We couldn't

talk meaningfully about revenues without including the numbers from

the most successful startups, and we don't have those. We often

discuss revenue growth with the earlier stage startups, because

that's how we gauge their progress, but when companies reach a

certain size it gets presumptuous for a seed investor to do that.In any case, companies' market caps do eventually become a function

of revenues, and post-money valuations of funding rounds are at

least guesses by pros about where those market caps will end up.The reason only 287 have valuations is that the rest have mostly

raised money on convertible notes, and although convertible notes

often have valuation caps, a valuation cap is merely an upper bound

on a valuation.[2]

We didn't try to accept a particular number. We have no way

of doing that even if we wanted to. We just tried to be significantly

pickier.[3]

Though you never know with bottlenecks, I'm guessing the next

one will be coordinating efforts among partners.[4]

I realize starting a company doesn't have to mean starting a

startup. There will be lots of people starting normal companies

too. But that's not relevant to an audience of investors.Geoff Ralston reports that in Silicon Valley it seemed thinkable

to start a startup in the mid 1980s. It would have started there.

But I know it didn't to undergraduates on the East Coast.[5]

This trend is one of the main causes of the increase in

economic inequality in the US since the mid twentieth century. The

person who would in 1950 have been the general manager of the x

division of Megacorp is now the founder of the x company, and owns

significant equity in it.[6]

If Congress passes the founder

visa in a non-broken form, that alone could in principle get

us up to 20x, since 95% of the world's population lives outside the

US.[7]

If idea clashes got bad enough, it could change what it means

to be a startup. We currently advise startups mostly to ignore

competitors. We tell them startups are competitive like running,

not like soccer; you don't have to go and steal the ball away from

the other team. But if idea clashes became common enough, maybe

you'd start to have to. That would be unfortunate.Thanks to Sam Altman, Paul Buchheit, Dalton Caldwell,

Patrick Collison, Jessica

Livingston, Andrew Mason, Geoff Ralston, and Garry Tan for reading

drafts of this.

How to Get Startup Ideas

Want to start a startup? Get funded by

Y Combinator.

November 2012The way to get startup ideas is not to try to think of startup

ideas. It's to look for problems, preferably problems you have

yourself.The very best startup ideas tend to have three things in common:

they're something the founders themselves want, that they themselves

can build, and that few others realize are worth doing. Microsoft,

Apple, Yahoo, Google, and Facebook all began this way.

ProblemsWhy is it so important to work on a problem you have? Among other

things, it ensures the problem really exists. It sounds obvious

to say you should only work on problems that exist. And yet by far

the most common mistake startups make is to solve problems no one

has.I made it myself. In 1995 I started a company to put art galleries

online. But galleries didn't want to be online. It's not how the

art business works. So why did I spend 6 months working on this

stupid idea? Because I didn't pay attention to users. I invented

a model of the world that didn't correspond to reality, and worked

from that. I didn't notice my model was wrong until I tried

to convince users to pay for what we'd built. Even then I took

embarrassingly long to catch on. I was attached to my model of the

world, and I'd spent a lot of time on the software. They had to

want it!Why do so many founders build things no one wants? Because they

begin by trying to think of startup ideas. That m.o. is doubly

dangerous: it doesn't merely yield few good ideas; it yields bad

ideas that sound plausible enough to fool you into working on them.At YC we call these "made-up" or "sitcom" startup ideas. Imagine

one of the characters on a TV show was starting a startup. The

writers would have to invent something for it to do. But coming

up with good startup ideas is hard. It's not something you can do

for the asking. So (unless they got amazingly lucky) the writers

would come up with an idea that sounded plausible, but was actually

bad.For example, a social network for pet owners. It doesn't sound

obviously mistaken. Millions of people have pets. Often they care

a lot about their pets and spend a lot of money on them. Surely

many of these people would like a site where they could talk to

other pet owners. Not all of them perhaps, but if just 2 or 3

percent were regular visitors, you could have millions of users.

You could serve them targeted offers, and maybe charge for premium

features.

[1]The danger of an idea like this is that when you run it by your

friends with pets, they don't say "I would never use this." They

say "Yeah, maybe I could see using something like that." Even when

the startup launches, it will sound plausible to a lot of people.

They don't want to use it themselves, at least not right now, but

they could imagine other people wanting it. Sum that reaction

across the entire population, and you have zero users.

[2]

WellWhen a startup launches, there have to be at least some users who

really need what they're making — not just people who could see

themselves using it one day, but who want it urgently. Usually

this initial group of users is small, for the simple reason that

if there were something that large numbers of people urgently needed

and that could be built with the amount of effort a startup usually

puts into a version one, it would probably already exist. Which

means you have to compromise on one dimension: you can either build

something a large number of people want a small amount, or something

a small number of people want a large amount. Choose the latter.

Not all ideas of that type are good startup ideas, but nearly all

good startup ideas are of that type.Imagine a graph whose x axis represents all the people who might

want what you're making and whose y axis represents how much they

want it. If you invert the scale on the y axis, you can envision

companies as holes. Google is an immense crater: hundreds of

millions of people use it, and they need it a lot. A startup just

starting out can't expect to excavate that much volume. So you

have two choices about the shape of hole you start with. You can

either dig a hole that's broad but shallow, or one that's narrow

and deep, like a well.Made-up startup ideas are usually of the first type. Lots of people

are mildly interested in a social network for pet owners.Nearly all good startup ideas are of the second type. Microsoft

was a well when they made Altair Basic. There were only a couple

thousand Altair owners, but without this software they were programming

in machine language. Thirty years later Facebook had the same

shape. Their first site was exclusively for Harvard students, of

which there are only a few thousand, but those few thousand users

wanted it a lot.When you have an idea for a startup, ask yourself: who wants this

right now? Who wants this so much that they'll use it even when

it's a crappy version one made by a two-person startup they've never

heard of? If you can't answer that, the idea is probably bad.

[3]You don't need the narrowness of the well per se. It's depth you

need; you get narrowness as a byproduct of optimizing for depth

(and speed). But you almost always do get it. In practice the

link between depth and narrowness is so strong that it's a good

sign when you know that an idea will appeal strongly to a specific

group or type of user.But while demand shaped like a well is almost a necessary condition

for a good startup idea, it's not a sufficient one. If Mark

Zuckerberg had built something that could only ever have appealed

to Harvard students, it would not have been a good startup idea.

Facebook was a good idea because it started with a small market

there was a fast path out of. Colleges are similar enough that if

you build a facebook that works at Harvard, it will work at any

college. So you spread rapidly through all the colleges. Once you

have all the college students, you get everyone else simply by

letting them in.Similarly for Microsoft: Basic for the Altair; Basic for other

machines; other languages besides Basic; operating systems;

applications; IPO.

SelfHow do you tell whether there's a path out of an idea? How do you

tell whether something is the germ of a giant company, or just a

niche product? Often you can't. The founders of Airbnb didn't

realize at first how big a market they were tapping. Initially

they had a much narrower idea. They were going to let hosts rent

out space on their floors during conventions. They didn't foresee

the expansion of this idea; it forced itself upon them gradually.

All they knew at first is that they were onto something. That's

probably as much as Bill Gates or Mark Zuckerberg knew at first.Occasionally it's obvious from the beginning when there's a path

out of the initial niche. And sometimes I can see a path that's

not immediately obvious; that's one of our specialties at YC. But

there are limits to how well this can be done, no matter how much

experience you have. The most important thing to understand about

paths out of the initial idea is the meta-fact that these are hard

to see.So if you can't predict whether there's a path out of an idea, how

do you choose between ideas? The truth is disappointing but

interesting: if you're the right sort of person, you have the right

sort of hunches. If you're at the leading edge of a field that's

changing fast, when you have a hunch that something is worth doing,

you're more likely to be right.In Zen and the Art of Motorcycle Maintenance, Robert Pirsig says:

You want to know how to paint a perfect painting? It's easy. Make

yourself perfect and then just paint naturally.

I've wondered about that passage since I read it in high school.

I'm not sure how useful his advice is for painting specifically,

but it fits this situation well. Empirically, the way to have good

startup ideas is to become the sort of person who has them.Being at the leading edge of a field doesn't mean you have to be

one of the people pushing it forward. You can also be at the leading

edge as a user. It was not so much because he was a programmer

that Facebook seemed a good idea to Mark Zuckerberg as because he

used computers so much. If you'd asked most 40 year olds in 2004

whether they'd like to publish their lives semi-publicly on the

Internet, they'd have been horrified at the idea. But Mark already

lived online; to him it seemed natural.Paul Buchheit says that people at the leading edge of a rapidly

changing field "live in the future." Combine that with Pirsig and

you get:

Live in the future, then build what's missing.

That describes the way many if not most of the biggest startups got

started. Neither Apple nor Yahoo nor Google nor Facebook were even

supposed to be companies at first. They grew out of things their

founders built because there seemed a gap in the world.If you look at the way successful founders have had their ideas,

it's generally the result of some external stimulus hitting a

prepared mind. Bill Gates and Paul Allen hear about the Altair and

think "I bet we could write a Basic interpreter for it." Drew Houston

realizes he's forgotten his USB stick and thinks "I really need to

make my files live online." Lots of people heard about the Altair.

Lots forgot USB sticks. The reason those stimuli caused those

founders to start companies was that their experiences had prepared

them to notice the opportunities they represented.The verb you want to be using with respect to startup ideas is not

"think up" but "notice." At YC we call ideas that grow naturally

out of the founders' own experiences "organic" startup ideas. The

most successful startups almost all begin this way.That may not have been what you wanted to hear. You may have

expected recipes for coming up with startup ideas, and instead I'm

telling you that the key is to have a mind that's prepared in the

right way. But disappointing though it may be, this is the truth.

And it is a recipe of a sort, just one that in the worst case takes

a year rather than a weekend.If you're not at the leading edge of some rapidly changing field,

you can get to one. For example, anyone reasonably smart can

probably get to an edge of programming (e.g. building mobile apps)

in a year. Since a successful startup will consume at least 3-5

years of your life, a year's preparation would be a reasonable

investment. Especially if you're also looking for a cofounder.

[4]You don't have to learn programming to be at the leading edge of a

domain that's changing fast. Other domains change fast. But while

learning to hack is not necessary, it is for the forseeable future

sufficient. As Marc Andreessen put it, software is eating the world,

and this trend has decades left to run.Knowing how to hack also means that when you have ideas, you'll be

able to implement them. That's not absolutely necessary (Jeff Bezos

couldn't) but it's an advantage. It's a big advantage, when you're

considering an idea like putting a college facebook online, if

instead of merely thinking "That's an interesting idea," you can

think instead "That's an interesting idea. I'll try building an

initial version tonight." It's even better when you're both a

programmer and the target user, because then the cycle of generating

new versions and testing them on users can happen inside one head.

NoticingOnce you're living in the future in some respect, the way to notice

startup ideas is to look for things that seem to be missing. If

you're really at the leading edge of a rapidly changing field, there

will be things that are obviously missing. What won't be obvious

is that they're startup ideas. So if you want to find startup

ideas, don't merely turn on the filter "What's missing?" Also turn

off every other filter, particularly "Could this be a big company?"

There's plenty of time to apply that test later. But if you're

thinking about that initially, it may not only filter out lots

of good ideas, but also cause you to focus on bad ones.Most things that are missing will take some time to see. You almost

have to trick yourself into seeing the ideas around you.But you know the ideas are out there. This is not one of those

problems where there might not be an answer. It's impossibly

unlikely that this is the exact moment when technological progress

stops. You can be sure people are going to build things in the

next few years that will make you think "What did I do before x?"And when these problems get solved, they will probably seem flamingly

obvious in retrospect. What you need to do is turn off the filters

that usually prevent you from seeing them. The most powerful is

simply taking the current state of the world for granted. Even the

most radically open-minded of us mostly do that. You couldn't get

from your bed to the front door if you stopped to question everything.But if you're looking for startup ideas you can sacrifice some of

the efficiency of taking the status quo for granted and start to

question things. Why is your inbox overflowing? Because you get

a lot of email, or because it's hard to get email out of your inbox?

Why do you get so much email? What problems are people trying to

solve by sending you email? Are there better ways to solve them?

And why is it hard to get emails out of your inbox? Why do you

keep emails around after you've read them? Is an inbox the optimal

tool for that?Pay particular attention to things that chafe you. The advantage

of taking the status quo for granted is not just that it makes life

(locally) more efficient, but also that it makes life more tolerable.

If you knew about all the things we'll get in the next 50 years but

don't have yet, you'd find present day life pretty constraining,

just as someone from the present would if they were sent back 50

years in a time machine. When something annoys you, it could be

because you're living in the future.When you find the right sort of problem, you should probably be

able to describe it as obvious, at least to you. When we started

Viaweb, all the online stores were built by hand, by web designers

making individual HTML pages. It was obvious to us as programmers

that these sites would have to be generated by software.

[5]Which means, strangely enough, that coming up with startup ideas

is a question of seeing the obvious. That suggests how weird this

process is: you're trying to see things that are obvious, and yet

that you hadn't seen.Since what you need to do here is loosen up your own mind, it may

be best not to make too much of a direct frontal attack on the

problem — i.e. to sit down and try to think of ideas. The best

plan may be just to keep a background process running, looking for

things that seem to be missing. Work on hard problems, driven

mainly by curiosity, but have a second self watching over your

shoulder, taking note of gaps and anomalies.

[6]Give yourself some time. You have a lot of control over the rate

at which you turn yours into a prepared mind, but you have less

control over the stimuli that spark ideas when they hit it. If

Bill Gates and Paul Allen had constrained themselves to come up

with a startup idea in one month, what if they'd chosen a month

before the Altair appeared? They probably would have worked on a

less promising idea. Drew Houston did work on a less promising

idea before Dropbox: an SAT prep startup. But Dropbox was a much

better idea, both in the absolute sense and also as a match for his

skills.

[7]A good way to trick yourself into noticing ideas is to work on

projects that seem like they'd be cool. If you do that, you'll

naturally tend to build things that are missing. It wouldn't seem

as interesting to build something that already existed.Just as trying to think up startup ideas tends to produce bad ones,

working on things that could be dismissed as "toys" often produces

good ones. When something is described as a toy, that means it has

everything an idea needs except being important. It's cool; users

love it; it just doesn't matter. But if you're living in the future

and you build something cool that users love, it may matter more

than outsiders think. Microcomputers seemed like toys when Apple

and Microsoft started working on them. I'm old enough to remember

that era; the usual term for people with their own microcomputers

was "hobbyists." BackRub seemed like an inconsequential science

project. The Facebook was just a way for undergrads to stalk one

another.At YC we're excited when we meet startups working on things that

we could imagine know-it-alls on forums dismissing as toys. To us

that's positive evidence an idea is good.If you can afford to take a long view (and arguably you can't afford

not to), you can turn "Live in the future and build what's missing"

into something even better:

Live in the future and build what seems interesting.

SchoolThat's what I'd advise college students to do, rather than trying

to learn about "entrepreneurship." "Entrepreneurship" is something

you learn best by doing it. The examples of the most successful

founders make that clear. What you should be spending your time

on in college is ratcheting yourself into the future. College is

an incomparable opportunity to do that. What a waste to sacrifice

an opportunity to solve the hard part of starting a startup — becoming

the sort of person who can have organic startup ideas — by

spending time learning about the easy part. Especially since

you won't even really learn about it, any more than you'd learn

about sex in a class. All you'll learn is the words for things.The clash of domains is a particularly fruitful source of ideas.

If you know a lot about programming and you start learning about

some other field, you'll probably see problems that software could

solve. In fact, you're doubly likely to find good problems in

another domain: (a) the inhabitants of that domain are not as likely

as software people to have already solved their problems with

software, and (b) since you come into the new domain totally ignorant,

you don't even know what the status quo is to take it for granted.So if you're a CS major and you want to start a startup, instead

of taking a class on entrepreneurship you're better off taking a

class on, say, genetics. Or better still, go work for a biotech

company. CS majors normally get summer jobs at computer hardware

or software companies. But if you want to find startup ideas, you

might do better to get a summer job in some unrelated field.

[8]Or don't take any extra classes, and just build things. It's no

coincidence that Microsoft and Facebook both got started in January.

At Harvard that is (or was) Reading Period, when students have no

classes to attend because they're supposed to be studying for finals.

[9]But don't feel like you have to build things that will become startups. That's

premature optimization. Just build things. Preferably with other

students. It's not just the classes that make a university such a

good place to crank oneself into the future. You're also surrounded

by other people trying to do the same thing. If you work together

with them on projects, you'll end up producing not just organic

ideas, but organic ideas with organic founding teams — and that,

empirically, is the best combination.Beware of research. If an undergrad writes something all his friends

start using, it's quite likely to represent a good startup idea.

Whereas a PhD dissertation is extremely unlikely to. For some

reason, the more a project has to count as research, the less likely

it is to be something that could be turned into a startup.

[10]

I think the reason is that the subset of ideas that count as research

is so narrow that it's unlikely that a project that satisfied that

constraint would also satisfy the orthogonal constraint of solving

users' problems. Whereas when students (or professors) build

something as a side-project, they automatically gravitate toward

solving users' problems — perhaps even with an additional energy

that comes from being freed from the constraints of research.

CompetitionBecause a good idea should seem obvious, when you have one you'll

tend to feel that you're late. Don't let that deter you. Worrying

that you're late is one of the signs of a good idea. Ten minutes

of searching the web will usually settle the question. Even if you

find someone else working on the same thing, you're probably not

too late. It's exceptionally rare for startups to be killed by

competitors — so rare that you can almost discount the possibility.

So unless you discover a competitor with the sort of lock-in that

would prevent users from choosing you, don't discard the idea.If you're uncertain, ask users. The question of whether you're too

late is subsumed by the question of whether anyone urgently needs

what you plan to make. If you have something that no competitor

does and that some subset of users urgently need, you have a

beachhead.

[11]The question then is whether that beachhead is big enough. Or more

importantly, who's in it: if the beachhead consists of people doing

something lots more people will be doing in the future, then it's

probably big enough no matter how small it is. For example, if

you're building something differentiated from competitors by the

fact that it works on phones, but it only works on the newest phones,

that's probably a big enough beachhead.Err on the side of doing things where you'll face competitors.

Inexperienced founders usually give competitors more credit than

they deserve. Whether you succeed depends far more on you than on

your competitors. So better a good idea with competitors than a

bad one without.You don't need to worry about entering a "crowded market" so long

as you have a thesis about what everyone else in it is overlooking.

In fact that's a very promising starting point. Google was that

type of idea. Your thesis has to be more precise than "we're going

to make an x that doesn't suck" though. You have to be able to

phrase it in terms of something the incumbents are overlooking.

Best of all is when you can say that they didn't have the courage

of their convictions, and that your plan is what they'd have done

if they'd followed through on their own insights. Google was that

type of idea too. The search engines that preceded them shied away

from the most radical implications of what they were doing — particularly

that the better a job they did, the faster users would

leave.A crowded market is actually a good sign, because it means both

that there's demand and that none of the existing solutions are

good enough. A startup can't hope to enter a market that's obviously

big and yet in which they have no competitors. So any startup that

succeeds is either going to be entering a market with existing

competitors, but armed with some secret weapon that will get them

all the users (like Google), or entering a market that looks small

but which will turn out to be big (like Microsoft).

[12]

FiltersThere are two more filters you'll need to turn off if you want to

notice startup ideas: the unsexy filter and the schlep filter.Most programmers wish they could start a startup by just writing

some brilliant code, pushing it to a server, and having users pay

them lots of money. They'd prefer not to deal with tedious problems

or get involved in messy ways with the real world. Which is a

reasonable preference, because such things slow you down. But this

preference is so widespread that the space of convenient startup

ideas has been stripped pretty clean. If you let your mind wander

a few blocks down the street to the messy, tedious ideas, you'll

find valuable ones just sitting there waiting to be implemented.The schlep filter is so dangerous that I wrote a separate essay

about the condition it induces, which I called

schlep blindness.

I gave Stripe as an example of a startup that benefited from turning

off this filter, and a pretty striking example it is. Thousands

of programmers were in a position to see this idea; thousands of

programmers knew how painful it was to process payments before

Stripe. But when they looked for startup ideas they didn't see

this one, because unconsciously they shrank from having to deal

with payments. And dealing with payments is a schlep for Stripe,

but not an intolerable one. In fact they might have had net less

pain; because the fear of dealing with payments kept most people

away from this idea, Stripe has had comparatively smooth sailing

in other areas that are sometimes painful, like user acquisition.

They didn't have to try very hard to make themselves heard by users,

because users were desperately waiting for what they were building.The unsexy filter is similar to the schlep filter, except it keeps

you from working on problems you despise rather than ones you fear.

We overcame this one to work on Viaweb. There were interesting

things about the architecture of our software, but we weren't

interested in ecommerce per se. We could see the problem was one

that needed to be solved though.Turning off the schlep filter is more important than turning off

the unsexy filter, because the schlep filter is more likely to be

an illusion. And even to the degree it isn't, it's a worse form

of self-indulgence. Starting a successful startup is going to be

fairly laborious no matter what. Even if the product doesn't entail

a lot of schleps, you'll still have plenty dealing with investors,

hiring and firing people, and so on. So if there's some idea you

think would be cool but you're kept away from by fear of the schleps

involved, don't worry: any sufficiently good idea will have as many.The unsexy filter, while still a source of error, is not as entirely

useless as the schlep filter. If you're at the leading edge of a

field that's changing rapidly, your ideas about what's sexy will

be somewhat correlated with what's valuable in practice. Particularly

as you get older and more experienced. Plus if you find an idea

sexy, you'll work on it more enthusiastically.

[13]

RecipesWhile the best way to discover startup ideas is to become the sort

of person who has them and then build whatever interests you,

sometimes you don't have that luxury. Sometimes you need an idea

now. For example, if you're working on a startup and your initial

idea turns out to be bad.For the rest of this essay I'll talk about tricks for coming up

with startup ideas on demand. Although empirically you're better

off using the organic strategy, you could succeed this way. You

just have to be more disciplined. When you use the organic method,

you don't even notice an idea unless it's evidence that something

is truly missing. But when you make a conscious effort to think

of startup ideas, you have to replace this natural constraint with

self-discipline. You'll see a lot more ideas, most of them bad,

so you need to be able to filter them.One of the biggest dangers of not using the organic method is the

example of the organic method. Organic ideas feel like inspirations.

There are a lot of stories about successful startups that began

when the founders had what seemed a crazy idea but "just knew" it

was promising. When you feel that about an idea you've had while

trying to come up with startup ideas, you're probably mistaken.When searching for ideas, look in areas where you have some expertise.

If you're a database expert, don't build a chat app for teenagers

(unless you're also a teenager). Maybe it's a good idea, but you

can't trust your judgment about that, so ignore it. There have to

be other ideas that involve databases, and whose quality you can

judge. Do you find it hard to come up with good ideas involving

databases? That's because your expertise raises your standards.

Your ideas about chat apps are just as bad, but you're giving

yourself a Dunning-Kruger pass in that domain.The place to start looking for ideas is things you need. There

must be things you need.

[14]One good trick is to ask yourself whether in your previous job you

ever found yourself saying "Why doesn't someone make x? If someone

made x we'd buy it in a second." If you can think of any x people

said that about, you probably have an idea. You know there's demand,

and people don't say that about things that are impossible to build.More generally, try asking yourself whether there's something unusual

about you that makes your needs different from most other people's.

You're probably not the only one. It's especially good if you're

different in a way people will increasingly be.If you're changing ideas, one unusual thing about you is the idea

you'd previously been working on. Did you discover any needs while

working on it? Several well-known startups began this way. Hotmail

began as something its founders wrote to talk about their previous

startup idea while they were working at their day jobs.

[15]A particularly promising way to be unusual is to be young. Some

of the most valuable new ideas take root first among people in their

teens and early twenties. And while young founders are at a

disadvantage in some respects, they're the only ones who really

understand their peers. It would have been very hard for someone

who wasn't a college student to start Facebook. So if you're a

young founder (under 23 say), are there things you and your friends

would like to do that current technology won't let you?The next best thing to an unmet need of your own is an unmet need

of someone else. Try talking to everyone you can about the gaps

they find in the world. What's missing? What would they like to

do that they can't? What's tedious or annoying, particularly in

their work? Let the conversation get general; don't be trying too

hard to find startup ideas. You're just looking for something to

spark a thought. Maybe you'll notice a problem they didn't consciously

realize they had, because you know how to solve it.When you find an unmet need that isn't your own, it may be somewhat

blurry at first. The person who needs something may not know exactly

what they need. In that case I often recommend that founders act

like consultants — that they do what they'd do if they'd been

retained to solve the problems of this one user. People's problems

are similar enough that nearly all the code you write this way will

be reusable, and whatever isn't will be a small price to start out

certain that you've reached the bottom of the well.

[16]One way to ensure you do a good job solving other people's problems

is to make them your own. When Rajat Suri of E la Carte decided

to write software for restaurants, he got a job as a waiter to learn

how restaurants worked. That may seem like taking things to extremes,

but startups are extreme. We love it when founders do such things.In fact, one strategy I recommend to people who need a new idea is

not merely to turn off their schlep and unsexy filters, but to seek

out ideas that are unsexy or involve schleps. Don't try to start

Twitter. Those ideas are so rare that you can't find them by looking

for them. Make something unsexy that people will pay you for.A good trick for bypassing the schlep and to some extent the unsexy

filter is to ask what you wish someone else would build, so that

you could use it. What would you pay for right now?Since startups often garbage-collect broken companies and industries,

it can be a good trick to look for those that are dying, or deserve

to, and try to imagine what kind of company would profit from their

demise. For example, journalism is in free fall at the moment.

But there may still be money to be made from something like journalism.

What sort of company might cause people in the future to say "this

replaced journalism" on some axis?But imagine asking that in the future, not now. When one company

or industry replaces another, it usually comes in from the side.

So don't look for a replacement for x; look for something that

people will later say turned out to be a replacement for x. And

be imaginative about the axis along which the replacement occurs.

Traditional journalism, for example, is a way for readers to get

information and to kill time, a way for writers to make money and

to get attention, and a vehicle for several different types of

advertising. It could be replaced on any of these axes (it has

already started to be on most).When startups consume incumbents, they usually start by serving

some small but important market that the big players ignore. It's

particularly good if there's an admixture of disdain in the big

players' attitude, because that often misleads them. For example,

after Steve Wozniak built the computer that became the Apple I, he

felt obliged to give his then-employer Hewlett-Packard the option

to produce it. Fortunately for him, they turned it down, and one

of the reasons they did was that it used a TV for a monitor, which

seemed intolerably déclassé to a high-end hardware company like HP

was at the time.

[17]Are there groups of

scruffy

but sophisticated users like the early

microcomputer "hobbyists" that are currently being ignored by the

big players? A startup with its sights set on bigger things can

often capture a small market easily by expending an effort that

wouldn't be justified by that market alone.Similarly, since the most successful startups generally ride some

wave bigger than themselves, it could be a good trick to look for

waves and ask how one could benefit from them. The prices of gene

sequencing and 3D printing are both experiencing Moore's Law-like

declines. What new things will we be able to do in the new world

we'll have in a few years? What are we unconsciously ruling out

as impossible that will soon be possible?

OrganicBut talking about looking explicitly for waves makes it clear that

such recipes are plan B for getting startup ideas. Looking for

waves is essentially a way to simulate the organic method. If

you're at the leading edge of some rapidly changing field, you don't

have to look for waves; you are the wave.Finding startup ideas is a subtle business, and that's why most

people who try fail so miserably. It doesn't work well simply to

try to think of startup ideas. If you do that, you get bad ones

that sound dangerously plausible. The best approach is more indirect:

if you have the right sort of background, good startup ideas will

seem obvious to you. But even then, not immediately. It takes

time to come across situations where you notice something missing.

And often these gaps won't seem to be ideas for companies, just

things that would be interesting to build. Which is why it's good

to have the time and the inclination to build things just because

they're interesting.Live in the future and build what seems interesting. Strange as

it sounds, that's the real recipe.

Notes[1]

This form of bad idea has been around as long as the web. It

was common in the 1990s, except then people who had it used to say

they were going to create a portal for x instead of a social network

for x. Structurally the idea is stone soup: you post a sign saying

"this is the place for people interested in x," and all those people

show up and you make money from them. What lures founders into

this sort of idea are statistics about the millions of people who

might be interested in each type of x. What they forget is that

any given person might have 20 affinities by this standard, and no

one is going to visit 20 different communities regularly.[2]

I'm not saying, incidentally, that I know for sure a social

network for pet owners is a bad idea. I know it's a bad idea the

way I know randomly generated DNA would not produce a viable organism.

The set of plausible sounding startup ideas is many times larger

than the set of good ones, and many of the good ones don't even

sound that plausible. So if all you know about a startup idea is

that it sounds plausible, you have to assume it's bad.[3]

More precisely, the users' need has to give them sufficient

activation energy to start using whatever you make, which can vary

a lot. For example, the activation energy for enterprise software

sold through traditional channels is very high, so you'd have to

be a lot better to get users to switch. Whereas the activation

energy required to switch to a new search engine is low. Which in

turn is why search engines are so much better than enterprise

software.[4]

This gets harder as you get older. While the space of ideas

doesn't have dangerous local maxima, the space of careers does.

There are fairly high walls between most of the paths people take

through life, and the older you get, the higher the walls become.[5]

It was also obvious to us that the web was going to be a big

deal. Few non-programmers grasped that in 1995, but the programmers

had seen what GUIs had done for desktop computers.[6]

Maybe it would work to have this second self keep a journal,

and each night to make a brief entry listing the gaps and anomalies

you'd noticed that day. Not startup ideas, just the raw gaps and

anomalies.[7]

Sam Altman points out that taking time to come up with an

idea is not merely a better strategy in an absolute sense, but also

like an undervalued stock in that so few founders do it.There's comparatively little competition for the best ideas, because

few founders are willing to put in the time required to notice them.

Whereas there is a great deal of competition for mediocre ideas,

because when people make up startup ideas, they tend to make up the

same ones.[8]

For the computer hardware and software companies, summer jobs

are the first phase of the recruiting funnel. But if you're good

you can skip the first phase. If you're good you'll have no trouble

getting hired by these companies when you graduate, regardless of

how you spent your summers.[9]

The empirical evidence suggests that if colleges want to help

their students start startups, the best thing they can do is leave

them alone in the right way.[10]

I'm speaking here of IT startups; in biotech things are different.[11]

This is an instance of a more general rule: focus on users,

not competitors. The most important information about competitors

is what you learn via users anyway.[12]

In practice most successful startups have elements of both.

And you can describe each strategy in terms of the other by adjusting

the boundaries of what you call the market. But it's useful to

consider these two ideas separately.[13]

I almost hesitate to raise that point though. Startups are

businesses; the point of a business is to make money; and with that

additional constraint, you can't expect you'll be able to spend all

your time working on what interests you most.[14]

The need has to be a strong one. You can retroactively

describe any made-up idea as something you need. But do you really

need that recipe site or local event aggregator as much as Drew

Houston needed Dropbox, or Brian Chesky and Joe Gebbia needed Airbnb?Quite often at YC I find myself asking founders "Would you use this

thing yourself, if you hadn't written it?" and you'd be surprised

how often the answer is no.[15]

Paul Buchheit points out that trying to sell something bad

can be a source of better ideas:"The best technique I've found for dealing with YC companies that

have bad ideas is to tell them to go sell the product ASAP (before

wasting time building it). Not only do they learn that nobody

wants what they are building, they very often come back with a

real idea that they discovered in the process of trying to sell

the bad idea."[16]

Here's a recipe that might produce the next Facebook, if

you're college students. If you have a connection to one of the

more powerful sororities at your school, approach the queen bees

thereof and offer to be their personal IT consultants, building

anything they could imagine needing in their social lives that

didn't already exist. Anything that got built this way would be

very promising, because such users are not just the most demanding

but also the perfect point to spread from.I have no idea whether this would work.[17]

And the reason it used a TV for a monitor is that Steve Wozniak

started out by solving his own problems. He, like most of his

peers, couldn't afford a monitor.Thanks to Sam Altman, Mike Arrington, Paul Buchheit, John Collison,

Patrick Collison, Garry Tan, and Harj Taggar for reading drafts of

this, and Marc Andreessen, Joe Gebbia, Reid Hoffman, Shel Kaphan,

Mike Moritz and Kevin Systrom for answering my questions about

startup history.Japanese TranslationItalian Translation

The Hardware Renaissance

Want to start a startup? Get funded by

Y Combinator.

October 2012One advantage of Y Combinator's early, broad focus is that we

see trends before most other people. And one of the most conspicuous

trends in the last batch was the large number of hardware startups.

Out of 84 companies, 7 were making hardware. On the whole

they've done better than the companies that weren't.They've faced resistance from investors of course. Investors have

a deep-seated bias against hardware. But investors' opinions are

a trailing indicator. The best founders are better at seeing the

future than the best investors, because the best founders are making

it.There is no one single force driving this trend. Hardware does

well on crowdfunding sites. The spread of tablets makes it

possible to build new things controlled

by and even incorporating

them. Electric motors

have improved.

Wireless connectivity of various types can now be taken for granted.

It's getting more straightforward to get things manufactured.

Arduinos, 3D printing, laser cutters, and more accessible CNC milling are making hardware easier to prototype.

Retailers are less of a bottleneck as customers increasingly buy

online.One question I can answer is why hardware is suddenly cool.

It always was cool.

Physical things are great. They just haven't

been as great a way to start a rapidly growing business

as software. But that rule may not be permanent. It's not even

that old; it only dates from about 1990. Maybe the advantage

of software will turn out to have been temporary. Hackers love to

build hardware, and customers love to buy it. So if the ease of

shipping hardware even approached the ease of shipping software,

we'd see a lot more hardware startups.It wouldn't be the first time something was a bad idea till it

wasn't. And it wouldn't be the first time investors learned that

lesson from founders.So if you want to work on hardware, don't be deterred from doing

it because you worry investors will discriminate against you. And

in particular, don't be deterred from applying to Y Combinator

with a hardware idea, because we're especially interested in hardware

startups.We know there's room for the next Steve Jobs.

But there's almost certainly also room for the first

<Your Name Here>.

Thanks to Sam Altman, Trevor Blackwell, David Cann, Sanjay Dastoor,

Paul Gerhardt, Cameron Robertson, Harj Taggar, and Garry Tan for reading drafts of this.A Hardware Renaissance while Software Eats the World?

Startup = Growth

Want to start a startup? Get funded by

Y Combinator.

September 2012A startup is a company designed to grow fast. Being newly founded

does not in itself make a company a startup. Nor is it necessary

for a startup to work on technology, or take venture funding, or

have some sort of "exit." The only essential thing is growth.

Everything else we associate with startups follows from growth.If you want to start one it's important to understand that. Startups

are so hard that you can't be pointed off to the side and hope to

succeed. You have to know that growth is what you're after. The

good news is, if you get growth, everything else tends to fall into

place. Which means you can use growth like a compass to make almost

every decision you face.

RedwoodsLet's start with a distinction that should be obvious but is often

overlooked: not every newly founded company is a startup. Millions

of companies are started every year in the US. Only a tiny fraction

are startups. Most are service businesses — restaurants, barbershops,

plumbers, and so on. These are not startups, except in a few unusual

cases. A barbershop isn't designed to grow fast. Whereas a search

engine, for example, is.When I say startups are designed to grow fast, I mean it in two

senses. Partly I mean designed in the sense of intended, because

most startups fail. But I also mean startups are different by

nature, in the same way a redwood seedling has a different destiny

from a bean sprout.That difference is why there's a distinct word, "startup," for

companies designed to grow fast. If all companies were essentially

similar, but some through luck or the efforts of their founders

ended up growing very fast, we wouldn't need a separate word. We

could just talk about super-successful companies and less successful

ones. But in fact startups do have a different sort of DNA from

other businesses. Google is not just a barbershop whose founders

were unusually lucky and hard-working. Google was different from

the beginning.To grow rapidly, you need to make something you can sell to a big

market. That's the difference between Google and a barbershop. A

barbershop doesn't scale.For a company to grow really big, it must (a) make something lots

of people want, and (b) reach and serve all those people. Barbershops

are doing fine in the (a) department. Almost everyone needs their

hair cut. The problem for a barbershop, as for any retail

establishment, is (b). A barbershop serves customers in person,

and few will travel far for a haircut. And even if they did, the

barbershop couldn't accomodate them.

[1]Writing software is a great way to solve (b), but you can still end

up constrained in (a). If you write software to teach Tibetan to

Hungarian speakers, you'll be able to reach most of the people who

want it, but there won't be many of them. If you make software

to teach English to Chinese speakers, however, you're in startup

territory.Most businesses are tightly constrained in (a) or (b). The distinctive

feature of successful startups is that they're not.

IdeasIt might seem that it would always be better to start a startup

than an ordinary business. If you're going to start a company, why

not start the type with the most potential? The catch is that this

is a (fairly) efficient market. If you write software to teach

Tibetan to Hungarians, you won't have much competition. If you

write software to teach English to Chinese speakers, you'll face

ferocious competition, precisely because that's such a larger prize.

[2]The constraints that limit ordinary companies also protect them.

That's the tradeoff. If you start a barbershop, you only have to

compete with other local barbers. If you start a search engine you

have to compete with the whole world.The most important thing that the constraints on a normal business

protect it from is not competition, however, but the difficulty of

coming up with new ideas. If you open a bar in a particular

neighborhood, as well as limiting your potential and protecting you

from competitors, that geographic constraint also helps define your

company. Bar + neighborhood is a sufficient idea for a small

business. Similarly for companies constrained in (a). Your niche

both protects and defines you.Whereas if you want to start a startup, you're probably going to

have to think of something fairly novel. A startup has to make

something it can deliver to a large market, and ideas of that type

are so valuable that all the obvious ones are already taken.That space of ideas has been so thoroughly picked over that a startup

generally has to work on something everyone else has overlooked.

I was going to write that one has to make a conscious effort to

find ideas everyone else has overlooked. But that's not how most

startups get started. Usually successful startups happen because

the founders are sufficiently different from other people that ideas

few others can see seem obvious to them. Perhaps later they step

back and notice they've found an idea in everyone else's blind spot,

and from that point make a deliberate effort to stay there.

[3]

But at the moment when successful startups get started, much of the

innovation is unconscious.What's different about successful founders is that they can see

different problems. It's a particularly good combination both to

be good at technology and to face problems that can be solved by

it, because technology changes so rapidly that formerly bad ideas

often become good without anyone noticing. Steve Wozniak's problem

was that he wanted his own computer. That was an unusual problem

to have in 1975. But technological change was about to make it a

much more common one. Because he not only wanted a computer but

knew how to build them, Wozniak was able to make himself one. And

the problem he solved for himself became one that Apple solved for

millions of people in the coming years. But by the time it was

obvious to ordinary people that this was a big market, Apple was

already established.Google has similar origins. Larry Page and Sergey Brin wanted to

search the web. But unlike most people they had the technical

expertise both to notice that existing search engines were not as

good as they could be, and to know how to improve them. Over the

next few years their problem became everyone's problem, as the web

grew to a size where you didn't have to be a picky search expert

to notice the old algorithms weren't good enough. But as happened

with Apple, by the time everyone else realized how important search

was, Google was entrenched.That's one connection between startup ideas and technology. Rapid

change in one area uncovers big, soluble problems in other areas.

Sometimes the changes are advances, and what they change is solubility.

That was the kind of change that yielded Apple; advances in chip

technology finally let Steve Wozniak design a computer he could

afford. But in Google's case the most important change was the

growth of the web. What changed there was not solubility but bigness.The other connection between startups and technology is that startups

create new ways of doing things, and new ways of doing things are,

in the broader sense of the word, new technology.

When a startup both begins with an

idea exposed by technological change and makes a product consisting

of technology in the narrower sense (what used to be called "high

technology"), it's easy to conflate the two. But the two connections

are distinct and in principle one could start a startup that was

neither driven by technological change, nor whose product consisted

of technology except in the broader sense.

[4]RateHow fast does a company have to grow to be considered a startup?

There's no precise answer to that. "Startup" is a pole, not a

threshold. Starting one is at first no more than a declaration of

one's ambitions. You're committing not just to starting a company,

but to starting a fast growing one, and you're thus committing to

search for one of the rare ideas of that type. But at first you

have no more than commitment. Starting a startup is like being an

actor in that respect. "Actor" too is a pole rather than a threshold.

At the beginning of his career, an actor is a waiter who goes to

auditions. Getting work makes him a successful actor, but he doesn't

only become an actor when he's successful.So the real question is not what growth rate makes a company a

startup, but what growth rate successful startups tend to have.

For founders that's more than a theoretical question, because it's

equivalent to asking if they're on the right path.The growth of a successful startup usually has three phases:

There's an initial period of slow or no growth while the startup

tries to figure out what it's doing. As the startup figures out how to make something lots of people

want and how to reach those people, there's a period of rapid

growth. Eventually a successful startup will grow into a big company.

Growth will slow, partly due to internal limits and partly because

the company is starting to bump up against the limits of the

markets it serves.

[5]

Together these three phases produce an S-curve. The phase whose

growth defines the startup is the second one, the ascent. Its

length and slope determine how big the company will be.The slope is the company's growth rate. If there's one number every

founder should always know, it's the company's growth rate. That's

the measure of a startup. If you don't know that number, you don't

even know if you're doing well or badly.When I first meet founders and ask what their growth rate is,

sometimes they tell me "we get about a hundred new customers a

month." That's not a rate. What matters is not the absolute number

of new customers, but the ratio of new customers to existing ones.

If you're really getting a constant number of new customers every

month, you're in trouble, because that means your growth rate is

decreasing.During Y Combinator we measure growth rate per week, partly because

there is so little time before Demo Day, and partly because startups

early on need frequent feedback from their users to tweak what

they're doing.

[6]A good growth rate during YC is 5-7% a week. If you can hit 10% a

week you're doing exceptionally well. If you can only manage 1%,

it's a sign you haven't yet figured out what you're doing.The best thing to measure the growth rate of is revenue. The next

best, for startups that aren't charging initially, is active users.

That's a reasonable proxy for revenue growth because whenever the

startup does start trying to make money, their revenues will probably

be a constant multiple of active users.

[7]

CompassWe usually advise startups to pick a growth rate they think they

can hit, and then just try to hit it every week. The key word here

is "just." If they decide to grow at 7% a week and they hit that

number, they're successful for that week. There's nothing more

they need to do. But if they don't hit it, they've failed in the

only thing that mattered, and should be correspondingly alarmed.Programmers will recognize what we're doing here. We're turning

starting a startup into an optimization problem. And anyone who

has tried optimizing code knows how wonderfully effective that sort

of narrow focus can be. Optimizing code means taking an existing

program and changing it to use less of something, usually time or

memory. You don't have to think about what the program should do,

just make it faster. For most programmers this is very satisfying

work. The narrow focus makes it a sort of puzzle, and you're

generally surprised how fast you can solve it.Focusing on hitting a growth rate reduces the otherwise bewilderingly

multifarious problem of starting a startup to a single problem.

You can use that target growth rate to make all your decisions for

you; anything that gets you the growth you need is ipso facto right.

Should you spend two days at a conference? Should you hire another

programmer? Should you focus more on marketing? Should you spend

time courting some big customer? Should you add x feature? Whatever

gets you your target growth rate.

[8]Judging yourself by weekly growth doesn't mean you can look no more

than a week ahead. Once you experience the pain of missing your

target one week (it was the only thing that mattered, and you failed

at it), you become interested in anything that could spare you such

pain in the future. So you'll be willing for example to hire another

programmer, who won't contribute to this week's growth but perhaps

in a month will have implemented some new feature that will get you

more users. But only if (a) the distraction of hiring someone

won't make you miss your numbers in the short term, and (b) you're

sufficiently worried about whether you can keep hitting your numbers

without hiring someone new.It's not that you don't think about the future, just that you think

about it no more than necessary.In theory this sort of hill-climbing could get a startup into

trouble. They could end up on a local maximum. But in practice

that never happens. Having to hit a growth number every week forces

founders to act, and acting versus not acting is the high bit of

succeeding. Nine times out of ten, sitting around strategizing is

just a form of procrastination. Whereas founders' intuitions about

which hill to climb are usually better than they realize. Plus the

maxima in the space of startup ideas are not spiky and isolated.

Most fairly good ideas are adjacent to even better ones.The fascinating thing about optimizing for growth is that it can

actually discover startup ideas. You can use the need for growth

as a form of evolutionary pressure. If you start out with some

initial plan and modify it as necessary to keep hitting, say, 10%

weekly growth, you may end up with a quite different company than

you meant to start. But anything that grows consistently at 10% a

week is almost certainly a better idea than you started with.There's a parallel here to small businesses. Just as the constraint

of being located in a particular neighborhood helps define a bar,

the constraint of growing at a certain rate can help define a

startup.You'll generally do best to follow that constraint wherever it leads

rather than being influenced by some initial vision, just as a

scientist is better off following the truth wherever it leads rather

than being influenced by what he wishes were the case. When Richard

Feynman said that the imagination of nature was greater than the

imagination of man, he meant that if you just keep following the

truth you'll discover cooler things than you could ever have made

up. For startups, growth is a constraint much like truth. Every

successful startup is at least partly a product of the imagination

of growth.

[9]

ValueIt's hard to find something that grows consistently at several

percent a week, but if you do you may have found something surprisingly

valuable. If we project forward we see why.

weeklyyearly

1%1.7x

2%2.8x

5%12.6x

7%33.7x

10%142.0x

A company that grows at 1% a week will grow 1.7x a year, whereas a

company that grows at 5% a week will grow 12.6x. A company making

$1000 a month (a typical number early in YC) and growing at 1% a

week will 4 years later be making $7900 a month, which is less than

a good programmer makes in salary in Silicon Valley. A startup

that grows at 5% a week will in 4 years be making $25 million a

month.

[10]Our ancestors must rarely have encountered cases of exponential

growth, because our intutitions are no guide here. What happens

to fast growing startups tends to surprise even the founders.Small variations in growth rate produce qualitatively different

outcomes. That's why there's a separate word for startups, and why

startups do things that ordinary companies don't, like raising money

and getting acquired. And, strangely enough, it's also why they

fail so frequently.Considering how valuable a successful startup can become, anyone

familiar with the concept of expected value would be surprised if

the failure rate weren't high. If a successful startup could make

a founder $100 million, then even if the chance of succeeding were

only 1%, the expected value of starting one would be $1 million.

And the probability of a group of sufficiently smart and determined

founders succeeding on that scale might be significantly over 1%.

For the right people — e.g. the young Bill Gates — the probability

might be 20% or even 50%. So it's not surprising that so many want

to take a shot at it. In an efficient market, the number of failed

startups should be proportionate to the size of the successes. And

since the latter is huge the former should be too.

[11]What this means is that at any given time, the great majority of

startups will be working on something that's never going to go

anywhere, and yet glorifying their doomed efforts with the grandiose

title of "startup."This doesn't bother me. It's the same with other high-beta vocations,

like being an actor or a novelist. I've long since gotten used to

it. But it seems to bother a lot of people, particularly those

who've started ordinary businesses. Many are annoyed that these

so-called startups get all the attention, when hardly any of them

will amount to anything.If they stepped back and looked at the whole picture they might be

less indignant. The mistake they're making is that by basing their

opinions on anecdotal evidence they're implicitly judging by the

median rather than the average. If you judge by the median startup,

the whole concept of a startup seems like a fraud. You have to

invent a bubble to explain why founders want to start them or

investors want to fund them. But it's a mistake to use the median

in a domain with so much variation. If you look at the average

outcome rather than the median, you can understand why investors

like them, and why, if they aren't median people, it's a rational

choice for founders to start them.

DealsWhy do investors like startups so much? Why are they so hot to

invest in photo-sharing apps, rather than solid money-making

businesses? Not only for the obvious reason.The test of any investment is the ratio of return to risk. Startups

pass that test because although they're appallingly risky, the

returns when they do succeed are so high. But that's not the only

reason investors like startups. An ordinary slower-growing business

might have just as good a ratio of return to risk, if both were

lower. So why are VCs interested only in high-growth companies?

The reason is that they get paid by getting their capital back,

ideally after the startup IPOs, or failing that when it's acquired.The other way to get returns from an investment is in the form of

dividends. Why isn't there a parallel VC industry that invests in

ordinary companies in return for a percentage of their profits?

Because it's too easy for people who control a private company to

funnel its revenues to themselves (e.g. by buying overpriced

components from a supplier they control) while making it look like

the company is making little profit. Anyone who invested in private

companies in return for dividends would have to pay close attention

to their books.The reason VCs like to invest in startups is not simply the returns,

but also because such investments are so easy to oversee. The

founders can't enrich themselves without also enriching the investors.

[12]Why do founders want to take the VCs' money? Growth, again. The

constraint between good ideas and growth operates in both directions.

It's not merely that you need a scalable idea to grow. If you have

such an idea and don't grow fast enough, competitors will. Growing

too slowly is particularly dangerous in a business with network

effects, which the best startups usually have to some degree.Almost every company needs some amount of funding to get started.

But startups often raise money even when they are or could be

profitable. It might seem foolish to sell stock in a profitable

company for less than you think it will later be worth, but it's

no more foolish than buying insurance. Fundamentally that's how

the most successful startups view fundraising. They could grow the

company on its own revenues, but the extra money and help supplied

by VCs will let them grow even faster. Raising money lets you

choose your growth rate.Money to grow faster is always at the command of the most successful

startups, because the VCs need them more than they need the VCs.

A profitable startup could if it wanted just grow on its own revenues.

Growing slower might be slightly dangerous, but chances are it

wouldn't kill them. Whereas VCs need to invest in startups, and

in particular the most successful startups, or they'll be out of

business. Which means that any sufficiently promising startup will

be offered money on terms they'd be crazy to refuse. And yet because

of the scale of the successes in the startup business, VCs can still

make money from such investments. You'd have to be crazy to believe

your company was going to become as valuable as a high growth rate

can make it, but some do.Pretty much every successful startup will get acquisition offers

too. Why? What is it about startups that makes other companies

want to buy them?

[13]Fundamentally the same thing that makes everyone else want the stock

of successful startups: a rapidly growing company is valuable. It's

a good thing eBay bought Paypal, for example, because Paypal is now

responsible for 43% of their sales and probably more of their growth.But acquirers have an additional reason to want startups. A rapidly

growing company is not merely valuable, but dangerous. If it keeps

expanding, it might expand into the acquirer's own territory. Most

product acquisitions have some component of fear. Even if an

acquirer isn't threatened by the startup itself, they might be

alarmed at the thought of what a competitor could do with it. And

because startups are in this sense doubly valuable to acquirers,

acquirers will often pay more than an ordinary investor would.

[14]

UnderstandThe combination of founders, investors, and acquirers forms a natural

ecosystem. It works so well that those who don't understand it are

driven to invent conspiracy theories to explain how neatly things

sometimes turn out. Just as our ancestors did to explain the

apparently too neat workings of the natural world. But there is

no secret cabal making it all work.If you start from the mistaken assumption that Instagram was

worthless, you have to invent a secret boss to force Mark Zuckerberg

to buy it. To anyone who knows Mark Zuckerberg, that is the reductio

ad absurdum of the initial assumption. The reason he bought Instagram

was that it was valuable and dangerous, and what made it so was

growth.If you want to understand startups, understand growth. Growth

drives everything in this world. Growth is why startups usually

work on technology — because ideas for fast growing companies are

so rare that the best way to find new ones is to discover those

recently made viable by change, and technology is the best source

of rapid change. Growth is why it's a rational choice economically

for so many founders to try starting a startup: growth makes the

successful companies so valuable that the expected value is high

even though the risk is too. Growth is why VCs want to invest in

startups: not just because the returns are high but also because

generating returns from capital gains is easier to manage than

generating returns from dividends. Growth explains why the most

successful startups take VC money even if they don't need to: it

lets them choose their growth rate. And growth explains why

successful startups almost invariably get acquisition offers. To

acquirers a fast-growing company is not merely valuable but dangerous

too.It's not just that if you want to succeed in some domain, you have

to understand the forces driving it. Understanding growth is what

starting a startup consists of. What you're really doing (and

to the dismay of some observers, all you're really doing) when you

start a startup is committing to solve a harder type of problem

than ordinary businesses do. You're committing to search for one

of the rare ideas that generates rapid growth. Because these ideas

are so valuable, finding one is hard. The startup is the embodiment

of your discoveries so far. Starting a startup is thus very much

like deciding to be a research scientist: you're not committing to

solve any specific problem; you don't know for sure which problems

are soluble; but you're committing to try to discover something no

one knew before. A startup founder is in effect an economic research

scientist. Most don't discover anything that remarkable, but some

discover relativity.

Notes[1]

Strictly speaking it's not lots of customers you need but a big

market, meaning a high product of number of customers times how

much they'll pay. But it's dangerous to have too few customers

even if they pay a lot, or the power that individual customers have

over you could turn you into a de facto consulting firm. So whatever

market you're in, you'll usually do best to err on the side of

making the broadest type of product for it.[2]

One year at Startup School David Heinemeier Hansson encouraged

programmers who wanted to start businesses to use a restaurant as

a model. What he meant, I believe, is that it's fine to start

software companies constrained in (a) in the same way a restaurant

is constrained in (b). I agree. Most people should not try to

start startups.[3]

That sort of stepping back is one of the things we focus on at

Y Combinator. It's common for founders to have discovered something

intuitively without understanding all its implications. That's

probably true of the biggest discoveries in any field.[4]

I got it wrong in "How to Make Wealth" when I said that a

startup was a small company that takes on a hard technical

problem. That is the most common recipe but not the only one.[5]

In principle companies aren't limited by the size of the markets

they serve, because they could just expand into new markets. But

there seem to be limits on the ability of big companies to do that.

Which means the slowdown that comes from bumping up against the

limits of one's markets is ultimately just another way in which

internal limits are expressed.It may be that some of these limits could be overcome by changing

the shape of the organization — specifically by sharding it.[6]

This is, obviously, only for startups that have already launched

or can launch during YC. A startup building a new database will

probably not do that. On the other hand, launching something small

and then using growth rate as evolutionary pressure is such a

valuable technique that any company that could start this way

probably should.[7]

If the startup is taking the Facebook/Twitter route and building

something they hope will be very popular but from which they don't

yet have a definite plan to make money, the growth rate has to be

higher, even though it's a proxy for revenue growth, because such

companies need huge numbers of users to succeed at all.Beware too of the edge case where something spreads rapidly but the

churn is high as well, so that you have good net growth till you run

through all the potential users, at which point it suddenly stops.[8]

Within YC when we say it's ipso facto right to do whatever gets

you growth, it's implicit that this excludes trickery like buying

users for more than their lifetime value, counting users as active

when they're really not, bleeding out invites at a regularly

increasing rate to manufacture a perfect growth curve, etc. Even

if you were able to fool investors with such tricks, you'd ultimately

be hurting yourself, because you're throwing off your own compass.[9]

Which is why it's such a dangerous mistake to believe that

successful startups are simply the embodiment of some brilliant

initial idea. What you're looking for initially is not so much a

great idea as an idea that could evolve into a great one. The

danger is that promising ideas are not merely blurry versions of

great ones. They're often different in kind, because the early

adopters you evolve the idea upon have different needs from the

rest of the market. For example, the idea that evolves into Facebook

isn't merely a subset of Facebook; the idea that evolves into

Facebook is a site for Harvard undergrads.[10]

What if a company grew at 1.7x a year for a really long time?

Could it not grow just as big as any successful startup? In principle

yes, of course. If our hypothetical company making $1000 a month

grew at 1% a week for 19 years, it would grow as big as a company

growing at 5% a week for 4 years. But while such trajectories may

be common in, say, real estate development, you don't see them much

in the technology business. In technology, companies that grow

slowly tend not to grow as big.[11]

Any expected value calculation varies from person to person

depending on their utility function for money. I.e. the first

million is worth more to most people than subsequent millions. How

much more depends on the person. For founders who are younger or

more ambitious the utility function is flatter. Which is probably

part of the reason the founders of the most successful startups of

all tend to be on the young side.[12]

More precisely, this is the case in the biggest winners, which

is where all the returns come from. A startup founder could pull

the same trick of enriching himself at the company's expense by

selling them overpriced components. But it wouldn't be worth it

for the founders of Google to do that. Only founders of failing

startups would even be tempted, but those are writeoffs from the

VCs' point of view anyway.[13]

Acquisitions fall into two categories: those where the acquirer

wants the business, and those where the acquirer just wants the

employees. The latter type is sometimes called an HR acquisition.

Though nominally acquisitions and sometimes on a scale that has a

significant effect on the expected value calculation for potential

founders, HR acquisitions are viewed by acquirers as more akin to

hiring bonuses.[14]

I once explained this to some founders who had recently arrived

from Russia. They found it novel that if you threatened a company

they'd pay a premium for you. "In Russia they just kill you," they

said, and they were only partly joking. Economically, the fact

that established companies can't simply eliminate new competitors

may be one of the most valuable aspects of the rule of law. And

so to the extent we see incumbents suppressing competitors via

regulations or patent suits, we should worry, not because it's a

departure from the rule of law per se but from what the rule of law

is aiming at.

Thanks to Sam Altman, Marc Andreessen, Paul Buchheit, Patrick

Collison, Jessica Livingston, Geoff Ralston, and Harj Taggar for

reading drafts of this.Arabic TranslationEstonian TranslationPortuguese TranslationItalian Translation

Black Swan Farming

Want to start a startup? Get funded by

Y Combinator.

September 2012I've done several types of work over the years but I don't know

another as counterintuitive as startup investing.The two most important things to understand about startup investing,

as a business, are (1) that effectively all the returns are

concentrated in a few big winners, and (2) that the best ideas look

initially like bad ideas.The first rule I knew intellectually, but didn't really grasp till

it happened to us. The total value of the companies we've funded

is around 10 billion, give or take a few. But just two companies,

Dropbox and Airbnb, account for about three quarters of it.In startups, the big winners are big to a degree that violates our

expectations about variation. I don't know whether these expectations

are innate or learned, but whatever the cause, we are just not

prepared for the 1000x variation in outcomes that one finds in

startup investing.That yields all sorts of strange consequences. For example, in

purely financial terms, there is probably at most one company in

each YC batch that will have a significant effect on our returns,

and the rest are just a cost of doing business.

[1]

I haven't

really assimilated that fact, partly because it's so counterintuitive,

and partly because we're not doing this just for financial reasons;

YC would be a pretty lonely place if we only had one company per

batch. And yet it's true.To succeed in a domain that violates your intuitions, you need to

be able to turn them off the way a pilot does when flying through

clouds.

[2]

You need to do what you know intellectually to be

right, even though it feels wrong.It's a constant battle for us. It's hard to make ourselves take

enough risks. When you interview a startup and think "they seem

likely to succeed," it's hard not to fund them. And yet, financially

at least, there is only one kind of success: they're either going

to be one of the really big winners or not, and if not it doesn't

matter whether you fund them, because even if they succeed the

effect on your returns will be insignificant. In the same day of

interviews you might meet some smart 19 year olds who aren't even

sure what they want to work on. Their chances of succeeding seem

small. But again, it's not their chances of succeeding that matter

but their chances of succeeding really big. The probability that

any group will succeed really big is microscopically small, but the

probability that those 19 year olds will might be higher than that

of the other, safer group.The probability that a startup will make it big is not simply a

constant fraction of the probability that they will succeed at all.

If it were, you could fund everyone who seemed likely to succeed

at all, and you'd get that fraction of big hits. Unfortunately

picking winners is harder than that. You have to ignore the elephant

in front of you, the likelihood they'll succeed, and focus instead

on the separate and almost invisibly intangible question of whether

they'll succeed really big.HarderThat's made harder by the fact that the best startup ideas seem at

first like bad ideas. I've written about this before: if a good

idea were obviously good, someone else would already have done it.

So the most successful founders tend to work on ideas that few

beside them realize are good. Which is not that far from a description

of insanity, till you reach the point where you see results.The first time Peter Thiel spoke at YC he drew a Venn diagram that

illustrates the situation perfectly. He drew two intersecting

circles, one labelled "seems like a bad idea" and the other "is a

good idea." The intersection is the sweet spot for startups.This concept is a simple one and yet seeing it as a Venn diagram

is illuminating. It reminds you that there is an intersection—that

there are good ideas that seem bad. It also reminds you that the

vast majority of ideas that seem bad are bad.The fact that the best ideas seem like bad ideas makes it even

harder to recognize the big winners. It means the probability of

a startup making it really big is not merely not a constant fraction

of the probability that it will succeed, but that the startups with

a high probability of the former will seem to have a disproportionately

low probability of the latter.History tends to get rewritten by big successes, so that in retrospect

it seems obvious they were going to make it big. For that reason

one of my most valuable memories is how lame Facebook sounded to

me when I first heard about it. A site for college students to

waste time? It seemed the perfect bad idea: a site (1) for a niche

market (2) with no money (3) to do something that didn't matter.One could have described Microsoft and Apple in exactly the same

terms.

[3]Harder StillWait, it gets worse. You not only have to solve this hard problem,

but you have to do it with no indication of whether you're succeeding.

When you pick a big winner, you won't know it for two years.Meanwhile, the one thing you can measure is dangerously

misleading. The one thing we can track precisely is how well the

startups in each batch do at fundraising after Demo Day. But we

know that's the wrong metric. There's no correlation between the

percentage of startups that raise money and the metric that does

matter financially, whether that batch of startups contains a big

winner or not.Except an inverse one. That's the scary thing: fundraising is not

merely a useless metric, but positively misleading. We're in a

business where we need to pick unpromising-looking outliers, and

the huge scale of the successes means we can afford to spread our

net very widely. The big winners could generate 10,000x returns.

That means for each big winner we could pick a thousand companies

that returned nothing and still end up 10x ahead.If we ever got to the point where 100% of the startups we funded

were able to raise money after Demo Day, it would almost certainly

mean we were being too conservative.

[4]It takes a conscious effort not to do that too. After 15 cycles

of preparing startups for investors and then watching how they do,

I can now look at a group we're interviewing through Demo Day

investors' eyes. But those are the wrong eyes to look through!We can afford to take at least 10x as much risk as Demo Day investors.

And since risk is usually proportionate to reward, if you can afford

to take more risk you should. What would it mean to take 10x more

risk than Demo Day investors? We'd have to be willing to fund 10x

more startups than they would. Which means that even if we're

generous to ourselves and assume that YC can on average triple a

startup's expected value, we'd be taking the right amount of risk

if only 30% of the startups were able to raise significant funding

after Demo Day.I don't know what fraction of them currently raise more after Demo

Day. I deliberately avoid calculating that number, because if you

start measuring something you start optimizing it, and I know it's

the wrong thing to optimize.

[5]

But the percentage is certainly

way over 30%. And frankly the thought of a 30% success rate at

fundraising makes my stomach clench. A Demo Day where only 30% of

the startups were fundable would be a shambles. Everyone would

agree that YC had jumped the shark. We ourselves would feel that

YC had jumped the shark. And yet we'd all be wrong.For better or worse that's never going to be more than a thought

experiment. We could never stand it. How about that for

counterintuitive? I can lay out what I know to be the right thing

to do, and still not do it. I can make up all sorts of plausible

justifications. It would hurt YC's brand (at least among the

innumerate) if we invested in huge numbers of risky startups that

flamed out. It might dilute the value of the alumni network.

Perhaps most convincingly, it would be demoralizing for us to be

up to our chins in failure all the time. But I know the real reason

we're so conservative is that we just haven't assimilated the fact

of 1000x variation in returns.We'll probably never be able to bring ourselves to take risks

proportionate to the returns in this business. The best we can

hope for is that when we interview a group and find ourselves

thinking "they seem like good founders, but what are investors going

to think of this crazy idea?" we'll continue to be able to say "who

cares what investors think?" That's what we thought about Airbnb,

and if we want to fund more Airbnbs we have to stay good at thinking

it.Notes[1]

I'm not saying that the big winners are all that matters, just

that they're all that matters financially for investors. Since

we're not doing YC mainly for financial reasons, the big winners

aren't all that matters to us. We're delighted to have funded

Reddit, for example. Even though we made comparatively little from

it, Reddit has had a big effect on the world, and it introduced us

to Steve Huffman and Alexis Ohanian, both of whom have become good

friends.Nor do we push founders to try to become one of the big winners if

they don't want to. We didn't "swing for the fences" in our own

startup (Viaweb, which was acquired for $50 million), and it would

feel pretty bogus to press founders to do something we didn't do.

Our rule is that it's up to the founders. Some want to take over

the world, and some just want that first few million. But we invest

in so many companies that we don't have to sweat any one outcome.

In fact, we don't have to sweat whether startups have exits at all.

The biggest exits are the only ones that matter financially, and

those are guaranteed in the sense that if a company becomes big

enough, a market for its shares will inevitably arise. Since the

remaining outcomes don't have a significant effect on returns, it's

cool with us if the founders want to sell early for a small amount,

or grow slowly and never sell (i.e. become a so-called lifestyle

business), or even shut the company down. We're sometimes disappointed

when a startup we had high hopes for doesn't do well, but this

disappointment is mostly the ordinary variety that anyone feels

when that happens.[2]

Without visual cues (e.g. the horizon) you can't distinguish

between gravity and acceleration. Which means if you're flying

through clouds you can't tell what the attitude of

the aircraft is. You could feel like you're flying straight and

level while in fact you're descending in a spiral. The solution

is to ignore what your body is telling you and listen only to your

instruments. But it turns out to be very hard to ignore what your

body is telling you. Every pilot knows about this

problem and yet

it is still a leading cause of accidents.[3]

Not all big hits follow this pattern though. The reason Google

seemed a bad idea was that there were already lots of search engines

and there didn't seem to be room for another.[4]

A startup's success at fundraising is a function of two things:

what they're selling and how good they are at selling it. And while

we can teach startups a lot about how to appeal to investors, even

the most convincing pitch can't sell an idea that investors don't

like. I was genuinely worried that Airbnb, for example, would not

be able to raise money after Demo Day. I couldn't convince Fred Wilson to fund them. They might not

have raised money at all but for the coincidence that Greg Mcadoo,

our contact at Sequoia, was one of a handful of VCs who understood

the vacation rental business, having spent much of the previous two

years investigating it.[5]

I calculated it once for the last batch before a consortium of

investors started offering investment automatically to every startup

we funded, summer 2010. At the time it was 94% (33 of 35 companies

that tried to raise money succeeded, and one didn't try because

they were already profitable). Presumably it's lower now because

of that investment; in the old days it was raise after Demo Day or

die.Thanks to Sam Altman, Paul Buchheit, Patrick Collison, Jessica

Livingston, Geoff Ralston, and Harj Taggar for reading drafts of

this.

The Top of My Todo List

April 2012A palliative care nurse called Bronnie Ware made a list of the

biggest regrets

of the dying. Her list seems plausible. I could see

myself — can see myself — making at least 4 of these

5 mistakes.If you had to compress them into a single piece of advice, it might

be: don't be a cog. The 5 regrets paint a portrait of post-industrial

man, who shrinks himself into a shape that fits his circumstances,

then turns dutifully till he stops.The alarming thing is, the mistakes that produce these regrets are

all errors of omission. You forget your dreams, ignore your family,

suppress your feelings, neglect your friends, and forget to be

happy. Errors of omission are a particularly dangerous type of

mistake, because you make them by default.I would like to avoid making these mistakes. But how do you avoid

mistakes you make by default? Ideally you transform your life so

it has other defaults. But it may not be possible to do that

completely. As long as these mistakes happen by default, you probably

have to be reminded not to make them. So I inverted the 5 regrets,

yielding a list of 5 commands

Don't ignore your dreams; don't work too much; say what you

think; cultivate friendships; be happy.

which I then put at the top of the file I use as a todo list.Japanese Translation

Writing and Speaking

March 2012I'm not a very good speaker. I say "um" a lot. Sometimes I have

to pause when I lose my train of thought. I wish I were a better

speaker. But I don't wish I were a better speaker like I wish I

were a better writer. What I really want is to have good ideas,

and that's a much bigger part of being a good writer than being a

good speaker.Having good ideas is most of writing well. If you know what you're

talking about, you can say it in the plainest words and you'll be

perceived as having a good style. With speaking it's the opposite:

having good ideas is an alarmingly small component of being a good

speaker.I first noticed this at a conference several years ago.

There was another speaker who was much better than me.

He had all of us roaring with laughter. I seemed awkward and

halting by comparison. Afterward I put my talk online like I usually

do. As I was doing it I tried to imagine what a transcript of the

other guy's talk would be like, and it was only then I realized he

hadn't said very much.Maybe this would have been obvious to someone who knew more about

speaking, but it was a revelation to me how much less ideas mattered

in speaking than writing.

[1]A few years later I heard a talk by someone who was not merely a

better speaker than me, but a famous speaker. Boy was he good. So

I decided I'd pay close attention to what he said, to learn how he

did it. After about ten sentences I found myself thinking "I don't

want to be a good speaker."Being a really good speaker is not merely orthogonal to having good ideas,

but in many ways pushes you in the opposite direction. For example,

when I give a talk, I usually write it out beforehand. I know that's

a mistake; I know delivering a

prewritten

talk makes it harder to

engage with an audience. The way to get the attention of an audience

is to give them your full attention, and when you're delivering

a prewritten talk, your attention is always divided between the

audience and the talk — even if you've memorized it. If you want

to engage an audience, it's better to start with no more than an outline

of what you want to say and

ad lib the individual sentences. But

if you do that, you might spend no more time thinking about each

sentence than it takes to say it.

[2]

Occasionally the stimulation

of talking to a live audience makes you think of new things, but

in general this is not going to generate ideas as well as writing

does, where you can spend as long on each sentence as you want.If you rehearse a prewritten speech enough, you can get

asymptotically close to the sort of engagement you get when speaking

ad lib. Actors do. But here again there's a tradeoff between

smoothness and ideas. All the time you spend practicing a talk,

you could instead spend making it better. Actors don't face

that temptation, except in the rare cases where they've written the

script, but any speaker does. Before I give a talk I can usually

be found sitting in a corner somewhere with a copy printed out on

paper, trying to rehearse it in my head. But I always end up

spending most of the time rewriting it instead. Every talk I give

ends up being given from a manuscript full of things crossed out

and rewritten. Which of course makes me um even more, because I

haven't had any time to practice the new bits.

[3]Depending on your audience, there are even worse tradeoffs than

these. Audiences like to be flattered; they like jokes; they like

to be swept off their feet by a vigorous stream of words. As you

decrease the intelligence of the audience, being a good speaker is

increasingly a matter of being a good bullshitter. That's true in

writing too of course, but the descent is steeper with talks. Any

given person is dumber as a member of an audience than as a reader.

Just as a speaker ad libbing can only spend as long thinking about

each sentence as it takes to say it, a person hearing a talk can

only spend as long thinking about each sentence as it takes to hear

it. Plus people in an audience are always affected by the reactions

of those around them, and the reactions that spread from person to

person in an audience are disproportionately the more brutish sort,

just as low notes travel through walls better than high ones. Every

audience is an incipient mob, and a good speaker uses that. Part

of the reason I laughed so much at the talk by the good speaker at

that conference was that everyone else did.

[4]So are talks useless? They're certainly inferior to the written

word as a source of ideas. But that's not all talks are good for.

When I go to a talk, it's usually because I'm interested in the

speaker. Listening to a talk is the closest most of us can get to

having a conversation with someone like the president, who doesn't

have time to meet individually with all the people who want to meet

him.Talks are also good at motivating me to do things. It's probably

no coincidence that so many famous speakers are described as

motivational speakers. That may be what public speaking is really

for. It's probably what it was originally for. The emotional

reactions you can elicit with a talk can be a powerful force.

I wish I could say that this force was more often used for good than

ill, but I'm not sure.Notes[1]

I'm not talking here about academic talks, which are a

different type of thing. While the

audience at an academic talk might appreciate a joke, they will (or

at least should) make a conscious effort to see what new ideas

you're presenting.[2]

That's the lower bound. In practice you can often do better,

because talks are usually about things you've written or talked

about before, and when you ad lib, you end up reproducing some of

those sentences. Like early medieval architecture, impromptu talks

are made of spolia. Which feels a bit dishonest, incidentally,

because you have to deliver these sentences as if you'd just thought

of them.[3]

Robert Morris points out that there is a way in which practicing

talks makes them better: reading a talk out loud can expose awkward

parts. I agree and in fact I read most things I write out loud at

least once for that reason.[4]

For sufficiently small audiences, it may not be true that being

part of an audience makes people dumber. The real decline seems

to set in when the audience gets too big for the talk to feel like

a conversation — maybe around 10 people.

Thanks to Sam Altman and Robert Morris for reading drafts

of this.

How Y Combinator Started

March 2012Y Combinator's 7th birthday was March 11. As usual we were so

busy we didn't notice till a few days after. I don't think we've

ever managed to remember our birthday on our birthday.

On March 11 2005, Jessica and I were walking home from dinner in

Harvard Square. Jessica was working at an investment bank at the

time, but she didn't like it much, so she had interviewed for a job

as director of marketing at a Boston VC fund. The VC fund was doing

what now seems a comically familiar thing for a VC fund to do:

taking a long time to make up their mind. Meanwhile I had been

telling Jessica all the things they should change about the VC

business essentially the ideas now underlying Y Combinator:

investors

should be making more, smaller investments, they should be funding

hackers instead of suits, they should be willing to fund younger

founders, etc.

At the time I had been thinking about doing some angel investing. I

had just given a talk to the undergraduate computer club at Harvard

about

how to start a

startup, and it

hit me afterward that although I had always

meant to do angel investing, 7 years had now passed since I got

enough money to do it, and I still hadn't started. I had also

been thinking about ways to work with Robert Morris and Trevor

Blackwell again. A few hours before I had

sent them an email trying to figure out what we could do together.

Between Harvard Square and my house the idea gelled. We'd start

our own investment firm and Jessica could work for that instead.

As we turned onto Walker Street we decided to do it. I agreed to

put $100k into the new fund and Jessica agreed to quit her job to

work for it. Over the next couple days I recruited Robert

and Trevor, who put in another $50k each. So YC

started with $200k.

Jessica was so happy to be able to quit her job and start her own

company that I took her picture

when we got home.

The company wasn't called Y Combinator yet. At first we called it

Cambridge Seed. But that name never saw the light of day, because

by the time we announced it a few days later, we'd changed the name

to Y Combinator. We realized early on that what we were doing could

be national in scope and we didn't want a name that tied us to one

place.

Initially we only had part of the idea. We were going to do

seed funding with standardized terms. Before YC, seed funding was

very haphazard. You'd get that first $10k from your friend's rich

uncle. The deal terms were often a disaster; often neither the

investor nor the founders nor the lawyer knew what the documents

should look like. Facebook's early history as a Florida LLC shows

how random things could be in those days. We were going to be

something there had not been before: a standard source of seed

funding.

We modelled YC on the seed funding we ourselves had taken

when we started Viaweb. We started Viaweb with $10k we got from

our friend Julian Weber,

the husband of Idelle Weber, whose

painting class I took as a grad student at Harvard. Julian knew

about business, but you would not describe him as a suit. Among

other things he'd been president of the National Lampoon. He was

also a lawyer, and got all our paperwork set up properly. In return

for $10k, getting us set up as a company, teaching us what

business was about, and remaining calm in times of crisis, Julian

got 10% of Viaweb. I remember thinking once what a good deal

Julian got. And then a second later I realized that without

Julian, Viaweb would never have made it. So even though it was a

good deal for him, it was a good deal for us too. That's why I

knew there was room for something like Y Combinator.

Initially we didn't have what turned out to be the most important

idea: funding startups synchronously, instead of asynchronously as

it had always been done before. Or rather we had the idea, but we

didn't realize its significance. We decided very early

that the first thing we'd do would

be to fund a bunch of startups over the coming summer. But we

didn't realize initially that this would be the way we'd do all our

investing. The reason we began by funding a bunch of startups at

once was not that we thought it would be a better way to fund

startups, but simply because we wanted to learn how to be angel

investors, and a summer program for undergrads seemed the fastest

way to do it. No one takes summer jobs that seriously. The

opportunity cost for a bunch of undergrads to spend a summer working

on startups was low enough that we wouldn't feel guilty encouraging

them to do it.

We knew students would already be making plans for the summer, so

we did what we're always telling startups to do: we launched fast.

Here are the

initial announcement

and description of what

was at the time called the Summer Founders Program.

We got lucky in that the length and structure of a summer program

turns out to be perfect for what we do.

The structure of the YC cycle is still almost identical to what

it was that first summer.

We also got lucky in who the first batch of founders were. We never

expected to make any money from that first batch. We thought of

the money we were investing as a combination of an educational expense

and a charitable donation. But the

founders in the first batch turned out to be surprisingly good.

And great people too. We're still friends with a lot of them today.

It's hard for people to realize now how inconsequential YC seemed at the

time. I can't blame people who didn't take us seriously, because

we ourselves didn't take that first summer program seriously in the

very beginning. But as the summer progressed we were increasingly

impressed by how well the startups were doing. Other people started

to be impressed too. Jessica and I invented a term, "the Y Combinator

effect," to describe the moment when the realization hit someone

that YC was not totally lame. When people came to YC to speak

at the dinners that first summer, they came in the spirit of someone

coming to address a Boy Scout troop. By the time they left the

building they were all saying some variant of "Wow, these

companies might actually succeed."

Now YC is well enough known that people are no longer surprised

when the companies we fund are legit, but it took a

while for reputation to catch up with reality. That's one of the

reasons we especially like funding ideas that might be dismissed

as "toys" because YC itself was dismissed as one initially.

When we saw how well it worked to fund companies synchronously,

we decided we'd keep doing that. We'd fund two batches of

startups a year.

We funded the second batch in Silicon Valley. That was

a last minute decision. In retrospect I think what pushed me over

the edge was going to Foo Camp that fall. The density of startup

people in the Bay Area was so much greater than in Boston, and the

weather was so nice. I remembered that from living there in the

90s. Plus I didn't want someone else to copy us and describe it

as the Y Combinator of Silicon Valley. I wanted YC to be the Y Combinator

of Silicon Valley. So doing the winter batch in California

seemed like one of those rare cases where the self-indulgent choice

and the ambitious one were the same.

If we'd had enough time to do what we wanted, Y Combinator would

have been in Berkeley. That was our favorite part of the Bay Area.

But we didn't have time to get a building in Berkeley. We didn't

have time to get our own building anywhere. The only way to get

enough space in time was to convince Trevor to let us take over

part of his (as it then seemed) giant building in Mountain View.

Yet again we lucked out, because Mountain View turned out to be the

ideal place to put something like YC. But even then we barely made

it. The first dinner in California, we had to warn all the founders

not to touch the walls, because the paint was still wet.

Defining Property

March 2012As a child I read a book of stories about a famous judge in eighteenth

century Japan called Ooka Tadasuke. One of the cases he decided

was brought by the owner of a food shop. A poor student who could

afford only rice was eating his rice while enjoying the delicious

cooking smells coming from the food shop. The owner wanted the

student to pay for the smells he was enjoying.The student was

stealing his smells!This story often comes to mind when I hear the RIAA and MPAA accusing

people of stealing music and movies.It sounds ridiculous to us to treat smells as property. But I can

imagine scenarios in which one could charge for smells. Imagine

we were living on a moon base where we had to buy air by the

liter. I could imagine air suppliers adding scents at an extra

charge.The reason it seems ridiculous to us to treat smells as property

is that it wouldn't work to. It would work on a moon base, though.What counts as property depends on what works to treat as property.

And that not only can change, but has changed. Humans may always

(for some definition of human and always) have treated small items

carried on one's person as property. But hunter gatherers didn't

treat land, for example, as property in the way we do.

[1]The reason so many people think of property as having a single

unchanging definition is that its definition changes very slowly.

[2]

But we are in the midst of such a change now. The record

labels and movie studios used to distribute what they made like air

shipped through tubes on a moon base. But with the arrival of

networks, it's as if we've moved to a planet with a breathable

atmosphere. Data moves like smells now. And through a combination

of wishful thinking and short-term greed, the labels and studios

have put themselves in the position of the food shop owner, accusing

us all of stealing their smells.(The reason I say short-term greed is that the underlying problem

with the labels and studios is that the people who run them are

driven by bonuses rather than equity. If they were driven by equity

they'd be looking for ways to take advantage of technological change

instead of fighting it. But building new things takes too long.

Their bonuses depend on this year's revenues, and the best way to

increase those is to extract more money from stuff they do already.)So what does this mean? Should people not be able to charge for

content? There's not a single yes or no answer to that question.

People should be able to charge for content when it works to charge

for content.But by "works" I mean something more subtle than "when they can get

away with it." I mean when people can charge for content without

warping society in order to do it. After all, the companies selling

smells on the moon base could continue to sell them on the Earth,

if they lobbied successfully for laws requiring us all to continue

to breathe through tubes down here too, even though we no longer

needed to.The crazy legal measures that the labels and studios have been

taking have a lot of that flavor. Newspapers and magazines are

just as screwed, but they are at least declining gracefully. The

RIAA and MPAA would make us breathe through tubes if they could.Ultimately it comes down to common sense. When you're abusing the

legal system by trying to use mass lawsuits against randomly chosen

people as a form of exemplary punishment, or lobbying for laws

that would break the Internet if they passed, that's ipso facto

evidence you're using a definition of property that doesn't work.This is where it's helpful to have working democracies and multiple

sovereign countries. If the world had a single, autocratic government,

the labels and studios could buy laws making the definition of

property be whatever they wanted. But fortunately there are still

some countries that are not copyright colonies of the US, and even

in the US, politicians

still seem to be afraid of actual voters, in sufficient numbers.

[3]The people running the US may not like it when voters or other

countries refuse to bend to their will, but ultimately it's in all

our interest that there's not a single point of attack for people

trying to warp the law to serve their own purposes. Private property

is an extremely useful idea — arguably one of our greatest inventions.

So far, each new definition of it has brought us increasing material

wealth.

[4]

It seems reasonable to suppose the newest one will

too. It would be a disaster if we all had to keep running an

obsolete version just because a few powerful people were too lazy

to upgrade.Notes[1]

If you want to learn more about hunter gatherers I strongly

recommend Elizabeth Marshall Thomas's The

Harmless People and The

Old Way.[2]

Change in the definition of property is driven mostly by

technological progress, however, and since technological progress

is accelerating, so presumably will the rate of change in the

definition of property. Which means it's all the more important

for societies to be able to respond gracefully to such changes,

because they will come at an ever increasing rate.[3]

As far as I know, the term "copyright colony" was first used

by Myles

Peterson.[4]

The state of technology isn't simply a function of

the definition of property. They each constrain the other. But

that being so, you can't mess with the definition of property without

affecting (and probably harming) the state of technology. The

history of the USSR offers a vivid illustration of that.Thanks to Sam Altman and Geoff Ralston for reading drafts

of this.Japanese Translation

Frighteningly Ambitious Startup Ideas

Want to start a startup? Get funded by

Y Combinator.

March 2012One of the more surprising things I've noticed while working

on Y Combinator is how frightening the most ambitious startup

ideas are. In this essay I'm going to demonstrate

this phenomenon by describing some. Any one of them

could make you a billionaire. That might sound like an attractive

prospect, and yet when I describe these ideas you may

notice you find yourself shrinking away from them.Don't worry, it's not a sign of weakness. Arguably it's a sign of

sanity. The biggest startup ideas are terrifying. And not just

because they'd be a lot of work. The biggest ideas seem to threaten

your identity: you wonder if you'd have enough ambition to carry

them through.There's a scene in Being John Malkovich where the nerdy hero

encounters a very attractive, sophisticated woman. She says to

him:

Here's the thing: If you ever got me, you wouldn't have a clue

what to do with me.

That's what these ideas say to us.This phenomenon is one of the most important things you can understand

about startups.

[1]

You'd expect big startup ideas to be

attractive, but actually they tend to repel you. And that has a

bunch of consequences. It means these ideas are invisible to most

people who try to think of startup ideas, because their subconscious

filters them out. Even the most ambitious people are probably best

off approaching them obliquely.1. A New Search EngineThe best ideas are just on the right side of impossible. I don't

know if this one is possible, but there are signs it might be.

Making a new search engine means competing with Google, and recently

I've noticed some cracks in their fortress.The point when it became clear to me that Microsoft had lost their

way was when they decided to get into the search business. That

was not a natural move for Microsoft. They did it because they

were afraid of Google, and Google was in the search business. But

this meant (a) Google was now setting Microsoft's agenda, and (b)

Microsoft's agenda consisted of stuff they weren't good at.Microsoft : Google :: Google : Facebook.That does not by itself mean

there's room for a new search engine, but lately when using Google

search I've found myself nostalgic for the old days, when

Google was true to its own slightly aspy self. Google used to give

me a page of the right answers, fast, with no clutter. Now the

results seem inspired by the Scientologist principle that what's

true is what's true for you. And the pages don't have the

clean, sparse feel they used to. Google search results used to

look like the output of a Unix utility. Now if I accidentally put

the cursor in the wrong place, anything might happen.The way to win here is to build the search engine all the hackers

use. A search engine whose users consisted of the top 10,000 hackers

and no one else would be in a very powerful position despite its

small size, just as Google was when it was that search engine. And

for the first time in over a decade the idea of switching seems

thinkable to me.Since anyone capable of starting this company is one of those 10,000

hackers, the route is at least straightforward: make the search

engine you yourself want. Feel free to make it excessively hackerish.

Make it really good for code search, for example. Would you like

search queries to be Turing complete? Anything that gets you those

10,000 users is ipso facto good.Don't worry if something you want to do will constrain you in the

long term, because if you don't get that initial core of users,

there won't be a long term. If you can just build something that

you and your friends genuinely prefer to Google, you're already

about 10% of the way to an IPO, just as Facebook was (though they

probably didn't realize it) when they got all the Harvard undergrads.2. Replace EmailEmail was not designed to be used the way we use it now. Email is

not a messaging protocol. It's a todo list. Or rather, my inbox

is a todo list, and email is the way things get onto it. But it

is a disastrously bad todo list.I'm open to different types of solutions to this problem, but I

suspect that tweaking the inbox is not enough, and that email has

to be replaced with a new protocol.

This new protocol should be a todo list protocol, not

a messaging protocol, although there is a degenerate case where

what someone wants you to do is: read the following text.As a todo list protocol, the new protocol should give more power

to the recipient than email does. I want there to be more restrictions

on what someone can put on my todo list. And when someone can put

something on my todo list, I want them to tell me more about what

they want from me. Do they want me to do something beyond just

reading some text? How important is it? (There obviously has to

be some mechanism to prevent people from saying everything is

important.) When does it have to be done?This is one of those ideas that's like an irresistible force meeting

an immovable object. On one hand, entrenched protocols are impossible

to replace. On the other, it seems unlikely that people in

100 years will still be living in the same email hell we do now.

And if email is going to get replaced eventually, why not now?If you do it right, you may be able to avoid the usual chicken

and egg problem new protocols face, because some of the most powerful

people in the world will be among the first to switch to it.

They're all at the mercy of email too.Whatever you build, make it fast. GMail has become painfully slow.

[2]

If you made something no better than GMail, but fast, that

alone would let you start to pull users away from GMail.GMail is slow because Google can't afford to spend a lot on it.

But people will pay for this. I'd have no problem paying $50 a month.

Considering how much time I spend in email, it's kind of scary to

think how much I'd be justified in paying. At least $1000 a month.

If I spend several hours a day reading and writing email, that would

be a cheap way to make my life better.3. Replace UniversitiesPeople are all over this idea lately, and I think they're onto

something. I'm reluctant to suggest that an institution that's

been around for a millennium is finished just because of some mistakes

they made in the last few decades, but certainly in the last few

decades US universities seem to have been headed down the wrong

path. One could do a lot better for a lot less money.I don't think universities will disappear. They won't be replaced

wholesale. They'll just lose the de facto monopoly on certain types

of learning that they once had. There will be many different ways

to learn different things, and some may look quite different from

universities. Y Combinator itself is arguably one of them.Learning is such a big problem that changing the way people do it

will have a wave of secondary effects. For example, the name of

the university one went to is treated by a lot of people (correctly

or not) as a credential in its own right. If learning breaks up

into many little pieces, credentialling may separate from it. There

may even need to be replacements for campus social life (and oddly

enough, YC even has aspects of that).You could replace high schools too, but there you face bureaucratic

obstacles that would slow down a startup. Universities seem the

place to start.4. Internet DramaHollywood has been slow to embrace the Internet. That was a

mistake, because I think we can now call a winner in the race between

delivery mechanisms, and it is the Internet, not cable.A lot of the reason is the horribleness of cable clients, also known

as TVs. Our family didn't wait for Apple TV. We hated our last

TV so much that a few months ago we replaced it with an iMac bolted

to the wall. It's a little inconvenient to control it with a

wireless mouse, but the overall experience is much better than the

nightmare UI we had to deal with before.Some of the attention people currently devote to watching

movies and TV can be stolen by things that seem completely unrelated,

like social networking apps. More can be stolen by things that are

a little more closely related, like games. But there will probably

always remain some residual demand for conventional drama, where

you sit passively and watch as a plot happens. So how do you deliver

drama via the Internet? Whatever you make will have to be on a

larger scale than Youtube clips. When people sit down to watch a

show, they want to know what they're going to get: either part

of a series with familiar characters, or a single longer "movie"

whose basic premise they know in advance.There are two ways delivery and payment could play out. Either

some company like Netflix or Apple will be the app store for

entertainment, and you'll reach audiences through them. Or the

would-be app stores will be too overreaching, or too technically

inflexible, and companies will arise to supply payment and streaming

a la carte to the producers of drama. If that's the way things

play out, there will also be a need for such infrastructure companies.5. The Next Steve JobsI was talking recently to someone who knew Apple well, and I asked

him if the people now running the company would be able to keep

creating new things the way Apple had under Steve Jobs. His answer

was simply "no." I already feared that would be the answer. I

asked more to see how he'd qualify it. But he didn't qualify it

at all. No, there will be no more great new stuff beyond whatever's

currently in the pipeline. Apple's

revenues may continue to rise for a long time, but as Microsoft

shows, revenue is a lagging indicator in the technology business.So if Apple's not going to make the next iPad, who is? None of the

existing players. None of them are run by product visionaries, and

empirically you can't seem to get those by hiring them. Empirically

the way you get a product visionary as CEO is for him to found the

company and not get fired. So the company that creates the next

wave of hardware is probably going to have to be a startup.I realize it sounds preposterously ambitious for a startup to try

to become as big as Apple. But no more ambitious than it was for

Apple to become as big as Apple, and they did it. Plus a startup

taking on this problem now has an advantage the original Apple

didn't: the example of Apple. Steve Jobs has shown us what's

possible. That helps would-be successors both directly, as Roger

Bannister did, by showing how much better you can do than people

did before, and indirectly, as Augustus did, by lodging the idea

in users' minds that a single person could unroll the future

for them.

[3]Now Steve is gone there's a vacuum we can all feel. If a new company

led boldly into the future of hardware, users would follow. The

CEO of that company, the "next Steve Jobs," might not measure up

to Steve Jobs. But he wouldn't have to. He'd just have to do a

better job than Samsung and HP and Nokia, and that seems pretty

doable.6. Bring Back Moore's LawThe last 10 years have reminded us what Moore's Law actually says.

Till about 2002 you could safely misinterpret it as promising that

clock speeds would double every 18 months. Actually what it says

is that circuit densities will double every 18 months. It used to

seem pedantic to point that out. Not any more. Intel can no longer

give us faster CPUs, just more of them.This Moore's Law is not as good as the old one. Moore's Law used

to mean that if your software was slow, all you had to do was wait,

and the inexorable progress of hardware would solve your problems.

Now if your software is slow you have to rewrite it to do more

things in parallel, which is a lot more work than waiting.It would be great if a startup could give us something of the old

Moore's Law back, by writing software that could make a large number

of CPUs look to the developer like one very fast CPU. There are

several ways to approach this problem. The most ambitious is to

try to do it automatically: to write a compiler that will parallelize

our code for us. There's a name for this compiler, the sufficiently

smart compiler, and it is a byword for impossibility. But is

it really impossible? Is there no configuration of the bits in

memory of a present day computer that is this compiler? If you

really think so, you should try to prove it, because that would be

an interesting result. And if it's not impossible but simply very

hard, it might be worth trying to write it. The expected value

would be high even if the chance of succeeding was low.The reason the expected value is so high is web services. If you

could write software that gave programmers the convenience of the

way things were in the old days, you could offer it to them as a

web service. And that would in turn mean that you got practically

all the users.Imagine there was another processor manufacturer that could still translate

increased circuit densities into increased clock speeds. They'd

take most of Intel's business. And since web services mean that

no one sees their processors anymore, by writing the sufficiently

smart compiler you could create a situation indistinguishable from

you being that manufacturer, at least for the server market.The least ambitious way of approaching the problem is to start from

the other end, and offer programmers more parallelizable Lego blocks

to build programs out of, like Hadoop and MapReduce. Then the

programmer still does much of the work of optimization.There's an intriguing middle ground where you build a semi-automatic

weapon—where there's a human in the loop. You make something

that looks to the user like the sufficiently smart compiler, but

inside has people, using highly developed optimization tools to

find and eliminate bottlenecks in users' programs. These people

might be your employees, or you might create a marketplace for

optimization.An optimization marketplace would be a way to generate the sufficiently

smart compiler piecemeal, because participants would immediately

start writing bots. It would be a curious state of affairs if you

could get to the point where everything could be done by bots,

because then you'd have made the sufficiently smart compiler, but

no one person would have a complete copy of it.I realize how crazy all this sounds. In fact, what I like about

this idea is all the different ways in which it's wrong. The whole

idea of focusing on optimization is counter to the general trend

in software development for the last several decades. Trying to

write the sufficiently smart compiler is by definition a mistake.

And even if it weren't, compilers are the sort of software that's

supposed to be created by open source projects, not companies. Plus

if this works it will deprive all the programmers who take pleasure

in making multithreaded apps of so much amusing complexity. The

forum troll I have by now internalized doesn't even know where to

begin in raising objections to this project. Now that's what I

call a startup idea.7. Ongoing DiagnosisBut wait, here's another that could face even greater resistance:

ongoing, automatic medical diagnosis.One of my tricks for generating startup ideas is to imagine the

ways in which we'll seem backward to future generations. And I'm

pretty sure that to people 50 or 100 years in the future, it will

seem barbaric that people in our era waited till they had symptoms

to be diagnosed with conditions like heart disease and cancer.For example, in 2004 Bill Clinton found he was feeling short of

breath. Doctors discovered that several of his arteries were over

90% blocked and 3 days later he had a quadruple bypass. It seems

reasonable to assume Bill Clinton has the best medical care available.

And yet even he had to wait till his arteries were over 90% blocked

to learn that the number was over 90%. Surely at some point in the

future we'll know these numbers the way we now know something like

our weight. Ditto for cancer. It will seem preposterous to future

generations that we wait till patients have physical symptoms to

be diagnosed with cancer. Cancer will show up on some sort of radar

screen immediately.(Of course, what shows up on the radar screen may be different from

what we think of now as cancer. I wouldn't be surprised if at any

given time we have ten or even hundreds of microcancers going at

once, none of which normally amount to anything.)A lot of the obstacles to ongoing diagnosis will come from the fact

that it's going against the grain of the medical profession. The

way medicine has always worked is that patients come to doctors

with problems, and the doctors figure out what's wrong. A lot of

doctors don't like the idea of going on the medical equivalent of

what lawyers call a "fishing expedition," where you go looking for

problems without knowing what you're looking for. They call the

things that get discovered this way "incidentalomas," and they are

something of a nuisance.For example, a friend of mine once had her brain scanned as part

of a study. She was horrified when the doctors running the study

discovered what appeared to be a large tumor. After further testing,

it turned out to be a harmless cyst. But it cost her a few days

of terror. A lot of doctors worry that if you start scanning people

with no symptoms, you'll get this on a giant scale: a huge number

of false alarms that make patients panic and require expensive and

perhaps even dangerous tests to resolve. But I think that's just

an artifact of current limitations. If people were scanned all the

time and we got better at deciding what was a real problem, my

friend would have known about this cyst her whole life and known

it was harmless, just as we do a birthmark.There is room for a lot of startups here.

In addition to the technical obstacles all

startups face, and the bureaucratic obstacles all medical startups

face, they'll be going against thousands of years of medical

tradition. But it will happen, and it will be a great thing—so

great that people in the future will feel as sorry for us as we do

for the generations that lived before anaesthesia and antibiotics.TacticsLet me conclude with some tactical advice. If you want to take on

a problem as big as the ones I've discussed, don't make a direct

frontal attack on it. Don't say, for example, that you're going

to replace email. If you do that you raise too many expectations.

Your employees and investors will constantly be asking "are we there

yet?" and you'll have an army of haters waiting to see you fail.

Just say you're building todo-list software. That sounds harmless.

People can notice you've replaced email when it's a fait accompli.

[4]Empirically, the way to do really big things seems to be to start

with deceptively small things. Want to dominate microcomputer

software? Start by writing a Basic interpreter for a machine with

a few thousand users. Want to make the universal web site? Start

by building a site for Harvard undergrads to stalk one another.Empirically, it's not just for other people that you need to start

small. You need to for your own sake. Neither Bill Gates nor Mark

Zuckerberg knew at first how big their companies were going to get.

All they knew was that they were onto something. Maybe it's a bad

idea to have really big ambitions initially, because the bigger

your ambition, the longer it's going to take, and the further you

project into the future, the more likely you'll get it wrong.I think the way to use these big ideas is not to try to identify a

precise point in the future and then ask yourself how to get from

here to there, like the popular image of a visionary. You'll be

better off if you operate like Columbus and just head in a general

westerly direction. Don't try to construct the future like a

building, because your current blueprint is almost certainly mistaken.

Start with something you know works, and when you expand, expand

westward.The popular image of the visionary is someone with a clear view of

the future, but empirically it may be better to have a blurry one.Notes[1]

It's also one of the most important things VCs fail to

understand about startups. Most expect founders to walk in with a

clear plan for the future, and judge them based on that. Few

consciously realize that in the biggest successes there is the least

correlation between the initial plan and what the startup eventually

becomes.[2]

This sentence originally read "GMail is painfully slow."

Thanks to Paul Buchheit for the correction.[3]

Roger Bannister is famous as the first person to run a mile

in under 4 minutes. But his world record only lasted 46 days. Once

he showed it could be done, lots of others followed. Ten years

later Jim Ryun ran a 3:59 mile as a high school junior.[4]

If you want to be the next Apple, maybe you don't even want to start

with consumer electronics. Maybe at first you make something hackers

use. Or you make something popular but apparently unimportant,

like a headset or router. All you need is a bridgehead.

Thanks to Sam Altman, Trevor Blackwell,

Paul Buchheit, Patrick Collison, Aaron Iba, Jessica

Livingston, Robert Morris, Harj Taggar and Garry Tan

for reading drafts of this.

A Word to the Resourceful

Want to start a startup? Get funded by

Y Combinator.

January 2012A year ago I noticed a pattern in the least successful startups

we'd funded: they all seemed hard to talk to. It felt as if there

was some kind of wall between us. I could never quite tell if they

understood what I was saying.This caught my attention because earlier we'd noticed a pattern

among the most successful startups, and it seemed to hinge on a

different quality. We found the startups that did best were the

ones with the sort of founders about whom we'd say "they can take

care of themselves." The startups that do best are fire-and-forget

in the sense that all you have to do is give them a lead, and they'll

close it, whatever type of lead it is. When they're raising money,

for example, you can do the initial intros knowing that if you

wanted to you could stop thinking about it at that point. You won't

have to babysit the round to make sure it happens. That type of

founder is going to come back with the money; the only question is

how much on what terms.It seemed odd that the outliers at the two ends of the spectrum

could be detected by what appeared to be unrelated tests. You'd

expect that if the founders at one end were distinguished by the

presence of quality x, at the other end they'd be distinguished by

lack of x. Was there some kind of inverse relation between

resourcefulness and being hard to talk to?It turns out there is, and the key to the mystery is the old adage

"a word to the wise is sufficient." Because this phrase is not

only overused, but overused in an indirect way (by prepending the

subject to some advice), most people who've heard it don't know

what it means. What it means is that if someone is wise, all you

have to do is say one word to them, and they'll understand immediately.

You don't have to explain in detail; they'll chase down all the

implications.In much the same way that all you have to do is give the right sort

of founder a one line intro to a VC, and he'll chase down the money.

That's the connection. Understanding all the implications — even the

inconvenient implications — of what someone tells you is a subset of

resourcefulness. It's conversational resourcefulness.Like real world resourcefulness, conversational resourcefulness

often means doing things you don't want to. Chasing down all the

implications of what's said to you can sometimes lead to uncomfortable

conclusions. The best word to describe the failure to do so is

probably "denial," though that seems a bit too narrow. A better

way to describe the situation would be to say that the unsuccessful

founders had the sort of conservatism that comes from weakness.

They traversed idea space as gingerly as a very old person

traverses the physical world.

[1]The unsuccessful founders weren't stupid. Intellectually they

were as capable as

the successful founders of following all the implications of what

one said to them. They just weren't eager to.So being hard to talk to was not what was killing the

unsuccessful startups. It

was a sign of an underlying lack of resourcefulness. That's what

was killing them. As well as

failing to chase down the implications of what was said to them,

the unsuccessful founders would also fail to chase down funding,

and users, and sources of new ideas. But the most immediate evidence

I had that something was amiss was that I couldn't talk to them.Notes[1]

A YC partner wrote:My feeling with the bad groups is that coming into office hours,

they've already decided what they're going to do and everything I

say is being put through an internal process in their heads, which

either desperately tries to munge what I've said into something

that conforms with their decision or just outright dismisses it and

creates a rationalization for doing so. They may not even be conscious

of this process but that's what I think is happening when you say

something to bad groups and they have that glazed over look. I don't

think it's confusion or lack of understanding per se, it's this

internal process at work.With the good groups, you can tell that everything you say is being

looked at with fresh eyes and even if it's dismissed, it's because

of some logical reason e.g. "we already tried that" or "from speaking

to our users that isn't what they'd like," etc. Those groups never

have that glazed over look.Thanks to Sam Altman, Patrick Collison, Aaron Iba, Jessica Livingston,

Robert Morris, Harj Taggar, and Garry Tan for reading drafts of

this.

Schlep Blindness

Want to start a startup? Get funded by

Y Combinator.

January 2012There are great startup ideas lying around unexploited right under

our noses. One reason we don't see them is a phenomenon I call

schlep blindness. Schlep was originally a Yiddish word but has

passed into general use in the US. It means a tedious, unpleasant

task.No one likes schleps, but hackers especially dislike them.

Most hackers who start startups wish they could do it by just writing

some clever software, putting it on a server somewhere, and watching

the money roll in—without ever having to talk to users, or negotiate

with other companies, or deal with other people's broken code.

Maybe that's possible, but I haven't seen it.One of the many things we do at Y Combinator is teach hackers about

the inevitability of schleps. No, you can't start a startup by

just writing code. I remember going through this realization myself.

There was a point in 1995 when I was still trying to convince myself

I could start a company by just writing code. But I soon learned

from experience that schleps are not merely inevitable, but pretty

much what business consists of. A company is defined by the schleps

it will undertake. And schleps should be dealt with the same way

you'd deal with a cold swimming pool: just jump in. Which is not

to say you should seek out unpleasant work per se, but that you

should never shrink from it if it's on the path to something great.The most dangerous thing about our dislike of schleps is that much

of it is unconscious. Your unconscious won't even let you see ideas

that involve painful schleps. That's schlep blindness.The phenomenon isn't limited to startups. Most people don't

consciously decide not to be in as good physical shape as Olympic

athletes, for example. Their unconscious mind decides for them,

shrinking from the work involved.The most striking example I know of schlep blindness is

Stripe, or

rather Stripe's idea. For over a decade, every hacker who'd ever

had to process payments online knew how painful the experience was.

Thousands of people must have known about this problem. And yet

when they started startups, they decided to build recipe sites, or

aggregators for local events. Why? Why work on problems few care

much about and no one will pay for, when you could fix one of the

most important components of the world's infrastructure? Because

schlep blindness prevented people from even considering the idea

of fixing payments.Probably no one who applied to Y Combinator to work on a recipe

site began by asking "should we fix payments, or build a recipe

site?" and chose the recipe site. Though the idea of fixing payments

was right there in plain sight, they never saw it, because their

unconscious mind shrank from the complications involved. You'd

have to make deals with banks. How do you do that? Plus you're

moving money, so you're going to have to deal with fraud, and people

trying to break into your servers. Plus there are probably all

sorts of regulations to comply with. It's a lot more intimidating

to start a startup like this than a recipe site.That scariness makes ambitious ideas doubly valuable. In addition

to their intrinsic value, they're like undervalued stocks in the

sense that there's less demand for them among founders. If you

pick an ambitious idea, you'll have less competition, because

everyone else will have been frightened off by the challenges

involved. (This is also true of starting a startup generally.)How do you overcome schlep blindness? Frankly, the most valuable

antidote to schlep blindness is probably ignorance. Most successful

founders would probably say that if they'd known when they were

starting their company about the obstacles they'd have to overcome,

they might never have started it. Maybe that's one reason the most

successful startups of all so often have young founders.In practice the founders grow with the problems. But no one seems

able to foresee that, not even older, more experienced founders.

So the reason younger founders have an advantage is that they make

two mistakes that cancel each other out. They don't know how much

they can grow, but they also don't know how much they'll need to.

Older founders only make the first mistake.Ignorance can't solve everything though. Some ideas so obviously

entail alarming schleps that anyone can see them. How do you see

ideas like that? The trick I recommend is to take yourself out of

the picture. Instead of asking "what problem should I solve?" ask

"what problem do I wish someone else would solve for me?" If someone

who had to process payments before Stripe had tried asking that,

Stripe would have been one of the first things they wished for.It's too late now to be Stripe, but there's plenty still broken in

the world, if you know how to see it.Thanks to Sam Altman, Paul Buchheit, Patrick Collison,

Aaron Iba, Jessica Livingston, Emmett Shear, and Harj Taggar

for reading drafts of this.

Snapshot: Viaweb, June 1998

January 2012A few hours before the Yahoo acquisition was announced in June 1998

I took a snapshot of Viaweb's

site. I thought it might be interesting to look at one day.The first thing one notices is is how tiny the pages are. Screens

were a lot smaller in 1998. If I remember correctly, our frontpage

used to just fit in the size window people typically used then.Browsers then (IE 6 was still 3 years in the future) had few fonts

and they weren't antialiased. If you wanted to make pages that

looked good, you had to render display text as images.You may notice a certain similarity between the Viaweb and Y Combinator logos. We did that

as an inside joke when we started YC. Considering how basic a red

circle is, it seemed surprising to me when we started Viaweb how

few other companies used one as their logo. A bit later I realized

why.On the Company

page you'll notice a mysterious individual called John McArtyem.

Robert Morris (aka Rtm) was so publicity averse after the

Worm that he

didn't want his name on the site. I managed to get him to agree

to a compromise: we could use his bio but not his name. He has

since relaxed a bit

on that point.Trevor graduated at about the same time the acquisition closed, so in the

course of 4 days he went from impecunious grad student to millionaire

PhD. The culmination of my career as a writer of press releases

was one celebrating

his graduation, illustrated with a drawing I did of him during

a meeting.(Trevor also appears as Trevino

Bagwell in our directory of web designers merchants could hire

to build stores for them. We inserted him as a ringer in case some

competitor tried to spam our web designers. We assumed his logo

would deter any actual customers, but it did not.)Back in the 90s, to get users you had to get mentioned in magazines

and newspapers. There were not the same ways to get found online

that there are today. So we used to pay a PR

firm $16,000 a month to get us mentioned in the press. Fortunately

reporters liked

us.In our advice about

getting traffic from search engines (I don't think the term SEO

had been coined yet), we say there are only 7 that matter: Yahoo,

AltaVista, Excite, WebCrawler, InfoSeek, Lycos, and HotBot. Notice

anything missing? Google was incorporated that September.We supported online transactions via a company called

Cybercash,

since if we lacked that feature we'd have gotten beaten up in product

comparisons. But Cybercash was so bad and most stores' order volumes

were so low that it was better if merchants processed orders like phone orders. We had a page in our site trying to talk merchants

out of doing real time authorizations.The whole site was organized like a funnel, directing people to the

test drive.

It was a novel thing to be able to try out software online. We put

cgi-bin in our dynamic urls to fool competitors about how our

software worked.We had some well

known users. Needless to say, Frederick's of Hollywood got the

most traffic. We charged a flat fee of $300/month for big stores,

so it was a little alarming to have users who got lots of traffic.

I once calculated how much Frederick's was costing us in bandwidth,

and it was about $300/month.Since we hosted all the stores, which together were getting just

over 10 million page views per month in June 1998, we consumed what

at the time seemed a lot of bandwidth. We had 2 T1s (3 Mb/sec)

coming into our offices. In those days there was no AWS. Even

colocating servers seemed too risky, considering how often things

went wrong with them. So we had our servers in our offices. Or

more precisely, in Trevor's office. In return for the unique

privilege of sharing his office with no other humans, he had to

share it with 6 shrieking tower servers. His office was nicknamed

the Hot Tub on account of the heat they generated. Most days his

stack of window air conditioners could keep up.For describing pages, we had a template language called RTML, which

supposedly stood for something, but which in fact I named after

Rtm. RTML was Common Lisp augmented by some macros and libraries,

and concealed under a structure editor that made it look like it

had syntax.Since we did continuous releases, our software didn't actually have

versions. But in those days the trade press expected versions, so

we made them up. If we wanted to get lots of attention, we made

the version number an

integer. That "version 4.0" icon was generated by our own

button generator, incidentally. The whole Viaweb site was made

with our software, even though it wasn't an online store, because

we wanted to experience what our users did.At the end of 1997, we released a general purpose shopping search

engine called Shopfind. It

was pretty advanced for the time. It had a programmable crawler

that could crawl most of the different stores online and pick out

the products.

Why Startup Hubs Work

Want to start a startup? Get funded by

Y Combinator.

October 2011If you look at a list of US cities sorted by population, the number

of successful startups per capita varies by orders of magnitude.

Somehow it's as if most places were sprayed with startupicide.I wondered about this for years. I could see the average town was

like a roach motel for startup ambitions: smart, ambitious people

went in, but no startups came out. But I was never able to figure

out exactly what happened inside the motel—exactly what was

killing all the potential startups.

[1]A couple weeks ago I finally figured it out. I was framing the

question wrong. The problem is not that most towns kill startups.

It's that death is the default for startups,

and most towns don't save them. Instead of thinking of most places

as being sprayed with startupicide, it's more accurate to think of

startups as all being poisoned, and a few places being sprayed with

the antidote.Startups in other places are just doing what startups naturally do:

fail. The real question is, what's saving startups in places

like Silicon Valley?

[2]EnvironmentI think there are two components to the antidote: being in a place

where startups are the cool thing to do, and chance meetings with

people who can help you. And what drives them both is the number

of startup people around you.The first component is particularly helpful in the first stage of

a startup's life, when you go from merely having an interest in

starting a company to actually doing it. It's quite a leap to start

a startup. It's an unusual thing to do. But in Silicon Valley it

seems normal.

[3]In most places, if you start a startup, people treat you as if

you're unemployed. People in the Valley aren't automatically

impressed with you just because you're starting a company, but they

pay attention. Anyone who's been here any amount of time knows not

to default to skepticism, no matter how inexperienced you seem or

how unpromising your idea sounds at first, because they've all seen

inexperienced founders with unpromising sounding ideas who a few

years later were billionaires.Having people around you care about what you're doing is an

extraordinarily powerful force. Even the

most willful people are susceptible to it. About a year after we

started Y Combinator I said something to a partner at a well known

VC firm that gave him the (mistaken) impression I was considering

starting another startup. He responded so eagerly that for about

half a second I found myself considering doing it.In most other cities, the prospect of starting a startup just doesn't

seem real. In the Valley it's not only real but fashionable. That

no doubt causes a lot of people to start startups who shouldn't.

But I think that's ok. Few people are suited to running a startup,

and it's very hard to predict beforehand which are (as I know all

too well from being in the business of trying to predict beforehand),

so lots of people starting startups who shouldn't is probably the

optimal state of affairs. As long as you're at a point in your

life when you can bear the risk of failure, the best way to find

out if you're suited to running a startup is to try

it.ChanceThe second component of the antidote is chance meetings with people

who can help you. This force works in both phases: both in the

transition from the desire to start a startup to starting one, and

the transition from starting a company to succeeding. The power

of chance meetings is more variable than people around you caring

about startups, which is like a sort of background radiation that

affects everyone equally, but at its strongest it is far stronger.Chance meetings produce miracles to compensate for the disasters

that characteristically befall startups. In the Valley, terrible

things happen to startups all the time, just like they do to startups

everywhere. The reason startups are more likely to make it here

is that great things happen to them too. In the Valley, lightning

has a sign bit.For example, you start a site for college students and you decide

to move to the Valley for the summer to work on it. And then on a

random suburban street in Palo Alto you happen to run into Sean

Parker, who understands the domain really well because he started

a similar startup himself, and also knows all the investors. And

moreover has advanced views, for 2004, on founders retaining control of their companies.You can't say precisely what the miracle will be, or even for sure

that one will happen. The best one can say is: if you're in a

startup hub, unexpected good things will probably happen to you,

especially if you deserve them.I bet this is true even for startups we fund. Even with us working

to make things happen for them on purpose rather than by accident,

the frequency of helpful chance meetings in the Valley is so high

that it's still a significant increment on what we can deliver.Chance meetings play a role like the role relaxation plays in having

ideas. Most people have had the experience of working hard on some

problem, not being able to solve it, giving up and going to bed,

and then thinking of the answer in the shower in the morning. What

makes the answer appear is letting your thoughts drift a bit—and thus drift off the wrong

path you'd been pursuing last night and onto the right one adjacent

to it.Chance meetings let your acquaintance drift in the same way taking

a shower lets your thoughts drift. The critical thing in both cases

is that they drift just the right amount. The meeting between Larry

Page and Sergey Brin was a good example. They let their acquaintance

drift, but only a little; they were both meeting someone they had

a lot in common with.For Larry Page the most important component of the antidote was

Sergey Brin, and vice versa. The antidote is

people. It's not the

physical infrastructure of Silicon Valley that makes it work, or

the weather, or anything like that. Those helped get it started,

but now that the reaction is self-sustaining what drives it is the

people.Many observers have noticed that one of the most distinctive things

about startup hubs is the degree to which people help one another

out, with no expectation of getting anything in return. I'm not

sure why this is so. Perhaps it's because startups are less of a

zero sum game than most types of business; they are rarely killed

by competitors. Or perhaps it's because so many startup founders

have backgrounds in the sciences, where collaboration is encouraged.A large part of YC's function is to accelerate that process. We're

a sort of Valley within the Valley, where the density of people

working on startups and their willingness to help one another are

both artificially amplified.NumbersBoth components of the antidote—an environment that encourages

startups, and chance meetings with people who help you—are

driven by the same underlying cause: the number of startup people

around you. To make a startup hub, you need a lot of people

interested in startups.There are three reasons. The first, obviously, is that if you don't

have enough density, the chance meetings don't happen.

[4]

The second is that different startups need such different things, so

you need a lot of people to supply each startup with what they need

most. Sean Parker was exactly what Facebook needed in 2004. Another

startup might have needed a database guy, or someone with connections

in the movie business.This is one of the reasons we fund such a large number of companies,

incidentally. The bigger the community, the greater the chance it

will contain the person who has that one thing you need most.The third reason you need a lot of people to make a startup hub is

that once you have enough people interested in the same problem,

they start to set the social norms. And it is a particularly

valuable thing when the atmosphere around you encourages you to do

something that would otherwise seem too ambitious. In most places

the atmosphere pulls you back toward the mean.I flew into the Bay Area a few days ago. I notice this every time

I fly over the Valley: somehow you can sense something is going on.

Obviously you can sense prosperity in how well kept a

place looks. But there are different kinds of prosperity. Silicon

Valley doesn't look like Boston, or New York, or LA, or DC. I tried

asking myself what word I'd use to describe the feeling the Valley

radiated, and the word that came to mind was optimism.Notes[1]

I'm not saying it's impossible to succeed in a city with few

other startups, just harder. If you're sufficiently good at

generating your own morale, you can survive without external

encouragement. Wufoo was based in Tampa and they succeeded. But

the Wufoos are exceptionally disciplined.[2]

Incidentally, this phenomenon is not limited to startups. Most

unusual ambitions fail, unless the person who has them manages to

find the right sort of community.[3]

Starting a company is common, but starting a startup is rare.

I've talked about the distinction between the two elsewhere, but

essentially a startup is a new business designed for scale. Most

new businesses are service businesses and except in rare cases those

don't scale.[4]

As I was writing this, I had a demonstration of the density of

startup people in the Valley. Jessica and I bicycled to University

Ave in Palo Alto to have lunch at the fabulous Oren's Hummus. As

we walked in, we met Charlie Cheever sitting near the door. Selina

Tobaccowala stopped to say hello on her way out. Then Josh Wilson

came in to pick up a take out order. After lunch we went to get

frozen yogurt. On the way we met Rajat Suri. When we got to the

yogurt place, we found Dave Shen there, and as we walked out we ran

into Yuri Sagalov. We walked with him for a block or so and we ran

into Muzzammil Zaveri, and then a block later we met Aydin Senkut.

This is everyday life in Palo Alto. I wasn't trying to meet people;

I was just having lunch. And I'm sure for every startup founder

or investor I saw that I knew, there were 5 more I didn't. If Ron

Conway had been with us he would have met 30 people he knew.Thanks to Sam Altman, Paul Buchheit, Jessica Livingston, and

Harj Taggar for reading drafts of this.

The Patent Pledge

August 2011I realized recently that we may be able to solve part of the patent

problem without waiting for the government.I've never been 100% sure whether patents help or hinder technological

progress. When I was a kid I thought they helped. I thought they

protected inventors from having their ideas stolen by big companies.

Maybe that was truer in the past, when more things were physical.

But regardless of whether patents are in general a good thing, there

do seem to be bad ways of using them. And since bad uses of patents

seem to be increasing, there is an increasing call for patent reform.The problem with patent reform is that it has to go through the

government. That tends to be slow. But recently I realized we can

also attack the problem downstream. As well as pinching off the

stream of patents at the point where they're issued, we may in some

cases be able to pinch it off at the point where they're used.One way of using patents that clearly does not encourage innovation

is when established companies with bad products use patents to

suppress small competitors with good products. This is the type

of abuse we may be able to decrease without having to go through

the government.The way to do it is to get the companies that are above pulling

this sort of trick to pledge publicly not to. Then the ones that

won't make such a pledge will be very conspicuous. Potential

employees won't want to work for them. And investors, too, will

be able to see that they're the sort of company that competes by

litigation rather than by making good products.Here's the pledge:

No first use of software patents against companies with less

than 25 people.

I've deliberately traded precision for brevity. The patent pledge

is not legally binding. It's like Google's "Don't be evil." They

don't define what evil is, but by publicly saying that, they're

saying they're willing to be held to a standard that, say, Altria

is not. And though constraining, "Don't be evil" has been good for

Google. Technology companies win by attracting the most productive

people, and the most productive people are attracted to employers

who hold themselves to a higher standard than the law requires.

[1]The patent pledge is in effect a narrower but open source "Don't

be evil." I encourage every technology company to adopt it. If

you want to help fix patents, encourage your employer to.Already most technology companies wouldn't sink to using patents

on startups. You don't see Google or Facebook suing startups for

patent infringement. They don't need to. So for the better technology

companies, the patent pledge requires no change in behavior. They're

just promising to do what they'd do anyway. And when all the

companies that won't use patents on startups have said so, the

holdouts will be very conspicuous.The patent pledge doesn't fix every problem with patents. It won't

stop patent trolls, for example; they're already pariahs. But the

problem the patent pledge does fix may be more serious than the

problem of patent trolls. Patent trolls are just parasites. A

clumsy parasite may occasionally kill the host, but that's not its

goal. Whereas companies that sue startups for patent infringement

generally do it with explicit goal of keeping their product off the

market.Companies that use patents on startups are attacking innovation at

the root. Now there's something any individual can do about this

problem, without waiting for the government: ask companies where

they stand.

Patent Pledge Site

Notes:[1]

Because the pledge is deliberately vague, we're going to need

common sense when intepreting it. And even more vice versa: the

pledge is vague in order to make people use common sense when

interpreting it.So for example I've deliberately avoided saying whether the 25

people have to be employees, or whether contractors count too. If

a company has to split hairs that fine about whether a suit would

violate the patent pledge, it's probably still a dick move.The Investment That Didn't Happen

Subject: Airbnb

March 2011Yesterday Fred Wilson published a remarkable post about missing

Airbnb. VCs miss good startups all the time, but it's extraordinarily

rare for one to talk about it publicly till long afterward. So

that post is further evidence what a rare bird Fred is. He's

probably the nicest VC I know.Reading Fred's post made me go back and look at the emails I exchanged

with him at the time, trying to convince him to invest in Airbnb.

It was quite interesting to read. You can see Fred's mind at work

as he circles the deal.Fred and the Airbnb founders have generously agreed to let me publish

this email exchange (with one sentence redacted about something

that's strategically important to Airbnb and not an important part

of the conversation). It's an interesting illustration of an element

of the startup ecosystem that few except the participants ever see:

investors trying to convince one another to invest in their portfolio

companies. Hundreds if not thousands of conversations of this type

are happening now, but if one has ever been published, I haven't

seen it. The Airbnbs themselves never even saw these emails at the

time.We do a lot of this behind the scenes stuff at YC, because we invest

in such a large number of companies, and we invest so early that

investors sometimes need a lot of convincing to see their merits.

I don't always try as hard as this though. Fred must

have found me quite annoying.

from: Paul Graham

to: Fred Wilson, AirBedAndBreakfast Founders

date: Fri, Jan 23, 2009 at 11:42 AM

subject: meet the airbedsOne of the startups from the batch that just started, AirbedAndBreakfast,

is in NYC right now meeting their users. (NYC is their biggest

market.) I'd recommend meeting them if your schedule allows.I'd been thinking to myself that though these guys were going to

do really well, I should introduce them to angels, because VCs would

never go for it. But then I thought maybe I should give you more

credit. You'll certainly like meeting them. Be sure to ask about

how they funded themselves with breakfast cereal.There's no reason this couldn't be as big as Ebay. And this team

is the right one to do it.--pgfrom: Brian Chesky

to: Paul Graham

cc: Nathan Blecharczyk, Joe Gebbia

date: Fri, Jan 23, 2009 at 11:40 AM

subject: Re: meet the airbedsPG,Thanks for the intro!Brianfrom: Paul Graham

to: Brian Chesky

cc: Nathan Blecharczyk, Joe Gebbia

date: Fri, Jan 23, 2009 at 12:38 PM

subject: Re: meet the airbedsIt's a longshot, at this stage, but if there was any VC who'd get

you guys, it would be Fred. He is the least suburban-golf-playing

VC I know.He likes to observe startups for a while before acting, so don't

be bummed if he seems ambivalent.--pgfrom: Fred Wilson

to: Paul Graham,

date: Sun, Jan 25, 2009 at 5:28 PM

subject: Re: meet the airbedsThanks PaulWe are having a bit of a debate inside our partnership about the

airbed concept. We'll finish that debate tomorrow in our weekly

meeting and get back to you with our thoughtsThanksFredfrom: Paul Graham

to: Fred Wilson

date: Sun, Jan 25, 2009 at 10:48 PM

subject: Re: meet the airbedsI'd recommend having the debate after meeting them instead of before.

We had big doubts about this idea, but they vanished on meeting the

guys.from: Fred Wilson

to: Paul Graham

date: Mon, Jan 26, 2009 at 11:08 AM

subject: RE: meet the airbedsWe are still very suspect of this idea but will take a meeting as

you suggestThanksfredfrom: Fred Wilson

to: Paul Graham, AirBedAndBreakfast Founders

date: Mon, Jan 26, 2009 at 11:09 AM

subject: RE: meet the airbedsAirbed team -Are you still in NYC?We'd like to meet if you areThanksfredfrom: Paul Graham

to: Fred Wilson

date: Mon, Jan 26, 2009 at 1:42 PM

subject: Re: meet the airbedsIdeas can morph. Practically every really big startup could say,

five years later, "believe it or not, we started out doing \_\_\_."

It just seemed a very good sign to me that these guys were actually

on the ground in NYC hunting down (and understanding) their users.

On top of several previous good signs.--pgfrom: Fred Wilson

to: Paul Graham

date: Sun, Feb 1, 2009 at 7:15 AM

subject: Re: meet the airbedsIt's interestingOur two junior team members were enthusiasticThe three "old guys" didn't get itfrom: Paul Graham

to: Fred Wilson

date: Mon, Feb 9, 2009 at 5:58 PM

subject: airbnbThe Airbeds just won the first poll among all the YC startups in

their batch by a landslide. In the past this has not been a 100%

indicator of success (if only anything were) but much better than

random.--pgfrom: Fred Wilson

to: Paul Graham

date: Fri, Feb 13, 2009 at 5:29 PM

subject: Re: airbnbI met them todayThey have an interesting businessI'm just not sure how big it's going to befredfrom: Paul Graham

to: Fred Wilson

date: Sat, Feb 14, 2009 at 9:50 AM

subject: Re: airbnbDid they explain the long-term goal of being the market in accommodation

the way eBay is in stuff? That seems like it would be huge. Hotels

now are like airlines in the 1970s before they figured out how to

increase their load factors.from: Fred Wilson

to: Paul Graham

date: Tue, Feb 17, 2009 at 2:05 PM

subject: Re: airbnbThey did but I am not sure I buy thatABNB reminds me of Etsy in that it facilitates real commerce in a

marketplace model directly between two peopleSo I think it can scale all the way to the bed and breakfast marketBut I am not sure they can take on the hotel marketI could be wrongBut even so, if you include short term room rental, second home

rental, bed and breakfast, and other similar classes of accommodations,

you get to a pretty big opportunityfredfrom: Paul Graham

to: Fred Wilson

date: Wed, Feb 18, 2009 at 12:21 AM

subject: Re: airbnbSo invest in them! They're very capital efficient. They would

make an investor's money go a long way.It's also counter-cyclical. They just arrived back from NYC, and

when I asked them what was the most significant thing they'd observed,

it was how many of their users actually needed to do these rentals

to pay their rents.--pgfrom: Fred Wilson

to: Paul Graham

date: Wed, Feb 18, 2009 at 2:21 AM

subject: Re: airbnbThere's a lot to likeI've done a few things, like intro it to my friends at Foundry who

were investors in Service Metrics and understand this modelI am also talking to my friend Mark Pincus who had an idea like

this a few years ago.So we are working on itThanks for the leadFredfrom: Paul Graham

to: Fred Wilson

date: Fri, Feb 20, 2009 at 10:00 PM

subject: airbnb already spreading to prosI know you're skeptical they'll ever get hotels, but there's a

continuum between private sofas and hotel rooms, and they just moved

one step further along it.[link to an airbnb user]This is after only a few months. I bet you they will get hotels

eventually. It will start with small ones. Just wait till all the

10-room pensiones in Rome discover this site. And once it spreads

to hotels, where is the point (in size of chain) at which it stops?

Once something becomes a big marketplace, you ignore it at your

peril.--pgfrom: Fred Wilson

to: Paul Graham

date: Sat, Feb 21, 2009 at 4:26 AM

subject: Re: airbnb already spreading to prosThat's true. It's also true that there are quite a few marketplaces

out there that serve this same marketIf you look at many of the people who list at ABNB, they list

elsewhere tooI am not negative on this one, I am interested, but we are still

in the gathering data phase.fred

Founder Control

Want to start a startup? Get funded by

Y Combinator.

December 2010Someone we funded is talking to VCs now, and asked me how common

it was for a startup's founders to retain control of the board after

a series A round. He said VCs told him this almost never happened.Ten years ago that was true. In the past, founders rarely kept

control of the board through a series A. The traditional series A

board consisted of two founders, two VCs, and one independent member.

More recently the recipe is often one founder, one VC, and one

independent. In either case the founders lose their majority.But not always. Mark Zuckerberg kept control of Facebook's board

through the series A and still has it today. Mark Pincus has kept

control of Zynga's too. But are these just outliers? How common

is it for founders to keep control after an A round? I'd heard of

several cases among the companies we've funded, but I wasn't sure

how many there were, so I emailed the ycfounders list.The replies surprised me. In a dozen companies we've funded, the

founders still had a majority of the board seats after the series

A round.I feel like we're at a tipping point here. A lot of VCs still act

as if founders retaining board control after a series A is unheard-of.

A lot of them try to make you feel bad if you even ask — as if

you're a noob or a control freak for wanting such a thing. But the

founders I heard from aren't noobs or control freaks. Or if they

are, they are, like Mark Zuckerberg, the kind of noobs and control

freaks VCs should be trying to fund more of.Founders retaining control after a series A is clearly heard-of.

And barring financial catastrophe, I think in the coming year it

will become the norm.Control of a company is a more complicated matter than simply

outvoting other parties in board meetings. Investors usually get

vetos over certain big decisions, like selling the company, regardless

of how many board seats they have. And board votes are rarely

split. Matters are decided in the discussion preceding the vote,

not in the vote itself, which is usually unanimous. But if opinion

is divided in such discussions, the side that knows it would lose

in a vote will tend to be less insistent. That's what board control

means in practice. You don't simply get to do whatever you want;

the board still has to act in the interest of the shareholders; but

if you have a majority of board seats, then your opinion about

what's in the interest of the shareholders will tend to prevail.So while board control is not total control, it's not imaginary

either. There's inevitably a difference in how things feel within

the company. Which means if it becomes the norm for founders to

retain board control after a series A, that will change the way

things feel in the whole startup world.The switch to the new norm may be surprisingly fast, because the

startups that can retain control tend to be the best ones. They're

the ones that set the trends, both for other startups and for VCs.A lot of the reason VCs are harsh when negotiating with startups

is that they're embarrassed to go back to their partners looking

like they got beaten. When they sign a termsheet, they want to be

able to brag about the good terms they got. A lot of them don't

care that much personally about whether founders keep board control.

They just don't want to seem like they had to make concessions.

Which means if letting the founders keep control stops being perceived

as a concession, it will rapidly become much more common.Like a lot of changes that have been forced on VCs, this change

won't turn out to be as big a problem as they might think. VCs will

still be able to convince; they just won't be able to compel. And

the startups where they have to resort to compulsion are not the

ones that matter anyway. VCs make most of their money from a few

big hits, and those aren't them.Knowing that founders will keep control of the board may even help

VCs pick better. If they know they can't fire the founders, they'll

have to choose founders they can trust. And that's who they should

have been choosing all along.Thanks to Sam Altman, John Bautista, Trevor Blackwell, Paul

Buchheit, Brian Chesky, Bill Clerico, Patrick Collison, Adam

Goldstein, James Lindenbaum, Jessica Livingston, and Fred Wilson

for reading drafts of this.

Tablets

December 2010I was thinking recently how inconvenient it was not to have a general

term for iPhones, iPads, and the corresponding things running

Android. The closest to a general term seems to be "mobile devices,"

but that (a) applies to any mobile phone, and (b) doesn't really

capture what's distinctive about the iPad.After a few seconds it struck me that what we'll end up calling

these things is tablets. The only reason we even consider calling

them "mobile devices" is that the iPhone preceded the iPad. If the

iPad had come first, we wouldn't think of the iPhone as a phone;

we'd think of it as a tablet small enough to hold up to your ear.The iPhone isn't so much a phone as a replacement for a phone.

That's an important distinction, because it's an early instance of

what will become a common pattern. Many if not most of the

special-purpose objects around us are going to be replaced by apps

running on tablets.This is already clear in cases like GPSes, music players, and

cameras. But I think it will surprise people how many things are

going to get replaced. We funded one startup that's

replacing keys.

The fact that you can change font sizes easily means the iPad

effectively replaces reading glasses. I wouldn't be surprised if

by playing some clever tricks with the accelerometer you could even

replace the bathroom scale.The advantages of doing things in software on a single device are

so great that everything that can get turned into software will.

So for the next couple years, a good recipe for startups

will be to look around you for things that people haven't realized

yet can be made unnecessary by a tablet app.In 1938 Buckminster Fuller coined the term ephemeralization to

describe the increasing tendency of physical machinery to be replaced

by what we would now call software. The reason tablets are going

to take over the world is not (just) that Steve Jobs and Co are

industrial design wizards, but because they have this force behind

them. The iPhone and the iPad have effectively drilled a hole that

will allow ephemeralization to flow into a lot of new areas. No one

who has studied the history of technology would want to underestimate

the power of that force.I worry about the power Apple could have with this force behind

them. I don't want to see another era of client monoculture like

the Microsoft one in the 80s and 90s. But if ephemeralization is

one of the main forces driving the spread of tablets, that suggests

a way to compete with Apple: be a better platform for it.It has turned out to be a great thing that Apple tablets have

accelerometers in them. Developers have used the accelerometer in

ways Apple could never have imagined. That's the nature of platforms.

The more versatile the tool, the less you can predict how people

will use it. So tablet makers should be thinking: what else can

we put in there? Not merely hardware, but software too. What else

can we give developers access to? Give hackers an inch and they'll

take you a mile.

Thanks to Sam Altman, Paul Buchheit, Jessica Livingston, and

Robert Morris for reading drafts of this.

What We Look for in Founders

Want to start a startup? Get funded by

Y Combinator.

October 2010

(I wrote this for Forbes, who asked me to write something

about the qualities we look for in founders. In print they had to cut

the last item because they didn't have room.)1. DeterminationThis has turned out to be the most important quality in startup

founders. We thought when we started Y Combinator that the most

important quality would be intelligence. That's the myth in the

Valley. And certainly you don't want founders to be stupid. But

as long as you're over a certain threshold of intelligence, what

matters most is determination. You're going to hit a lot of

obstacles. You can't be the sort of person who gets demoralized

easily.Bill Clerico and Rich Aberman of WePay

are a good example. They're

doing a finance startup, which means endless negotiations with big,

bureaucratic companies. When you're starting a startup that depends

on deals with big companies to exist, it often feels like they're

trying to ignore you out of existence. But when Bill Clerico starts

calling you, you may as well do what he asks, because he is not

going away.

2. FlexibilityYou do not however want the sort of determination implied by phrases

like "don't give up on your dreams." The world of startups is so

unpredictable that you need to be able to modify your dreams on the

fly. The best metaphor I've found for the combination of determination

and flexibility you need is a running back.

He's determined to get

downfield, but at any given moment he may need to go sideways or

even backwards to get there.The current record holder for flexibility may be Daniel Gross of

Greplin. He applied to YC with

some bad ecommerce idea. We told

him we'd fund him if he did something else. He thought for a second,

and said ok. He then went through two more ideas before settling

on Greplin. He'd only been working on it for a couple days when

he presented to investors at Demo Day, but he got a lot of interest.

He always seems to land on his feet.

3. ImaginationIntelligence does matter a lot of course. It seems like the type

that matters most is imagination. It's not so important to be able

to solve predefined problems quickly as to be able to come up with

surprising new ideas. In the startup world, most good ideas

seem

bad initially. If they were obviously good, someone would already

be doing them. So you need the kind of intelligence that produces

ideas with just the right level of craziness.Airbnb is that kind of idea.

In fact, when we funded Airbnb, we

thought it was too crazy. We couldn't believe large numbers of

people would want to stay in other people's places. We funded them

because we liked the founders so much. As soon as we heard they'd

been supporting themselves by selling Obama and McCain branded

breakfast cereal, they were in. And it turned out the idea was on

the right side of crazy after all.

4. NaughtinessThough the most successful founders are usually good people, they

tend to have a piratical gleam in their eye. They're not Goody

Two-Shoes type good. Morally, they care about getting the big

questions right, but not about observing proprieties. That's why

I'd use the word naughty rather than evil. They delight in

breaking

rules, but not rules that matter. This quality may be redundant

though; it may be implied by imagination.Sam Altman of Loopt

is one of the most successful alumni, so we

asked him what question we could put on the Y Combinator application

that would help us discover more people like him. He said to ask

about a time when they'd hacked something to their advantage—hacked in the sense of beating the system, not breaking into

computers. It has become one of the questions we pay most attention

to when judging applications.

5. FriendshipEmpirically it seems to be hard to start a startup with just

one

founder. Most of the big successes have two or three. And the

relationship between the founders has to be strong. They must

genuinely like one another, and work well together. Startups do

to the relationship between the founders what a dog does to a sock:

if it can be pulled apart, it will be.Emmett Shear and Justin Kan of Justin.tv

are a good example of close

friends who work well together. They've known each other since

second grade. They can practically read one another's minds. I'm

sure they argue, like all founders, but I have never once sensed

any unresolved tension between them.Thanks to Jessica Livingston and Chris Steiner for reading drafts of this.

The New Funding Landscape

Want to start a startup? Get funded by

Y Combinator.

October 2010After barely changing at all for decades, the startup funding

business is now in what could, at least by comparison, be called

turmoil. At Y Combinator we've seen dramatic changes in the funding

environment for startups. Fortunately one of them is much higher

valuations.The trends we've been seeing are probably not YC-specific. I wish

I could say they were, but the main cause is probably just that we

see trends first—partly because the startups we fund are very

plugged into the Valley and are quick to take advantage of anything

new, and partly because we fund so many that we have enough data

points to see patterns clearly.What we're seeing now, everyone's probably going to be seeing in

the next couple years. So I'm going to explain what we're seeing,

and what that will mean for you if you try to raise money.Super-AngelsLet me start by describing what the world of startup funding used

to look like. There used to be two sharply differentiated types

of investors: angels and venture capitalists. Angels are individual

rich people who invest small amounts of their own money, while VCs

are employees of funds that invest large amounts of other people's.For decades there were just those two types of investors, but now

a third type has appeared halfway between them: the so-called

super-angels.

[1]

And VCs have been provoked by their arrival

into making a lot of angel-style investments themselves. So the

previously sharp line between angels and VCs has become hopelessly

blurred.There used to be a no man's land between angels and VCs. Angels

would invest $20k to $50k apiece, and VCs usually a million or more.

So an angel round meant a collection of angel investments that

combined to maybe $200k, and a VC round meant a series A round in

which a single VC fund (or occasionally two) invested $1-5 million.The no man's land between angels and VCs was a very inconvenient

one for startups, because it coincided with the amount many wanted

to raise. Most startups coming out of Demo Day wanted to raise

around $400k. But it was a pain to stitch together that much out

of angel investments, and most VCs weren't interested in investments

so small. That's the fundamental reason the super-angels have

appeared. They're responding to the market.The arrival of a new type of investor is big news for startups,

because there used to be only two and they rarely competed with one

another. Super-angels compete with both angels and VCs. That's

going to change the rules about how to raise money. I don't know

yet what the new rules will be, but it looks like most of the changes

will be for the better.A super-angel has some of the qualities of an angel, and some of

the qualities of a VC. They're usually individuals, like angels.

In fact many of the current super-angels were initially angels of

the classic type. But like VCs, they invest other people's money.

This allows them to invest larger amounts than angels: a typical

super-angel investment is currently about $100k. They make investment

decisions quickly, like angels. And they make a lot more investments

per partner than VCs—up to 10 times as many.The fact that super-angels invest other people's money makes them

doubly alarming to VCs. They don't just compete for startups; they

also compete for investors. What super-angels really are is a new

form of fast-moving, lightweight VC fund. And those of us in the

technology world know what usually happens when something comes

along that can be described in terms like that. Usually it's the

replacement.Will it be? As of now, few of the startups that take money from

super-angels are ruling out taking VC money. They're just postponing

it. But that's still a problem for VCs. Some of the startups that

postpone raising VC money may do so well on the angel money they

raise that they never bother to raise more. And those who do raise

VC rounds will be able to get higher valuations when they do. If

the best startups get 10x higher valuations when they raise series

A rounds, that would cut VCs' returns from winners at least tenfold.

[2]So I think VC funds are seriously threatened by the super-angels.

But one thing that may save them to some extent is the uneven

distribution of startup outcomes: practically all the returns are

concentrated in a few big successes. The expected value of a startup

is the percentage chance it's Google. So to the extent that winning

is a matter of absolute returns, the super-angels could win practically

all the battles for individual startups and yet lose the war, if

they merely failed to get those few big winners. And there's a

chance that could happen, because the top VC funds have better

brands, and can also do more for their portfolio companies.

[3]Because super-angels make more investments per partner, they have

less partner per investment. They can't pay as much attention to

you as a VC on your board could. How much is that extra attention

worth? It will vary enormously from one partner to another. There's

no consensus yet in the general case. So for now this is something

startups are deciding individually.Till now, VCs' claims about how much value they added were sort of

like the government's. Maybe they made you feel better, but you

had no choice in the matter, if you needed money on the scale only

VCs could supply. Now that VCs have competitors, that's going to

put a market price on the help they offer. The interesting thing

is, no one knows yet what it will be.Do startups that want to get really big need the sort of advice and

connections only the top VCs can supply? Or would super-angel money

do just as well? The VCs will say you need them, and the super-angels

will say you don't. But the truth is, no one knows yet, not even

the VCs and super-angels themselves. All the super-angels know

is that their new model seems promising enough to be worth trying,

and all the VCs know is that it seems promising enough to worry

about.RoundsWhatever the outcome, the conflict between VCs and super-angels is

good news for founders. And not just for the obvious reason that

more competition for deals means better terms. The whole shape of

deals is changing.One of the biggest differences between angels and VCs is the amount

of your company they want. VCs want a lot. In a series A round

they want a third of your company, if they can get it. They don't

care much how much they pay for it, but they want a lot because the

number of series A investments they can do is so small. In a

traditional series A investment, at least one partner from the VC

fund takes a seat on your board.

[4]

Since board seats last about

5 years and each partner can't handle more than about 10 at once,

that means a VC fund can only do about 2 series A deals per partner

per year. And that means they need to get as much of the company

as they can in each one. You'd have to be a very promising startup

indeed to get a VC to use up one of his 10 board seats for only a

few percent of you.Since angels generally don't take board seats, they don't have this

constraint. They're happy to buy only a few percent of you. And

although the super-angels are in most respects mini VC funds, they've

retained this critical property of angels. They don't take board

seats, so they don't need a big percentage of your company.Though that means you'll get correspondingly less attention from

them, it's good news in other respects. Founders never really liked

giving up as much equity as VCs wanted. It was a lot of the company

to give up in one shot. Most founders doing series A deals would

prefer to take half as much money for half as much stock, and then

see what valuation they could get for the second half of the stock

after using the first half of the money to increase its value. But

VCs never offered that option.Now startups have another alternative. Now it's easy to raise angel

rounds about half the size of series A rounds. Many of the startups

we fund are taking this route, and I predict that will be true of

startups in general.A typical big angel round might be $600k on a convertible note with

a valuation cap of $4 million premoney. Meaning that when the note

converts into stock (in a later round, or upon acquisition), the

investors in that round will get .6 / 4.6, or 13% of the company.

That's a lot less than the 30 to 40% of the company you usually

give up in a series A round if you do it so early.

[5]But the advantage of these medium-sized rounds is not just that

they cause less dilution. You also lose less control. After an

angel round, the founders almost always still have control of the

company, whereas after a series A round they often don't. The

traditional board structure after a series A round is two founders,

two VCs, and a (supposedly) neutral fifth person. Plus series A

terms usually give the investors a veto over various kinds of

important decisions, including selling the company. Founders usually

have a lot of de facto control after a series A, as long as things

are going well. But that's not the same as just being able to do

what you want, like you could before.A third and quite significant advantage of angel rounds is that

they're less stressful to raise. Raising a traditional series A

round has in the past taken weeks, if not months. When a VC firm

can only do 2 deals per partner per year, they're careful about

which they do. To get a traditional series A round you have to go

through a series of meetings, culminating in a full partner meeting

where the firm as a whole says yes or no. That's the really scary

part for founders: not just that series A rounds take so long, but

at the end of this long process the VCs might still say no. The

chance of getting rejected after the full partner meeting averages

about 25%. At some firms it's over 50%.Fortunately for founders, VCs have been getting a lot faster.

Nowadays Valley VCs are more likely to take 2 weeks than 2 months.

But they're still not as fast as angels and super-angels, the most

decisive of whom sometimes decide in hours.Raising an angel round is not only quicker, but you get feedback

as it progresses. An angel round is not an all or nothing thing

like a series A. It's composed of multiple investors with varying

degrees of seriousness, ranging from the upstanding ones who commit

unequivocally to the jerks who give you lines like "come back to

me to fill out the round." You usually start collecting money from

the most committed investors and work your way out toward the

ambivalent ones, whose interest increases as the round fills up.But at each point you know how you're doing. If investors turn

cold you may have to raise less, but when investors in an angel

round turn cold the process at least degrades gracefully, instead

of blowing up in your face and leaving you with nothing, as happens

if you get rejected by a VC fund after a full partner meeting.

Whereas if investors seem hot, you can not only close the round

faster, but now that convertible notes are becoming the norm,

actually raise the price to reflect demand.ValuationHowever, the VCs have a weapon they can use against the super-angels,

and they have started to use it. VCs have started making angel-sized

investments too. The term "angel round" doesn't mean that all the

investors in it are angels; it just describes the structure of the

round. Increasingly the participants include VCs making investments

of a hundred thousand or two. And when VCs invest in angel rounds

they can do things that super-angels don't like. VCs are quite

valuation-insensitive in angel rounds—partly because they are

in general, and partly because they don't care that much about the

returns on angel rounds, which they still view mostly as a way to

recruit startups for series A rounds later. So VCs who invest in

angel rounds can blow up the valuations for angels and super-angels

who invest in them.

[6]Some super-angels seem to care about valuations. Several turned

down YC-funded startups after Demo Day because their valuations

were too high. This was not a problem for the startups; by definition

a high valuation means enough investors were willing to accept it.

But it was mysterious to me that the super-angels would quibble

about valuations. Did they not understand that the big returns

come from a few big successes, and that it therefore mattered far

more which startups you picked than how much you paid for them?After thinking about it for a while and observing certain other

signs, I have a theory that explains why the super-angels may be

smarter than they seem. It would make sense for super-angels to

want low valuations if they're hoping to invest in startups that

get bought early. If you're hoping to hit the next Google, you

shouldn't care if the valuation is 20 million. But if you're looking

for companies that are going to get bought for 30 million, you care.

If you invest at 20 and the company gets bought for 30, you only

get 1.5x. You might as well buy Apple.So if some of the super-angels were looking for companies that could

get acquired quickly, that would explain why they'd care about

valuations. But why would they be looking for those? Because

depending on the meaning of "quickly," it could actually be very

profitable. A company that gets acquired for 30 million is a failure

to a VC, but it could be a 10x return for an angel, and moreover,

a quick 10x return. Rate of return is what matters in

investing—not the multiple you get, but the multiple per year.

If a super-angel gets 10x in one year, that's a higher rate of

return than a VC could ever hope to get from a company that took 6

years to go public. To get the same rate of return, the VC would

have to get a multiple of 10^6—one million x. Even Google

didn't come close to that.So I think at least some super-angels are looking for companies

that will get bought. That's the only rational explanation for

focusing on getting the right valuations, instead of the right

companies. And if so they'll be different to deal with than VCs.

They'll be tougher on valuations, but more accommodating if you want

to sell early.PrognosisWho will win, the super-angels or the VCs? I think the answer to

that is, some of each. They'll each become more like one another.

The super-angels will start to invest larger amounts, and the VCs

will gradually figure out ways to make more, smaller investments

faster. A decade from now the players will be hard to tell apart,

and there will probably be survivors from each group.What does that mean for founders? One thing it means is that the

high valuations startups are presently getting may not last forever.

To the extent that valuations are being driven up by price-insensitive

VCs, they'll fall again if VCs become more like super-angels and

start to become more miserly about valuations. Fortunately if this

does happen it will take years.The short term forecast is more competition between investors, which

is good news for you. The super-angels will try to undermine the

VCs by acting faster, and the VCs will try to undermine the

super-angels by driving up valuations. Which for founders will

result in the perfect combination: funding rounds that close fast,

with high valuations.But remember that to get that combination, your startup will have

to appeal to both super-angels and VCs. If you don't seem like you

have the potential to go public, you won't be able to use VCs to

drive up the valuation of an angel round.There is a danger of having VCs in an angel round: the so-called

signalling risk. If VCs are only doing it in the hope of investing

more later, what happens if they don't? That's a signal to everyone

else that they think you're lame.How much should you worry about that? The seriousness of signalling

risk depends on how far along you are. If by the next time you

need to raise money, you have graphs showing rising revenue or

traffic month after month, you don't have to worry about any signals

your existing investors are sending. Your results will speak for

themselves.

[7]Whereas if the next time you need to raise money you won't yet have

concrete results, you may need to think more about the message your

investors might send if they don't invest more. I'm not sure yet

how much you have to worry, because this whole phenomenon of VCs

doing angel investments is so new. But my instincts tell me you

don't have to worry much. Signalling risk smells like one of those

things founders worry about that's not a real problem. As a rule,

the only thing that can kill a good startup is the startup itself.

Startups hurt themselves way more often than competitors hurt them,

for example. I suspect signalling risk is in this category too.One thing YC-funded startups have been doing to mitigate the risk

of taking money from VCs in angel rounds is not to take too much

from any one VC. Maybe that will help, if you have the luxury of

turning down money.Fortunately, more and more startups will. After decades of competition

that could best be described as intramural, the startup funding

business is finally getting some real competition. That should

last several years at least, and maybe a lot longer. Unless there's

some huge market crash, the next couple years are going to be a

good time for startups to raise money. And that's exciting because

it means lots more startups will happen.

Notes[1]

I've also heard them called "Mini-VCs" and "Micro-VCs." I

don't know which name will stick.There were a couple predecessors. Ron Conway had angel funds

starting in the 1990s, and in some ways First Round Capital is closer to a

super-angel than a VC fund.[2]

It wouldn't cut their overall returns tenfold, because investing

later would probably (a) cause them to lose less on investments

that failed, and (b) not allow them to get as large a percentage

of startups as they do now. So it's hard to predict precisely what

would happen to their returns.[3]

The brand of an investor derives mostly from the success of

their portfolio companies. The top VCs thus have a big brand

advantage over the super-angels. They could make it self-perpetuating

if they used it to get all the best new startups. But I don't think

they'll be able to. To get all the best startups, you have to do

more than make them want you. You also have to want them; you have

to recognize them when you see them, and that's much harder.

Super-angels will snap up stars that VCs miss. And that will cause

the brand gap between the top VCs and the super-angels gradually

to erode.[4]

Though in a traditional series A round VCs put two partners

on your board, there are signs now that VCs may begin to conserve

board seats by switching to what used to be considered an angel-round

board, consisting of two founders and one VC. Which is also to the

founders' advantage if it means they still control the company.[5]

In a series A round, you usually have to give up more than

the actual amount of stock the VCs buy, because they insist you

dilute yourselves to set aside an "option pool" as well. I predict

this practice will gradually disappear though.[6]

The best thing for founders, if they can get it, is a convertible

note with no valuation cap at all. In that case the money invested

in the angel round just converts into stock at the valuation of the

next round, no matter how large. Angels and super-angels tend not

to like uncapped notes. They have no idea how much of the company

they're buying. If the company does well and the valuation of the

next round is high, they may end up with only a sliver of it. So

by agreeing to uncapped notes, VCs who don't care about valuations

in angel rounds can make offers that super-angels hate to match.[7]

Obviously signalling risk is also not a problem if you'll

never need to raise more money. But startups are often mistaken

about that.Thanks to Sam Altman, John Bautista, Patrick Collison, James

Lindenbaum, Reid Hoffman, Jessica Livingston and Harj Taggar

for reading drafts

of this.

Where to See Silicon Valley

Want to start a startup? Get funded by

Y Combinator.

October 2010Silicon Valley proper is mostly suburban sprawl. At first glance

it doesn't seem there's anything to see. It's not the sort of place

that has conspicuous monuments. But if you look, there are subtle

signs you're in a place that's different from other places.1. Stanford

UniversityStanford is a strange place. Structurally it is to an ordinary

university what suburbia is to a city. It's enormously spread out,

and feels surprisingly empty much of the time. But notice the

weather. It's probably perfect. And notice the beautiful mountains

to the west. And though you can't see it, cosmopolitan San Francisco

is 40 minutes to the north. That combination is much of the reason

Silicon Valley grew up around this university and not some other

one.2. University

AveA surprising amount of the work of the Valley is done in the cafes

on or just off University Ave in Palo Alto. If you visit on a

weekday between 10 and 5, you'll often see founders pitching

investors. In case you can't tell, the founders are the ones leaning

forward eagerly, and the investors are the ones sitting back with

slightly pained expressions.3. The Lucky

OfficeThe office at 165 University Ave was Google's first. Then it was

Paypal's. (Now it's Wepay's.) The interesting thing about it is

the location. It's a smart move to put a startup in a place with

restaurants and people walking around instead of in an office park,

because then the people who work there want to stay there, instead

of fleeing as soon as conventional working hours end. They go out

for dinner together, talk about ideas, and then come back and

implement them.It's important to realize that Google's current location in an

office park is not where they started; it's just where they were

forced to move when they needed more space. Facebook was till

recently across the street, till they too had to move because they

needed more space.4. Old

Palo AltoPalo Alto was not originally a suburb. For the first 100 years or

so of its existence, it was a college town out in the countryside.

Then in the mid 1950s it was engulfed in a wave of suburbia that

raced down the peninsula. But Palo Alto north of Oregon expressway

still feels noticeably different from the area around it. It's one

of the nicest places in the Valley. The buildings are old (though

increasingly they are being torn down and replaced with generic

McMansions) and the trees are tall. But houses are very

expensive—around $1000 per square foot. This is post-exit

Silicon Valley.

5. Sand

Hill RoadIt's interesting to see the VCs' offices on the north side of Sand

Hill Road precisely because they're so boringly uniform. The

buildings are all more or less the same, their exteriors express

very little, and they are arranged in a confusing maze. (I've been

visiting them for years and I still occasionally get lost.) It's

not a coincidence. These buildings are a pretty accurate reflection

of the VC business.If you go on a weekday you may see groups of founders there to meet

VCs. But mostly you won't see anyone; bustling is the last word

you'd use to describe the atmos. Visiting Sand Hill Road reminds

you that the opposite of "down and dirty" would be "up and clean."6. Castro

StreetIt's a tossup whether Castro Street or University Ave should be

considered the heart of the Valley now. University Ave would have

been 10 years ago. But Palo Alto is getting expensive. Increasingly

startups are located in Mountain View, and Palo Alto is a place

they come to meet investors. Palo Alto has a lot of different

cafes, but there is one that clearly dominates in Mountain View:

Red

Rock.7. GoogleGoogle spread out from its first building in Mountain View

to a lot of the surrounding ones. But because the

buildings were built at different times by different people,

the place doesn't have the sterile, walled-off feel that a typical

large company's headquarters have. It definitely has a flavor of

its own though. You sense there is something afoot. The general

atmos is vaguely utopian; there are lots of Priuses, and people who

look like they drive them.You can't get into Google unless you know someone there. It's very

much worth seeing inside if you can, though. Ditto for Facebook,

at the end of California Ave in Palo Alto, though there is nothing

to see outside.8. Skyline

DriveSkyline Drive runs along the crest of the Santa Cruz mountains. On

one side is the Valley, and on the other is the sea—which

because it's cold and foggy and has few harbors, plays surprisingly

little role in the lives of people in the Valley, considering how

close it is. Along some parts of Skyline the dominant trees are

huge redwoods, and in others they're live oaks. Redwoods mean those

are the parts where the fog off the coast comes in at night; redwoods

condense rain out of fog. The MROSD manages a collection of great walking trails off

Skyline.9. 280Silicon Valley has two highways running the length of it: 101, which

is pretty ugly, and 280, which is one of the more beautiful highways

in the world. I always take 280 when I have a choice. Notice the

long narrow lake to the west? That's the San Andreas Fault. It

runs along the base of the hills, then heads uphill through Portola

Valley. One of the MROSD trails runs right along

the fault. A string of rich neighborhoods runs along the

foothills to the west of 280: Woodside, Portola Valley, Los Altos

Hills, Saratoga, Los Gatos.SLAC goes right under 280 a little bit south of Sand Hill Road. And a couple miles south of that is the Valley's equivalent of the "Welcome to Las Vegas" sign: The Dish.

NotesI skipped the Computer

History Museum because this is a list of where to see the Valley

itself, not where to see artifacts from it. I also skipped San

Jose. San Jose calls itself the capital of Silicon Valley, but

when people in the Valley use the phrase "the city," they mean San

Francisco. San Jose is a dotted line on a map.Thanks to Sam Altman, Paul Buchheit, Patrick Collison, and Jessica Livingston

for reading drafts of this.

High Resolution Fundraising

Want to start a startup? Get funded by

Y Combinator.

September 2010The reason startups have been using

more convertible notes in angel

rounds is that they make deals close faster. By making it easier

for startups to give different prices to different investors, they

help them break the sort of deadlock that happens when investors

all wait to see who else is going to invest.By far the biggest influence on investors' opinions of a startup

is the opinion of other investors. There are very, very few who

simply decide for themselves. Any startup founder can tell you the

most common question they hear from investors is not about the

founders or the product, but "who else is investing?"That tends to produce deadlocks. Raising an old-fashioned

fixed-size equity round can take weeks, because all the angels sit around

waiting for the others to commit, like competitors in a bicycle

sprint who deliberately ride slowly at the start so they can follow

whoever breaks first.Convertible notes let startups beat such deadlocks by rewarding

investors willing to move first with lower (effective) valuations.

Which they deserve because they're taking more risk. It's much

safer to invest in a startup Ron Conway has already invested in;

someone who comes after him should pay a higher price.The reason convertible notes allow more flexibility in price is

that valuation caps aren't actual valuations, and notes are cheap

and easy to do. So you can do high-resolution fundraising: if you

wanted you could have a separate note with a different cap for each

investor.That cap need not simply rise monotonically. A startup could

also give better deals to investors they expected to help

them most. The point is simply that different investors,

whether because of the help they offer or their willingness to

commit, have different values for

startups, and their terms should reflect that.Different terms for different investors is

clearly the way of the future. Markets always evolve toward higher

resolution. You may not need to use convertible notes to do it.

With sufficiently lightweight standardized equity terms (and some

changes in investors' and lawyers' expectations about equity rounds)

you might be able to do the same thing with equity instead of debt.

Either would be fine with startups, so long as they can easily

change their valuation.Deadlocks weren't the only problem with fixed-size equity rounds.

Another was that startups had to decide in advance how much to

raise. I think it's a mistake for a startup to fix upon a specific

number. If investors are easily convinced, the startup should raise more

now, and if investors are skeptical, the startup should take a

smaller amount and use that to get the company to the point where

it's more convincing.It's just not reasonable to expect startups to pick an optimal round

size in advance, because that depends on the reactions of investors,

and those are impossible to predict.Fixed-size, multi-investor angel rounds are such a bad idea for

startups that one wonders why things were ever done that way. One

possibility is that this custom reflects the way investors like to

collude when they can get away with it. But I think the actual

explanation is less sinister. I think angels (and their lawyers)

organized rounds this way in unthinking imitation of VC series A

rounds. In a series A, a fixed-size equity round with a lead makes

sense, because there is usually just one big investor, who is

unequivocally the lead. Fixed-size series A rounds already are

high res. But the more investors you have in a round, the less

sense it makes for everyone to get the same price.The most interesting question here may be what high res fundraising

will do to the world of investors. Bolder investors will now get

rewarded with lower prices. But more important, in a

hits-driven business, is that they'll be able to get into the deals

they want. Whereas the "who else is investing?" type of investors

will not only pay higher prices, but may not be able to get into

the best deals at all.Thanks to Immad Akhund, Sam Altman, John Bautista, Pete Koomen,

Jessica Livingston, Dan Siroker, Harj Taggar, and

Fred Wilson for reading drafts of this.

What Happened to Yahoo

Want to start a startup? Get funded by

Y Combinator.

August 2010When I went to work for Yahoo after they bought our startup in 1998,

it felt like the center of the world. It was supposed to be the

next big thing. It was supposed to be what Google turned out to

be.What went wrong? The problems that hosed Yahoo go back a long time,

practically to the beginning of the company. They were already

very visible when I got there in 1998. Yahoo had two problems

Google didn't: easy money, and ambivalence about being a technology

company.MoneyThe first time I met Jerry Yang, we thought we were meeting for

different reasons. He thought we were meeting so he could check

us out in person before buying us. I thought we were meeting so we

could show him our new technology, Revenue Loop. It was a way of

sorting shopping search results. Merchants bid a percentage of

sales for traffic, but the results were sorted not by the bid but

by the bid times the average amount a user would buy. It was

like the algorithm Google uses now to sort ads, but this was in the

spring of 1998, before Google was founded.Revenue Loop was the optimal sort for shopping search, in the sense

that it sorted in order of how much money Yahoo would make from

each link. But it wasn't just optimal in that sense. Ranking

search results by user behavior also makes search better. Users

train the search: you can start out finding matches based on mere

textual similarity, and as users buy more stuff the search results

get better and better.Jerry didn't seem to care. I was confused. I was showing him

technology that extracted the maximum value from search traffic,

and he didn't care? I couldn't tell whether I was explaining it

badly, or he was just very poker faced.I didn't realize the answer till later, after I went to work at

Yahoo. It was neither of my guesses. The reason Yahoo didn't care

about a technique that extracted the full value of traffic was that

advertisers were already overpaying for it. If Yahoo merely extracted

the actual value, they'd have made less.Hard as it is to believe now, the big money then was in banner ads.

Advertisers were willing to pay ridiculous amounts for banner ads.

So Yahoo's sales force had evolved to exploit this source of revenue.

Led by a large and terrifyingly formidable man called Anil Singh,

Yahoo's sales guys would fly out to Procter & Gamble and come back

with million dollar orders for banner ad impressions.The prices seemed cheap compared to print, which was what advertisers,

for lack of any other reference, compared them to. But they were

expensive compared to what they were worth. So these big, dumb

companies were a dangerous source of revenue to depend on. But

there was another source even more dangerous: other Internet startups.By 1998, Yahoo was the beneficiary of a de facto Ponzi scheme.

Investors were excited about the Internet. One reason they were

excited was Yahoo's revenue growth. So they invested in new Internet

startups. The startups then used the money to buy ads on Yahoo to

get traffic. Which caused yet more revenue growth for Yahoo, and

further convinced investors the Internet was worth investing in.

When I realized this one day, sitting in my cubicle, I jumped up

like Archimedes in his bathtub, except instead of "Eureka!" I was

shouting "Sell!"Both the Internet startups and the Procter & Gambles were doing

brand advertising. They didn't care about targeting. They just

wanted lots of people to see their ads. So traffic became the thing

to get at Yahoo. It didn't matter what type.

[1]It wasn't just Yahoo. All the search engines were doing it. This

was why they were trying to get people to start calling them "portals"

instead of "search engines." Despite the actual meaning of the word

portal, what they meant by it was a site where users would find

what they wanted on the site itself, instead of just passing through

on their way to other destinations, as they did at a search engine.I remember telling David Filo in late 1998 or early 1999 that Yahoo

should buy Google, because I and most of the other programmers in

the company were using it instead of Yahoo for search. He told me

that it wasn't worth worrying about. Search was only 6% of our

traffic, and we were growing at 10% a month. It wasn't worth doing

better.I didn't say "But search traffic is worth more than other traffic!"

I said "Oh, ok." Because I didn't realize either how much search

traffic was worth. I'm not sure even Larry and Sergey did then.

If they had, Google presumably wouldn't have expended any effort

on enterprise search.If circumstances had been different, the people running Yahoo might

have realized sooner how important search was. But they had the

most opaque obstacle in the world between them and the truth: money.

As long as customers were writing big checks for banner ads, it was

hard to take search seriously. Google didn't have that to distract

them.HackersBut Yahoo also had another problem that made it hard to change

directions. They'd been thrown off balance from the start by their

ambivalence about being a technology company.One of the weirdest things about Yahoo when I went to work there

was the way they insisted on calling themselves a "media company."

If you walked around their offices, it seemed like a software

company. The cubicles were full of programmers writing code, product

managers thinking about feature lists and ship dates, support people

(yes, there were actually support people) telling users to restart

their browsers, and so on, just like a software company. So why

did they call themselves a media company?One reason was the way they made money: by selling ads. In 1995

it was hard to imagine a technology company making money that way.

Technology companies made money by selling their software to users.

Media companies sold ads. So they must be a media company.Another big factor was the fear of Microsoft. If anyone at Yahoo

considered the idea that they should be a technology company, the

next thought would have been that Microsoft would crush them.It's hard for anyone much younger than me to understand the fear

Microsoft still inspired in 1995. Imagine a company with several

times the power Google has now, but way meaner. It was perfectly

reasonable to be afraid of them. Yahoo watched them crush the first

hot Internet company, Netscape. It was reasonable to worry that

if they tried to be the next Netscape, they'd suffer the same fate.

How were they to know that Netscape would turn out to be Microsoft's

last victim?It would have been a clever move to pretend to be a media company

to throw Microsoft off their scent. But unfortunately Yahoo actually

tried to be one, sort of. Project managers at Yahoo were called

"producers," for example, and the different parts of the company

were called "properties." But what Yahoo really needed to be was a

technology company, and by trying to be something else, they ended

up being something that was neither here nor there. That's why

Yahoo as a company has never had a sharply defined identity.The worst consequence of trying to be a media company was that they

didn't take programming seriously enough. Microsoft (back in the

day), Google, and Facebook have all had hacker-centric cultures.

But Yahoo treated programming as a commodity. At Yahoo, user-facing software

was controlled by product managers and designers. The job of

programmers was just to take the work of the product managers and

designers the final step, by translating it into code.One obvious result of this practice was that when Yahoo built things,

they often weren't very good. But that wasn't the worst problem.

The worst problem was that they hired bad programmers.Microsoft (back in the day), Google, and Facebook have all been

obsessed with hiring the best programmers. Yahoo wasn't. They

preferred good programmers to bad ones, but they didn't have the

kind of single-minded, almost obnoxiously elitist focus on hiring

the smartest people that the big winners have had. And when you

consider how much competition there was for programmers when they

were hiring, during the Bubble, it's not surprising that the quality

of their programmers was uneven.In technology, once you have bad programmers, you're doomed. I

can't think of an instance where a company has sunk into technical

mediocrity and recovered. Good programmers want to work with other

good programmers. So once the quality of programmers at your company

starts to drop, you enter a death spiral from which there is no

recovery.

[2]At Yahoo this death spiral started early. If there was ever a time when

Yahoo was a Google-style talent magnet, it was over by the time I

got there in 1998.The company felt prematurely old. Most technology companies

eventually get taken over by suits and middle managers. At Yahoo

it felt as if they'd deliberately accelerated this process. They

didn't want to be a bunch of hackers. They wanted to be suits. A

media company should be run by suits.The first time I visited Google, they had about 500 people, the

same number Yahoo had when I went to work there. But boy did things

seem different. It was still very much a hacker-centric culture.

I remember talking to some programmers in the cafeteria about the

problem of gaming search results (now known as SEO), and they asked

"what should we do?" Programmers at Yahoo wouldn't have asked that.

Theirs was not to reason why; theirs was to build what product

managers spec'd. I remember coming away from Google thinking "Wow,

it's still a startup."There's not much we can learn from Yahoo's first fatal flaw. It's

probably too much to hope any company could avoid being damaged by

depending on a bogus source of revenue. But startups can learn an

important lesson from the second one. In the software business,

you can't afford not to have a hacker-centric culture.Probably the most impressive commitment I've heard to having a

hacker-centric culture came from Mark Zuckerberg, when he spoke at

Startup School in 2007. He said that in the early days Facebook

made a point of hiring programmers even for jobs that would not

ordinarily consist of programming, like HR and marketing.So which companies need to have a hacker-centric culture? Which

companies are "in the software business" in this respect? As Yahoo

discovered, the area covered by this rule is bigger than most people

realize. The answer is: any company that needs to have good software.Why would great programmers want to work for a company that didn't

have a hacker-centric culture, as long as there were others that

did? I can imagine two reasons: if they were paid a huge amount,

or if the domain was interesting and none of the companies in it

were hacker-centric. Otherwise you can't attract good programmers

to work in a suit-centric culture. And without good programmers

you won't get good software, no matter how many people you put on

a task, or how many procedures you establish to ensure "quality."Hacker culture

often seems kind of irresponsible. That's why people

proposing to destroy it use phrases like "adult supervision." That

was the phrase they used at Yahoo. But there are worse things than

seeming irresponsible. Losing, for example.

Notes[1]

The closest we got to targeting when I was there was when we

created pets.yahoo.com in order to provoke a bidding war between 3

pet supply startups for the spot as top sponsor.[2]

In theory you could beat the death spiral by buying good

programmers instead of hiring them. You can get programmers

who would never have come to you as employees by buying their

startups. But so far the only companies smart enough

to do this are companies smart enough not to need to.Thanks to Trevor Blackwell, Jessica Livingston, and

Geoff Ralston for

reading drafts of this.

The Future of Startup Funding

Want to start a startup? Get funded by

Y Combinator.

August 2010Two years ago I

wrote about what I called "a huge, unexploited

opportunity in startup funding:" the growing disconnect between

VCs, whose current business model requires them to invest large

amounts, and a large class of startups that need less than they

used to. Increasingly, startups want a couple hundred thousand

dollars, not a couple million.

[1]The opportunity is a lot less unexploited now. Investors have

poured into this territory from both directions. VCs are much more

likely to make angel-sized investments than they were a year ago.

And meanwhile the past year has seen a dramatic increase in a new

type of investor: the super-angel, who operates like an angel, but

using other people's money, like a VC.Though a lot of investors are entering this territory, there is

still room for more. The distribution of investors should mirror

the distribution of startups, which has the usual power law dropoff.

So there should be a lot more people investing tens or hundreds of

thousands than millions.

[2]In fact, it may be good for angels that there are more people doing

angel-sized deals, because if angel rounds become more legitimate,

then startups may start to opt for angel rounds even when they

could, if they wanted, raise series A rounds from VCs. One reason

startups prefer series A rounds is that they're more prestigious.

But if angel investors become more active and better known, they'll

increasingly be able to compete with VCs in brand.Of course, prestige isn't the main reason to prefer a series A

round. A startup will probably get more attention from investors

in a series A round than an angel round. So if a startup is choosing

between an angel round and an A round from a good VC fund, I usually

advise them to take the A round.

[3]But while series A rounds aren't going away, I think VCs should be

more worried about super-angels than vice versa. Despite their

name, the super-angels are really mini VC funds, and they clearly

have existing VCs in their sights.They would seem to have history on their side.

The pattern here seems the same

one we see when startups and established companies enter a new

market. Online video becomes possible, and YouTube plunges right

in, while existing media companies embrace it only half-willingly,

driven more by fear than hope, and aiming more to protect their

turf than to do great things for users. Ditto for PayPal. This

pattern is repeated over and over, and it's usually the invaders

who win. In this case the super-angels are the invaders. Angel

rounds are their whole business, as online video was for YouTube.

Whereas VCs who make angel investments mostly do it as a way to

generate deal flow for series A rounds.

[4]On the other hand, startup investing is a very strange business.

Nearly all the returns are concentrated in a few big winners. If

the super-angels merely fail to invest in (and to some extent

produce) the big winners, they'll be out of business, even if they

invest in all the others.VCsWhy don't VCs start doing smaller series A rounds? The sticking

point is board seats. In a traditional series A round, the partner

whose deal it is takes a seat on the startup's board. If we assume

the average startup runs for 6 years and a partner can bear to be

on 12 boards at once, then a VC fund can do 2 series A deals per

partner per year.It has always seemed to me the solution is to take fewer board

seats. You don't have to be on the board to help a startup. Maybe

VCs feel they need the power that comes with board membership to

ensure their money isn't wasted. But have they tested that theory?

Unless they've tried not taking board seats and found their returns

are lower, they're not bracketing the problem.I'm not saying VCs don't help startups. The good ones help them a

lot. What I'm saying is that the kind of help that matters, you

may not have to be a board member to give.

[5]How will this all play out? Some VCs will probably adapt, by doing

more, smaller deals. I wouldn't be surprised if by streamlining

their selection process and taking fewer board seats, VC funds could

do 2 to 3 times as many series A rounds with no loss of quality.But other VCs will make no more than superficial changes. VCs are

conservative, and the threat to them isn't mortal. The VC funds

that don't adapt won't be violently displaced. They'll edge gradually

into a different business without realizing it. They'll still do

what they will call series A rounds, but these will increasingly

be de facto series B rounds.

[6]In such rounds they won't get the 25 to 40% of the company they do

now. You don't give up as much of the company in later rounds

unless something is seriously wrong. Since the VCs who don't adapt

will be investing later, their returns from winners may be smaller.

But investing later should also mean they have fewer losers. So

their ratio of risk to return may be the same or even better.

They'll just have become a different, more conservative, type of

investment.AngelsIn the big angel rounds that increasingly compete with series A

rounds, the investors won't take as much equity as VCs do now. And

VCs who try to compete with angels by doing more, smaller deals

will probably find they have to take less equity to do it. Which

is good news for founders: they'll get to keep more of the company.The deal terms of angel rounds will become less restrictive

too—not just less restrictive than series A terms, but less

restrictive than angel terms have traditionally been.In the future, angel rounds will less often be for specific amounts

or have a lead investor. In the old days, the standard m.o. for

startups was to find one angel to act as the lead investor. You'd

negotiate a round size and valuation with the lead, who'd supply

some but not all of the money. Then the startup and the lead would

cooperate to find the rest.The future of angel rounds looks more like this: instead of a fixed

round size, startups will do a rolling close, where they take money

from investors one at a time till they feel they have enough.

[7]

And though there's going to be one investor who gives them the first

check, and his or her help in recruiting other investors will

certainly be welcome, this initial investor will no longer be the

lead in the old sense of managing the round. The startup will now

do that themselves.There will continue to be lead investors in the sense of investors

who take the lead in advising a startup. They may also make

the biggest investment. But they won't always have to be the one

terms are negotiated with, or be the first money in, as they have

in the past. Standardized paperwork will do away with the need to

negotiate anything except the valuation, and that will get easier

too.If multiple investors have to share a valuation, it will be whatever

the startup can get from the first one to write a check, limited

by their guess at whether this will make later investors balk. But

there may not have to be just one valuation. Startups are increasingly

raising money on convertible notes, and convertible notes have not

valuations but at most valuation caps: caps on what the

effective valuation will be when the debt converts to equity (in a

later round, or upon acquisition if that happens first). That's

an important difference because it means a startup could do multiple

notes at once with different caps. This is now starting to happen,

and I predict it will become more common.SheepThe reason things are moving this way is that the old way sucked

for startups. Leads could (and did) use a fixed size round as a

legitimate-seeming way of saying what all founders hate to hear:

I'll invest if other people will. Most investors, unable to judge

startups for themselves, rely instead on the opinions of other

investors. If everyone wants in, they want in too; if not, not.

Founders hate this because it's a recipe for deadlock, and delay

is the thing a startup can least afford. Most investors know this

m.o. is lame, and few say openly that they're doing it. But the

craftier ones achieve the same result by offering to lead rounds

of fixed size and supplying only part of the money. If the startup

can't raise the rest, the lead is out too. How could they go ahead

with the deal? The startup would be underfunded!In the future, investors will increasingly be unable to offer

investment subject to contingencies like other people investing.

Or rather, investors who do that will get last place in line.

Startups will go to them only to fill up rounds that are mostly

subscribed. And since hot startups tend to have rounds that are

oversubscribed, being last in line means they'll probably miss the

hot deals. Hot deals and successful startups are not identical,

but there is a significant correlation.

[8]

So investors who won't invest unilaterally will have lower returns.Investors will probably find they do better when deprived of this

crutch anyway. Chasing hot deals doesn't make investors choose

better; it just makes them feel better about their choices. I've

seen feeding frenzies both form and fall apart many times, and as

far as I can tell they're mostly random.

[9]

If investors can

no longer rely on their herd instincts, they'll have to think more

about each startup before investing. They may be surprised how

well this works.Deadlock wasn't the only disadvantage of letting a lead investor

manage an angel round. The investors would not infrequently collude

to push down the valuation. And rounds took too long to close,

because however motivated the lead was to get the round closed, he

was not a tenth as motivated as the startup.Increasingly, startups are taking charge of their own angel rounds.

Only a few do so far, but I think we can already declare the old

way dead, because those few are the best startups. They're the

ones in a position to tell investors how the round is going to work.

And if the startups you want to invest in do things a certain way,

what difference does it make what the others do?TractionIn fact, it may be slightly misleading to say that angel rounds

will increasingly take the place of series A rounds. What's really

happening is that startup-controlled rounds are taking the place

of investor-controlled rounds.This is an instance of a very important meta-trend, one that Y

Combinator itself has been based on from the beginning: founders

are becoming increasingly powerful relative to investors. So if

you want to predict what the future of venture funding will be like,

just ask: how would founders like it to be? One by one, all the

things founders dislike about raising money are going to get

eliminated.

[10]Using that heuristic, I'll predict a couple more things. One is

that investors will increasingly be unable to wait for startups to

have "traction" before they put in significant money. It's hard

to predict in advance which startups will succeed. So most investors

prefer, if they can, to wait till the startup is already succeeding,

then jump in quickly with an offer. Startups hate this as well,

partly because it tends to create deadlock, and partly because it

seems kind of slimy. If you're a promising startup but don't yet

have significant growth, all the investors are your friends in

words, but few are in actions. They all say they love you, but

they all wait to invest. Then when you start to see growth, they

claim they were your friend all along, and are aghast at the thought

you'd be so disloyal as to leave them out of your round. If founders

become more powerful, they'll be able to make investors give them

more money upfront.(The worst variant of this behavior is the tranched deal, where the

investor makes a small initial investment, with more to follow if

the startup does well. In effect, this structure gives the investor

a free option on the next round, which they'll only take if it's

worse for the startup than they could get in the open market.

Tranched deals are an abuse. They're increasingly rare, and they're

going to get rarer.)

[11]Investors don't like trying to predict which startups will succeed,

but increasingly they'll have to. Though the way that happens won't

necessarily be that the behavior of existing investors will change;

it may instead be that they'll be replaced by other investors with

different behavior—that investors who understand startups

well enough to take on the hard problem of predicting their trajectory

will tend to displace suits whose skills lie more in raising money

from LPs.SpeedThe other thing founders hate most about fundraising is how long

it takes. So as founders become more powerful, rounds should start

to close faster.Fundraising is still terribly distracting for startups. If you're

a founder in the middle of raising a round, the round is the top idea in your mind, which means working on the

company isn't. If a round takes 2 months to close, which is

reasonably fast by present standards, that means 2 months during

which the company is basically treading water. That's the worst

thing a startup could do.So if investors want to get the best deals, the way to do it will

be to close faster. Investors don't need weeks to make up their

minds anyway. We decide based on about 10 minutes of reading an

application plus 10 minutes of in person interview, and we only

regret about 10% of our decisions. If we can decide in 20 minutes,

surely the next round of investors can decide in a couple days.

[12]There are a lot of institutionalized delays in startup funding: the

multi-week mating dance with investors; the distinction between

termsheets and deals; the fact that each series A has enormously

elaborate, custom paperwork. Both founders and investors tend to

take these for granted. It's the way things have always been. But

ultimately the reason these delays exist is that they're to the

advantage of investors. More time gives investors more information

about a startup's trajectory, and it also tends to make startups

more pliable in negotiations, since they're usually short of money.These conventions weren't designed to drag out the funding process,

but that's why they're allowed to persist. Slowness is to the

advantage of investors, who have in the past been the ones with the

most power. But there is no need for rounds to take months or even

weeks to close, and once founders realize that, it's going to stop.

Not just in angel rounds, but in series A rounds too. The future

is simple deals with standard terms, done quickly.One minor abuse that will get corrected in the process is option

pools. In a traditional series A round, before the VCs invest they

make the company set aside a block of stock for future hires—usually

between 10 and 30% of the company. The point is to ensure this

dilution is borne by the existing shareholders. The practice isn't

dishonest; founders know what's going on. But it makes deals

unnecessarily complicated. In effect the valuation is 2 numbers.

There's no need to keep doing this.

[13]The final thing founders want is to be able to sell some of

their own stock in later rounds. This won't be a change,

because the practice is now quite common. A lot of investors

hated the idea, but the world hasn't exploded as a result,

so it will happen more, and more openly.SurpriseI've talked here about a bunch of changes that will be forced on

investors as founders become more powerful. Now the good news:

investors may actually make more money as a result.A couple days ago an interviewer

asked

me if founders having more

power would be better or worse for the world. I was surprised,

because I'd never considered that question. Better or worse, it's

happening. But after a second's reflection, the answer seemed

obvious. Founders understand their companies better than investors,

and it has to be better if the people with more knowledge have more

power.One of the mistakes novice pilots make is overcontrolling the

aircraft: applying corrections too vigorously, so the aircraft

oscillates about the desired configuration instead of approaching

it asymptotically. It seems probable that investors have till now

on average been overcontrolling their portfolio companies. In a

lot of startups, the biggest source of stress for the founders is

not competitors but investors. Certainly it was for us at Viaweb.

And this is not a new phenomenon: investors were James Watt's biggest

problem too. If having less power prevents investors from

overcontrolling startups, it should be better not just for founders

but for investors too.Investors may end up with less stock per startup, but startups will

probably do better with founders more in control, and there will

almost certainly be more of them. Investors all compete with one

another for deals, but they aren't one another's main competitor.

Our main competitor is employers. And so far that competitor is

crushing us. Only a tiny fraction of people who could start a

startup do. Nearly all customers choose the competing product, a

job. Why? Well, let's look at the product we're offering. An

unbiased review would go something like this:

Starting a startup gives you more freedom and the opportunity to

make a lot more money than a job, but it's also hard work and at

times very stressful.

Much of the stress comes from dealing with investors. If reforming

the investment process removed that stress, we'd make our product

much more attractive. The kind of people who make good startup

founders don't mind dealing with technical problems—they enjoy

technical problems—but they hate the type of problems investors

cause.Investors have no

idea that when they maltreat one startup, they're preventing 10

others from happening, but they are. Indirectly, but they are. So

when investors stop trying to squeeze a little more out of their

existing deals, they'll find they're net ahead, because so many

more new deals appear.One of our axioms at Y Combinator is not to think of deal flow as

a zero-sum game. Our main focus is to encourage more startups to happen,

not to win a larger share of the existing stream. We've found this

principle very useful, and we think as it spreads outward it will

help later stage investors as well."Make something people want"

applies to us too.Notes[1]

In this essay I'm talking mainly about software startups.

These points don't apply to types of startups that are still expensive

to start, e.g. in energy or biotech.Even the cheap kinds of startups will generally raise large amounts

at some point, when they want to hire a lot of people. What has

changed is how much they can get done before that.[2]

It's not the distribution of good startups that has a power

law dropoff, but the distribution of potentially good startups,

which is to say, good deals. There are lots of potential winners,

from which a few actual winners emerge with superlinear certainty.[3]

As I was writing this, I asked some founders who'd taken

series A rounds from top VC funds whether it was worth it, and they

unanimously said yes.The quality of investor is more important than the type of round,

though. I'd take an angel round from good angels over a series A

from a mediocre VC.[4]

Founders also worry that taking an angel investment from a

VC means they'll look bad if the VC declines to participate in the

next round. The trend of VC angel investing is so new that it's

hard to say how justified this worry is.Another danger, pointed out by Mitch Kapor, is that if VCs are only

doing angel deals to generate series A deal flow, then their

incentives aren't aligned with the founders'. The founders want

the valuation of the next round to be high, and the VCs want it to

be low. Again, hard to say yet how much of a problem this will be.[5]

Josh Kopelman pointed out that another way to be on fewer

boards at once is to take board seats for shorter periods.[6]

Google was in this respect as so many others the pattern for

the future. It would be great for VCs if the similarity extended

to returns. That's probably too much to hope for, but the returns

may be somewhat higher, as I explain later.[7]

Doing a rolling close doesn't mean the company is always

raising money. That would be a distraction. The point of a rolling

close is to make fundraising take less time, not more. With a

classic fixed sized round, you don't get any money till all the

investors agree, and that often creates a situation where they all

sit waiting for the others to act. A rolling close usually prevents

this.

[8]

There are two (non-exclusive) causes of hot deals: the quality

of the company, and domino effects among investors. The former is

obviously a better predictor of success.[9]

Some of the randomness is concealed by the fact that investment

is a self fulfilling prophecy.[10]

The shift in power to founders is exaggerated now because

it's a seller's market. On the next downtick it will seem like I

overstated the case. But on the next uptick after that, founders

will seem more powerful than ever.[11]

More generally, it will become less common for the same

investor to invest in successive rounds, except when exercising an

option to maintain their percentage. When the same investor invests

in successive rounds, it often means the startup isn't getting

market price. They may not care; they may prefer to work with an

investor they already know; but as the investment market becomes

more efficient, it will become increasingly easy to get market price

if they want it. Which in turn means the investment community will

tend to become more stratified.[12]

The two 10 minuteses have 3 weeks between them so founders

can get cheap plane tickets, but except for that they could be

adjacent.[13]

I'm not saying option pools themselves will go away. They're

an administrative convenience. What will go away is investors

requiring them.

Thanks to Sam Altman, John Bautista, Trevor Blackwell,

Paul Buchheit, Jeff Clavier,

Patrick Collison, Ron Conway, Matt Cohler, Chris Dixon, Mitch Kapor,

Josh Kopelman, Pete Koomen, Carolynn Levy, Jessica Livingston, Ariel

Poler, Geoff Ralston, Naval Ravikant, Dan Siroker, Harj Taggar, and

Fred Wilson

for reading drafts of this.

The Acceleration of Addictiveness

July 2010What hard liquor, cigarettes, heroin, and crack have in common is

that they're all more concentrated forms of less addictive predecessors.

Most if not all the things we describe as addictive are. And the

scary thing is, the process that created them is accelerating.We wouldn't want to stop it. It's the same process that cures

diseases: technological progress. Technological progress means

making things do more of what we want. When the thing we want is

something we want to want, we consider technological progress good.

If some new technique makes solar cells x% more efficient, that

seems strictly better. When progress concentrates something we

don't want to want—when it transforms opium into heroin—it seems

bad. But it's the same process at work.

[1]No one doubts this process is accelerating, which means increasing

numbers of things we like will be transformed into things we like

too much.

[2]As far as I know there's no word for something we like too much.

The closest is the colloquial sense of "addictive." That usage has

become increasingly common during my lifetime. And it's clear why:

there are an increasing number of things we need it for. At the

extreme end of the spectrum are crack and meth. Food has been

transformed by a combination of factory farming and innovations in

food processing into something with way more immediate bang for the

buck, and you can see the results in any town in America. Checkers

and solitaire have been replaced by World of Warcraft and FarmVille.

TV has become much more engaging, and even so it can't compete with Facebook.The world is more addictive than it was 40 years ago. And unless

the forms of technological progress that produced these things are

subject to different laws than technological progress in general,

the world will get more addictive in the next 40 years than it did

in the last 40.The next 40 years will bring us some wonderful things. I don't

mean to imply they're all to be avoided. Alcohol is a dangerous

drug, but I'd rather live in a world with wine than one without.

Most people can coexist with alcohol; but you have to be careful.

More things we like will mean more things we have to be careful

about.Most people won't, unfortunately. Which means that as the world

becomes more addictive, the two senses in which one can live a

normal life will be driven ever further apart. One sense of "normal"

is statistically normal: what everyone else does. The other is the

sense we mean when we talk about the normal operating range of a

piece of machinery: what works best.These two senses are already quite far apart. Already someone

trying to live well would seem eccentrically abstemious in most of

the US. That phenomenon is only going to become more pronounced.

You can probably take it as a rule of thumb from now on that if

people don't think you're weird, you're living badly.Societies eventually develop antibodies to addictive new things.

I've seen that happen with cigarettes. When cigarettes first

appeared, they spread the way an infectious disease spreads through

a previously isolated population. Smoking rapidly became a

(statistically) normal thing. There were ashtrays everywhere. We

had ashtrays in our house when I was a kid, even though neither of

my parents smoked. You had to for guests.As knowledge spread about the dangers of smoking, customs changed.

In the last 20 years, smoking has been transformed from something

that seemed totally normal into a rather seedy habit: from something

movie stars did in publicity shots to something small huddles of

addicts do outside the doors of office buildings. A lot of the

change was due to legislation, of course, but the legislation

couldn't have happened if customs hadn't already changed.It took a while though—on the order of 100 years. And unless the

rate at which social antibodies evolve can increase to match the

accelerating rate at which technological progress throws off new

addictions, we'll be increasingly unable to rely on customs to

protect us.

[3]

Unless we want to be canaries in the coal mine

of each new addiction—the people whose sad example becomes a

lesson to future generations—we'll have to figure out for ourselves

what to avoid and how. It will actually become a reasonable strategy

(or a more reasonable strategy) to suspect

everything new.In fact, even that won't be enough. We'll have to worry not just

about new things, but also about existing things becoming more

addictive. That's what bit me. I've avoided most addictions, but

the Internet got me because it became addictive while I was using

it.

[4]Most people I know have problems with Internet addiction. We're

all trying to figure out our own customs for getting free of it.

That's why I don't have an iPhone, for example; the last thing I

want is for the Internet to follow me out into the world.

[5]

My latest trick is taking long hikes. I used to think running was a

better form of exercise than hiking because it took less time. Now

the slowness of hiking seems an advantage, because the longer I

spend on the trail, the longer I have to think without interruption.Sounds pretty eccentric, doesn't it? It always will when you're

trying to solve problems where there are no customs yet to guide

you. Maybe I can't plead Occam's razor; maybe I'm simply eccentric.

But if I'm right about the acceleration of addictiveness, then this

kind of lonely squirming to avoid it will increasingly be the fate

of anyone who wants to get things done. We'll increasingly be

defined by what we say no to.

Notes[1]

Could you restrict technological progress to areas where you

wanted it? Only in a limited way, without becoming a police state.

And even then your restrictions would have undesirable side effects.

"Good" and "bad" technological progress aren't sharply differentiated,

so you'd find you couldn't slow the latter without also slowing the

former. And in any case, as Prohibition and the "war on drugs"

show, bans often do more harm than good.[2]

Technology has always been accelerating. By Paleolithic

standards, technology evolved at a blistering pace in the Neolithic

period.[3]

Unless we mass produce social customs. I suspect the recent

resurgence of evangelical Christianity in the US is partly a reaction

to drugs. In desperation people reach for the sledgehammer; if

their kids won't listen to them, maybe they'll listen to God. But

that solution has broader consequences than just getting kids to

say no to drugs. You end up saying no to

science as well.

I worry we may be heading for a future in which only a few people

plot their own itinerary through no-land, while everyone else books

a package tour. Or worse still, has one booked for them by the

government.[4]

People commonly use the word "procrastination" to describe

what they do on the Internet. It seems to me too mild to describe

what's happening as merely not-doing-work. We don't call it

procrastination when someone gets drunk instead of working.[5]

Several people have told me they like the iPad because it

lets them bring the Internet into situations where a laptop would

be too conspicuous. In other words, it's a hip flask. (This is

true of the iPhone too, of course, but this advantage isn't as

obvious because it reads as a phone, and everyone's used to those.)Thanks to Sam Altman, Patrick Collison, Jessica Livingston, and

Robert Morris for reading drafts of this.

The Top Idea in Your Mind

Want to start a startup? Get funded by

Y Combinator.

July 2010I realized recently that what one thinks about in the shower in the

morning is more important than I'd thought. I knew it was a good

time to have ideas. Now I'd go further: now I'd say it's hard to

do a really good job on anything you don't think about in the shower.Everyone who's worked on difficult problems is probably familiar

with the phenomenon of working hard to figure something out, failing,

and then suddenly seeing the answer a bit later while doing something

else. There's a kind of thinking you do without trying to. I'm

increasingly convinced this type of thinking is not merely helpful

in solving hard problems, but necessary. The tricky part is, you

can only control it indirectly.

[1]I think most people have one top idea in their mind at any given

time. That's the idea their thoughts will drift toward when they're

allowed to drift freely. And this idea will thus tend to get all

the benefit of that type of thinking, while others are starved of

it. Which means it's a disaster to let the wrong idea become the

top one in your mind.What made this clear to me was having an idea I didn't want as the

top one in my mind for two long stretches.I'd noticed startups got way less done when they started raising

money, but it was not till we ourselves raised money that I understood

why. The problem is not the actual time it takes to meet with

investors. The problem is that once you start raising money, raising

money becomes the top idea in your mind. That becomes what you

think about when you take a shower in the morning. And that means

other questions aren't.I'd hated raising money when I was running Viaweb, but I'd forgotten

why I hated it so much. When we raised money for Y Combinator, I

remembered. Money matters are particularly likely to become the

top idea in your mind. The reason is that they have to be. It's

hard to get money. It's not the sort of thing that happens by

default. It's not going to happen unless you let it become the

thing you think about in the shower. And then you'll make little

progress on anything else you'd rather be working on.

[2](I hear similar complaints from friends who are professors. Professors

nowadays seem to have become professional fundraisers who do a

little research on the side. It may be time to fix that.)The reason this struck me so forcibly is that for most of the

preceding 10 years I'd been able to think about what I wanted. So

the contrast when I couldn't was sharp. But I don't think this

problem is unique to me, because just about every startup I've seen

grinds to a halt when they start raising money or talking

to acquirers.You can't directly control where your thoughts drift. If you're

controlling them, they're not drifting. But you can control them

indirectly, by controlling what situations you let yourself get

into. That has been the lesson for me: be careful what you let

become critical to you. Try to get yourself into situations where

the most urgent problems are ones you want to think about.You don't have complete control, of course. An emergency could

push other thoughts out of your head. But barring emergencies you

have a good deal of indirect control over what becomes the top idea

in your mind.I've found there are two types of thoughts especially worth

avoiding thoughts like the Nile Perch in the way they push

out more interesting ideas. One I've already mentioned: thoughts

about money. Getting money is almost by definition an attention

sink.

The other is disputes. These too are engaging in the

wrong way: they have the same velcro-like shape as genuinely

interesting ideas, but without the substance. So avoid disputes

if you want to get real work done.

[3]Even Newton fell into this trap. After publishing his theory of

colors in 1672 he found himself distracted by disputes for years,

finally concluding that the only solution was to stop publishing:

I see I have made myself a slave to Philosophy, but if I get free

of Mr Linus's business I will resolutely bid adew to it eternally,

excepting what I do for my privat satisfaction or leave to come

out after me. For I see a man must either resolve to put out

nothing new or become a slave to defend it.

[4]

Linus and his students at Liege were among the more tenacious

critics. Newton's biographer Westfall seems to feel he was

overreacting:

Recall that at the time he wrote, Newton's "slavery" consisted

of five replies to Liege, totalling fourteen printed pages, over

the course of a year.

I'm more sympathetic to Newton. The problem was not the 14 pages,

but the pain of having this stupid controversy constantly reintroduced

as the top idea in a mind that wanted so eagerly to think about

other things.Turning the other cheek turns out to have selfish advantages.

Someone who does you an injury hurts you twice: first by the injury

itself, and second by taking up your time afterward thinking about

it. If you learn to ignore injuries you can at least avoid the

second half. I've found I can to some extent avoid thinking about

nasty things people have done to me by telling myself: this doesn't

deserve space in my head. I'm always delighted to find I've forgotten

the details of disputes, because that means I hadn't been thinking

about them. My wife thinks I'm more forgiving than she is, but my

motives are purely selfish.I suspect a lot of people aren't sure what's the top idea in their

mind at any given time. I'm often mistaken about it. I tend to

think it's the idea I'd want to be the top one, rather than the one

that is. But it's easy to figure this out: just take a shower.

What topic do your thoughts keep returning to? If it's not what

you want to be thinking about, you may want to change something.Notes[1]

No doubt there are already names for this type of thinking, but

I call it "ambient thought."[2]

This was made particularly clear in our case, because neither

of the funds we raised was difficult, and yet in both cases the

process dragged on for months. Moving large amounts of money around

is never something people treat casually. The attention required

increases with the amount—maybe not linearly, but definitely

monotonically.[3]

Corollary: Avoid becoming an administrator, or your job will

consist of dealing with money and disputes.[4]

Letter to Oldenburg, quoted in Westfall, Richard, Life of

Isaac Newton, p. 107.Thanks to Sam Altman, Patrick Collison, Jessica Livingston,

and Robert Morris for reading drafts of this.

How to Lose Time and Money

July 2010When we sold our startup in 1998 I suddenly got a lot of money. I

now had to think about something I hadn't had to think about before:

how not to lose it. I knew it was possible to go from rich to

poor, just as it was possible to go from poor to rich. But while

I'd spent a lot of the past several years studying the paths from

poor to rich,

I knew practically nothing about the paths from rich

to poor. Now, in order to avoid them, I had to learn where they

were.So I started to pay attention to how fortunes are lost. If you'd

asked me as a kid how rich people became poor, I'd have said by

spending all their money. That's how it happens in books and movies,

because that's the colorful way to do it. But in fact the way most

fortunes are lost is not through excessive expenditure, but through

bad investments.It's hard to spend a fortune without noticing. Someone with ordinary

tastes would find it hard to blow through more than a few tens of

thousands of dollars without thinking "wow, I'm spending a lot of

money." Whereas if you start trading derivatives, you can lose a

million dollars (as much as you want, really) in the blink of an

eye.In most people's minds, spending money on luxuries sets off alarms

that making investments doesn't. Luxuries seem self-indulgent.

And unless you got the money by inheriting it or winning a lottery,

you've already been thoroughly trained that self-indulgence leads

to trouble. Investing bypasses those alarms. You're not spending

the money; you're just moving it from one asset to another. Which

is why people trying to sell you expensive things say "it's an

investment."The solution is to develop new alarms. This can be a tricky business,

because while the alarms that prevent you from overspending are so

basic that they may even be in our DNA, the ones that prevent you

from making bad investments have to be learned, and are sometimes

fairly counterintuitive.A few days ago I realized something surprising: the situation with

time is much the same as with money. The most dangerous way to

lose time is not to spend it having fun, but to spend it doing fake

work. When you spend time having fun, you know you're being

self-indulgent. Alarms start to go off fairly quickly. If I woke

up one morning and sat down on the sofa and watched TV all day, I'd

feel like something was terribly wrong. Just thinking about it

makes me wince. I'd start to feel uncomfortable after sitting on

a sofa watching TV for 2 hours, let alone a whole day.And yet I've definitely had days when I might as well have sat in

front of a TV all day—days at the end of which, if I asked myself

what I got done that day, the answer would have been: basically,

nothing. I feel bad after these days too, but nothing like as bad

as I'd feel if I spent the whole day on the sofa watching TV. If

I spent a whole day watching TV I'd feel like I was descending into

perdition. But the same alarms don't go off on the days when I get

nothing done, because I'm doing stuff that seems, superficially,

like real work. Dealing with email, for example. You do it sitting

at a desk. It's not fun. So it must be work.With time, as with money, avoiding pleasure is no longer enough to

protect you. It probably was enough to protect hunter-gatherers,

and perhaps all pre-industrial societies. So nature and nurture

combine to make us avoid self-indulgence. But the world has gotten

more complicated: the most dangerous traps now are new behaviors

that bypass our alarms about self-indulgence by mimicking more

virtuous types. And the worst thing is, they're not even fun.

Thanks to Sam Altman, Trevor Blackwell, Patrick Collison, Jessica

Livingston, and Robert Morris for reading drafts of this.

Organic Startup Ideas

Want to start a startup? Get funded by

Y Combinator.

April 2010The best way to come up with startup ideas is to ask yourself the

question: what do you wish someone would make for you?There are two types of startup ideas: those that grow organically

out of your own life, and those that you decide, from afar, are

going to be necessary to some class of users other than you. Apple

was the first type. Apple happened because Steve Wozniak wanted a

computer. Unlike most people who wanted computers, he could design

one, so he did. And since lots of other people wanted the same

thing, Apple was able to sell enough of them to get the company

rolling. They still rely on this principle today, incidentally.

The iPhone is the phone Steve Jobs wants.

[1]Our own startup, Viaweb, was of the second type. We made software

for building online stores. We didn't need this software ourselves.

We weren't direct marketers. We didn't even know when we started

that our users were called "direct marketers." But we were

comparatively old when we started the company (I was 30 and Robert

Morris was 29), so we'd seen enough to know users would need this

type of software.

[2]There is no sharp line between the two types of ideas, but

the most successful startups seem to be closer to the Apple type

than the Viaweb type. When he was writing that first Basic interpreter

for the Altair, Bill Gates was writing something he would use, as

were Larry and Sergey when they wrote the first versions of Google.Organic ideas are generally preferable to the made up kind, but

particularly so when the founders are young. It takes experience

to predict what other people will want. The worst ideas we see at

Y Combinator are from young founders making things they think other

people will want.So if you want to start a startup and don't know yet what you're

going to do, I'd encourage you to focus initially on organic ideas.

What's missing or broken in your daily life? Sometimes if you just

ask that question you'll get immediate answers. It must have seemed

obviously broken to Bill Gates that you could only program the

Altair in machine language.You may need to stand outside yourself a bit to see brokenness,

because you tend to get used to it and take it for granted. You

can be sure it's there, though. There are always great ideas sitting

right under our noses. In 2004 it was ridiculous that Harvard

undergrads were still using a Facebook printed on paper. Surely

that sort of thing should have been online.There are ideas that obvious lying around now. The reason you're

overlooking them is the same reason you'd have overlooked the idea

of building Facebook in 2004: organic startup ideas usually don't

seem like startup ideas at first. We know now that Facebook was

very successful, but put yourself back in 2004. Putting undergraduates'

profiles online wouldn't have seemed like much of a startup idea.

And in fact, it wasn't initially a startup idea. When Mark spoke

at a YC dinner this winter he said he wasn't trying to start a

company when he wrote the first version of Facebook. It was just

a project. So was the Apple I when Woz first started working on

it. He didn't think he was starting a company. If these guys had

thought they were starting companies, they might have been tempted

to do something more "serious," and that would have been a mistake.So if you want to come up with organic startup ideas, I'd encourage

you to focus more on the idea part and less on the startup part.

Just fix things that seem broken, regardless of whether it seems

like the problem is important enough to build a company on. If you

keep pursuing such threads it would be hard not to end up making

something of value to a lot of people, and when you do, surprise,

you've got a company.

[3]Don't be discouraged if what you produce initially is something

other people dismiss as a toy. In fact, that's a good sign.

That's probably why everyone else has been overlooking the idea. The first

microcomputers were dismissed as toys. And the first planes, and

the first cars. At this point, when someone comes to us with

something that users like but that we could envision forum trolls

dismissing as a toy, it makes us especially likely to invest.While young founders are at a disadvantage when coming up with

made-up ideas, they're the best source of organic ones, because

they're at the forefront of technology. They use the latest stuff.

They only just decided what to use, so why wouldn't they? And

because they use the latest stuff, they're in a position to discover

valuable types of fixable brokenness first.There's nothing more valuable than an unmet need that is just

becoming fixable. If you find something broken that you can fix

for a lot of people, you've found a gold mine. As with an actual

gold mine, you still have to work hard to get the gold out of it.

But at least you know where the seam is, and that's the hard part.Notes[1]

This suggests a way to predict areas where Apple will be weak:

things Steve Jobs doesn't use. E.g. I doubt he is much into gaming.

[2]

In retrospect, we should have become direct marketers. If

I were doing Viaweb again, I'd open our own online store. If we

had, we'd have understood users a lot better. I'd encourage anyone

starting a startup to become one of its users, however unnatural it

seems.[3]

Possible exception: It's hard to compete directly with open source software.

You can build things for programmers, but there has to be some part

you can charge for.Thanks to Sam Altman, Trevor Blackwell, and Jessica Livingston

for reading drafts of this.

Apple's Mistake

Want to start a startup? Get funded by

Y Combinator.

November 2009I don't think Apple realizes how badly the App Store approval process

is broken. Or rather, I don't think they realize how much it matters

that it's broken.The way Apple runs the App Store has harmed their reputation with

programmers more than anything else they've ever done.

Their reputation with programmers used to be great.

It used to be the most common complaint you heard

about Apple was that their fans admired them too uncritically.

The App Store has changed that. Now a lot of programmers

have started to see Apple as evil.How much of the goodwill Apple once had with programmers have they

lost over the App Store? A third? Half? And that's just so far.

The App Store is an ongoing karma leak.\* \* \*How did Apple get into this mess? Their fundamental problem is

that they don't understand software.They treat iPhone apps the way they treat the music they sell through

iTunes. Apple is the channel; they own the user; if you want to

reach users, you do it on their terms. The record labels agreed,

reluctantly. But this model doesn't work for software. It doesn't

work for an intermediary to own the user. The software business

learned that in the early 1980s, when companies like VisiCorp showed

that although the words "software" and "publisher" fit together,

the underlying concepts don't. Software isn't like music or books.

It's too complicated for a third party to act as an intermediary

between developer and user. And yet that's what Apple is trying

to be with the App Store: a software publisher. And a particularly

overreaching one at that, with fussy tastes and a rigidly enforced

house style.If software publishing didn't work in 1980, it works even less now

that software development has evolved from a small number of big

releases to a constant stream of small ones. But Apple doesn't

understand that either. Their model of product development derives

from hardware. They work on something till they think it's finished,

then they release it. You have to do that with hardware, but because

software is so easy to change, its design can benefit from evolution.

The standard way to develop applications now is to launch fast and

iterate. Which means it's a disaster to have long, random delays

each time you release a new version.Apparently Apple's attitude is that developers should be more careful

when they submit a new version to the App Store. They would say

that. But powerful as they are, they're not powerful enough to

turn back the evolution of technology. Programmers don't use

launch-fast-and-iterate out of laziness. They use it because it

yields the best results. By obstructing that process, Apple is

making them do bad work, and programmers hate that as much as Apple

would.How would Apple like it if when they discovered a serious bug in

OS X, instead of releasing a software update immediately, they had

to submit their code to an intermediary who sat on it for a month

and then rejected it because it contained an icon they didn't like?By breaking software development, Apple gets the opposite of what

they intended: the version of an app currently available in the App

Store tends to be an old and buggy one. One developer told me:

As a result of their process, the App Store is full of half-baked

applications. I make a new version almost every day that I release

to beta users. The version on the App Store feels old and crappy.

I'm sure that a lot of developers feel this way: One emotion is

"I'm not really proud about what's in the App Store", and it's

combined with the emotion "Really, it's Apple's fault."

Another wrote:

I believe that they think their approval process helps users by

ensuring quality. In reality, bugs like ours get through all the

time and then it can take 4-8 weeks to get that bug fix approved,

leaving users to think that iPhone apps sometimes just don't work.

Worse for Apple, these apps work just fine on other platforms

that have immediate approval processes.

Actually I suppose Apple has a third misconception: that all the

complaints about App Store approvals are not a serious problem.

They must hear developers complaining. But partners and suppliers

are always complaining. It would be a bad sign if they weren't;

it would mean you were being too easy on them. Meanwhile the iPhone

is selling better than ever. So why do they need to fix anything?They get away with maltreating developers, in the short term, because

they make such great hardware. I just bought a new 27" iMac a

couple days ago. It's fabulous. The screen's too shiny, and the

disk is surprisingly loud, but it's so beautiful that you can't

make yourself care.So I bought it, but I bought it, for the first time, with misgivings.

I felt the way I'd feel buying something made in a country with a

bad human rights record. That was new. In the past when I bought

things from Apple it was an unalloyed pleasure. Oh boy! They make

such great stuff. This time it felt like a Faustian bargain. They

make such great stuff, but they're such assholes. Do I really want

to support this company?\* \* \*Should Apple care what people like me think? What difference does

it make if they alienate a small minority of their users?There are a couple reasons they should care. One is that these

users are the people they want as employees. If your company seems

evil, the best programmers won't work for you. That hurt Microsoft

a lot starting in the 90s. Programmers started to feel sheepish

about working there. It seemed like selling out. When people from

Microsoft were talking to other programmers and they mentioned where

they worked, there were a lot of self-deprecating jokes about having

gone over to the dark side. But the real problem for Microsoft

wasn't the embarrassment of the people they hired. It was the

people they never got. And you know who got them? Google and

Apple. If Microsoft was the Empire, they were the Rebel Alliance.

And it's largely because they got more of the best people that

Google and Apple are doing so much better than Microsoft today.Why are programmers so fussy about their employers' morals? Partly

because they can afford to be. The best programmers can work

wherever they want. They don't have to work for a company they

have qualms about.But the other reason programmers are fussy, I think, is that evil

begets stupidity. An organization that wins by exercising power

starts to lose the ability to win by doing better work. And it's

not fun for a smart person to work in a place where the best ideas

aren't the ones that win. I think the reason Google embraced "Don't

be evil" so eagerly was not so much to impress the outside world

as to inoculate themselves against arrogance.

[1]That has worked for Google so far. They've become more

bureaucratic, but otherwise they seem to have held true to their

original principles. With Apple that seems less the case. When you

look at the famous

1984 ad

now, it's easier to imagine Apple as the

dictator on the screen than the woman with the hammer.

[2]

In fact, if you read the dictator's speech it sounds uncannily like a

prophecy of the App Store.

We have triumphed over the unprincipled dissemination of facts.We have created, for the first time in all history, a garden of

pure ideology, where each worker may bloom secure from the pests

of contradictory and confusing truths.

The other reason Apple should care what programmers think of them

is that when you sell a platform, developers make or break you. If

anyone should know this, Apple should. VisiCalc made the Apple II.And programmers build applications for the platforms they use. Most

applications—most startups, probably—grow out of personal projects.

Apple itself did. Apple made microcomputers because that's what

Steve Wozniak wanted for himself. He couldn't have afforded a

minicomputer.

[3]

Microsoft likewise started out making interpreters

for little microcomputers because

Bill Gates and Paul Allen were interested in using them. It's a

rare startup that doesn't build something the founders use.The main reason there are so many iPhone apps is that so many programmers

have iPhones. They may know, because they read it in an article,

that Blackberry has such and such market share. But in practice

it's as if RIM didn't exist. If they're going to build something,

they want to be able to use it themselves, and that means building

an iPhone app.So programmers continue to develop iPhone apps, even though Apple

continues to maltreat them. They're like someone stuck in an abusive

relationship. They're so attracted to the iPhone that they can't

leave. But they're looking for a way out. One wrote:

While I did enjoy developing for the iPhone, the control they

place on the App Store does not give me the drive to develop

applications as I would like. In fact I don't intend to make any

more iPhone applications unless absolutely necessary.

[4]

Can anything break this cycle? No device I've seen so far could.

Palm and RIM haven't a hope. The only credible contender is Android.

But Android is an orphan; Google doesn't really care about it, not

the way Apple cares about the iPhone. Apple cares about the iPhone

the way Google cares about search.\* \* \*Is the future of handheld devices one locked down by Apple? It's

a worrying prospect. It would be a bummer to have another grim

monoculture like we had in the 1990s. In 1995, writing software

for end users was effectively identical with writing Windows

applications. Our horror at that prospect was the single biggest

thing that drove us to start building web apps.At least we know now what it would take to break Apple's lock.

You'd have to get iPhones out of programmers' hands. If programmers

used some other device for mobile web access, they'd start to develop

apps for that instead.How could you make a device programmers liked better than the iPhone?

It's unlikely you could make something better designed. Apple

leaves no room there. So this alternative device probably couldn't

win on general appeal. It would have to win by virtue of some

appeal it had to programmers specifically.One way to appeal to programmers is with software. If you

could think of an application programmers had to have, but that

would be impossible in the circumscribed world of the iPhone,

you could presumably get them to switch.That would definitely happen if programmers started to use handhelds

as development machines—if handhelds displaced laptops the

way laptops displaced desktops. You need more control of a development

machine than Apple will let you have over an iPhone.Could anyone make a device that you'd carry around in your pocket

like a phone, and yet would also work as a development machine?

It's hard to imagine what it would look like. But I've learned

never to say never about technology. A phone-sized device that

would work as a development machine is no more miraculous by present

standards than the iPhone itself would have seemed by the standards

of 1995.My current development machine is a MacBook Air, which I use with

an external monitor and keyboard in my office, and by itself when

traveling. If there was a version half the size I'd prefer it.

That still wouldn't be small enough to carry around everywhere like

a phone, but we're within a factor of 4 or so. Surely that gap is

bridgeable. In fact, let's make it an

RFS. Wanted:

Woman with hammer.Notes[1]

When Google adopted "Don't be evil," they were still so small

that no one would have expected them to be, yet.

[2]

The dictator in the 1984 ad isn't Microsoft, incidentally;

it's IBM. IBM seemed a lot more frightening in those days, but

they were friendlier to developers than Apple is now.[3]

He couldn't even afford a monitor. That's why the Apple

I used a TV as a monitor.[4]

Several people I talked to mentioned how much they liked the

iPhone SDK. The problem is not Apple's products but their policies.

Fortunately policies are software; Apple can change them instantly

if they want to. Handy that, isn't it?Thanks to Sam Altman, Trevor Blackwell, Ross Boucher,

James Bracy, Gabor Cselle,

Patrick Collison, Jason Freedman, John Gruber, Joe Hewitt, Jessica Livingston,

Robert Morris, Teng Siong Ong, Nikhil Pandit, Savraj Singh, and Jared Tame for reading drafts of this.Russian Translation

What Startups Are Really Like

Want to start a startup? Get funded by

Y Combinator.

October 2009(This essay is derived from a talk at the 2009 Startup School.)I wasn't sure what to talk about at Startup School, so I decided

to ask the founders of the startups we'd funded. What hadn't I

written about yet?I'm in the unusual position of being able to test the essays I write

about startups. I hope the ones on other topics are right, but I

have no way to test them. The ones on startups get tested by about

70 people every 6 months.So I sent all the founders an email asking what surprised them about

starting a startup. This amounts to asking what I got wrong, because

if I'd explained things well enough, nothing should have surprised

them.I'm proud to report I got one response saying:

What surprised me the most is that everything was actually

fairly predictable!

The bad news is that I got over 100 other responses listing the

surprises they encountered.There were very clear patterns in the responses; it was remarkable

how often several people had been surprised by exactly the same

thing. These were the biggest:

1. Be Careful with CofoundersThis was the surprise mentioned by the most founders. There were

two types of responses: that you have to be careful who you pick

as a cofounder, and that you have to work hard to maintain your

relationship.What people wished they'd paid more attention to when choosing

cofounders was character and commitment, not ability. This was

particularly true with startups that failed. The lesson: don't

pick cofounders who will flake.Here's a typical reponse:

You haven't seen someone's true colors unless you've worked

with them on a startup.

The reason character is so important is that it's tested more

severely than in most other situations. One founder said explicitly

that the relationship between founders was more important than

ability:

I would rather cofound a startup with a friend than a stranger

with higher output. Startups are so hard and emotional that

the bonds and emotional and social support that come with

friendship outweigh the extra output lost.

We learned this lesson a long time ago. If you look at the YC

application, there are more questions about the commitment and

relationship of the founders than their ability.Founders of successful startups talked less about choosing cofounders

and more about how hard they worked to maintain their relationship.

One thing that surprised me is how the relationship of startup

founders goes from a friendship to a marriage. My relationship

with my cofounder went from just being friends to seeing each

other all the time, fretting over the finances and cleaning up

shit. And the startup was our baby. I summed it up once like

this: "It's like we're married, but we're not fucking."

Several people used that word "married." It's a far more intense

relationship than you usually see between coworkers—partly because

the stresses are so much greater, and partly because at first the

founders are the whole company. So this relationship has to be

built of top quality materials and carefully maintained. It's the

basis of everything.

2. Startups Take Over Your LifeJust as the relationship between cofounders is more intense than

it usually is between coworkers, so is the relationship between the

founders and the company. Running a startup is not like having a

job or being a student, because it never stops. This is so foreign

to most people's experience that they don't get it till it happens.

[1]

I didn't realize I would spend almost every waking moment either

working or thinking about our startup. You enter a whole

different way of life when it's your company vs. working for

someone else's company.

It's exacerbated by the fast pace of startups, which makes it seem

like time slows down:

I think the thing that's been most surprising to me is how one's

perspective on time shifts. Working on our startup, I remember

time seeming to stretch out, so that a month was a huge interval.

In the best case, total immersion can be exciting:

It's surprising how much you become consumed by your startup,

in that you think about it day and night, but never once does

it feel like "work."

Though I have to say, that quote is from someone we funded this

summer. In a couple years he may not sound so chipper.

3. It's an Emotional Roller-coasterThis was another one lots of people were surprised about. The ups

and downs were more extreme than they were prepared for.In a startup, things seem great one moment and hopeless the next.

And by next, I mean a couple hours later.

The emotional ups and downs were the biggest surprise for me.

One day, we'd think of ourselves as the next Google and dream

of buying islands; the next, we'd be pondering how to let our

loved ones know of our utter failure; and on and on.

The hard part, obviously, is the lows. For a lot of founders that

was the big surprise:

How hard it is to keep everyone motivated during rough days or

weeks, i.e. how low the lows can be.

After a while, if you don't have significant success to cheer you

up, it wears you out:

Your most basic advice to founders is "just don't die," but the

energy to keep a company going in lieu of unburdening success

isn't free; it is siphoned from the founders themselves.

There's a limit to how much you can take. If you get to the point

where you can't keep working anymore, it's not the end of the world.

Plenty of famous founders have had some failures along the way.

4. It Can Be FunThe good news is, the highs are also very high. Several founders

said what surprised them most about doing a startup was how fun it

was:

I think you've left out just how fun it is to do a startup. I

am more fulfilled in my work than pretty much any of my friends

who did not start companies.

What they like most is the freedom:

I'm surprised by how much better it feels to be working on

something that is challenging and creative, something I believe

in, as opposed to the hired-gun stuff I was doing before. I

knew it would feel better; what's surprising is how much better.

Frankly, though, if I've misled people here, I'm not eager to fix

that. I'd rather have everyone think starting a startup is grim

and hard than have founders go into it expecting it to be fun, and

a few months later saying "This is supposed to be fun? Are you

kidding?"The truth is, it wouldn't be fun for most people. A lot of what

we try to do in the application process is to weed out the people

who wouldn't like it, both for our sake and theirs.The best way to put it might be that starting a startup is fun the

way a survivalist training course would be fun, if you're into that

sort of thing. Which is to say, not at all, if you're not.

5. Persistence Is the KeyA lot of founders were surprised how important persistence was in

startups. It was both a negative and a positive surprise: they were

surprised both by the degree of persistence required

Everyone said how determined and resilient you must be, but

going through it made me realize that the determination required

was still understated.

and also by the degree to which persistence alone was able to

dissolve obstacles:

If you are persistent, even problems that seem out of your

control (i.e. immigration) seem to work themselves out.

Several founders mentioned specifically how much more important

persistence was than intelligence.

I've been surprised again and again by just how much more

important persistence is than raw intelligence.

This applies not just to intelligence but to ability in general,

and that's why so many people said character was more important in

choosing cofounders.

6. Think Long-TermYou need persistence because everything takes longer than you expect.

A lot of people were surprised by that.

I'm continually surprised by how long everything can take.

Assuming your product doesn't experience the explosive growth

that very few products do, everything from development to

dealmaking (especially dealmaking) seems to take 2-3x longer

than I always imagine.

One reason founders are surprised is that because they work fast,

they expect everyone else to. There's a shocking amount of shear

stress at every point where a startup touches a more bureaucratic

organization, like a big company or a VC fund. That's why fundraising

and the enterprise market kill and maim so many startups.

[2]But I think the reason most founders are surprised by how long it

takes is that they're overconfident. They think they're going to

be an instant success, like YouTube or Facebook. You tell them

only 1 out of 100 successful startups has a trajectory like that,

and they all think "we're going to be that 1."Maybe they'll listen to one of the more successful founders:

The top thing I didn't understand before going into it is that

persistence is the name of the game. For the vast majority of

startups that become successful, it's going to be a really

long journey, at least 3 years and probably 5+.

There is a positive side to thinking longer-term. It's not just

that you have to resign yourself to everything taking longer than

it should. If you work patiently it's less stressful, and you can

do better work:

Because we're relaxed, it's so much easier to have fun doing

what we do. Gone is the awkward nervous energy fueled by the

desperate need to not fail guiding our actions. We can concentrate

on doing what's best for our company, product, employees and

customers.

That's why things get so much better when you hit ramen profitability.

You can shift into a different mode of working.7. Lots of Little ThingsWe often emphasize how rarely startups win simply because they hit

on some magic idea. I think founders have now gotten that into

their heads. But a lot were surprised to find this also applies

within startups. You have to do lots of different things:

It's much more of a grind than glamorous. A timeslice selected

at random would more likely find me tracking down a weird DLL

loading bug on Swedish Windows, or tracking down a bug in the

financial model Excel spreadsheet the night before a board

meeting, rather than having brilliant flashes of strategic

insight.

Most hacker-founders would like to spend all their time programming.

You won't get to, unless you fail. Which can be transformed into:

If you spend all your time programming, you will fail.The principle extends even into programming. There is rarely a

single brilliant hack that ensures success:

I learnt never to bet on any one feature or deal or anything

to bring you success. It is never a single thing. Everything

is just incremental and you just have to keep doing lots of

those things until you strike something.

Even in the rare cases where a clever hack makes your fortune, you

probably won't know till later:

There is no such thing as a killer feature. Or at least you

won't know what it is.

So the best strategy is to try lots of different things. The reason

not to put all your eggs in one basket is not the usual one,

which applies even when you know which basket is best. In a startup

you don't even know that.

8. Start with Something MinimalLots of founders mentioned how important it was to launch with the

simplest possible thing. By this point everyone knows you should

release fast and iterate. It's practically a mantra at YC. But

even so a lot of people seem to have been burned by not doing it:

Build the absolute smallest thing that can be considered a

complete application and ship it.

Why do people take too long on the first version? Pride, mostly.

They hate to release something that could be better. They worry

what people will say about them. But you have to overcome this:

Doing something "simple" at first glance does not mean you

aren't doing something meaningful, defensible, or valuable.

Don't worry what people will say. If your first version is so

impressive that trolls don't make fun of it, you waited too long

to launch.

[3]One founder said this should be your approach to all programming,

not just startups, and I tend to agree.

Now, when coding, I try to think "How can I write this such

that if people saw my code, they'd be amazed at how little there

is and how little it does?"

Over-engineering is poison. It's not like doing extra work for

extra credit. It's more like telling a lie that you then have to

remember so you don't contradict it.

9. Engage UsersProduct development is a conversation with the user that doesn't

really start till you launch. Before you launch, you're like a

police artist before he's shown the first version of his sketch to

the witness.It's so important to launch fast that it may be better to think of

your initial version not as a product, but as a trick for getting

users to start talking to you.

I learned to think about the initial stages of a startup as a

giant experiment. All products should be considered experiments,

and those that have a market show promising results extremely

quickly.

Once you start talking to users, I guarantee you'll be surprised

by what they tell you.

When you let customers tell you what they're after, they will

often reveal amazing details about what they find valuable as

well what they're willing to pay for.

The surprise is generally positive as well as negative. They won't

like what you've built, but there will be other things they would

like that would be trivially easy to implement. It's not till you

start the conversation by launching the wrong thing that they can

express (or perhaps even realize) what they're looking for.

10. Change Your IdeaTo benefit from engaging with users you have to be willing to change

your idea. We've always encouraged founders to see a startup idea

as a hypothesis rather than a blueprint. And yet they're still

surprised how well it works to change the idea.

Normally if you complain about something being hard, the general

advice is to work harder. With a startup, I think you should

find a problem that's easy for you to solve. Optimizing in

solution-space is familiar and straightforward, but you can

make enormous gains playing around in problem-space.

Whereas mere determination, without flexibility, is a greedy algorithm

that may get you nothing more than a mediocre local maximum:

When someone is determined, there's still a danger that they'll

follow a long, hard path that ultimately leads nowhere.

You want to push forward, but at the same time twist and turn to

find the most promising path. One founder put it very succinctly:

Fast iteration is the key to success.

One reason this advice is so hard to follow is that people don't

realize how hard it is to judge startup ideas, particularly their

own. Experienced founders learn to keep an open mind:

Now I don't laugh at ideas anymore, because I realized how

terrible I was at knowing if they were good or not.

You can never tell what will work. You just have to do whatever

seems best at each point. We do this with YC itself. We still

don't know if it will work, but it seems like a decent hypothesis.

11. Don't Worry about CompetitorsWhen you think you've got a great idea, it's sort of like having a

guilty conscience about something. All someone has to do is look

at you funny, and you think "Oh my God, they know."These alarms are almost always false:

Companies that seemed like competitors and threats at first

glance usually never were when you really looked at it. Even

if they were operating in the same area, they had a different

goal.

One reason people overreact to competitors is that they overvalue

ideas. If ideas really were the key, a competitor with the same

idea would be a real threat. But it's usually execution that

matters:

All the scares induced by seeing a new competitor pop up are

forgotten weeks later. It always comes down to your own product

and approach to the market.

This is generally true even if competitors get lots of attention.

Competitors riding on lots of good blogger perception aren't

really the winners and can disappear from the map quickly. You

need consumers after all.

Hype doesn't make satisfied users, at least not for something as

complicated as technology.12. It's Hard to Get UsersA lot of founders complained about how hard it was to get users,

though.

I had no idea how much time and effort needed to go into attaining

users.

This is a complicated topic. When you can't get users, it's hard

to say whether the problem is lack of exposure, or whether the

product's simply bad. Even good products can be blocked by switching

or integration costs:

Getting people to use a new service is incredibly difficult.

This is especially true for a service that other companies can

use, because it requires their developers to do work. If you're

small, they don't think it is urgent.

[4]

The sharpest criticism of YC came from a founder who said we didn't

focus enough on customer acquisition:

YC preaches "make something people want" as an engineering task,

a never ending stream of feature after feature until enough

people are happy and the application takes off. There's very

little focus on the cost of customer acquisition.

This may be true; this may be something we need to fix, especially

for applications like games. If you make something where the

challenges are mostly technical, you can rely on word of mouth,

like Google did. One founder was surprised by how well that worked

for him:

There is an irrational fear that no one will buy your product.

But if you work hard and incrementally make it better, there

is no need to worry.

But with other types of startups you may win less by features and

more by deals and marketing.

13. Expect the Worst with DealsDeals fall through. That's a constant of the startup world. Startups

are powerless, and good startup ideas generally seem wrong. So

everyone is nervous about closing deals with you, and you have no

way to make them.This is particularly true with investors:

In retrospect, it would have been much better if we had operated

under the assumption that we would never get any additional

outside investment. That would have focused us on finding

revenue streams early.

My advice is generally pessimistic. Assume you won't get money,

and if someone does offer you any, assume you'll never get any more.

If someone offers you money, take it. You say it a lot, but I

think it needs even more emphasizing. We had the opportunity

to raise a lot more money than we did last year and I wish we

had.

Why do founders ignore me? Mostly because they're optimistic by

nature. The mistake is to be optimistic about things you can't

control. By all means be optimistic about your ability to make

something great. But you're asking for trouble if you're optimistic

about big companies or investors.

14. Investors Are CluelessA lot of founders mentioned how surprised they were by the cluelessness

of investors:

They don't even know about the stuff they've invested in. I

met some investors that had invested in a hardware device and

when I asked them to demo the device they had difficulty switching

it on.

Angels are a bit better than VCs, because they usually have startup

experience themselves:

VC investors don't know half the time what they are talking

about and are years behind in their thinking. A few were great,

but 95% of the investors we dealt with were unprofessional,

didn't seem to be very good at business or have any kind of

creative vision. Angels were generally much better to talk to.

Why are founders surprised that VCs are clueless? I think it's

because they seem so formidable.The reason VCs seem formidable is that it's their profession to.

You get to be a VC by convincing asset managers to trust you with

hundreds of millions of dollars. How do you do that? You have to

seem confident, and you have to seem like you understand technology.

[5]

15. You May Have to Play GamesBecause investors are so bad at judging you, you have to work harder

than you should at selling yourself. One founder said the thing

that surprised him most was

The degree to which feigning certitude impressed investors.

This is the thing that has surprised me most about YC founders'

experiences. This summer we invited some of the alumni to talk to

the new startups about fundraising, and pretty much 100% of their

advice was about investor psychology. I thought I was cynical about

VCs, but the founders were much more cynical.

A lot of what startup founders do is just posturing. It works.

VCs themselves have no idea of the extent to which the startups

they like are the ones that are best at selling themselves to VCs.

[6]

It's exactly the same phenomenon we saw a step earlier. VCs get

money by seeming confident to LPs, and founders get money by seeming

confident to VCs.

16. Luck Is a Big FactorWith two such random linkages in the path between startups and

money, it shouldn't be surprising that luck is a big factor in

deals. And yet a lot of founders are surprised by it.

I didn't realize how much of a role luck plays and how much is

outside of our control.

If you think about famous startups, it's pretty clear how big a

role luck plays. Where would Microsoft be if IBM insisted on an

exclusive license for DOS?Why are founders fooled by this? Business guys probably aren't,

but hackers are used to a world where skill is paramount, and you

get what you deserve.

When we started our startup, I had bought the hype of the startup

founder dream: that this is a game of skill. It is, in some

ways. Having skill is valuable. So is being determined as all

hell. But being lucky is the critical ingredient.

Actually the best model would be to say that the outcome is the

product of skill, determination, and luck. No matter how much

skill and determination you have, if you roll a zero for luck, the

outcome is zero.These quotes about luck are not from founders whose startups failed.

Founders who fail quickly tend to blame themselves. Founders who

succeed quickly don't usually realize how lucky they were. It's

the ones in the middle who see how important luck is.

17. The Value of CommunityA surprising number of founders said what surprised them most about

starting a startup was the value of community. Some meant the

micro-community of YC founders:

The immense value of the peer group of YC companies, and facing

similar obstacles at similar times.

which shouldn't be that surprising, because that's why it's structured

that way. Others were surprised at the value of the startup community

in the larger sense:

How advantageous it is to live in Silicon Valley, where you

can't help but hear all the cutting-edge tech and startup news,

and run into useful people constantly.

The specific thing that surprised them most was the general spirit

of benevolence:

One of the most surprising things I saw was the willingness of

people to help us. Even people who had nothing to gain went out

of their way to help our startup succeed.

and particularly how it extended all the way to the top:

The surprise for me was how accessible important and interesting

people are. It's amazing how easily you can reach out to people

and get immediate feedback.

This is one of the reasons I like being part of this world. Creating

wealth is not a zero-sum game, so you don't have to stab people in

the back to win.

18. You Get No RespectThere was one surprise founders mentioned that I'd forgotten about:

that outside the startup world, startup founders get no respect.

In social settings, I found that I got a lot more respect when

I said, "I worked on Microsoft Office" instead of "I work at a

small startup you've never heard of called x."

Partly this is because the rest of the world just doesn't get

startups, and partly it's yet another consequence of the fact that

most good startup ideas seem bad:

If you pitch your idea to a random person, 95% of the time

you'll find the person instinctively thinks the idea will be a

flop and you're wasting your time (although they probably won't

say this directly).

Unfortunately this extends even to dating:

It surprised me that being a startup founder does not get you

more admiration from women.

I did know about that, but I'd forgotten.

19. Things Change as You GrowThe last big surprise founders mentioned is how much things changed

as they grew. The biggest change was that you got to program even

less:

Your job description as technical founder/CEO is completely

rewritten every 6-12 months. Less coding, more

managing/planning/company building, hiring, cleaning up messes,

and generally getting things in place for what needs to happen

a few months from now.

In particular, you now have to deal with employees, who often have

different motivations:

I knew the founder equation and had been focused on it since I

knew I wanted to start a startup as a 19 year old. The employee

equation is quite different so it took me a while to get it

down.

Fortunately, it can become a lot less stressful once you reach

cruising altitude:

I'd say 75% of the stress is gone now from when we first started.

Running a business is so much more enjoyable now. We're more

confident. We're more patient. We fight less. We sleep more.

I wish I could say it was this way for every startup that succeeded,

but 75% is probably on the high side.

The Super-PatternThere were a few other patterns, but these were the biggest. One's

first thought when looking at them all is to ask if there's a

super-pattern, a pattern to the patterns.I saw it immediately, and so did a YC founder I read the list to.

These are supposed to be the surprises, the things I didn't tell

people. What do they all have in common? They're all things I

tell people. If I wrote a new essay with the same outline as this

that wasn't summarizing the founders' responses, everyone would say

I'd run out of ideas and was just repeating myself.What is going on here?When I look at the responses, the common theme is that

starting a startup was like I said, but way more so. People just

don't seem to get how different it is till they do it. Why? The

key to that mystery is to ask, how different from what? Once you

phrase it that way, the answer is obvious: from a job. Everyone's

model of work is a job. It's completely pervasive. Even if you've

never had a job, your parents probably did, along with practically

every other adult you've met.Unconsciously, everyone expects a startup to be like a job, and

that explains most of the surprises. It explains why people are

surprised how carefully you have to choose cofounders and how hard

you have to work to maintain your relationship. You don't have to

do that with coworkers. It explains why the ups and downs are

surprisingly extreme. In a job there is much more damping. But

it also explains why the good times are surprisingly good: most

people can't imagine such freedom. As you go down the list, almost

all the surprises are surprising in how much a startup differs from

a job.You probably can't overcome anything so pervasive as the model of

work you grew up with. So the best solution is to be consciously

aware of that. As you go into a startup, you'll be thinking "everyone

says it's really extreme." Your next thought will probably be "but

I can't believe it will be that bad." If you want to avoid being

surprised, the next thought after that should be: "and the reason

I can't believe it will be that bad is that my model of work is a

job."

Notes[1]

Graduate students might understand it. In grad school you

always feel you should be working on your thesis. It doesn't end

every semester like classes do.[2]

The best way for a startup to engage with slow-moving

organizations is to fork off separate processes to deal with them.

It's when they're on the critical path that they kill you—when

you depend on closing a deal to move forward. It's worth taking

extreme measures to avoid that.[3]

This is a variant of Reid Hoffman's principle that if you

aren't embarrassed by what you launch with, you waited too long to

launch.[4]

The question to ask about what you've built is not whether it's

good, but whether it's good enough to supply the activation energy

required.[5]

Some VCs seem to understand technology because they actually

do, but that's overkill; the defining test is whether you can talk

about it well enough to convince limited partners.[6]

This is the same phenomenon you see with defense contractors

or fashion brands. The dumber the customers, the more effort you

expend on the process of selling things to them rather than making

the things you sell.Thanks: to Jessica Livingston for reading drafts of this,

and to all the founders who responded to my email.Related:Startups in 13 SentencesThe Hardest Lessons for Startups to LearnHow Not to DieThe 18 Mistakes That Kill StartupsA Fundraising Survival GuideRussian TranslationKorean TranslationHebrew Translation

Persuade xor Discover

September 2009When meeting people you don't know very well, the convention is

to seem extra friendly. You smile and say "pleased to meet you,"

whether you are or not. There's nothing dishonest about this.

Everyone knows that these little social lies aren't meant

to be taken literally, just as everyone knows that

"Can you pass the salt?" is only grammatically a question.I'm perfectly willing to smile and say "pleased to meet you"

when meeting new people. But there is another set of

customs for being ingratiating in print that are not so

harmless.The reason there's a convention of being ingratiating in print

is that most essays are written to persuade.

And as any politician could tell

you, the way to persuade people is not just to baldly state the

facts. You have to add a spoonful of sugar to make the medicine

go down.For example, a politician announcing the cancellation of

a government program will not merely say "The

program is canceled." That would seem offensively

curt. Instead he'll spend most of his time talking about the

noble effort made by the people who worked on it.The reason these conventions are more dangerous is that they

interact with the ideas. Saying "pleased to meet you" is just

something you prepend to a conversation, but the sort of spin

added by politicians is woven through it. We're starting to

move from social lies to real lies.Here's an example of a paragraph from an essay I wrote about

labor unions. As written,

it tends to offend people who like unions.

People who think the labor movement was the creation of heroic

union organizers have a problem to explain: why are unions shrinking

now? The best they can do is fall back on the default explanation

of people living in fallen civilizations. Our ancestors were

giants. The workers of the early twentieth century must have had

a moral courage that's lacking today.

Now here's the same paragraph rewritten to please instead of

offending them:

Early union organizers made heroic sacrifices to improve conditions

for workers. But though

labor unions are shrinking now, it's not because present union

leaders are any less courageous. An employer couldn't get away

with hiring thugs to beat up union leaders today, but if they

did, I see no reason to believe today's union leaders would shrink

from the challenge. So I think it would be a mistake to attribute

the decline of unions to some kind of decline in the people who

run them. Early union leaders were heroic, certainly, but we

should not suppose that if unions have declined, it's because

present union leaders are somehow inferior. The cause must be

external.

[1]

It makes the same point: that it can't have been the personal

qualities of early union organizers that made unions successful,

but must have been some external factor, or otherwise present-day

union leaders would have to be inferior people. But written this

way it seems like a defense of present-day union organizers rather

than an attack on early ones. That makes it more persuasive to

people who like unions, because it seems sympathetic to their cause.I believe everything I wrote in the second version. Early union

leaders did make heroic sacrifices. And

present union leaders probably would rise to the occasion if

necessary. People tend to; I'm skeptical about the idea of "the

greatest generation."

[2]If I believe everything I said in the second version, why didn't I

write it that way? Why offend people needlessly?Because I'd rather offend people than pander to them,

and if you write about controversial topics you have to choose one or the other. The degree of

courage of past or present union leaders is beside the point; all

that matters for the argument is that they're the same.

But if you want to please

people who are mistaken, you can't simply tell the truth. You're

always going to have to add some sort of padding to protect their

misconceptions from bumping against reality.Most writers do. Most writers write to persuade, if only out of

habit or politeness. But I don't write to persuade; I write to

figure out. I write to persuade a hypothetical perfectly unbiased

reader.Since the custom is to write to persuade the actual reader, someone

who doesn't will seem arrogant. In fact, worse than arrogant: since

readers are used to essays that try to please someone, an essay

that displeases one side in a dispute reads as an attempt to pander

to the other. To a lot of pro-union readers, the first paragraph

sounds like the sort of thing a right-wing radio talk show host

would say to stir up his followers. But it's not. Something that

curtly contradicts one's beliefs can be hard to distinguish from a

partisan attack on them, but though they can end up in the same

place they come from different sources.Would it be so bad to add a few extra words, to make people feel

better? Maybe not. Maybe I'm excessively attached to conciseness.

I write code the same way I write essays,

making pass after pass

looking for anything I can cut. But I have a legitimate reason for

doing this. You don't know what the ideas are until you get them

down to the fewest words.

[3]The danger of the second paragraph

is not merely that it's longer. It's that you start to lie to

yourself. The ideas start to get mixed together with the spin

you've added to get them past the readers' misconceptions.I think the goal of an essay should be to discover

surprising things. That's my goal, at least.

And most surprising means most different from what people currently

believe. So writing to persuade and writing to discover are

diametrically opposed. The more your conclusions disagree with

readers' present beliefs, the more effort you'll have to expend on

selling your ideas rather than having them. As you accelerate,

this drag increases, till eventually you reach a point where 100%

of your energy is devoted to overcoming it and you can't go any

faster.It's hard enough to overcome one's own misconceptions without having

to think about how to get the resulting ideas past other people's.

I worry that if I wrote to persuade, I'd start to shy away unconsciously

from ideas I knew would be hard to sell. When I notice something

surprising, it's usually very faint at first. There's nothing more

than a slight stirring of discomfort. I don't want anything to get

in the way of noticing it consciously.

Notes[1]

I had a strange feeling of being back in high school writing

this. To get a good grade you had to both write the sort of pious

crap you were expected to, but also seem to be writing with conviction.

The solution was a kind of method acting. It was revoltingly

familiar to slip back into it.[2]

Exercise for the reader:

rephrase that thought to please the same people the first version

would offend.[3]

Come to think of it, there is one way in which I deliberately

pander to readers, because it doesn't change the number of words:

I switch person. This flattering distinction seems so natural to

the average reader that they probably don't notice even when I

switch in mid-sentence, though you tend to notice when it's done

as conspicuously as this.Thanks to Jessica Livingston and Robert Morris

for reading drafts of this.Note: An earlier version of this essay began by talking

about why people dislike Michael Arrington. I now believe that

was mistaken, and that most people don't dislike him for the

same reason I did when I first met him, but simply because

he writes about controversial things.

Post-Medium Publishing

September 2009Publishers of all types, from news to music, are unhappy that

consumers won't pay for content anymore. At least, that's how they

see it.In fact consumers never really were paying for content, and publishers

weren't really selling it either. If the content was what they

were selling, why has the price of books or music or movies always

depended mostly on the format? Why didn't better content cost more?

[1]A copy of Time costs $5 for 58 pages, or 8.6 cents a page.

The Economist costs $7 for 86 pages, or 8.1 cents a page. Better

journalism is actually slightly cheaper.Almost every form of publishing has been organized as if the medium

was what they were selling, and the content was irrelevant. Book

publishers, for example, set prices based on the cost of producing

and distributing books. They treat the words printed in the book

the same way a textile manufacturer treats the patterns printed on

its fabrics.Economically, the print media are in the business of marking up

paper. We can all imagine an old-style editor getting a scoop and

saying "this will sell a lot of papers!" Cross out that final S and

you're describing their business model. The reason they make less

money now is that people don't need as much paper.A few months ago I ran into a friend in a cafe. I had a copy of

the New York Times, which I still occasionally buy on weekends. As

I was leaving I offered it to him, as I've done countless times

before in the same situation. But this time something new happened.

I felt that sheepish feeling you get when you offer someone something

worthless. "Do you, er, want a printout of yesterday's news?" I

asked. (He didn't.)Now that the medium is evaporating, publishers have nothing left

to sell. Some seem to think they're going to sell content—that

they were always in the content business, really. But they weren't,

and it's unclear whether anyone could be.SellingThere have always been people in the business of selling information,

but that has historically been a distinct business from publishing.

And the business of selling information to consumers has always

been a marginal one. When I was a kid there were people who used

to sell newsletters containing stock tips, printed on colored paper

that made them hard for the copiers of the day to reproduce. That

is a different world, both culturally and economically, from the

one publishers currently inhabit.People will pay for information they think they can make money from.

That's why they paid for those stock tip newsletters, and why

companies pay now for Bloomberg terminals and Economist Intelligence

Unit reports. But will people pay for information otherwise?

History offers little encouragement.If audiences were willing to pay more for better content, why wasn't

anyone already selling it to them? There was no reason you couldn't

have done that in the era of physical media. So were the print

media and the music labels simply overlooking this opportunity? Or

is it, rather, nonexistent?What about iTunes? Doesn't that show people will pay for content?

Well, not really. iTunes is more of a tollbooth than a store. Apple

controls the default path onto the iPod. They offer a convenient

list of songs, and whenever you choose one they ding your credit

card for a small amount, just below the threshold of attention.

Basically, iTunes makes money by taxing people, not selling them

stuff. You can only do that if you own the channel, and even then

you don't make much from it, because a toll has to be ignorable to

work. Once a toll becomes painful, people start to find ways around

it, and that's pretty easy with digital content.The situation is much the same with digital books. Whoever controls

the device sets the terms. It's in their interest for content to

be as cheap as possible, and since they own the channel, there's a

lot they can do to drive prices down. Prices will fall even further

once writers realize they don't need publishers. Getting a book

printed and distributed is a daunting prospect for a writer, but

most can upload a file.Is software a counterexample? People pay a lot for desktop software,

and that's just information. True, but I don't think publishers

can learn much from software. Software companies can charge a lot

because (a) many of the customers are businesses, who get in

trouble

if they use pirated versions, and (b) though in form merely

information, software is treated by both maker and purchaser as a

different type of thing from a song or an article. A Photoshop

user needs Photoshop in a way that no one needs a particular song

or article.That's why there's a separate word, "content," for information

that's not software. Software is a different business. Software

and content blur together in some of the most lightweight software,

like casual games. But those are usually free. To make money the

way software companies do, publishers would have to become software

companies, and being publishers gives them no particular head start

in that domain.

[2]The most promising countertrend is the premium cable channel. People

still pay for those. But broadcasting isn't publishing: you're not

selling a copy of something. That's one reason the movie business

hasn't seen their revenues decline the way the news and music

businesses have. They only have one foot in publishing.To the extent the movie business can avoid becoming publishers,

they may avoid publishing's problems. But there are limits to how

well they'll be able to do that. Once publishing—giving people

copies—becomes the most natural way of distributing your content,

it probably doesn't work to stick to old forms of distribution just

because you make more that way. If free copies of your content are

available online, then you're competing with publishing's form of

distribution, and that's just as bad as being a publisher.Apparently some people in the music business hope to retroactively

convert it away from publishing, by getting listeners to pay for

subscriptions. It seems unlikely that will work if they're just

streaming the same files you can get as mp3s.NextWhat happens to publishing if you can't sell content? You have two

choices: give it away and make money from it indirectly, or find

ways to embody it in things people will pay for.The first is probably the future of most current media.

Give music

away and make money from concerts and t-shirts. Publish articles

for free and make money from one of a dozen permutations of

advertising. Both publishers and investors are down on advertising

at the moment, but it has more potential than they realize.I'm not claiming that potential will be realized by the existing

players. The optimal

ways to make money from the written word

probably require different words written by different people.It's harder to say what will happen to movies. They could evolve

into ads. Or they could return to their roots and make going to

the theater a treat. If they made the experience good enough,

audiences might start to prefer it to watching pirated movies at

home.

[3]

Or maybe the movie business will dry up, and the people

working in it will go to work for game developers.I don't know how big embodying information in physical form will

be. It may be surprisingly large; people overvalue

physical stuff.

There should remain some market for printed books, at least.I can see the evolution of book publishing in the books on my

shelves. Clearly at some point in the 1960s the big publishing

houses started to ask: how cheaply can we make books before people

refuse to buy them? The answer turned out to be one step short of

phonebooks. As long as it isn't floppy, consumers still perceive

it as a book.That worked as long as buying printed books was the only way to

read them. If printed books are optional, publishers will have to

work harder to entice people to buy them. There should be some

market, but it's hard to foresee how big, because its size will

depend not on macro trends like the amount people read, but on the

ingenuity of individual publishers.

[4]Some magazines may thrive by focusing on the magazine as a physical

object. Fashion magazines could be made lush in a way that would

be hard to match digitally, at least for a while. But this is

probably not an option for most magazines.I don't know exactly what the future will look like, but I'm not

too worried about it. This sort of change tends to create as many

good things as it kills. Indeed, the really interesting question is not

what will happen to existing forms, but what new forms will appear.The reason I've been writing about existing forms is that I don't

know what new forms will appear. But though I can't predict

specific winners, I can offer a recipe for recognizing them. When

you see something that's taking advantage of new technology to give

people something they want that they couldn't have before, you're

probably looking at a winner. And when you see something that's

merely reacting to new technology in an attempt to preserve some

existing source of revenue, you're probably looking at a loser.

Notes[1]

I don't like the word "content" and tried for a while to avoid

using it, but I have to admit there's no other word that means the

right thing. "Information" is too general.Ironically, the main reason I don't like "content" is the thesis

of this essay. The word suggests an undifferentiated slurry, but

economically that's how both publishers and audiences treat it.

Content is information you don't need.[2]

Some types of publishers would be at a disadvantage trying

to enter the software business. Record labels, for example, would

probably find it more natural to expand into casinos than software,

because the kind of people who run them would be more at home at

the mafia end of the business spectrum than the don't-be-evil end.[3]

I never watch movies in theaters anymore. The tipping point

for me was the ads they show first.[4]

Unfortunately, making physically nice books will only be a

niche within a niche. Publishers are more likely to resort to

expedients like selling autographed copies, or editions with the

buyer's picture on the cover.Thanks to Michael Arrington, Trevor Blackwell, Steven Levy, Robert

Morris, and Geoff Ralston for reading drafts of this.

The List of N Things

September 2009I bet you the current issue of Cosmopolitan has an article

whose title begins with a number. "7 Things He Won't Tell You about

Sex," or something like that. Some popular magazines

feature articles of this type on the cover of every

issue. That can't be happening by accident. Editors must know

they attract readers.Why do readers like the list of n things so much? Mainly because

it's easier to read than a regular article.

[1]

Structurally, the list of n things is a degenerate case of essay.

An essay can go anywhere the writer wants. In a list of n things

the writer agrees to constrain himself to a collection of points

of roughly equal importance, and he tells the reader explicitly

what they are.Some of the work of reading an article is understanding its

structure—figuring out what in high school we'd have called

its "outline." Not explicitly, of course, but someone who really

understands an article probably has something in his brain afterward

that corresponds to such an outline. In a list of n things, this

work is done for you. Its structure is an exoskeleton.As well as being explicit, the structure is guaranteed to be of the

simplest possible type: a few main points with few to no subordinate

ones, and no particular connection between them.Because the main points are unconnected, the list of n things is

random access. There's no thread of reasoning you have to follow. You could

read the list in any order. And because the points are independent

of one another, they work like watertight compartments in an

unsinkable ship. If you get bored with, or can't understand, or

don't agree with one point, you don't have to give up on the article.

You can just abandon that one and skip to the next. A list of n

things is parallel and therefore fault tolerant.There are times when this format is what a writer wants. One, obviously,

is when what you have to say actually is a list of n

things. I once wrote an essay about the mistakes that kill startups, and a few people made fun of me

for writing something whose title began with a number. But in that

case I really was trying to make a complete catalog of a number of

independent things. In fact, one of the questions I was trying to

answer was how many there were.There are other less legitimate reasons for using this format. For

example, I use it when I get close to a deadline. If I have to

give a talk and I haven't started it a few days beforehand, I'll

sometimes play it safe and make the talk a list of n things.The list of n things is easier for writers as well as readers. When

you're writing a real essay, there's always a chance you'll hit a

dead end. A real essay is a train of thought, and some trains of

thought just peter out. That's an alarming possibility when you

have to give a talk in a few days. What if you run out of ideas?

The compartmentalized structure of the list of n things protects

the writer from his own stupidity in much the same way it protects

the reader. If you run out of ideas on one point, no problem: it

won't kill the essay. You can take out the whole point if you need

to, and the essay will still survive.Writing a list of n things is so relaxing. You think of n/2 of

them in the first 5 minutes. So bang, there's the structure, and

you just have to fill it in. As you think of more points, you just

add them to the end. Maybe you take out or rearrange or combine a

few, but at every stage you have a valid (though initially low-res)

list of n things. It's like the sort of programming where you write

a version 1 very quickly and then gradually modify it, but at every

point have working code—or the style of painting where you begin

with a complete but very blurry sketch done in an hour, then spend

a week cranking up the resolution.Because the list of n things is easier for writers too, it's not

always a damning sign when readers prefer it. It's not necessarily

evidence readers are lazy; it could also mean they don't have

much confidence in the writer. The list of n things is in that

respect the cheeseburger of essay forms. If you're eating at a

restaurant you suspect is bad, your best bet is to order the

cheeseburger. Even a bad cook can make a decent cheeseburger. And

there are pretty strict conventions about what a cheeseburger should

look like. You can assume the cook isn't going to try something

weird and artistic. The list of n things similarly limits the

damage that can be done by a bad writer. You know it's going to

be about whatever the title says, and the format prevents the writer

from indulging in any flights of fancy.Because the list of n things is the easiest essay form, it should

be a good one for beginning writers. And in fact it is what most

beginning writers are taught. The classic 5 paragraph essay is

really a list of n things for n = 3. But the students writing them

don't realize they're using the same structure as the articles they

read in Cosmopolitan. They're not allowed to include the numbers,

and they're expected to spackle over the gaps with gratuitous

transitions ("Furthermore...") and cap the thing at either end with

introductory and concluding paragraphs so it will look superficially

like a real essay.

[2]It seems a fine plan to start students off with the list of n things.

It's the easiest form. But if we're going to do that, why not do

it openly? Let them write lists of n things like the pros, with

numbers and no transitions or "conclusion."There is one case where the list of n things is a dishonest format:

when you use it to attract attention by falsely claiming the list

is an exhaustive one. I.e. if you write an article that purports

to be about the 7 secrets of success. That kind of title is the

same sort of reflexive challenge as a whodunit. You have to at least

look at the article to check whether they're the same 7 you'd list.

Are you overlooking one of the secrets of success? Better check.It's fine to put "The" before the number if you really believe

you've made an exhaustive list. But evidence suggests most things

with titles like this are linkbait.The greatest weakness of the list of n things is that there's so

little room for new thought. The main point of essay writing, when

done right, is the new ideas you have while doing it. A real essay,

as the name implies, is

dynamic: you don't know what you're going

to write when you start. It will be about whatever you discover

in the course of writing it.This can only happen in a very limited way in a list of n things.

You make the title first, and that's what it's going to be about.

You can't have more new ideas in the writing than will fit in the

watertight compartments you set up initially. And your brain seems

to know this: because you don't have room for new ideas, you don't

have them.Another advantage of admitting to beginning writers that the 5

paragraph essay is really a list of n things is that we can warn

them about this. It only lets you experience the defining

characteristic of essay writing on a small scale: in thoughts of a

sentence or two. And it's particularly dangerous that the 5 paragraph

essay buries the list of n things within something that looks like

a more sophisticated type of essay. If you don't know you're using

this form, you don't know you need to escape it.Notes[1]

Articles of this type are also startlingly popular on Delicious,

but I think that's because

delicious/popular

is driven by bookmarking,

not because Delicious users are stupid. Delicious users are

collectors, and a list of n things seems particularly collectible

because it's a collection itself.[2]

Most "word problems" in school math textbooks are similarly

misleading. They look superficially like the application of math

to real problems, but they're not. So if anything they reinforce

the impression that math is merely a complicated but pointless

collection of stuff to be memorized.Russian Translation

The Anatomy of Determination

Want to start a startup? Get funded by

Y Combinator.

September 2009Like all investors, we spend a lot of time trying to learn how to

predict which startups will succeed. We probably spend more time

thinking about it than most, because we invest the earliest.

Prediction is usually all we have to rely on.We learned quickly that the most important predictor of success is

determination. At first we thought it might be intelligence.

Everyone likes to believe that's what makes startups succeed. It

makes a better story that a company won because its founders were

so smart. The PR people and reporters who spread such stories

probably believe them themselves. But while it certainly helps to

be smart, it's not the deciding factor. There are plenty of people

as smart as Bill Gates who achieve nothing.In most domains, talent is overrated compared to determination—partly

because it makes a better story, partly because it gives onlookers

an excuse for being lazy, and partly because after a while determination

starts to look like talent.I can't think of any field in which determination is overrated, but

the relative importance of determination and talent probably do

vary somewhat. Talent probably matters more in types of work that

are purer, in the sense that one is solving mostly a single type

of problem instead of many different types. I suspect determination

would not take you as far in math as it would in, say, organized

crime.I don't mean to suggest by this comparison that types of work that

depend more on talent are always more admirable. Most people would

agree it's more admirable to be good at math than memorizing long

strings of digits, even though the latter depends more on natural

ability.Perhaps one reason people believe startup founders win by being

smarter is that intelligence does matter more in technology startups

than it used to in earlier types of companies. You probably do

need to be a bit smarter to dominate Internet search than you had

to be to dominate railroads or hotels or newspapers. And that's

probably an ongoing trend. But even in the highest of high tech

industries, success still depends more on determination than brains.If determination is so important, can we isolate its components?

Are some more important than others? Are there some you can

cultivate?The simplest form of determination is sheer willfulness. When you

want something, you must have it, no matter what.A good deal of willfulness must be inborn, because it's common to

see families where one sibling has much more of it than another.

Circumstances can alter it, but at the high end of the scale, nature

seems to be more important than nurture. Bad circumstances can

break the spirit of a strong-willed person, but I don't think there's

much you can do to make a weak-willed person stronger-willed.Being strong-willed is not enough, however. You also have to be

hard on yourself. Someone who was strong-willed but self-indulgent

would not be called determined. Determination implies your willfulness

is balanced by discipline.That word balance is a significant one. The more willful you are,

the more disciplined you have to be. The stronger your will, the

less anyone will be able to argue with you except yourself. And

someone has to argue with you, because everyone has base impulses,

and if you have more will than discipline you'll just give into

them and end up on a local maximum like drug addiction.We can imagine will and discipline as two fingers squeezing a

slippery melon seed. The harder they squeeze, the further the seed

flies, but they must both squeeze equally or the seed spins off

sideways.If this is true it has interesting implications, because discipline

can be cultivated, and in fact does tend to vary quite a lot in the

course of an individual's life. If determination is effectively

the product of will and discipline, then you can become more

determined by being more disciplined.

[1]Another consequence of the melon seed model is that the more willful

you are, the more dangerous it is to be undisciplined. There seem

to be plenty of examples to confirm that. In some very energetic

people's lives you see something like wing flutter, where they

alternate between doing great work and doing absolutely nothing.

Externally this would look a lot like bipolar disorder.The melon seed model is inaccurate in at least one respect, however:

it's static. In fact the dangers of indiscipline increase with

temptation. Which means, interestingly, that determination tends

to erode itself. If you're sufficiently determined to achieve great

things, this will probably increase the number of temptations around

you. Unless you become proportionally more disciplined, willfulness

will then get the upper hand, and your achievement will revert to

the mean.That's why Julius Caesar thought thin men so dangerous. They weren't

tempted by the minor perquisites of power.The melon seed model implies it's possible to be too disciplined.

Is it? I think there probably are people whose willfulness is

crushed down by excessive discipline, and who would achieve more

if they weren't so hard on themselves. One reason the young sometimes

succeed where the old fail is that they don't realize how incompetent

they are. This lets them do a kind of deficit spending. When they

first start working on something, they overrate their achievements.

But that gives them confidence to keep working, and their performance

improves. Whereas someone clearer-eyed would see their initial

incompetence for what it was, and perhaps be discouraged from

continuing.There's one other major component of determination: ambition. If

willfulness and discipline are what get you to your destination,

ambition is how you choose it.I don't know if it's exactly right to say that ambition is a component

of determination, but they're not entirely orthogonal. It would

seem a misnomer if someone said they were very determined to do

something trivially easy.And fortunately ambition seems to be quite malleable; there's a lot

you can do to increase it. Most people don't know how ambitious

to be, especially when they're young. They don't know what's hard,

or what they're capable of. And this problem is exacerbated by

having few peers. Ambitious people are rare, so if everyone is

mixed together randomly, as they tend to be early in people's lives,

then the ambitious ones won't have many ambitious peers. When you

take people like this and put them together with other ambitious

people, they bloom like dying plants given water. Probably most

ambitious people are starved for the sort of encouragement they'd

get from ambitious peers, whatever their age.

[2]Achievements also tend to increase your ambition. With each step

you gain confidence to stretch further next time.So here in sum is how determination seems to work: it consists of

willfulness balanced with discipline, aimed by ambition. And

fortunately at least two of these three qualities can be cultivated.

You may be able to increase your strength of will somewhat; you can

definitely learn self-discipline; and almost everyone is practically

malnourished when it comes to ambition.I feel like I understand determination a bit better now. But only

a bit: willfulness, discipline, and ambition are all concepts almost

as complicated as determination.

[3]Note too that determination and talent are not the whole story.

There's a third factor in achievement: how much you like the work.

If you really love working on something,

you don't need determination to drive you; it's what you'd do anyway.

But most types of work have aspects one doesn't like, because most

types of work consist of doing things for other people, and it's

very unlikely that the tasks imposed by their needs will happen to

align exactly with what you want to do.Indeed, if you want to create the most wealth,

the way to do it is to focus more on their needs than your interests,

and make up the difference with determination.Notes[1]

Loosely speaking. What I'm claiming with the melon seed model

is more like determination is proportionate to wd^m - k|w - d|^n,

where w is will and d discipline.[2]

Which means one of the best ways to help a society generally

is to create events and institutions that bring ambitious

people together. It's like pulling the control rods out of a

reactor: the energy they emit encourages other ambitious people,

instead of being absorbed by the normal people they're usually

surrounded with.Conversely, it's probably a mistake to do as some European countries

have done and try to ensure none of your universities is significantly

better than the others.[3]

For example, willfulness clearly has two subcomponents,

stubbornness and energy. The first alone yields someone who's

stubbornly inert. The second alone yields someone flighty.

As willful people get older or otherwise lose their energy, they

tend to become merely stubborn.

Thanks to Sam Altman, Jessica Livingston, and Robert Morris

for reading drafts of this.Italian TranslationPortuguese TranslationRussian Translation

What Kate Saw in Silicon Valley

August 2009Kate Courteau is the architect who designed Y Combinator's office.

Recently we managed to recruit her to help us run YC when she's not

busy with architectural projects. Though she'd heard a lot about

YC since the beginning, the last 9 months have been a total immersion.I've been around the startup world for so long that it seems normal

to me, so I was curious to hear what had surprised her most about

it. This was her list:1. How many startups fail.Kate knew in principle that startups

were very risky, but she was surprised to see how constant the

threat of failure was — not just for the minnows, but even for the

famous startups whose founders came to speak at YC dinners.

2. How much startups' ideas change.As usual, by Demo Day about

half the startups were doing something significantly different than

they started with. We encourage that. Starting a startup is like

science in that you have to follow the truth wherever it leads. In

the rest of the world, people don't start things till they're sure

what they want to do, and once started they tend continue on their

initial path even if it's mistaken.

3. How little money it can take to start a startup.In Kate's

world, everything is still physical and expensive. You can barely

renovate a bathroom for the cost of starting a startup.

4. How scrappy founders are.That was her actual word. I agree

with her, but till she mentioned this it never occurred to me how

little this quality is appreciated in most of the rest of the world.

It wouldn't be a compliment in most organizations to call someone

scrappy.What does it mean, exactly? It's basically the diminutive form of

belligerent. Someone who's scrappy manages to be both threatening

and undignified at the same time. Which seems to me exactly what

one would want to be, in any kind of work. If you're not threatening,

you're probably not doing anything new, and dignity is merely a

sort of plaque.

5. How tech-saturated Silicon Valley is."It seems like everybody

here is in the industry." That isn't literally true, but there is

a qualitative difference between Silicon Valley and other places.

You tend to keep your voice down, because there's a good chance the

person at the next table would know some of the people you're talking

about. I never felt that in Boston. The good news is, there's

also a good chance the person at the next table could help you in

some way.

6. That the speakers at YC were so consistent in their advice.

Actually, I've noticed this too. I always worry the speakers will

put us in an embarrassing position by contradicting what we tell the

startups, but it happens surprisingly rarely.When I asked her what specific things she remembered speakers always

saying, she mentioned: that the way to succeed was to launch something

fast, listen to users, and then iterate; that startups required

resilience because they were always an emotional rollercoaster; and

that most VCs were sheep.I've been impressed by how consistently the speakers advocate

launching fast and iterating. That was contrarian advice 10 years

ago, but it's clearly now the established practice.

7. How casual successful startup founders are.Most of the famous

founders in Silicon Valley are people you'd overlook on the street.

It's not merely that they don't dress up. They don't project any

kind of aura of power either. "They're not trying to impress

anyone."Interestingly, while Kate said that she could never pick out

successful founders, she could recognize VCs, both by the way they

dressed and the way they carried themselves.

8. How important it is for founders to have people to ask for advice.(I swear I didn't prompt this one.) Without advice "they'd just

be sort of lost." Fortunately, there are a lot of people to help

them. There's a strong tradition within YC of helping other YC-funded

startups. But we didn't invent that idea: it's just a slightly

more concentrated form of existing Valley culture.

9. What a solitary task startups are.Architects are constantly

interacting face to face with other people, whereas doing a technology

startup, at least, tends to require long stretches of uninterrupted

time to work. "You could do it in a box."By inverting this list, we can get a portrait of the "normal" world.

It's populated by people who talk a lot with one another as they

work slowly but harmoniously on conservative, expensive projects

whose destinations are decided in advance, and who carefully adjust

their manner to reflect their position in the hierarchy.That's also a fairly accurate description of the past. So startup

culture may not merely be different in the way you'd expect any

subculture to be, but a leading indicator.Japanese Translation

The Trouble with the Segway

July 2009The Segway hasn't delivered on its initial promise, to put it mildly.

There are several reasons why, but one is that people don't want

to be seen riding them. Someone riding a Segway looks like a dork.My friend Trevor Blackwell built

his own Segway,

which we called

the Segwell. He also built a one-wheeled version,

the Eunicycle,

which looks exactly like a regular unicycle till you realize the

rider isn't pedaling. He has ridden them both to downtown Mountain

View to get coffee. When he rides the Eunicycle, people smile at

him. But when he rides the Segwell, they shout abuse from their

cars: "Too lazy to walk, ya fuckin homo?"Why do Segways provoke this reaction? The reason you look like a

dork riding a Segway is that you look smug. You don't seem to

be working hard enough.Someone riding a motorcycle isn't working any harder. But because

he's sitting astride it, he seems to be making an effort. When

you're riding a Segway you're just standing there. And someone who's

being whisked along while seeming to do no work—someone in a sedan

chair, for example—can't help but look smug.Try this thought experiment and it becomes clear: imagine something

that worked like the Segway, but that you rode with one foot in

front of the other, like a skateboard. That wouldn't seem nearly

as uncool.So there may be a way to capture more of the market Segway hoped

to reach: make a version that doesn't look so easy for the rider.

It would also be helpful if the styling was in the tradition of

skateboards or bicycles rather than medical devices.Curiously enough, what got Segway into this problem was that the

company was itself a kind of Segway. It was too easy for them;

they were too successful raising money. If they'd had to grow the

company gradually, by iterating through several versions they sold

to real users, they'd have learned pretty quickly that people looked

stupid riding them. Instead they had enough to work in secret. They

had focus groups aplenty, I'm sure, but they didn't have the people

yelling insults out of cars. So they never realized they were

zooming confidently down a blind alley.

Ramen Profitable

Want to start a startup? Get funded by

Y Combinator.

July 2009Now that the term "ramen profitable" has become widespread, I ought

to explain precisely what the idea entails.Ramen profitable means a startup makes just enough to pay the

founders' living expenses. This is a different form of profitability

than startups have traditionally aimed for. Traditional profitability

means a big bet is finally paying off, whereas the main importance

of ramen profitability is that it buys you time.

[1]In the past, a startup would usually become profitable only

after raising and spending quite a lot of money. A company making

computer hardware might not become profitable for 5 years, during

which they spent $50 million. But when they did

they might have revenues of $50 million a year. This kind of

profitability means the startup has succeeded.Ramen profitability is the other extreme: a startup that becomes

profitable after 2 months, even though its revenues are only $3000

a month, because the only employees are a couple 25 year old founders

who can live on practically nothing. Revenues of $3000 a month do

not mean the company has succeeded.

But it does share something with the one

that's profitable in the traditional way: they don't need to raise

money to survive.Ramen profitability is an unfamiliar idea to most people because

it only recently became feasible. It's still not feasible for a

lot of startups; it would not be for most biotech startups, for

example; but it is for many software startups because they're now

so cheap. For many, the only real cost is the founders'

living expenses.The main significance of this type of profitability is that you're

no longer at the mercy of investors. If you're still losing money,

then eventually you'll either have to raise more

or shut down. Once you're

ramen profitable this painful choice goes away.

You can still raise money, but you don't have to do it now.\* \* \*The most obvious advantage of not needing money is that

you can get better terms. If investors know you need money, they'll

sometimes take advantage of you. Some may even deliberately

stall, because they know that as you run out of money you'll become

increasingly pliable.But there are also three less obvious advantages of ramen profitability.

One is that it makes you more attractive to investors. If you're

already profitable, on however small a scale, it shows that (a) you

can get at least someone to pay you, (b) you're serious about

building things people want, and (c) you're disciplined enough to

keep expenses low.This is reassuring to investors, because you've addressed three of

their biggest worries. It's common for them to fund companies that

have smart founders and a big market, and yet still fail. When

these companies fail, it's usually because (a) people wouldn't pay

for what they made, e.g. because it was too hard to sell to them,

or the market wasn't ready yet, (b) the founders solved the wrong

problem, instead of paying attention to what users needed, or (c)

the company spent too much and burned through their funding before

they started to make money. If you're ramen profitable, you're

already avoiding these mistakes.Another advantage of ramen profitability is that it's good for

morale. A company

tends to feel rather theoretical when you first start it. It's

legally a company, but you feel like you're lying when you call it

one. When people start to pay you significant amounts, the company

starts to feel real. And your own living expenses are the milestone

you feel most, because at that point the future flips state. Now

survival is the default, instead of dying.A morale boost on that scale is very valuable in a startup, because

the moral weight of running a startup is what makes it hard. Startups

are still very rare. Why don't more people do it? The financial

risk? Plenty of 25 year olds save nothing anyway. The long hours?

Plenty of people work just as long hours in regular jobs. What keeps

people from starting startups is the fear of having so much

responsibility. And this is not an irrational fear: it really is

hard to bear. Anything that takes some of that weight off you will

greatly increase your chances of surviving.A startup that reaches ramen profitability may be more likely

to succeed than not. Which is pretty exciting, considering the

bimodal distribution of outcomes in startups: you either fail or

make a lot of money.The fourth advantage of ramen profitability is the least obvious

but may be the most important. If you don't need to raise money,

you don't have to interrupt working on the company to do it.Raising money is terribly distracting.

You're lucky if your

productivity is a third of what it was before. And it can last for

months.I didn't understand (or rather, remember) precisely why raising

money was so distracting till earlier this year. I'd noticed that

startups we funded would usually grind to a halt when they switched

to raising money, but I didn't remember exactly why till YC raised

money itself. We had a comparatively easy time of it; the first

people I asked said yes; but it took months to work out the

details, and during that time I got hardly any real work done. Why?

Because I thought about it all the time.At any given time there tends to be one problem that's the most

urgent for a startup. This is what you think about as you fall

asleep at night and when you take a shower in the morning. And

when you start raising money, that becomes the problem you think

about. You only take one shower in the morning, and if you're

thinking about investors during it, then you're not thinking about

the product.Whereas if you can choose when you raise money, you can pick a time

when you're not in the middle of something else, and you can probably

also insist that the round close fast. You may even be able to

avoid having the round occupy your thoughts, if you don't care

whether it closes.\* \* \*Ramen profitable means no more than the definition implies. It

does not, for example, imply that you're "bootstrapping" the

startup—that you're never going to take money from investors.

Empirically that doesn't seem to work very well. Few startups

succeed without taking investment. Maybe as startups get cheaper

it will become more common. On the other hand, the money is there,

waiting to be invested. If startups need it less, they'll be able

to get it on better terms, which will make them more inclined to

take it. That will tend to produce an equilibrium.

[2]Another thing ramen profitability doesn't imply is Joe Kraus's idea

that you should put your

business model in beta when you put your

product in beta. He believes you should get

people to pay you from the beginning. I think that's too constraining.

Facebook didn't, and they've done better than most startups. Making

money right away was not only unnecessary for them, but probably

would have been harmful. I do think Joe's rule could be useful for

many startups, though. When founders seem unfocused, I sometimes

suggest they try to get customers to pay them for something, in the

hope that this constraint will prod them into action.The difference between Joe's idea and ramen profitability is that

a ramen profitable company doesn't have to be making money the way

it ultimately will. It just has to be making money. The most

famous example is Google, which initially made money by licensing

search to sites like Yahoo.Is there a downside to ramen profitability? Probably the biggest

danger is that it might turn you into a consulting firm. Startups

have to be product companies, in the sense of making a single thing

that everyone uses. The defining quality of startups is that they

grow fast, and consulting just can't scale the way a product can.

[3]

But it's pretty easy to make $3000 a month consulting; in

fact, that would be a low rate for contract programming. So there

could be a temptation to slide into consulting, and telling

yourselves you're a ramen profitable startup, when in fact

you're not a startup at all.It's ok to do a little consulting-type work at first. Startups

usually have to do something weird at first. But remember

that ramen profitability is not the destination. A startup's

destination is to grow really big; ramen profitability is a trick

for not dying en route.Notes[1]

The "ramen" in "ramen profitable" refers to instant ramen,

which is just about the cheapest food available.Please do not take the term literally. Living on instant ramen

would be very unhealthy. Rice and beans are a better source of

food. Start by investing in a rice cooker, if you don't have one.Rice and Beans for 2n

olive oil or butter

n yellow onions

other fresh vegetables; experiment

3n cloves garlic

n 12-oz cans white, kidney, or black beans

n cubes Knorr beef or vegetable bouillon

n teaspoons freshly ground black pepper

3n teaspoons ground cumin

n cups dry rice, preferably brown

Put rice in rice cooker. Add water as specified on rice package.

(Default: 2 cups water per cup of rice.) Turn on rice cooker and

forget about it.Chop onions and other vegetables and fry in oil, over fairly low

heat, till onions are glassy. Put in chopped garlic, pepper, cumin,

and a little more fat, and stir. Keep heat low. Cook another 2 or

3 minutes, then add beans (don't drain the beans), and stir. Throw

in the bouillon cube(s), cover, and cook on lowish heat for at least

10 minutes more. Stir vigilantly to avoid sticking.If you want to save money, buy beans in giant cans from discount

stores. Spices are also much cheaper when bought in bulk.

If there's an Indian grocery store near you, they'll have big

bags of cumin for the same price as the little jars in supermarkets.[2]

There's a good chance that a shift in power from investors

to founders would actually increase the size of the venture business.

I think investors currently err too far on the side of being harsh

to founders. If they were forced to stop, the whole venture business

would work better, and you might see something like the increase

in trade you always see when restrictive laws are removed.Investors

are one of the biggest sources of pain for founders; if they stopped

causing so much pain, it would be better to be a founder; and if

it were better to be a founder, more people would do it.[3]

It's conceivable that a startup could grow big by transforming

consulting into a form that would scale. But if they did that

they'd really be a product company.Thanks to Jessica Livingston for reading drafts of this.Japanese Translation

Maker's Schedule, Manager's Schedule

"...the mere consciousness of an engagement will sometimes worry a whole day." Charles Dickens

July 2009One reason programmers dislike meetings so much is that they're on

a different type of schedule from other people. Meetings cost them

more.There are two types of schedule, which I'll call the manager's

schedule and the maker's schedule. The manager's schedule is for

bosses. It's embodied in the traditional appointment book, with

each day cut into one hour intervals. You can block off several

hours for a single task if you need to, but by default you change

what you're doing every hour.When you use time that way, it's merely a practical problem to meet

with someone. Find an open slot in your schedule, book them, and

you're done.Most powerful people are on the manager's schedule. It's the

schedule of command. But there's another way of using time that's

common among people who make things, like programmers and writers.

They generally prefer to use time in units of half a day at least.

You can't write or program well in units of an hour. That's barely

enough time to get started.When you're operating on the maker's schedule, meetings are a

disaster. A single meeting can blow a whole afternoon, by breaking

it into two pieces each too small to do anything hard in. Plus you

have to remember to go to the meeting. That's no problem for someone

on the manager's schedule. There's always something coming on the

next hour; the only question is what. But when someone on the

maker's schedule has a meeting, they have to think about it.For someone on the maker's schedule, having a meeting is like

throwing an exception. It doesn't merely cause you to switch from

one task to another; it changes the mode in which you work.I find one meeting can sometimes affect a whole day. A meeting

commonly blows at least half a day, by breaking up a morning or

afternoon. But in addition there's sometimes a cascading effect.

If I know the afternoon is going to be broken up, I'm slightly less

likely to start something ambitious in the morning. I know this

may sound oversensitive, but if you're a maker, think of your own

case. Don't your spirits rise at the thought of having an entire

day free to work, with no appointments at all? Well, that means

your spirits are correspondingly depressed when you don't. And

ambitious projects are by definition close to the limits of your

capacity. A small decrease in morale is enough to kill them off.Each type of schedule works fine by itself. Problems arise when

they meet. Since most powerful people operate on the manager's

schedule, they're in a position to make everyone resonate at their

frequency if they want to. But the smarter ones restrain themselves,

if they know that some of the people working for them need long

chunks of time to work in.Our case is an unusual one. Nearly all investors, including all

VCs I know, operate on the manager's schedule. But

Y Combinator

runs on the maker's schedule. Rtm and Trevor and I do because we

always have, and Jessica does too, mostly, because she's gotten

into sync with us.I wouldn't be surprised if there start to be more companies like

us. I suspect founders may increasingly be able to resist, or at

least postpone, turning into managers, just as a few decades ago

they started to be able to resist switching from jeans

to suits.How do we manage to advise so many startups on the maker's schedule?

By using the classic device for simulating the manager's schedule

within the maker's: office hours. Several times a week I set aside

a chunk of time to meet founders we've funded. These chunks of

time are at the end of my working day, and I wrote a signup program

that ensures all the appointments within a given set of office hours

are clustered at the end. Because they come at the end of my day

these meetings are never an interruption. (Unless their working

day ends at the same time as mine, the meeting presumably interrupts

theirs, but since they made the appointment it must be worth it to

them.) During busy periods, office hours sometimes get long enough

that they compress the day, but they never interrupt it.

When we were working on our own startup, back in the 90s, I evolved

another trick for partitioning the day. I used to program from

dinner till about 3 am every day, because at night no one could

interrupt me. Then I'd sleep till about 11 am, and come in and

work until dinner on what I called "business stuff." I never thought

of it in these terms, but in effect I had two workdays each day,

one on the manager's schedule and one on the maker's.When you're operating on the manager's schedule you can do something

you'd never want to do on the maker's: you can have speculative

meetings. You can meet someone just to get to know one another.

If you have an empty slot in your schedule, why not? Maybe it will

turn out you can help one another in some way.Business people in Silicon Valley (and the whole world, for that

matter) have speculative meetings all the time. They're effectively

free if you're on the manager's schedule. They're so common that

there's distinctive language for proposing them: saying that you

want to "grab coffee," for example.Speculative meetings are terribly costly if you're on the maker's

schedule, though. Which puts us in something of a bind. Everyone

assumes that, like other investors, we run on the manager's schedule.

So they introduce us to someone they think we ought to meet, or

send us an email proposing we grab coffee. At this point we have

two options, neither of them good: we can meet with them, and lose

half a day's work; or we can try to avoid meeting them, and probably

offend them.Till recently we weren't clear in our own minds about the source

of the problem. We just took it for granted that we had to either

blow our schedules or offend people. But now that I've realized

what's going on, perhaps there's a third option: to write something

explaining the two types of schedule. Maybe eventually, if the

conflict between the manager's schedule and the maker's schedule

starts to be more widely understood, it will become less of a

problem.Those of us on the maker's schedule are willing to compromise. We

know we have to have some number of meetings. All we ask from those

on the manager's schedule is that they understand the cost.

Thanks to Sam Altman, Trevor Blackwell, Paul Buchheit, Jessica Livingston,

and Robert Morris for reading drafts of this.Related:How to Do What You LoveGood and Bad ProcrastinationTurkish TranslationFrench TranslationKorean TranslationGerman Translation

A Local Revolution?

April 2009Recently I realized I'd been holding two ideas in my head that would explode if combined.The first is that startups may represent a new economic phase, on the scale of the Industrial Revolution. I'm not sure of this, but there seems a decent chance it's true. People are dramatically more

productive as founders or early employees of startups—imagine how much less Larry and Sergey would have achieved if they'd gone to work for a big company—and that scale of improvement can change social customs.The second idea is that startups are a type of business that flourishes in certain places that specialize in it—that Silicon Valley specializes in startups in the same way Los Angeles specializes in movies, or New York in finance. [1]What if both are true? What if startups are both a new economic phase and also a type of business that only flourishes in certain centers?If so, this revolution is going to be particularly revolutionary. All previous revolutions have spread. Agriculture, cities, and industrialization all spread widely. If startups end up being like the movie business, with just a handful of centers and one dominant one, that's going to have novel consequences.There are already signs that startups may not spread particularly well. The spread of startups seems to be proceeding slower than the spread of the Industrial Revolution, despite the fact that communication is so much faster now.Within a few decades of the founding of Boulton & Watt there were steam engines scattered over northern Europe and North America. Industrialization didn't spread much beyond those regions for a while. It only spread to places where there was a strong middle class—countries where a private citizen could make a fortune without having it confiscated. Otherwise it wasn't worth investing in factories. But in a country with a strong middle class it was easy for industrial techniques to take root. An individual mine or factory owner could decide to install a steam engine, and within a few years he could probably find someone local to make him one. So steam engines spread fast. And they spread widely, because the locations of mines and factories were determined by features like rivers, harbors, and sources of raw materials.

[2]Startups don't seem to spread so well, partly because they're more a social than a technical phenomenon, and partly because they're not tied to geography. An individual European manufacturer could import industrial techniques and they'd work fine. This doesn't seem to work so well with startups: you need a community of expertise, as you do in the movie business. [3]

Plus there aren't the same forces driving startups to spread. Once railroads or electric power grids were invented, every region had to have them. An area without railroads or power was a rich potential market. But this isn't true with startups. There's no need for a Microsoft of France or Google of Germany.Governments may decide they want to encourage startups locally, but government policy can't call them into being the way a genuine need could.How will this all play out? If I had to predict now, I'd say that startups will spread, but very slowly, because their spread will be driven not by government policies (which won't work) or by market need (which doesn't exist) but, to the extent that it happens at all, by the same random factors that have caused startup culture to spread thus far. And such random factors will increasingly be outweighed by the pull of existing startup hubs.Silicon Valley is where it is because William Shockley wanted to move back to Palo Alto, where he grew up, and the experts he lured west to work with him liked it so much they stayed. Seattle owes much of its position as a tech center to the same cause: Gates and Allen wanted to move home. Otherwise Albuquerque might have Seattle's place in the rankings. Boston is a tech center because it's the intellectual capital of the US and probably the world. And if Battery Ventures hadn't turned down Facebook, Boston would be significantly bigger now on the startup radar screen.But of course it's not a coincidence that Facebook got funded in the Valley and not Boston. There are more and bolder investors in Silicon Valley than in Boston, and even undergrads know it.Boston's case illustrates the difficulty you'd have establishing a new startup hub this late in the game. If you wanted to create a startup hub by reproducing the way existing ones happened, the

way to do it would be to establish a first-rate research university in a place so nice that rich people wanted to live there. Then the town would be hospitable to both groups you need: both founders and investors. That's the combination that yielded Silicon Valley. But Silicon Valley didn't have Silicon Valley to compete with. If you tried now to create a startup hub by planting a great university in a nice place, it would have a harder time getting started, because many of the best startups it produced would be sucked away to existing startup hubs.Recently I suggested a potential shortcut:

pay startups to move. Once you had enough good startups in one place, it would create a self-sustaining chain reaction. Founders would start to move there without being paid, because that was where their peers were, and investors would appear too, because that was where the deals were.In practice I doubt any government would have the balls to try this, or the brains to do it right. I didn't mean it as a practical suggestion, but more as an exploration of the lower bound of what it would take to create a startup hub deliberately.The most likely scenario is (1) that no government will successfully establish a startup hub, and (2) that the spread of startup culture will thus be driven by the random factors that have driven it so far, but (3) that these factors will be increasingly outweighed by the pull of existing startup hubs. Result: this revolution, if it is one, will be unusually localized.

Notes[1]

There are two very different types of startup: one kind that evolves naturally, and one kind that's called into being to "commercialize" a scientific discovery. Most computer/software startups are now the first type, and most pharmaceutical startups the second. When I talk about startups in this essay, I mean type I startups. There is no difficulty making type II startups spread: all you have to do is fund medical research labs; commercializing whatever new discoveries the boffins throw off is as straightforward as building a new airport. Type II startups neither require nor produce startup culture. But that means having type II startups won't get you type I startups. Philadelphia is a case in point: lots of type II startups, but hardly any type I.Incidentally, Google may appear to be an instance of a type II startup, but it wasn't. Google is not pagerank commercialized. They could have used another algorithm and everything would have turned out the same. What made Google Google is that they cared about doing search well at a critical point in the evolution of the web.[2]

Watt didn't invent the steam engine. His critical invention was a refinement that made steam engines dramatically more efficient: the separate condenser. But that oversimplifies his role. He had such a different attitude to the problem and approached it with such energy that he transformed the field. Perhaps the most accurate way to put it would be to say that Watt reinvented the steam engine.[3]

The biggest counterexample here is Skype. If you're doing

something that would get shut down in the US, it becomes an

advantage to be located elsewhere. That's why Kazaa took

the place of Napster. And the expertise and connections the

founders gained from running Kazaa helped ensure the success

of Skype.Thanks to Patrick Collison, Jessica Livingston, and Fred Wilson for reading drafts of this.

Why Twitter is a Big Deal

April 2009Om Malik is the most recent of many people

to ask why Twitter is such a big deal.The reason is that it's a new messaging

protocol, where you don't specify the recipients.

New protocols are rare. Or more precisely, new

protocols that take off are.

There are only a handful of commonly used ones: TCP/IP

(the Internet), SMTP (email), HTTP (the web), and so on. So any

new protocol is a big deal. But Twitter is a protocol owned

by a private company. That's even rarer.Curiously, the fact that the founders of Twitter

have been slow to monetize it may in the long run

prove to be an advantage. Because they haven't tried

to control it too much, Twitter feels to everyone like

previous protocols. One forgets it's owned by a

private company. That must have made it easier for

Twitter to spread.

The Founder Visa

April 2009I usually avoid politics, but since we now seem to have an administration that's open to suggestions, I'm going to risk making one. The single biggest thing the government could do to increase the number of startups in this country is a policy that would cost nothing: establish a new class of visa for startup founders.The biggest constraint on the number of new startups that get created in the US is not tax policy or employment law or even Sarbanes-Oxley. It's that we won't let the people who want to start them into the country.Letting just 10,000 startup founders into the country each year could have a visible effect on the economy. If we assume 4 people per startup, which is probably an overestimate, that's 2500 new companies. Each year. They wouldn't all grow as big as Google, but out of 2500 some would come close.By definition these 10,000 founders wouldn't be taking jobs from Americans: it could be part of the terms of the visa that they couldn't work for existing companies, only new ones they'd founded. In fact they'd cause there to be

more jobs for Americans, because the companies they started would hire more employees as they grew.The tricky part might seem to be how one defined a startup. But that could be solved quite easily: let the market decide. Startup investors work hard to find the best startups. The government could not do better than to piggyback on their expertise, and use investment by recognized startup investors as the test of whether a company was a real startup.How would the government decide who's a startup investor? The same way they decide what counts as a university for student visas. We'll establish our own accreditation procedure. We know who one another are.10,000 people is a drop in the bucket by immigration standards, but would represent a huge increase in the pool of startup founders. I think this would have such a visible effect on the economy that it would make the legislator who introduced the bill famous. The only way to know for sure would be to try it, and that would cost practically nothing.

Thanks to Trevor Blackwell, Paul Buchheit, Jeff Clavier, David Hornik, Jessica Livingston, Greg Mcadoo, Aydin Senkut, and Fred Wilson for reading drafts of this.Related:The United States of Entrepreneurs About Half of VC-Backed Company Founders are Immigrants

Five Founders

April 2009Inc recently asked me who I thought were the 5 most

interesting startup founders of the last 30 years. How do

you decide who's the most interesting? The best test seemed

to be influence: who are the 5

who've influenced me most? Who do I use as examples when I'm

talking to companies we fund? Who do I find myself quoting?1. Steve JobsI'd guess Steve is the most influential founder not just for me but

for most people you could ask. A lot of startup culture is Apple

culture. He was the original young founder. And while the concept

of "insanely great" already existed in the arts, it was a novel

idea to introduce into a company in the 1980s.More remarkable still, he's stayed interesting for 30 years. People

await new Apple products the way they'd await new books by a popular

novelist. Steve may not literally design them, but they wouldn't

happen if he weren't CEO.Steve is clever and driven, but so are a lot of people in the Valley.

What makes him unique is his

sense of

design. Before him, most

companies treated design as a frivolous extra. Apple's competitors

now know better.2. TJ RodgersTJ Rodgers isn't as famous as Steve Jobs, but he may be the best

writer among Silicon Valley CEOs. I've probably learned more from

him about the startup way of thinking than from anyone else. Not

so much from specific things he's written as by reconstructing the

mind that produced them: brutally candid; aggressively garbage-collecting

outdated ideas; and yet driven by pragmatism rather than ideology.The first essay of his that I read was so electrifying that I

remember exactly where I was at the time. It was

High

Technology Innovation: Free Markets or Government Subsidies? and

I was downstairs in the Harvard Square T Station. It felt as if

someone had flipped on a light switch inside my head.3. Larry & SergeyI'm sorry to treat Larry and Sergey as one person. I've always

thought that was unfair to them. But it does seem as if Google was a

collaboration.Before Google, companies in Silicon Valley already knew it was

important to have the best hackers. So they claimed, at least.

But Google pushed this idea further than anyone had before. Their

hypothesis seems to have been that, in the initial stages at least,

all you need is good hackers: if you hire all the smartest people

and put them to work on a problem where their success can be measured,

you win. All the other stuff—which includes all the stuff that

business schools think business consists of—you can figure out

along the way. The results won't be perfect, but they'll be optimal.

If this was their hypothesis, it's now been verified experimentally.4. Paul BuchheitFew know this, but one person, Paul Buchheit, is responsible for

three of the best things Google has done. He was the original

author of GMail, which is the most impressive thing Google has after

search. He also wrote the first prototype of AdSense, and was the

author of Google's mantra "Don't be evil."PB made a point in a talk once that I now mention to every startup

we fund: that it's better, initially, to make a small number of

users really love you than a large number kind of like you. If I

could tell startups only

ten sentences,

this would be one of them.Now he's cofounder of a startup called Friendfeed. It's only a

year old, but already everyone in the Valley is watching them.

Someone responsible for three of the biggest ideas at Google is

going to come up with more.5. Sam AltmanI was told I shouldn't mention founders of YC-funded companies in

this list. But Sam Altman can't be stopped by such flimsy rules.

If he wants to be on this list, he's going to be.Honestly, Sam is, along with Steve Jobs, the founder I refer to

most when I'm advising startups. On questions of design, I ask

"What would Steve do?" but on questions of strategy or ambition I

ask "What would Sama do?"What I learned from meeting Sama is that the doctrine of the elect

applies to startups. It applies way less than most people think:

startup investing does not consist of trying to pick winners the

way you might in a horse race. But there are a few people with

such force of will that they're going to get whatever they want.

Relentlessly Resourceful

Want to start a startup? Get funded by

Y Combinator.

March 2009A couple days ago I finally got being a good startup founder down

to two words: relentlessly resourceful.Till then the best I'd managed was to get the opposite quality down

to one: hapless. Most dictionaries say hapless means unlucky. But

the dictionaries are not doing a very good job. A team that outplays

its opponents but loses because of a bad decision by the referee

could be called unlucky, but not hapless. Hapless implies passivity.

To be hapless is to be battered by circumstances—to let the world

have its way with you, instead of having your way with the world.

[1]Unfortunately there's no antonym of hapless, which makes it difficult

to tell founders what to aim for. "Don't be hapless" is not much

of rallying cry.It's not hard to express the quality we're looking for in metaphors.

The best is probably a running back. A good running back is not

merely determined, but flexible as well. They want to get downfield,

but they adapt their plans on the fly.Unfortunately this is just a metaphor, and not a useful one to most

people outside the US. "Be like a running back" is no better than

"Don't be hapless."But finally I've figured out how to express this quality directly.

I was writing a talk for

investors, and I had to explain what to

look for in founders. What would someone who was the opposite of

hapless be like? They'd be relentlessly resourceful. Not merely

relentless. That's not enough to make things go your way except

in a few mostly uninteresting domains. In any interesting domain,

the difficulties will be novel. Which means you can't simply plow

through them, because you don't know initially how hard they are;

you don't know whether you're about to plow through a block of foam

or granite. So you have to be resourceful. You have to keep

trying new things.Be relentlessly resourceful.That sounds right, but is it simply a description

of how to be successful in general? I don't think so. This isn't

the recipe for success in writing or painting, for example. In

that kind of work the recipe is more to be actively curious.

Resourceful implies the obstacles are external, which they generally

are in startups. But in writing and painting they're mostly internal;

the obstacle is your own obtuseness.

[2]There probably are other fields where "relentlessly resourceful"

is the recipe for success. But though other fields may share it,

I think this is the best short description we'll find of what makes

a good startup founder. I doubt it could be made more precise.Now that we know what we're looking for, that leads to other

questions. For example, can this quality be taught? After four

years of trying to teach it to people, I'd say that yes, surprisingly

often it can. Not to everyone, but to many people.

[3]

Some

people are just constitutionally passive, but others have a latent

ability to be relentlessly resourceful that only needs to be brought

out.This is particularly true of young people who have till now always

been under the thumb of some kind of authority. Being relentlessly

resourceful is definitely not the recipe for success in big companies,

or in most schools. I don't even want to think what the recipe is

in big companies, but it is certainly longer and messier, involving

some combination of resourcefulness, obedience, and building

alliances.Identifying this quality also brings us closer to answering a

question people often wonder about: how many startups there could

be. There is not, as some people seem to think, any economic upper

bound on this number. There's no reason to believe there is any

limit on the amount of newly created wealth consumers can absorb,

any more than there is a limit on the number of theorems that can

be proven. So probably the limiting factor on the number of startups

is the pool of potential founders. Some people would make good

founders, and others wouldn't. And now that we can say what makes

a good founder, we know how to put an upper bound on the size of

the pool.This test is also useful to individuals. If you want to know whether

you're the right sort of person to start a startup, ask yourself

whether you're relentlessly resourceful. And if you want to know

whether to recruit someone as a cofounder, ask if they are.You can even use it tactically. If I were running a startup, this

would be the phrase I'd tape to the mirror. "Make something people

want" is the destination, but "Be relentlessly resourceful" is how

you get there.

Notes[1]

I think the reason the dictionaries are wrong is that the

meaning of the word has shifted. No one writing a dictionary from

scratch today would say that hapless meant unlucky. But a couple

hundred years ago they might have. People were more at the mercy

of circumstances in the past, and as a result a lot of the words

we use for good and bad outcomes have origins in words about luck.When I was living in Italy, I was once trying to tell someone

that I hadn't had much success in doing something, but I couldn't

think of the Italian word for success. I spent some time trying

to describe the word I meant. Finally she said "Ah! Fortuna!"[2]

There are aspects of startups where the recipe is to be

actively curious. There can be times when what you're doing is

almost pure discovery. Unfortunately these times are a small

proportion of the whole. On the other hand, they are in research

too.[3]

I'd almost say to most people, but I realize (a) I have no

idea what most people are like, and (b) I'm pathologically optimistic

about people's ability to change.Thanks to Trevor Blackwell and Jessica Livingston for reading drafts

of this.

How to Be an Angel Investor

March 2009(This essay is derived from a talk at AngelConf.)When we sold our startup in 1998 I thought one day I'd do some angel

investing. Seven years later I still hadn't started. I put it off

because it seemed mysterious and complicated. It turns out to be

easier than I expected, and also more interesting.The part I thought was hard, the mechanics of investing, really

isn't. You give a startup money and they give you stock. You'll

probably get either preferred stock, which means stock with extra

rights like getting your money back first in a sale, or convertible

debt, which means (on paper) you're lending the company money, and

the debt converts to stock at the next sufficiently big funding

round.

[1]There are sometimes minor tactical advantages to using one or the

other. The paperwork for convertible debt is simpler. But really

it doesn't matter much which you use. Don't spend much time worrying

about the details of deal terms, especially when you first start

angel investing. That's not how you win at this game. When you

hear people talking about a successful angel investor, they're not

saying "He got a 4x liquidation preference." They're saying "He

invested in Google."That's how you win: by investing in the right startups. That is

so much more important than anything else that I worry I'm misleading

you by even talking about other things.MechanicsAngel investors often syndicate deals, which means they join together

to invest on the same terms. In a syndicate there is usually a

"lead" investor who negotiates the terms with the startup. But not

always: sometimes the startup cobbles together a syndicate of

investors who approach them independently, and the startup's lawyer

supplies the paperwork.The easiest way to get started in angel investing is to find a

friend who already does it, and try to get included in his syndicates.

Then all you have to do is write checks.Don't feel like you have to join a syndicate, though. It's not that

hard to do it yourself. You can just use the standard

series AA

documents Wilson Sonsini and Y Combinator published online.

You should of course have your lawyer review everything. Both you

and the startup should have lawyers. But the lawyers don't have

to create the agreement from scratch.

[2]

When you negotiate terms with a startup, there are two numbers you

care about: how much money you're putting in, and the valuation of

the company. The valuation determines how much stock you get. If

you put $50,000 into a company at a pre-money valuation of $1

million, then the post-money valuation is $1.05 million, and you

get .05/1.05, or 4.76% of the company's stock.If the company raises more money later, the new investor will take

a chunk of the company away from all the existing shareholders just

as you did. If in the next round they sell 10% of the company to

a new investor, your 4.76% will be reduced to 4.28%.That's ok. Dilution is normal. What saves you from being mistreated

in future rounds, usually, is that you're in the same boat as the

founders. They can't dilute you without diluting themselves just

as much. And they won't dilute themselves unless they end up

net ahead. So in theory, each further

round of investment leaves you

with a smaller share of an even more valuable company, till after

several more rounds you end up with .5% of the company at the point

where it IPOs, and you are very happy because your $50,000 has

become $5 million.

[3]The agreement by which you invest should have provisions that

let you contribute to

future rounds to maintain your percentage. So it's your choice

whether you get diluted.

[4]

If the company does really well,

you eventually will, because eventually the valuations will get so

high it's not worth it for you.How much does an angel invest? That varies enormously, from $10,000

to hundreds of thousands or in rare cases even millions. The upper

bound is obviously the total amount the founders want to raise.

The lower bound is 5-10% of the total or $10,000, whichever

is greater. A typical angel round these days might be $150,000

raised from 5 people.Valuations don't vary as much. For angel rounds it's rare to see

a valuation lower than half a million or higher than 4 or 5 million.

4 million is starting to be VC territory.How do you decide what valuation to offer? If you're part of a

round led by someone else, that problem is solved for you. But

what if you're investing by yourself? There's no real answer.

There is no rational way to value an early stage startup. The

valuation reflects nothing more than the strength of the company's

bargaining position. If they really want you, either because they

desperately need money, or you're someone who can help them a lot,

they'll let you invest at a low valuation. If they don't need you,

it will be higher. So guess. The startup may not have any more

idea what the number should be than you do.

[5]Ultimately it doesn't matter much. When angels make a lot of money

from a deal, it's not because they invested at a valuation of $1.5

million instead of $3 million. It's because the company was really

successful.I can't emphasize that too much. Don't get hung up on mechanics

or deal terms. What you should spend your time thinking about is

whether the company is good.(Similarly, founders also should not get hung up on deal

terms, but should spend their time thinking about how to make the

company good.)There's a second less obvious component of an angel investment: how

much you're expected to help the startup. Like the amount you

invest, this can vary a lot. You don't have to do anything if you

don't want to; you could simply be a source of money. Or you can

become a de facto employee of the company. Just make sure that you

and the startup agree in advance about roughly how much you'll do

for them.Really hot companies sometimes have high standards for angels. The

ones everyone wants to invest in practically audition investors,

and only take money from people who are famous and/or will work

hard for them. But don't feel like you have to put in a lot of

time or you won't get to invest in any good startups. There is a

surprising lack of correlation between how hot a deal a startup is

and how well it ends up doing. Lots of hot startups will end up

failing, and lots of startups no one likes will end up succeeding.

And the latter are so desperate for money that they'll take it from

anyone at a low valuation.

[6]Picking WinnersIt would be nice to be able to pick those out, wouldn't it? The

part of angel investing that has most effect on your returns, picking

the right companies, is also the hardest. So you should practically

ignore (or more precisely, archive, in the Gmail sense) everything

I've told you so far. You may need to refer to it at some point,

but it is not the central issue.The central issue is picking the right startups. What "Make something

people want" is for startups, "Pick the right startups" is for

investors. Combined they yield "Pick the startups that will make

something people want."How do you do that? It's not as simple as picking startups that

are already making something wildly popular. By then it's

too late for angels. VCs will already be onto them. As an angel,

you have to pick startups before they've got a hit—either

because they've made something great but users don't realize it

yet, like Google early on, or because they're still an iteration

or two away from the big hit, like Paypal when they were making

software for transferring money between PDAs.To be a good angel investor, you have to be a good judge of potential.

That's what it comes down to. VCs can be fast followers. Most of

them don't try to predict what will win. They just try to notice

quickly when something already is winning. But angels have to be

able to predict.

[7]One interesting consequence of this fact is that there are a lot

of people out there who have never even made an angel investment

and yet are already better angel investors than they realize.

Someone who doesn't know the first thing about the mechanics of

venture funding but knows what a successful startup founder looks

like is actually far ahead of someone who knows termsheets inside

out, but thinks

"hacker" means someone who breaks into computers.

If you can recognize good startup founders by empathizing with

them—if you both resonate at the same frequency—then

you may already be a better startup picker than the median professional

VC.

[8]Paul Buchheit, for example, started angel investing about a year

after me, and he was pretty much immediately as good as me at picking

startups. My extra year of experience was rounding error compared

to our ability to empathize with founders.What makes a good founder? If there were a word that meant the

opposite of hapless, that would be the one. Bad founders seem

hapless. They may be smart, or not, but somehow events overwhelm

them and they get discouraged and give up. Good founders make

things happen the way they want. Which is not to say they force

things to happen in a predefined way. Good founders have a healthy

respect for reality. But they are relentlessly resourceful. That's

the closest I can get to the opposite of hapless. You want to fund

people who are relentlessly resourceful.Notice we started out talking about things, and now we're talking

about people. There is an ongoing debate between investors which

is more important, the people, or the idea—or more precisely,

the market. Some, like Ron Conway, say it's the people—that

the idea will change, but the people are the foundation of the

company. Whereas Marc Andreessen says he'd back ok founders in a

hot market over great founders in a bad one.

[9]These two positions are not so far apart as they seem, because good

people find good markets. Bill Gates would probably have ended up

pretty rich even if IBM hadn't happened to drop the PC standard in

his lap.I've thought a lot about the disagreement between the investors who

prefer to bet on people and those who prefer to bet on markets.

It's kind of surprising that it even exists. You'd expect opinions

to have converged more.But I think I've figured out what's going on. The three most

prominent people I know who favor markets are Marc, Jawed Karim,

and Joe Kraus. And all three of them, in their own startups,

basically flew into a thermal: they hit a market growing so fast

that it was all they could do to keep up with it. That kind of

experience is hard to ignore. Plus I think they underestimate

themselves: they think back to how easy it felt to ride that huge

thermal upward, and they think "anyone could have done it." But

that isn't true; they are not ordinary people.So as an angel investor I think you want to go with Ron Conway and

bet on people. Thermals happen, yes, but no one can predict

them—not even the founders, and certainly not you as an

investor. And only good people can ride the thermals if they hit

them anyway.Deal FlowOf course the question of how to choose startups presumes you

have startups to choose between. How do you find them? This is

yet another problem that gets solved for you by syndicates. If you

tag along on a friend's investments, you don't have to find startups.The problem is not finding startups, exactly, but finding a stream

of reasonably high quality ones. The traditional way to do this

is through contacts. If you're friends with a lot of investors and

founders, they'll send deals your way. The Valley basically runs

on referrals. And once you start to become known as reliable,

useful investor, people will refer lots of deals to you. I certainly

will.There's also a newer way to find startups, which is to come to

events like Y Combinator's Demo Day, where a batch of newly created

startups presents to investors all at once. We have two Demo Days

a year, one in March and one in August. These are basically mass

referrals.But events like Demo Day only account for a fraction of matches

between startups and investors. The personal referral is still the

most common route. So if you want to hear about new startups, the

best way to do it is to get lots of referrals.The best way to get lots of referrals is to invest in startups. No

matter how smart and nice you seem, insiders will be reluctant to

send you referrals until you've proven yourself by doing a couple

investments. Some smart, nice guys turn out to be flaky,

high-maintenance investors. But once you prove yourself as a good

investor, the deal flow, as they call it, will increase rapidly in

both quality and quantity. At the extreme, for someone like Ron

Conway, it is basically identical with the deal flow of the whole

Valley.So if you want to invest seriously, the way to get started is to

bootstrap yourself off your existing connections, be a good investor

in the startups you meet that way, and eventually you'll start a

chain reaction. Good investors are rare, even in Silicon Valley.

There probably aren't more than a couple hundred serious angels in the whole

Valley, and yet they're probably the single most important ingredient

in making the Valley what it is. Angels are the limiting reagent

in startup formation.If there are only a couple hundred serious angels in the Valley,

then by deciding to become one you could single-handedly make the pipeline

for startups in Silicon Valley significantly wider. That is kind

of mind-blowing.Being GoodHow do you be a good angel investor? The first thing you need is

to be decisive. When we talk to founders about good and bad

investors, one of the ways we describe the good ones is to say "he

writes checks." That doesn't mean the investor says yes to everyone.

Far from it. It means he makes up his mind quickly,

and follows through. You may be thinking, how hard could that be?

You'll see when you try it. It follows from the nature of angel

investing that the decisions are hard. You have to guess early,

at the stage when the most promising ideas still seem counterintuitive,

because if they were obviously good, VCs would already have funded

them.Suppose it's 1998. You come across a startup founded by a couple

grad students. They say they're going to work on Internet search.

There are already a bunch of big public companies doing search.

How can these grad students possibly compete with them? And does

search even matter anyway? All the search engines are trying to

get people to start calling them "portals" instead. Why would you

want to invest in a startup run by a couple of nobodies who are

trying to compete with large, aggressive companies in an area they

themselves have declared passe? And yet the grad students seem

pretty smart. What do you do?There's a hack for being decisive when you're inexperienced: ratchet

down the size of your investment till it's an amount you wouldn't

care too much about losing. For every rich person (you probably

shouldn't try angel investing unless you think of yourself as rich)

there's some amount that would be painless, though annoying, to

lose. Till you feel comfortable investing, don't invest more than

that per startup.For example, if you have $5 million in investable assets, it would

probably be painless (though annoying) to lose $15,000. That's

less than .3% of your net worth. So start by making 3 or 4 $15,000

investments. Nothing will teach you about angel investing like

experience. Treat the first few as an educational expense. $60,000

is less than a lot of graduate programs. Plus you get equity.What's really uncool is to be strategically indecisive: to string

founders along while trying to gather more information about the

startup's trajectory.

[10]

There's always a temptation to do that,

because you just have so little to go on, but you have to consciously

resist it. In the long term it's to your advantage to be good.The other component of being a good angel investor is simply to be

a good person. Angel investing is not a business where you make

money by screwing people over. Startups create wealth, and

creating wealth is not a zero sum game. No one has to lose for you

to win. In fact, if you mistreat the founders you invest in, they'll

just get demoralized and the company will do worse. Plus your

referrals will dry up. So I recommend being good.The most successful angel investors I know are all basically good

people. Once they invest in a company, all they want to do is help

it. And they'll help people they haven't invested in too. When

they do favors they don't seem to keep track of them. It's too

much overhead. They just try to help everyone, and assume good

things will flow back to them somehow. Empirically that seems to

work.

Notes[1]

Convertible debt can be either capped at a particular valuation,

or can be done at a discount to whatever the valuation turns out

to be when it converts. E.g. convertible debt at a discount of 30%

means when it converts you get stock as if you'd invested at a 30%

lower valuation. That can be useful in cases where you can't or

don't want to figure out what the valuation should be. You leave

it to the next investor. On the other hand, a lot of investors

want to know exactly what they're getting, so they will only do

convertible debt with a cap.[2]

The expensive part of creating an agreement from scratch is

not writing the agreement, but bickering at several hundred

dollars an hour over the details. That's why the series AA paperwork

aims at a middle ground. You can just start from the compromise

you'd have reached after lots of back and forth.When you fund a startup, both your lawyers should be specialists

in startups. Do not use ordinary corporate lawyers for this. Their

inexperience makes them overbuild: they'll create huge, overcomplicated

agreements, and spend hours arguing over irrelevant things.In the Valley, the top startup law firms are Wilson Sonsini, Orrick,

Fenwick & West, Gunderson Dettmer, and Cooley Godward. In Boston

the best are Goodwin Procter, Wilmer Hale, and Foley Hoag.[3]

Your mileage may vary.[4]

These anti-dilution provisions also protect you against

tricks like a later investor trying to steal the company by doing

another round that values the company at $1. If you have a competent

startup lawyer handle the deal for you, you should be protected

against such tricks initially. But it could become a problem later.

If a big VC firm wants to invest in the startup after you, they may

try to make you take out your anti-dilution protections. And if

they do the startup will be pressuring you to agree. They'll tell

you that if you don't, you're going to kill their deal with the VC.

I recommend you solve this problem by having a gentlemen's agreement

with the founders: agree with them in advance that you're not going

to give up your anti-dilution protections. Then it's up to them

to tell VCs early on.The reason you don't want to give them up is the following scenario.

The VCs recapitalize the company, meaning they give it additional

funding at a pre-money valuation of zero. This wipes out the

existing shareholders, including both you and the founders. They

then grant the founders lots of options, because they need them to

stay around, but you get nothing.Obviously this is not a nice thing to do. It doesn't happen often.

Brand-name VCs wouldn't recapitalize a company just to steal a few

percent from an angel. But there's a continuum here. A less

upstanding, lower-tier VC might be tempted to do it to steal a big

chunk of stock.I'm not saying you should always absolutely refuse to give up your

anti-dilution protections. Everything is a negotiation. If you're

part of a powerful syndicate, you might be able to give up legal

protections and rely on social ones. If you invest in a deal led

by a big angel like Ron Conway, for example, you're pretty well

protected against being mistreated, because any VC would think twice

before crossing him. This kind of protection is one of the reasons

angels like to invest in syndicates.[5]

Don't invest so much, or at such a low valuation, that you

end up with an excessively large share of a startup, unless you're

sure your money will be the last they ever need. Later stage

investors won't invest in a company if the founders don't have

enough equity left to motivate them. I talked to a VC recently who

said he'd met with a company he really liked, but he turned

them down because investors already owned more than half of it.

Those investors probably thought they'd been pretty clever by getting

such a large chunk of this desirable company, but in fact they were

shooting themselves in the foot.[6]

At any given time I know of at least 3 or 4 YC alumni who I

believe will be big successes but who are running on vapor,

financially, because investors don't yet get what they're doing.

(And no, unfortunately, I can't tell you who they are. I can't

refer a startup to an investor I don't know.)[7]

There are some VCs who can predict instead of reacting. Not

surprisingly, these are the most successful ones.[8]

It's somewhat sneaky of me to put it this way, because the

median VC loses money. That's one of the most surprising things

I've learned about VC while working on Y Combinator. Only a fraction

of VCs even have positive returns. The rest exist to satisfy demand

among fund managers for venture capital as an asset class. Learning

this explained a lot about some of the VCs I encountered when we

were working on Viaweb.[9]

VCs also generally say they prefer great markets to great

people. But what they're really saying is they want both. They're

so selective that they only even consider great people. So when

they say they care above all about big markets, they mean that's

how they choose between great people.[10]

Founders rightly dislike the sort of investor who says he's

interested in investing but doesn't want to lead. There are

circumstances where this is an acceptable excuse, but more often

than not what it means is "No, but if you turn out to be a hot deal,

I want to be able to claim retroactively I said yes."If you like a startup enough to invest in it, then invest in it.

Just use the standard series

AA terms and write them a check.

Thanks to Sam Altman, Paul Buchheit, Jessica Livingston,

Robert Morris, and Fred Wilson for reading drafts of this.

Comment on this essay.

Why TV Lost

March 2009About twenty years ago people noticed computers and TV were on a

collision course and started to speculate about what they'd produce

when they converged. We now know the answer: computers. It's clear

now that even by using the word "convergence" we were giving TV too

much credit. This won't be convergence so much as replacement.

People may still watch things they call "TV shows," but they'll

watch them mostly on computers.What decided the contest for computers? Four forces, three of which

one could have predicted, and one that would have been harder to.One predictable cause of victory is that the Internet is an open

platform. Anyone can build whatever they want on it, and the market

picks the winners. So innovation happens at hacker speeds instead

of big company speeds.The second is Moore's Law, which has worked its usual magic on

Internet bandwidth.

[1]The third reason computers won is piracy. Users prefer it

not just because it's free, but because it's

more convenient. Bittorrent and YouTube have already trained a new

generation of viewers that the place to watch shows is on a computer

screen.

[2]The somewhat more surprising force was one specific type of innovation:

social applications. The average teenage kid has a pretty much

infinite capacity for talking to their friends. But they can't

physically be with them all the time. When I was in high school

the solution was the telephone. Now it's social networks, multiplayer

games, and various messaging applications. The way you reach them

all is through a computer.

[3]

Which means every teenage kid (a)

wants a computer with an Internet connection, (b) has an incentive

to figure out how to use it, and (c) spends countless hours in front

of it.This was the most powerful force of all. This was what made everyone

want computers. Nerds got computers because they liked them. Then

gamers got them to play games on. But it was connecting to other

people that got everyone else: that's what made even grandmas and

14 year old girls want computers.

After decades of running an IV drip right into their audience,

people in the entertainment business had understandably come to

think of them as rather passive. They thought they'd be able to

dictate the way shows reached audiences. But they underestimated

the force of their desire to connect with one another.Facebook killed TV. That is wildly oversimplified, of course, but

probably as close to the truth as you can get in three words.\_\_\_The TV networks already seem, grudgingly, to see where things are

going, and have responded by putting their stuff, grudgingly, online.

But they're still dragging their heels. They still seem to wish

people would watch shows on TV instead, just as newspapers that put

their stories online still seem to wish people would wait till the

next morning and read them printed on paper. They should both just

face the fact that the Internet is the primary medium.They'd be in a better position if they'd done that earlier. When

a new medium arises that's powerful enough to make incumbents

nervous, then it's probably powerful enough to win, and the best

thing they can do is jump in immediately.Whether they like it or not, big changes are coming, because the

Internet dissolves the two cornerstones of broadcast media:

synchronicity and locality. On the Internet, you don't have to

send everyone the same signal, and you don't have to send it to

them from a local source. People will watch what they want when

they want it, and group themselves according to whatever shared

interest they feel most strongly. Maybe their strongest shared

interest will be their physical location, but I'm guessing not.

Which means local TV is probably dead. It was an artifact of

limitations imposed by old technology. If someone were creating

an Internet-based TV company from scratch now, they might have some

plan for shows aimed at specific regions, but it wouldn't be a top

priority.Synchronicity and locality are tied together. TV network affiliates

care what's on at 10 because that delivers viewers for local news

at 11. This connection adds more brittleness than strength, however:

people don't watch what's on at 10 because they want to watch the

news afterward.TV networks will fight these trends, because they don't have

sufficient flexibility to adapt to them. They're hemmed in by local

affiliates in much the same way car companies are hemmed in by

dealers and unions. Inevitably, the people running the networks

will take the easy route and try to keep the old model running for

a couple more years, just as the record labels have done.A recent article in the Wall Street Journal described how TV networks

were trying to add more live shows, partly as a way to make viewers

watch TV synchronously instead of watching recorded shows when it

suited them. Instead of delivering what viewers want, they're

trying to force them to change their habits to suit the networks'

obsolete business model. That never works unless you have a monopoly

or cartel to enforce it, and even then it only works temporarily.The other reason networks like live shows is that they're cheaper

to produce. There they have the right idea, but they haven't

followed it to its conclusion. Live content can be way cheaper

than networks realize, and the way to take advantage of dramatic

decreases in cost is to

increase volume. The networks are prevented

from seeing this whole line of reasoning because they still think

of themselves as being in the broadcast business—as sending one

signal to everyone.

[4]\_\_\_Now would be a good time to start any company that competes with

TV networks. That's what a lot of Internet startups are, though

they may not have had this as an explicit goal. People only have

so many leisure hours a day, and TV is premised on such long sessions

(unlike Google, which prides itself on sending users on their way

quickly) that anything that takes up their time is competing with

it. But in addition to such indirect competitors, I think TV

companies will increasingly face direct ones.Even in cable TV, the long tail was lopped off prematurely by the

threshold you had to get over to start a new channel. It will be

longer on the Internet, and there will be more mobility within it.

In this new world, the existing players will only have the advantages

any big company has in its market.That will change the balance of power between the networks and the

people who produce shows. The networks used to be gatekeepers.

They distributed your work, and sold advertising on it. Now the

people who produce a show can distribute it themselves. The main

value networks supply now is ad sales. Which will tend to put them

in the position of service providers rather than publishers.Shows will change even more. On the Internet there's no reason to

keep their current format, or even the fact that they have a single

format. Indeed, the more interesting sort of convergence that's

coming is between shows and games. But on the question of what

sort of entertainment gets distributed on the Internet in 20 years,

I wouldn't dare to make any predictions, except that things will

change a lot. We'll get whatever the most imaginative people can

cook up. That's why the Internet won.

Notes[1]

Thanks to Trevor Blackwell for this point. He adds: "I

remember the eyes of phone companies gleaming in the early 90s when

they talked about convergence. They thought most programming would

be on demand, and they would implement it and make a lot of money.

It didn't work out. They assumed that their local network infrastructure

would be critical to do video on-demand, because you couldn't

possibly stream it from a few data centers over the internet. At

the time (1992) the entire cross-country Internet bandwidth wasn't

enough for one video stream. But wide-area bandwidth increased more

than they expected and they were beaten by iTunes and Hulu."[2]

Copyright owners tend to focus on the aspect they see of

piracy, which is the lost revenue. They therefore think what drives

users to do it is the desire to get something for free. But iTunes

shows that people will pay for stuff online, if you make it easy.

A significant component of piracy is simply that it offers a better

user experience.[3]

Or a phone that is actually a computer. I'm not making any

predictions about the size of the device that will replace TV, just

that it will have a browser and get data via the Internet.[4]

Emmett Shear writes: "I'd argue the long tail for sports may

be even larger than the long tail for other kinds of content. Anyone

can broadcast a high school football game that will be interesting

to 10,000 people or so, even if the quality of production is not

so good."

Thanks to Sam Altman, Trevor Blackwell, Nancy Cook, Michael Seibel.

Emmett Shear, and Fred Wilson for reading drafts of this.Japanese Translation

Can You Buy a Silicon Valley? Maybe.

February 2009A lot of cities look at Silicon Valley and ask "How could we make

something like that happen here?" The

organic way to do it is to

establish a first-rate university in a place where rich people want

to live. That's how Silicon Valley happened. But could you shortcut

the process by funding startups?Possibly. Let's consider what it would take.The first thing to understand is that encouraging startups is a

different problem from encouraging startups in a particular city.

The latter is much more expensive.People sometimes think they could improve the startup scene in their

town by starting something like Y

Combinator there, but in fact it

will have near zero effect. I know because Y Combinator itself had

near zero effect on Boston when we were based there half the year.

The people we funded came from all over the country (indeed, the

world) and afterward they went wherever they could get more

funding—which generally meant Silicon Valley.The seed funding business is not a regional business, because at

that stage startups are mobile. They're just a couple founders with

laptops.

[1]If you want to encourage startups in a particular city, you have

to fund startups that won't leave. There are two ways to do that:

have rules preventing them from leaving, or fund them at the point

in their life when they naturally take root. The first approach

is a mistake, because it becomes a filter for selecting bad startups.

If your terms force startups to do things they don't want to, only

the desperate ones will take your money.Good startups will move to another city as a condition of funding.

What they won't do is agree not to move the next time they need

funding. So the only way to get them to stay is to give them enough

that they never need to leave.\_\_\_How much would that take? If you want to keep startups from leaving

your town, you have to give them enough that they're not tempted

by an offer from Silicon Valley VCs that requires them to move. A

startup would be able to refuse such an offer if they had grown to

the point where they were (a) rooted in your town and/or (b) so

successful that VCs would fund them even if they didn't move.How much would it cost to grow a startup to that point? A minimum

of several hundred thousand dollars. Wufoo

seem to have rooted

themselves in Tampa on $118k, but they're an extreme case. On

average it would take at least half a million.So if it seems too good to be true to think you could grow a local

silicon valley by giving startups $15-20k each like Y Combinator,

that's because it is. To make them stick around you'd have to give

them at least 20 times that much.However, even that is an interesting prospect. Suppose to be on

the safe side it would cost a million dollars per startup. If you

could get startups to stick to your town for a million apiece, then

for a billion dollars you could bring in a thousand startups. That probably wouldn't push you past Silicon Valley itself,

but it might get you second place.For the price of a football stadium, any town that was decent to

live in could make itself one of the biggest startup hubs in the

world.What's more, it wouldn't take very long. You could probably do

it in five years. During the term of one mayor. And it would get

easier over time, because the more startups you had in town, the

less it would take to get new ones to move there. By the time you

had a thousand startups in town, the VCs wouldn't be trying so hard

to get them to move to Silicon Valley; instead they'd be opening

local offices. Then you'd really be in good shape. You'd have

started a self-sustaining chain reaction like the one that drives

the Valley.\_\_\_But now comes the hard part. You have to pick the startups. How

do you do that? Picking startups is a rare and valuable skill, and

the handful of people who have it are not readily hireable. And

this skill is so hard to measure that if a government did try to

hire people with it, they'd almost certainly get the wrong ones.For example, a city could give money to a VC fund to establish a

local branch, and let them make the choices. But only a bad VC

fund would take that deal. They wouldn't seem bad to the city

officials. They'd seem very impressive. But they'd be bad at

picking startups. That's the characteristic failure mode of VCs.

All VCs look impressive to limited partners. The difference between

the good ones and the bad ones only becomes visible in the other

half of their jobs: choosing and advising startups.

[2]What you really want is a pool of local angel investors—people

investing money they made from their own startups. But unfortunately

you run into a chicken and egg problem here. If your city isn't

already a startup hub, there won't be people there who got rich

from startups. And there is no way I can think of that a city could

attract angels from outside. By definition they're rich. There's

no incentive that would make them move.

[3]However, a city could select startups by piggybacking on the expertise

of investors who weren't local. It would be pretty straightforward

to make a list of the most eminent Silicon Valley angels and from

that to generate a list of all the startups they'd invested in. If

a city offered these companies a million dollars each to move, a

lot of the earlier stage ones would probably take it.Preposterous as this plan sounds, it's probably the most efficient

way a city could select good startups.It would hurt the startups somewhat to be separated from their

original investors. On the other hand, the extra million dollars

would give them a lot more runway.\_\_\_Would the transplanted startups survive? Quite possibly. The only

way to find out would be to try it. It would be a pretty cheap

experiment, as civil expenditures go. Pick 30 startups that eminent

angels have recently invested in, give them each a million dollars

if they'll relocate to your city, and see what happens after a year.

If they seem to be thriving, you can try importing startups on a

larger scale.Don't be too legalistic about the conditions under which they're

allowed to leave. Just have a gentlemen's agreement.Don't try to do it on the cheap and pick only 10 for the initial

experiment. If you do this on too small a scale you'll just guarantee

failure. Startups need to be around other startups. 30 would be

enough to feel like a community.Don't try to make them all work in some renovated warehouse you've

made into an "incubator." Real startups prefer to work in their

own spaces.In fact, don't impose any restrictions on the startups at all.

Startup founders are mostly hackers,

and hackers are much more

constrained by gentlemen's agreements than regulations. If they

shake your hand on a promise, they'll keep it. But show them a

lock and their first thought is how to pick it.Interestingly, the 30-startup experiment could be done by any

sufficiently rich private citizen. And what pressure it would

put on the city if it worked.

[4]\_\_\_Should the city take stock in return for the money?

In principle they're entitled to, but how would they choose valuations

for the startups? You couldn't just give them all the same valuation:

that would be too low for some (who'd turn you down) and too high

for others (because it might make their next round a "down round").

And since we're assuming we're doing this without being able to

pick startups, we also have to assume we can't value them, since

that's practically the same thing.Another reason not to take stock in the startups is that startups

are often involved in disreputable things. So are established

companies, but they don't get blamed for it. If someone gets

murdered by someone they met on Facebook, the press will treat the

story as if it were about Facebook. If someone gets murdered by

someone they met at a supermarket, the press will just treat it as

a story about a murder. So understand that if you invest in startups,

they might build things that get used for pornography, or file-sharing,

or the expression of unfashionable opinions. You should probably

sponsor this project jointly with your political opponents, so they

can't use whatever the startups do as a club to beat you with.It would be too much of a political liability just to give

the startups the money, though. So the best plan would be to

make it convertible debt, but which didn't convert except in

a really big round, like $20 million.\_\_\_How well this scheme worked would depend on the

city. There are

some towns, like Portland, that would be easy to turn into startup

hubs, and others, like Detroit, where it would really be an uphill

battle. So be honest with yourself about the sort of town you have

before you try this.It will be easier in proportion to how much your town resembles San

Francisco. Do you have good weather? Do people live downtown, or

have they abandoned the center for the suburbs? Would the city be

described as "hip" and "tolerant," or as reflecting "traditional

values?" Are there good universities nearby? Are there walkable

neighborhoods? Would nerds feel at home? If you answered yes to

all these questions, you might be able not only to pull off this

scheme, but to do it for less than a million per startup.I realize the chance of any city having

the political will to carry out this plan is microscopically

small. I just wanted to explore what it would take if one did.

How hard would it be to jumpstart a silicon valley? It's

fascinating to think this prize might be within

the reach of so many cities. So even though they'll all still

spend the money on the stadium, at least now someone can ask them:

why did you choose to do that instead of becoming a serious

rival to Silicon Valley?

Notes[1]

What people who start these supposedly local seed firms always

find is that (a) their applicants come from all over, not just the

local area, and (b) the local startups also apply to the other seed

firms. So what ends up happening is that the applicant pool gets

partitioned by quality rather than geography.[2]

Interestingly, the bad VCs fail by choosing startups run by

people like them—people who are good presenters, but have no

real substance. It's a case of the fake leading the fake. And

since everyone involved is so plausible, the LPs who invest in these

funds have no idea what's happening till they measure their returns.[3]

Not even being a tax haven, I suspect. That makes some rich

people move, but not the type who would make good angel investors

in startups.[4]

Thanks to Michael Keenan for pointing this out.Thanks to Trevor Blackwell, Jessica Livingston, Robert

Morris, and Fred Wilson for reading drafts of this.

What I've Learned from Hacker News

February 2009Hacker News was two years

old last week. Initially it was supposed to be a side project—an

application to sharpen Arc on, and a place for current and future

Y Combinator founders to exchange news. It's grown bigger and taken

up more time than I expected, but I don't regret that because I've

learned so much from working on it.GrowthWhen we launched in February 2007, weekday traffic was around 1600

daily uniques. It's since grown to around 22,000. This growth

rate is a bit higher than I'd like. I'd like the site to grow,

since a site that isn't growing at least slowly is probably dead.

But I wouldn't want it to grow as large as Digg or Reddit—mainly

because that would dilute the character of the site, but also because

I don't want to spend all my time dealing with scaling.I already have problems enough with that. Remember, the original

motivation for HN was to test a new programming language, and

moreover one that's focused on experimenting with language design,

not performance. Every time the site gets slow, I fortify myself

by recalling McIlroy and Bentley's famous quote

The key to performance is elegance, not battalions of special

cases.

and look for the bottleneck I can remove with least code. So far

I've been able to keep up, in the sense that performance has remained

consistently mediocre despite 14x growth. I don't know what I'll

do next, but I'll probably think of something.This is my attitude to the site generally. Hacker News is an

experiment, and an experiment in a very young field. Sites of this

type are only a few years old. Internet conversation generally is

only a few decades old. So we've probably only discovered a fraction

of what we eventually will.That's why I'm so optimistic about HN. When a technology is this

young, the existing solutions are usually terrible; which means it

must be possible to do much better; which means many problems that

seem insoluble aren't. Including, I hope, the problem that has

afflicted so many previous communities: being ruined by growth.DilutionUsers have worried about that since the site was a few months old.

So far these alarms have been false, but they may not always be.

Dilution is a hard problem. But probably soluble; it doesn't mean

much that open conversations have "always" been destroyed by growth

when "always" equals 20 instances.But it's important to remember we're trying to solve a new problem,

because that means we're going to have to try new things, most of

which probably won't work. A couple weeks ago I tried displaying

the names of users with the highest average comment scores in orange.

[1]

That was a mistake. Suddenly a culture that had been more

or less united was divided into haves and have-nots. I didn't

realize how united the culture had been till I saw it divided. It

was painful to watch.

[2]So orange usernames won't be back. (Sorry about that.) But there

will be other equally broken-seeming ideas in the future, and the

ones that turn out to work will probably seem just as broken as

those that don't.Probably the most important thing I've learned about dilution is

that it's measured more in behavior than users. It's bad behavior

you want to keep out more than bad people. User behavior turns out

to be surprisingly malleable. If people are

expected to behave

well, they tend to; and vice versa.Though of course forbidding bad behavior does tend to keep away bad

people, because they feel uncomfortably constrained in a place where

they have to behave well. But this way of keeping them out is

gentler and probably also more effective than overt barriers.It's pretty clear now that the broken windows theory applies to

community sites as well. The theory is that minor forms of bad

behavior encourage worse ones: that a neighborhood with lots of

graffiti and broken windows becomes one where robberies occur. I

was living in New York when Giuliani introduced the reforms that

made the broken windows theory famous, and the transformation was

miraculous. And I was a Reddit user when the opposite happened

there, and the transformation was equally dramatic.I'm not criticizing Steve and Alexis. What happened to Reddit

didn't happen out of neglect. From the start they had a policy of

censoring nothing except spam. Plus Reddit had different goals

from Hacker News. Reddit was a startup, not a side project; its

goal was to grow as fast as possible. Combine rapid growth and

zero censorship, and the result is a free for all. But I don't

think they'd do much differently if they were doing it again.

Measured by traffic, Reddit is much more successful than Hacker

News.But what happened to Reddit won't inevitably happen to HN. There

are several local maxima. There can be places that are free for

alls and places that are more thoughtful, just as there are in the

real world; and people will behave differently depending on which

they're in, just as they do in the real world.I've observed this in the wild. I've seen people cross-posting on

Reddit and Hacker News who actually took the trouble to write two

versions, a flame for Reddit and a more subdued version for HN.SubmissionsThere are two major types of problems a site like Hacker News needs

to avoid: bad stories and bad comments. So far the danger of bad

stories seems smaller. The stories on the frontpage now are still

roughly the ones that would have been there when HN started.I once thought I'd have to weight votes to keep crap off the

frontpage, but I haven't had to yet. I wouldn't have predicted the

frontpage would hold up so well, and I'm not sure why it has.

Perhaps only the more thoughtful users care enough to submit and

upvote links, so the marginal cost of one random new user approaches

zero. Or perhaps the frontpage protects itself, by advertising what type of submission is expected.The most dangerous thing for the frontpage is stuff that's too easy

to upvote. If someone proves a new theorem, it takes some work by

the reader to decide whether or not to upvote it. An amusing cartoon

takes less. A rant with a rallying cry as the title takes zero,

because people vote it up without even reading it.Hence what I call the Fluff Principle: on a user-voted news site,

the links that are easiest to judge will take over unless you take

specific measures to prevent it.Hacker News has two kinds of protections against fluff. The most

common types of fluff links are banned as off-topic. Pictures of

kittens, political diatribes, and so on are explicitly banned. This

keeps out most fluff, but not all of it. Some links are both fluff,

in the sense of being very short, and also on topic.There's no single solution to that. If a link is just an empty

rant, editors will sometimes kill it even if it's on topic in the

sense of being about hacking, because it's not on topic by the real

standard, which is to engage one's intellectual curiosity. If the

posts on a site are characteristically of this type I sometimes ban

it, which means new stuff at that url is auto-killed. If a post

has a linkbait title, editors sometimes rephrase it to be more

matter-of-fact. This is especially necessary with links whose

titles are rallying cries, because otherwise they become implicit

"vote up if you believe such-and-such" posts, which are the most

extreme form of fluff.The techniques for dealing with links have to evolve, because the

links do. The existence of aggregators has already affected what

they aggregate. Writers now deliberately write things to draw traffic

from aggregators—sometimes even specific ones. (No, the irony

of this statement is not lost on me.) Then there are the more

sinister mutations, like linkjacking—posting a paraphrase of

someone else's article and submitting that instead of the original.

These can get a lot of upvotes, because a lot of what's good in an

article often survives; indeed, the closer the paraphrase is to

plagiarism, the more survives.

[3]I think it's important that a site that kills submissions provide

a way for users to see what got killed if they want to. That keeps

editors honest, and just as importantly, makes users confident

they'd know if the editors stopped being honest. HN users can do

this by flipping a switch called showdead in their profile.

[4]CommentsBad comments seem to be a harder problem than bad submissions.

While the quality of links on the frontpage of HN hasn't changed

much, the quality of the median comment may have decreased somewhat.There are two main kinds of badness in comments: meanness and

stupidity. There is a lot of overlap between the two—mean

comments are disproportionately likely also to be dumb—but

the strategies for dealing with them are different. Meanness is

easier to control. You can have rules saying one shouldn't be mean,

and if you enforce them it seems possible to keep a lid on meanness.Keeping a lid on stupidity is harder, perhaps because stupidity is

not so easily distinguishable. Mean people are more likely to know

they're being mean than stupid people are to know they're being

stupid.The most dangerous form of stupid comment is not the long but

mistaken argument, but the dumb joke. Long but mistaken arguments

are actually quite rare. There is a strong correlation between

comment quality and length; if you wanted to compare the quality

of comments on community sites, average length would be a good

predictor. Probably the cause is human nature rather than anything

specific to comment threads. Probably it's simply that stupidity

more often takes the form of having few ideas than wrong ones.Whatever the cause, stupid comments tend to be short. And since

it's hard to write a short comment that's distinguished for the

amount of information it conveys, people try to distinguish them

instead by being funny. The most tempting format for stupid comments

is the supposedly witty put-down, probably because put-downs are

the easiest form of humor.

[5]

So one advantage of forbidding

meanness is that it also cuts down on these.Bad comments are like kudzu: they take over rapidly. Comments have

much more effect on new comments than submissions have on new

submissions. If someone submits a lame article, the other submissions

don't all become lame. But if someone posts a stupid comment on a

thread, that sets the tone for the region around it. People reply

to dumb jokes with dumb jokes.Maybe the solution is to add a delay before people can respond to

a comment, and make the length of the delay inversely proportional

to some prediction of its quality. Then dumb threads would grow

slower.

[6]

PeopleI notice most of the techniques I've described are conservative:

they're aimed at preserving the character of the site rather than

enhancing it. I don't think that's a bias of mine. It's due to

the shape of the problem. Hacker News had the good fortune to start

out good, so in this case it's literally a matter of preservation.

But I think this principle would also apply to sites with different

origins.The good things in a community site come from people more than

technology; it's mainly in the prevention of bad things that

technology comes into play. Technology certainly can enhance

discussion. Nested comments do, for example. But I'd rather use

a site with primitive features and smart, nice users than a more

advanced one whose users were idiots or trolls.So the most important thing a community site can do is attract the

kind of people it wants. A site trying to be as big as possible

wants to attract everyone. But a site aiming at a particular subset

of users has to attract just those—and just as importantly,

repel everyone else. I've made a conscious effort to do this on

HN. The graphic design is as plain as possible, and the site rules

discourage dramatic link titles. The goal is that the only thing

to interest someone arriving at HN for the first time should be the

ideas expressed there.The downside of tuning a site to attract certain people is that,

to those people, it can be too attractive. I'm all too aware how

addictive Hacker News can be. For me, as for many users, it's a

kind of virtual town square. When I want to take a break from

working, I walk into the square, just as I might into Harvard Square

or University Ave in the physical world.

[7]

But an online square is

more dangerous than a physical one. If I spent half the day loitering

on University Ave, I'd notice. I have to walk a mile to get there,

and sitting in a cafe feels different from working. But visiting

an online forum takes just a click, and feels superficially very

much like working. You may be wasting your time, but you're not

idle. Someone is wrong on the Internet, and you're fixing the

problem.Hacker News is definitely useful. I've learned a lot from things

I've read on HN. I've written several essays that began as comments

there. So I wouldn't want the site to go away. But I would like

to be sure it's not a net drag on productivity. What a disaster

that would be, to attract thousands of smart people to a site that

caused them to waste lots of time. I wish I could be 100% sure

that's not a description of HN.I feel like the addictiveness of games and social applications is

still a mostly unsolved problem. The situation now is like it was

with crack in the 1980s: we've invented terribly addictive new

things, and we haven't yet evolved ways to protect ourselves from

them. We will eventually, and that's one of the problems I hope

to focus on next.

Notes[1]

I tried ranking users by both average and median comment

score, and average (with the high score thrown out) seemed the more

accurate predictor of high quality. Median may be the more accurate

predictor of low quality though.[2]

Another thing I learned from this experiment is that if you're

going to distinguish between people, you better be sure you do it

right. This is one problem where rapid prototyping doesn't work.Indeed, that's the intellectually honest argument for not discriminating

between various types of people. The reason not to do it is not

that everyone's the same, but that it's bad to do wrong and hard

to do right.[3]

When I catch egregiously linkjacked posts I replace the url

with that of whatever they copied. Sites that habitually linkjack

get banned.[4]

Digg is notorious for its lack of transparency. The root of

the problem is not that the guys running Digg are especially sneaky,

but that they use the wrong algorithm for generating their frontpage.

Instead of bubbling up from the bottom as they get more votes, as

on Reddit, stories start at the top and get pushed down by new

arrivals.The reason for the difference is that Digg is derived from Slashdot,

while Reddit is derived from Delicious/popular. Digg is Slashdot

with voting instead of editors, and Reddit is Delicious/popular

with voting instead of bookmarking. (You can still see fossils of

their origins in their graphic design.)Digg's algorithm is very vulnerable to gaming, because any story

that makes it onto the frontpage is the new top story. Which in

turn forces Digg to respond with extreme countermeasures. A lot

of startups have some kind of secret about the subterfuges they had

to resort to in the early days, and I suspect Digg's is the extent

to which the top stories were de facto chosen by human editors.[5]

The dialog on Beavis and Butthead was composed largely of

these, and when I read comments on really bad sites I can hear them

in their voices.[6]

I suspect most of the techniques for discouraging stupid

comments have yet to be discovered. Xkcd implemented a particularly

clever one in its IRC channel: don't allow the same thing twice.

Once someone has said "fail," no one can ever say it again. This

would penalize short comments especially, because they have less

room to avoid collisions in.Another promising idea is the stupid

filter, which is just like a

probabilistic spam filter, but trained on corpora of stupid and

non-stupid comments instead.You may not have to kill bad comments to solve the problem. Comments

at the bottom of a long thread are rarely seen, so it may be enough

to incorporate a prediction of quality in the comment sorting

algorithm.[7]

What makes most suburbs so demoralizing is that there's no

center to walk to.

Thanks to Justin Kan, Jessica Livingston, Robert Morris,

Alexis Ohanian, Emmet Shear, and Fred Wilson for reading drafts of

this.

Comment on this essay.

Startups in 13 Sentences

Want to start a startup? Get funded by

Y Combinator.

Watch how this essay was

written.

February 2009One of the things I always tell startups is a principle I learned

from Paul Buchheit: it's better to make a few people really happy

than to make a lot of people semi-happy. I was saying recently to

a reporter that if I could only tell startups 10 things, this would

be one of them. Then I thought: what would the other 9 be?When I made the list there turned out to be 13:

1. Pick good cofounders.Cofounders are for a startup what location is for real estate. You

can change anything about a house except where it is. In a startup

you can change your idea easily, but changing your cofounders is

hard.

[1]

And the success of a startup is almost always a function

of its founders.2. Launch fast.The reason to launch fast is not so much that it's critical to get

your product to market early, but that you haven't really started

working on it till you've launched. Launching teaches you what you

should have been building. Till you know that you're wasting your

time. So the main value of whatever you launch with is as a pretext

for engaging users.3. Let your idea evolve.This is the second half of launching fast. Launch fast and iterate.

It's a big mistake to treat a startup as if it were merely a matter

of implementing some brilliant initial idea. As in an essay, most

of the ideas appear in the implementing.4. Understand your users.You can envision the wealth created by a startup as a rectangle,

where one side is the number of users and the other is how much you

improve their lives.

[2]

The second dimension is the one you have

most control over. And indeed, the growth in the first will be

driven by how well you do in the second. As in science, the hard

part is not answering questions but asking them: the hard part is

seeing something new that users lack. The better you understand

them the better the odds of doing that. That's why so many successful

startups make something the founders needed.5. Better to make a few users love you than a lot ambivalent.Ideally you want to make large numbers of users love you, but you

can't expect to hit that right away. Initially you have to choose

between satisfying all the needs of a subset of potential users,

or satisfying a subset of the needs of all potential users. Take

the first. It's easier to expand userwise than satisfactionwise.

And perhaps more importantly, it's harder to lie to yourself. If

you think you're 85% of the way to a great product, how do you know

it's not 70%? Or 10%? Whereas it's easy to know how many users

you have.6. Offer surprisingly good customer service.Customers are used to being maltreated. Most of the companies they

deal with are quasi-monopolies that get away with atrocious customer

service. Your own ideas about what's possible have been unconsciously

lowered by such experiences. Try making your customer service not

merely good, but

surprisingly good. Go out of your way to make

people happy. They'll be overwhelmed; you'll see. In the earliest

stages of a startup, it pays to offer customer service on a level

that wouldn't scale, because it's a way of learning about your

users.7. You make what you measure.I learned this one from Joe Kraus.

[3]

Merely measuring something

has an uncanny tendency to improve it. If you want to make your

user numbers go up, put a big piece of paper on your wall and every

day plot the number of users. You'll be delighted when it goes up

and disappointed when it goes down. Pretty soon you'll start

noticing what makes the number go up, and you'll start to do more

of that. Corollary: be careful what you measure.8. Spend little.I can't emphasize enough how important it is for a startup to be cheap.

Most startups fail before they make something people want, and the

most common form of failure is running out of money. So being cheap

is (almost) interchangeable with iterating rapidly.

[4]

But it's

more than that. A culture of cheapness keeps companies young in

something like the way exercise keeps people young.9. Get ramen profitable."Ramen profitable" means a startup makes just enough to pay the

founders' living expenses. It's not rapid prototyping for business

models (though it can be), but more a way of hacking the investment

process. Once you cross over into ramen profitable, it completely

changes your relationship with investors. It's also great for

morale.10. Avoid distractions.Nothing kills startups like distractions. The worst type are those

that pay money: day jobs, consulting, profitable side-projects.

The startup may have more long-term potential, but you'll always

interrupt working on it to answer calls from people paying you now.

Paradoxically, fundraising is this type of distraction, so try to

minimize that too.11. Don't get demoralized.Though the immediate cause of death in a startup tends to be running

out of money, the underlying cause is usually lack of focus. Either

the company is run by stupid people (which can't be fixed with

advice) or the people are smart but got demoralized. Starting a

startup is a huge moral weight. Understand this and make a conscious

effort not to be ground down by it, just as you'd be careful to

bend at the knees when picking up a heavy box.12. Don't give up.Even if you get demoralized, don't give up. You can get surprisingly

far by just not giving up. This isn't true in all fields. There

are a lot of people who couldn't become good mathematicians no

matter how long they persisted. But startups aren't like that.

Sheer effort is usually enough, so long as you keep morphing your

idea.13. Deals fall through.One of the most useful skills we learned from Viaweb was not getting

our hopes up. We probably had 20 deals of various types fall

through. After the first 10 or so we learned to treat deals as

background processes that we should ignore till they terminated.

It's very dangerous to morale to start to depend on deals closing,

not just because they so often don't, but because it makes them

less likely to.

Having gotten it down to 13 sentences, I asked myself which I'd

choose if I could only keep one.Understand your users. That's the key. The essential task in a

startup is to create wealth; the dimension of wealth you have most

control over is how much you improve users' lives; and the hardest

part of that is knowing what to make for them. Once you know what

to make, it's mere effort to make it, and most decent hackers are

capable of that.Understanding your users is part of half the principles in this

list. That's the reason to launch early, to understand your users.

Evolving your idea is the embodiment of understanding your users.

Understanding your users well will tend to push you toward making

something that makes a few people deeply happy. The most important

reason for having surprisingly good customer service is that it

helps you understand your users. And understanding your users will

even ensure your morale, because when everything else is collapsing

around you, having just ten users who love you will keep you going.Notes[1]

Strictly speaking it's impossible without a time machine.[2]

In practice it's more like a ragged comb.[3]

Joe thinks one of the founders of Hewlett Packard said it first,

but he doesn't remember which.[4]

They'd be interchangeable if markets stood still. Since they

don't, working twice as fast is better than having twice as much

time.Turkish TranslationSpanish TranslationBulgarian TranslationJapanese TranslationPersian Translation

Keep Your Identity Small

February 2009I finally realized today why politics and religion yield such

uniquely useless discussions.As a rule, any mention of religion on an online forum degenerates

into a religious argument. Why? Why does this happen with religion

and not with Javascript or baking or other topics people talk about

on forums?What's different about religion is that people don't feel they need

to have any particular expertise to have opinions about

it. All they need is strongly held beliefs, and anyone can have

those. No thread about Javascript will grow as fast as one about

religion, because people feel they have to be over some threshold

of expertise to post comments about that. But on religion everyone's

an expert.Then it struck me: this is the problem with politics too. Politics,

like religion, is a topic where there's no threshold of expertise

for expressing an opinion. All you need is strong convictions.Do religion and politics have something in common that explains

this similarity? One possible explanation is that they deal with

questions that have no definite answers, so there's no back pressure

on people's opinions. Since no one can be proven wrong, every

opinion is equally valid, and sensing this, everyone lets fly with

theirs.But this isn't true. There are certainly some political questions

that have definite answers, like how much a new government policy

will cost. But the more precise political questions suffer the

same fate as the vaguer ones.I think what religion and politics have in common is that they

become part of people's identity, and people can never have a

fruitful argument about something that's part of their identity.

By definition they're partisan.Which topics engage people's identity depends on the people, not

the topic. For example, a discussion about a battle that included

citizens of one or more of the countries involved would probably

degenerate into a political argument. But a discussion today about

a battle that took place in the Bronze Age probably wouldn't. No

one would know what side to be on. So it's not politics that's the

source of the trouble, but identity. When people say a discussion

has degenerated into a religious war, what they really mean is that

it has started to be driven mostly by people's identities.

[1]Because the point at which this happens depends on the people rather

than the topic, it's a mistake to conclude that because a question

tends to provoke religious wars, it must have no answer. For example,

the question of the relative merits of programming languages often

degenerates into a religious war, because so many programmers

identify as X programmers or Y programmers. This sometimes leads

people to conclude the question must be unanswerable—that all

languages are equally good. Obviously that's false: anything else

people make can be well or badly designed; why should this be

uniquely impossible for programming languages? And indeed, you can

have a fruitful discussion about the relative merits of programming

languages, so long as you exclude people who respond from identity.More generally, you can have a fruitful discussion about a topic

only if it doesn't engage the identities of any of the

participants. What makes politics and religion such minefields is

that they engage so many people's identities. But you could in

principle have a useful conversation about them with some people.

And there are other topics that might seem harmless, like the

relative merits of Ford and Chevy pickup trucks, that you couldn't

safely talk about with others.The most intriguing thing about this theory, if it's right, is that

it explains not merely which kinds of discussions to avoid, but how

to have better ideas. If people can't think clearly about anything

that has become part of their identity, then all other things being

equal, the best plan is to let as few things into your identity as

possible.

[2]Most people reading this will already be fairly tolerant. But there

is a step beyond thinking of yourself as x but tolerating y: not

even to consider yourself an x. The more labels you have for

yourself, the dumber they make you.Notes[1]

When that happens, it tends to happen fast, like a core going

critical. The threshold for participating goes down to zero, which

brings in more people. And they tend to say incendiary things,

which draw more and angrier counterarguments.[2]

There may be some things it's a net win to include in your

identity. For example, being a scientist. But arguably that is

more of a placeholder than an actual label—like putting NMI on a

form that asks for your middle initial—because it doesn't commit

you to believing anything in particular. A scientist isn't committed

to believing in natural selection in the same way a bibilical

literalist is committed to rejecting it. All he's committed to is

following the evidence wherever it leads.Considering yourself a scientist is equivalent to putting a sign

in a cupboard saying "this cupboard must be kept empty." Yes,

strictly speaking, you're putting something in the cupboard, but

not in the ordinary sense.Thanks to Sam Altman, Trevor Blackwell, Paul Buchheit, and Robert

Morris for reading drafts of this.Russian TranslationPortuguese TranslationRomanian Translation

After Credentials

December 2008A few months ago I read a New York Times article on South

Korean cram schools that said

Admission to the right university can make or break an ambitious

young South Korean.

A parent added:

"In our country, college entrance exams determine 70 to 80 percent

of a person's future."

It was striking how old fashioned this sounded. And

yet when I was in high school it wouldn't have seemed too far off

as a description of the US. Which means things must have been

changing here.The course of people's lives in the US now seems to be determined

less by credentials and more by performance than it was 25 years

ago. Where you go to college still matters, but not like it used

to.What happened?\_\_\_\_\_Judging people by their academic credentials was in its time an

advance. The practice seems to have begun in China, where starting

in 587 candidates for the imperial civil service had to take an

exam on classical literature. [1] It was also a test of wealth,

because the knowledge it tested was so specialized that passing

required years of expensive training. But though wealth was a

necessary condition for passing, it was not a sufficient one. By

the standards of the rest of the world in 587, the Chinese system

was very enlightened. Europeans didn't introduce formal civil

service exams till the nineteenth century, and even then they seem

to have been influenced by the Chinese example.Before credentials, government positions were obtained mainly by

family influence, if not outright bribery. It was a great step

forward to judge people by their performance on a test. But by no

means a perfect solution. When you judge people that way, you tend

to get cram schools—which they did in Ming China and nineteenth

century England just as much as in present day South Korea.What cram schools are, in effect, is leaks in a seal. The use of

credentials

was an attempt to seal off the direct transmission of power between

generations, and cram schools represent that power finding holes

in the seal. Cram schools turn wealth in one generation into

credentials in the next.It's hard to beat this phenomenon, because the schools adjust to suit

whatever the tests measure. When the tests are narrow and

predictable, you get cram schools on the classic model, like those

that prepared candidates for Sandhurst (the British West Point) or

the classes American students take now to improve their SAT scores.

But as the tests get broader, the schools do too. Preparing a

candidate for the Chinese imperial civil service exams took years,

as prep school does today. But the raison d'etre of all these

institutions has been the same: to beat the system. [2]\_\_\_\_\_History suggests that, all other things being equal, a society

prospers in proportion to its ability to prevent parents from

influencing their children's success directly. It's a fine thing

for parents to help their children indirectly—for example,

by helping them to become smarter or more disciplined, which then

makes them more successful. The problem comes when parents use

direct methods: when they are able to use their own wealth or power

as a substitute for their children's qualities.Parents will tend to do this when they can. Parents will die for

their kids, so it's not surprising to find they'll also push their

scruples to the limits for them. Especially if other parents are

doing it.Sealing off this force has a double advantage. Not only does a

society get "the best man for the job," but

parents' ambitions are diverted from direct methods to indirect

ones—to actually trying to raise their kids well.But we should expect it to be very hard to contain parents' efforts

to obtain an unfair advantage for their kids. We're dealing with

one of the most powerful forces in human nature. We shouldn't expect

naive solutions to work, any more than we'd expect naive solutions

for keeping heroin out of a prison to work.\_\_\_\_\_The obvious way to solve the problem is to make credentials better.

If the tests a society uses are currently hackable, we can study

the way people beat them and try to plug the holes. You can use

the cram schools to show you where most of the holes are. They

also tell you when you're succeeding in fixing them: when cram

schools become less popular.A more general solution

would be to push for increased transparency, especially at critical

social bottlenecks like college admissions. In the US this process

still shows many outward signs of corruption. For example, legacy

admissions. The official story is that legacy status doesn't carry

much weight, because all it does is break ties: applicants are

bucketed by ability, and legacy status is only used to decide between

the applicants in the bucket that straddles the cutoff. But what

this means is that a university can make legacy status have as much

or as little weight as they want, by adjusting the size of the

bucket that straddles the cutoff.By gradually chipping away at the abuse of credentials, you could

probably make them more airtight. But what a long fight it would

be. Especially when the institutions administering the tests don't

really want them to be airtight.\_\_\_\_\_Fortunately there's a better way to prevent the direct transmission

of power between generations. Instead of trying to make credentials

harder to hack, we can also make them matter less.Let's think about what credentials are for. What they are,

functionally, is a way of predicting performance. If you could

measure actual performance, you wouldn't need them.So why did they even evolve? Why haven't we just been measuring

actual performance? Think about where credentialism first appeared:

in selecting candidates for large organizations. Individual

performance is hard to measure in large organizations, and the

harder performance is to measure, the more important it is

to predict it. If an organization could immediately and cheaply

measure the performance of recruits, they wouldn't need to examine

their credentials. They could take everyone and keep just the good

ones.Large organizations can't do this. But a bunch of small organizations

in a market can come close. A market takes every organization and

keeps just the good ones. As organizations get smaller, this

approaches taking every person and keeping just the good ones. So

all other things being equal, a society consisting of more, smaller

organizations will care less about credentials.\_\_\_\_\_That's what's been happening in the US. That's why those quotes

from Korea sound so old fashioned. They're talking about an economy

like America's a few decades ago, dominated by a few big companies.

The route for the ambitious in that sort of environment is to join

one and climb to the top. Credentials matter a lot then. In the

culture of a large organization, an elite pedigree becomes a self-fulfilling

prophecy.This doesn't work in small companies. Even if your colleagues were

impressed by your credentials, they'd soon be parted from you if

your performance didn't match, because the company would go out of

business and the people would be dispersed.In a world of small companies, performance is all anyone cares

about. People hiring for a startup don't care whether you've even

graduated from college, let alone which one. All they care about

is what you can do. Which is in fact all that should matter, even

in a large organization. The reason credentials have such prestige

is that for so long the large organizations

in a society tended to be the most powerful. But in the US at least

they don't have the monopoly on power they once did, precisely

because they can't measure (and thus reward) individual performance.

Why spend twenty years climbing the corporate ladder when you can

get rewarded directly by the market?I realize I see a more exaggerated version of the change than most

other people. As a partner at an early stage venture funding firm,

I'm like a jumpmaster shoving people out of the old world of

credentials and into the new one of performance. I'm an agent of

the change I'm seeing. But I don't think I'm imagining it. It was

not so easy 25 years ago for an ambitious person to choose to be

judged directly by the market. You had to go through bosses, and

they were influenced by where you'd been to college.\_\_\_\_\_What made it possible for small organizations to succeed in America?

I'm still not entirely sure. Startups are certainly a large part

of it. Small organizations can develop new ideas faster than large

ones, and new ideas are increasingly valuable.But I don't think startups account for all the shift from credentials

to measurement. My friend Julian Weber told me that when he went

to work for a New York law firm in the 1950s they paid associates

far less than firms do today. Law firms then made no pretense of

paying people according to the value of the work they'd done. Pay

was based on seniority. The younger employees were paying their

dues. They'd be rewarded later.The same principle prevailed at industrial companies. When my

father was working at Westinghouse in the 1970s, he had people

working for him who made more than he did, because they'd been there

longer.Now companies increasingly have to pay employees market price for

the work they do. One reason is that employees no longer trust

companies to deliver

deferred rewards: why work to accumulate

deferred rewards at a company that might go bankrupt, or be taken

over and have all its implicit obligations wiped out? The other

is that some companies broke ranks and started to pay young employees

large amounts. This was particularly true in consulting, law, and

finance, where it led to the phenomenon of yuppies. The word is

rarely used today because it's no longer surprising to see a 25

year old with money, but in 1985 the sight of a 25 year old

professional able to afford a new BMW was so novel that it

called forth a new word.The classic yuppie worked for a small organization. He didn't work

for General Widget, but for the law firm that handled General

Widget's acquisitions or the investment bank that floated their

bond issues.Startups and yuppies entered the American conceptual vocabulary

roughly simultaneously in the late 1970s and early 1980s. I don't

think there was a causal connection. Startups happened because

technology started to change so fast that big companies could no

longer keep a lid on the smaller ones. I don't think the rise of

yuppies was inspired by it; it seems more as if there was a change

in the social conventions (and perhaps the laws) governing the way

big companies worked. But the two phenomena rapidly fused to produce

a principle that now seems obvious: paying energetic young people

market rates, and getting correspondingly high performance from

them.At about the same time the US economy rocketed out of the doldrums

that had afflicted it for most of the 1970s. Was there a connection?

I don't know enough to say, but it felt like it at the time. There

was a lot of energy released.\_\_\_\_\_Countries worried about their competitiveness are right to be

concerned about the number of startups started within them. But

they would do even better to examine the underlying principle. Do

they let energetic young people get paid market rate for the work

they do? The young are the test, because when people aren't rewarded

according to performance, they're invariably rewarded according to

seniority instead.All it takes is a few beachheads in your economy that pay for

performance. Measurement spreads like heat. If one part of a

society is better at measurement than others, it tends to push the

others to do better. If people who are young but smart and driven

can make more by starting their own companies than by working for

existing ones, the existing companies are forced to pay more to

keep them. So market rates gradually permeate every organization,

even the government. [3]The measurement of performance will tend to push even the organizations

issuing credentials into line. When we were kids I used to annoy

my sister by ordering her to do things I knew she was about to do

anyway. As credentials are superseded by performance, a similar

role is the best former gatekeepers can hope for. Once credential

granting institutions are no longer in the self-fullfilling prophecy

business, they'll have to work harder to predict the future.\_\_\_\_\_Credentials are a step beyond bribery and influence. But they're

not the final step. There's an even better way to block the

transmission of power between generations: to encourage the trend

toward an economy made of more, smaller units. Then you can measure

what credentials merely predict.No one likes the transmission of power between generations—not

the left or the right. But the market forces favored by the right

turn out to be a better way of preventing it than the credentials

the left are forced to fall back on.The era of credentials began to end when the power of large

organizations peaked

in the late twentieth century. Now we seem

to be entering a new era based on measurement. The reason the new

model has advanced so rapidly is that it works so much better. It

shows no sign of slowing.Notes[1] Miyazaki, Ichisada

(Conrad Schirokauer trans.), China's Examination Hell: The Civil

Service Examinations of Imperial China, Yale University Press,

1981.Scribes in ancient Egypt took exams, but they were more the type

of proficiency test any apprentice might have to pass.[2] When I say the

raison d'etre of prep schools is to get kids into better colleges,

I mean this in the narrowest sense. I'm not saying that's all prep

schools do, just that if they had zero effect on college admissions

there would be far less demand for them.[3] Progressive tax

rates will tend to damp this effect, however, by decreasing the

difference between good and bad measurers.Thanks to Trevor Blackwell, Sarah Harlin, Jessica Livingston, and David

Sloo for reading drafts of this.

Could VC be a Casualty of the Recession?

December 2008(I originally wrote this at the request of a company producing

a report about entrepreneurship. Unfortunately after reading it

they decided it was too controversial to include.)

VC funding will probably dry up somewhat during the present recession,

like it usually does in bad times. But this time the result may

be different. This time the number of new startups may not decrease.

And that could be dangerous for VCs.When VC funding dried up after the Internet Bubble, startups dried

up too. There were not a lot of new startups being founded in

2003. But startups aren't tied to VC the way they were 10 years

ago. It's now possible for VCs and startups to diverge. And if

they do, they may not reconverge once the economy gets better.The reason startups no longer depend so much on VCs is one that

everyone in the startup business knows by now: it has gotten much

cheaper to start a startup. There are four main reasons: Moore's

law has made hardware cheap; open source has made software free;

the web has made marketing and distribution free; and more powerful

programming languages mean development teams can be smaller. These

changes have pushed the cost of starting a startup down into the

noise. In a lot of startups—probaby most startups funded by

Y Combinator—the biggest expense is simply the founders'

living expenses. We've had startups that were profitable on revenues

of $3000 a month.$3000 is insignificant as revenues go. Why should anyone care about

a startup making $3000 a month? Because, although insignificant

as revenue, this amount of money can change a startup's

funding situation completely.Someone running a startup is always calculating in the back of their

mind how much "runway" they have—how long they have till the

money in the bank runs out and they either have to be profitable,

raise more money, or go out of business. Once you cross the threshold

of profitability, however low, your runway becomes infinite. It's

a qualitative change, like the stars turning into lines and

disappearing when the Enterprise accelerates to warp speed. Once

you're profitable you don't need investors' money. And because

Internet startups have become so cheap to run, the threshold of

profitability can be trivially low. Which means many Internet

startups don't need VC-scale investments anymore. For many startups,

VC funding has, in the language of VCs, gone from a must-have to a

nice-to-have.This change happened while no one was looking, and its effects have

been largely masked so far. It was during the trough after the

Internet Bubble that it became trivially cheap to start a startup,

but few realized it because startups were so out of fashion. When

startups came back into fashion, around 2005, investors were starting

to write checks again. And while founders may not have needed VC

money the way they used to, they were willing to take it if

offered—partly because there was a tradition of startups

taking VC money, and partly because startups, like dogs, tend to

eat when given the opportunity. As long as VCs were writing checks,

founders were never forced to explore the limits of how little they

needed them. There were a few startups who hit these limits

accidentally because of their unusual circumstances—most

famously 37signals, which hit the limit because they crossed into

startup land from the other direction: they started as a consulting

firm, so they had revenue before they had a product.VCs and founders are like two components that used to be bolted

together. Around 2000 the bolt was removed. Because the components

have so far been subjected to the same forces, they still seem to

be joined together, but really one is just resting on the other.

A sharp impact would make them fly apart. And the present recession

could be that impact.Because of Y Combinator's position at the extreme end of the spectrum,

we'd be the first to see signs of a separation between founders and

investors, and we are in fact seeing it. For example, though the

stock market crash does seem to have made investors more cautious,

it doesn't seem to have had any effect on the number of people who

want to start startups. We take applications for funding every 6

months. Applications for the current funding cycle closed on October

17, well after the markets tanked, and even so we got a record

number, up 40% from the same cycle a year before.Maybe things will be different a year from now, if the economy

continues to get worse, but so far there is zero slackening of

interest among potential founders. That's different from the way

things felt in 2001. Then there was a widespread feeling among

potential founders that startups were over, and that one should

just go to grad school. That isn't happening this time, and part

of the reason is that even in a bad economy it's not that hard to

build something that makes $3000 a month. If investors stop writing

checks, who cares?We also see signs of a divergence between founders and investors

in the attitudes of existing startups we've funded. I was talking

to one recently that had a round fall through at the last minute

over the sort of trifle that breaks deals when investors feel they

have the upper hand—over an uncertainty about whether the

founders had correctly filed their 83(b) forms, if you can believe

that. And yet this startup is obviously going to succeed: their

traffic and revenue graphs look like a jet taking off. So I asked

them if they wanted me to introduce them to more investors. To my

surprise, they said no—that they'd just spent four months

dealing with investors, and they were actually a lot happier now

that they didn't have to. There was a friend they wanted to hire

with the investor money, and now they'd have to postpone that. But

otherwise they felt they had enough in the bank to make it to

profitability. To make sure, they were moving to a cheaper apartment.

And in this economy I bet they got a good deal on it.I've detected this "investors aren't worth the trouble" vibe from

several YC founders I've talked to recently. At least one startup

from the most recent (summer) cycle may not even raise angel money,

let alone VC. Ticketstumbler

made it to profitability on Y Combinator's $15,000 investment and

they hope not to need more. This surprised even us. Although YC

is based on the idea of it being cheap to start a startup, we never

anticipated that founders would grow successful startups on nothing

more than YC funding.If founders decide VCs aren't worth the trouble, that could be bad

for VCs. When the economy bounces back in a few years and they're

ready to write checks again, they may find that founders have moved

on.There is a founder community just as there's a VC community. They

all know one another, and techniques spread rapidly between them.

If one tries a new programming language or a new hosting provider

and gets good results, 6 months later half of them are using it.

And the same is true for funding. The current generation of founders

want to raise money from VCs, and Sequoia specifically, because

Larry and Sergey took money from VCs, and Sequoia specifically.

Imagine what it would do to the VC business if the next hot company

didn't take VC at all.VCs think they're playing a zero sum game. In fact, it's not even

that. If you lose a deal to Benchmark, you lose that deal, but VC

as an industry still wins. If you lose a deal to None, all VCs

lose.This recession may be different from the one after the Internet

Bubble. This time founders may keep starting startups. And if

they do, VCs will have to keep writing checks, or they could become

irrelevant.Thanks to Sam Altman, Trevor Blackwell, David Hornik, Jessica

Livingston, Robert Morris, and Fred Wilson for reading drafts of

this.Russian Translation

The High-Res Society

December 2008For nearly all of history the success of a society was proportionate

to its ability to assemble large and disciplined organizations.

Those who bet on economies of scale generally won, which meant the

largest organizations were the most successful ones.Things have already changed so much that this is hard for us to

believe, but till just a few decades ago the largest organizations

tended to be the most progressive. An ambitious kid graduating

from college in 1960 wanted to work in the huge, gleaming offices

of Ford, or General Electric, or NASA. Small meant small-time.

Small in 1960 didn't mean a cool little startup. It meant uncle

Sid's shoe store.When I grew up in the 1970s, the idea of the "corporate ladder" was

still very much alive. The standard plan was to try to get into a

good college, from which one would be drafted into some organization

and then rise to positions of gradually increasing responsibility.

The more ambitious merely hoped to climb the same ladder faster.

[1]But in the late twentieth century something changed. It turned out

that economies of scale were not the only force at work. Particularly

in technology, the increase in speed one could get from smaller

groups started to trump the advantages of size.The future turned out to be different from the one we were expecting

in 1970. The domed cities and flying cars we expected have failed

to materialize. But fortunately so have the jumpsuits with badges

indicating our specialty and rank. Instead of being dominated by

a few, giant tree-structured organizations, it's now looking like

the economy of the future will be a fluid network of smaller,

independent units.It's not so much that large organizations stopped working. There's

no evidence that famously successful organizations like the Roman

army or the British East India Company were any less afflicted by

protocol and politics than organizations of the same size today.

But they were competing against opponents who couldn't change the

rules on the fly by discovering new technology. Now it turns out

the rule "large and disciplined organizations win" needs to have a

qualification appended: "at games that change slowly." No one knew

till change reached a sufficient speed.Large organizations will start to do worse now, though,

because for the first time in history they're no longer getting the

best people. An ambitious kid graduating from college now doesn't

want to work for a big company. They want to work for the hot

startup that's rapidly growing into one. If they're really ambitious,

they want to start it.

[2]This doesn't mean big companies will disappear. To say that

startups will succeed implies that big companies will exist, because

startups that succeed either become big companies or are acquired

by them.

[3]

But large organizations will probably never again

play the leading role they did up till the last quarter of the

twentieth century.It's kind of surprising that a trend that lasted so long would ever

run out. How often does it happen that a rule works for thousands

of years, then switches polarity?The millennia-long run of bigger-is-better left us with a lot of

traditions that are now obsolete,

but extremely deeply rooted.

Which means the ambitious can now do arbitrage on them. It will

be very valuable to understand precisely which ideas to keep and

which can now be discarded.The place to look is where the spread of smallness began: in the

world of startups.There have always been occasional cases, particularly in the US,

of ambitious people who grew the ladder under them instead of

climbing it. But till recently this was an anomalous route that

tended to be followed only by outsiders. It was no coincidence

that the great industrialists of the nineteenth century had so

little formal education. As huge as their companies eventually

became, they were all essentially mechanics and shopkeepers at

first. That was a social step no one with a college education would

take if they could avoid it. Till the rise of technology startups,

and in particular, Internet startups, it was very unusual for

educated people to start their own businesses.The eight men who left Shockley Semiconductor to found Fairchild

Semiconductor, the original Silicon Valley startup, weren't even

trying to start a company at first. They were just looking for a

company willing to hire them as a group. Then one of their parents

introduced them to a small investment bank that offered to find

funding for them to start their own, so they did. But starting a

company was an alien idea to them; it was something they backed

into.

[4]Now I would guess that practically every Stanford or Berkeley

undergrad who knows how to program has at least considered the idea

of starting a startup. East Coast universities are not far behind,

and British universities only a little behind them. This pattern

suggests that attitudes at Stanford and Berkeley are not an anomaly,

but a leading indicator. This is the way the world is going.Of course, Internet startups are still only a fraction of the world's

economy. Could a trend based on them be that powerful?I think so. There's no reason to suppose there's any limit to the

amount of work that could be done in this area. Like science,

wealth seems to expand fractally. Steam power was a sliver of the

British economy when Watt started working on it. But his work led

to more work till that sliver had expanded into something bigger

than the whole economy of which it had initially been a part.The same thing could happen with the Internet. If Internet startups

offer the best opportunity for ambitious people, then a lot of

ambitious people will start them, and this bit of the economy will

balloon in the usual fractal way.Even if Internet-related applications only become a tenth of the

world's economy, this component will set the tone for the rest.

The most dynamic part of the economy always does, in everything

from salaries to standards of dress. Not just because of its

prestige, but because the principles underlying the most dynamic

part of the economy tend to be ones that work.For the future, the trend to bet on seems to be networks of small,

autonomous groups whose performance is measured individually. And

the societies that win will be the ones with the least impedance.As with the original industrial revolution, some societies are going

to be better at this than others. Within a generation of its birth

in England, the Industrial Revolution had spread to continental

Europe and North America. But it didn't spread everywhere. This

new way of doing things could only take root in places that were

prepared for it. It could only spread to places that already had

a vigorous middle class.There is a similar social component to the transformation that began

in Silicon Valley in the 1960s. Two new kinds of techniques were

developed there: techniques for building integrated circuits, and

techniques for building a new type of company designed to grow fast

by creating new technology. The techniques for building integrated

circuits spread rapidly to other countries. But the techniques for

building startups didn't. Fifty years later, startups are ubiquitous

in Silicon Valley and common in a handful of other US cities, but

they're still an anomaly in most of the world.Part of the reason—possibly the main reason—that startups

have not spread as broadly as the Industrial Revolution did is their

social disruptiveness. Though it brought many social changes, the

Industrial Revolution was not fighting the principle that bigger

is better. Quite the opposite: the two dovetailed beautifully.

The new industrial companies adapted the customs of existing large

organizations like the military and the civil service, and the

resulting hybrid worked well. "Captains of industry" issued orders

to "armies of workers," and everyone knew what they were supposed

to do.Startups seem to go more against the grain, socially. It's hard

for them to flourish in societies that value hierarchy and stability,

just as it was hard for industrialization to flourish in societies

ruled by people who stole at will from the merchant class. But

there were already a handful of countries past that stage when the

Industrial Revolution happened. There do not seem to be that many

ready this time.

Notes[1]

One of the bizarre consequences of this model was that the usual

way to make more money was to become a manager. This is one of the

things startups fix.[2]

There are a lot of reasons American car companies have been

doing so much worse than Japanese car companies, but at least one

of them is a cause for optimism: American graduates have more

options.[3]

It's possible that companies will one day be able to grow big

in revenues without growing big in people, but we are not very far

along that trend yet.[4]

Lecuyer, Christophe, Making Silicon Valley, MIT Press, 2006.Thanks to Trevor Blackwell, Paul Buchheit, Jessica Livingston,

and Robert Morris for reading drafts of this.

The Other Half of "Artists Ship"

November 2008One of the differences between big companies and startups is that

big companies tend to have developed procedures to protect themselves

against mistakes. A startup walks like a toddler, bashing

into things and falling over all the time. A big company is more

deliberate.The gradual accumulation of checks in an organization is a kind of

learning, based on disasters that have happened to it or others

like it. After giving a contract to a supplier who goes bankrupt

and fails to deliver, for example, a company might require all

suppliers to prove they're solvent before submitting bids.As companies grow they invariably get more such checks, either in

response to disasters they've suffered, or (probably more often)

by hiring people from bigger companies who bring with them customs

for protecting against new types of disasters.It's natural for organizations to learn from mistakes. The problem

is, people who propose new checks almost never consider that the

check itself has a cost.Every check has a cost. For example, consider the case of making

suppliers verify their solvency. Surely that's mere prudence? But

in fact it could have substantial costs. There's obviously the

direct cost in time of the people on both sides who supply and check

proofs of the supplier's solvency. But the real costs are the ones

you never hear about: the company that would be the best supplier,

but doesn't bid because they can't spare the effort to get verified.

Or the company that would be the best supplier, but falls just short

of the threshold for solvency—which will of course have been set

on the high side, since there is no apparent cost of increasing it.Whenever someone in an organization proposes to add a new check,

they should have to explain not just the benefit but the cost. No

matter how bad a job they did of analyzing it, this meta-check would

at least remind everyone there had to be a cost, and send them

looking for it.If companies started doing that, they'd find some surprises. Joel

Spolsky recently spoke at Y Combinator about selling software to

corporate customers. He said that in most companies software costing

up to about $1000 could be bought by individual managers without

any additional approvals. Above that threshold, software purchases

generally had to be approved by a committee. But babysitting this

process was so expensive for software vendors that it didn't make

sense to charge less than $50,000. Which means if you're making

something you might otherwise have charged $5000 for, you have to

sell it for $50,000 instead.The purpose of the committee is presumably to ensure that the company

doesn't waste money. And yet the result is that the company pays

10 times as much.Checks on purchases will always be expensive, because the harder

it is to sell something to you, the more it has to cost. And not

merely linearly, either. If you're hard enough to sell to, the

people who are best at making things don't want to bother. The

only people who will sell to you are companies that specialize in

selling to you. Then you've sunk to a whole new level of inefficiency.

Market mechanisms no longer protect you, because the good suppliers

are no longer in the market.Such things happen constantly to the biggest organizations of all,

governments. But checks instituted by governments can cause much

worse problems than merely overpaying. Checks instituted by

governments can cripple a country's whole economy. Up till about

1400, China was richer and more technologically advanced than Europe.

One reason Europe pulled ahead was that the Chinese government

restricted long trading voyages. So it was left to the Europeans

to explore and eventually to dominate the rest of the world, including

China.In more recent times, Sarbanes-Oxley has practically destroyed the

US IPO market. That wasn't the intention of the legislators who

wrote it. They just wanted to add a few more checks on public

companies. But they forgot to consider the cost. They forgot that

companies about to go public are usually rather stretched, and that

the weight of a few extra checks that might be easy for General

Electric to bear are enough to prevent younger companies from being

public at all.Once you start to think about the cost of checks, you can start to

ask other interesting questions. Is the cost increasing or decreasing?

Is it higher in some areas than others? Where does it increase

discontinuously? If large organizations started to ask questions

like that, they'd learn some frightening things.I think the cost of checks may actually be increasing. The reason

is that software plays an increasingly important role in companies,

and the people who write software are particularly harmed by checks.Programmers are unlike many types of workers in that the best ones

actually prefer to work hard. This doesn't seem to be the case in

most types of work. When I worked in fast food, we didn't prefer

the busy times. And when I used to mow lawns, I definitely didn't

prefer it when the grass was long after a week of rain.Programmers, though, like it better when they write more code. Or

more precisely, when they release more code. Programmers like to

make a difference. Good ones, anyway.For good programmers, one of the best things about working for a

startup is that there are few checks on releases. In true startups,

there are no external checks at all. If you have an idea for a new

feature in the morning, you can write it and push it to the production

servers before lunch. And when you can do that, you have more

ideas.At big companies, software has to go through various approvals

before it can be launched. And the cost of doing this can be

enormous—in fact, discontinuous. I was talking recently to a

group of three programmers whose startup had been acquired a few

years before by a big company. When they'd been independent, they

could release changes instantly. Now, they said, the absolute

fastest they could get code released on the production servers was

two weeks.This didn't merely make them less productive. It made them hate

working for the acquirer.Here's a sign of how much programmers like to be able to work hard:

these guys would have paid to be able to release code immediately,

the way they used to. I asked them if they'd trade 10% of the

acquisition price for the ability to release code immediately, and

all three instantly said yes. Then I asked what was the maximum

percentage of the acquisition price they'd trade for it. They said

they didn't want to think about it, because they didn't want to

know how high they'd go, but I got the impression it might be as

much as half.They'd have sacrificed hundreds of thousands of dollars, perhaps

millions, just to be able to deliver more software to users. And

you know what? It would have been perfectly safe to let them. In

fact, the acquirer would have been better off; not only wouldn't

these guys have broken anything, they'd have gotten a lot more done.

So the acquirer is in fact getting worse performance at greater

cost. Just like the committee approving software purchases.And just as the greatest danger of being hard to sell to is not

that you overpay but that the best suppliers won't even sell to

you, the greatest danger of applying too many checks to your

programmers is not that you'll make them unproductive, but that

good programmers won't even want to work for you.Steve Jobs's famous maxim "artists ship" works both ways. Artists

aren't merely capable of shipping. They insist on it. So if you

don't let people ship, you won't have any artists.

Why to Start a Startup in a Bad Economy

Want to start a startup? Get funded by

Y Combinator.

October 2008The economic situation is apparently so grim that some experts fear

we may be in for a stretch as bad as the mid seventies.When Microsoft and Apple were founded.As those examples suggest, a recession may not be such a bad time

to start a startup. I'm not claiming it's a particularly good time

either. The truth is more boring: the state of the economy doesn't

matter much either way.If we've learned one thing from funding so many startups, it's that

they succeed or fail based on the qualities of the founders. The

economy has some effect, certainly, but as a predictor of success

it's rounding error compared to the founders.Which means that what matters is who you are, not when you do it.

If you're the right sort of person, you'll win even in a bad economy.

And if you're not, a good economy won't save you. Someone who

thinks "I better not start a startup now, because the economy is

so bad" is making the same mistake as the people who thought during

the Bubble "all I have to do is start a startup, and I'll be rich."So if you want to improve your chances, you should think far more

about who you can recruit as a cofounder than the state of the

economy. And if you're worried about threats to the survival of

your company, don't look for them in the news. Look in the mirror.But for any given team of founders, would it not pay to wait till

the economy is better before taking the leap? If you're starting

a restaurant, maybe, but not if you're working on technology.

Technology progresses more or less independently of the stock market.

So for any given idea, the payoff for acting fast in a bad economy

will be higher than for waiting. Microsoft's first product was a

Basic interpreter for the Altair. That was exactly what the world

needed in 1975, but if Gates and Allen had decided to wait a few

years, it would have been too late.Of course, the idea you have now won't be the last you have. There

are always new ideas. But if you have a specific idea you want to

act on, act now.That doesn't mean you can ignore the economy. Both customers and investors

will be feeling pinched. It's not necessarily a problem if customers

feel pinched: you may even be able to benefit from it, by making

things that save money.

Startups often make things cheaper, so in

that respect they're better positioned to prosper in a recession

than big companies.Investors are more of a problem. Startups generally need to raise

some amount of external funding, and investors tend to be less

willing to invest in bad times. They shouldn't be. Everyone knows

you're supposed to buy when times are bad and sell when times are

good. But of course what makes investing so counterintuitive is

that in equity markets, good times are defined as everyone thinking

it's time to buy. You have to be a contrarian to be correct, and

by definition only a minority of investors can be.So just as investors in 1999 were tripping over one another trying

to buy into lousy startups, investors in 2009 will presumably be

reluctant to invest even in good ones.You'll have to adapt to this. But that's nothing new: startups

always have to adapt to the whims of investors. Ask any founder

in any economy if they'd describe investors as fickle, and watch

the face they make. Last year you had to be prepared to explain

how your startup was viral. Next year you'll have to explain how

it's recession-proof.(Those are both good things to be. The mistake investors make is

not the criteria they use but that they always tend to focus on one

to the exclusion of the rest.)Fortunately the way to make a startup recession-proof is to do

exactly what you should do anyway: run it as cheaply as possible.

For years I've been telling founders that the surest route to success

is to be the cockroaches of the corporate world. The immediate

cause of death in a startup is always running out of money. So the

cheaper your company is to operate, the harder it is to kill.

And fortunately it has gotten very cheap to run a startup. A recession

will if anything make it cheaper still.If nuclear winter really is here, it may be safer to be a cockroach

even than to keep your job. Customers may drop off individually

if they can no longer afford you, but you're not going to lose them

all at once; markets don't "reduce headcount."What if you quit your job to start a startup that fails, and you

can't find another? That could be a problem if you work in sales or

marketing. In those fields it can take months to find a new

job in a bad economy. But hackers seem to be more liquid. Good

hackers can always get some kind of job. It might not be your dream

job, but you're not going to starve.Another advantage of bad times is that there's less competition.

Technology trains leave the station at regular intervals. If

everyone else is cowering in a corner, you may have a whole car to

yourself.You're an investor too. As a founder, you're buying stock with

work: the reason Larry and Sergey are so rich is not so much that

they've done work worth tens of billions of dollars, but that they

were the first investors in Google. And like any investor you

should buy when times are bad.Were you nodding in agreement, thinking "stupid investors" a few

paragraphs ago when I was talking about how investors are reluctant

to put money into startups in bad markets, even though that's the

time they should rationally be most willing to buy? Well, founders

aren't much better. When times get bad, hackers go to grad school.

And no doubt that will happen this time too. In fact, what makes

the preceding paragraph true is that most readers won't believe

it—at least to the extent of acting on it.So maybe a recession is a good time to start a startup. It's hard

to say whether advantages like lack of competition outweigh

disadvantages like reluctant investors. But it doesn't matter much

either way. It's the people that matter. And for a given set of

people working on a given technology, the time to act is always

now.Russian TranslationChinese TranslationJapanese Translation

A Fundraising Survival Guide

Want to start a startup? Get funded by

Y Combinator.

August 2008Raising money is the second hardest part of starting a startup.

The hardest part is making something people want: most startups

that die, die because they didn't do that. But the second biggest

cause of death is probably the difficulty of raising money.

Fundraising is brutal.One reason it's so brutal is simply the brutality of markets. People

who've spent most of their lives in schools or big companies may

not have been exposed to that. Professors and bosses usually feel

some sense of responsibility toward you; if you make a valiant

effort and fail, they'll cut you a break. Markets are less forgiving.

Customers don't care how hard you worked, only whether you solved

their problems.Investors evaluate startups the way customers evaluate products,

not the way bosses evaluate employees. If you're making a valiant

effort and failing, maybe they'll invest in your next startup, but

not this one.But raising money from investors is harder than selling to

customers, because there are so few of them. There's

nothing like an efficient market. You're unlikely to have more

than 10 who are interested; it's difficult to talk to more. So the

randomness of any one investor's behavior can really affect you.Problem number 3: investors are very random. All investors, including

us, are by ordinary standards incompetent. We constantly have to

make decisions about things we don't understand, and more often

than not we're wrong.And yet a lot is at stake. The amounts invested by different types

of investors vary from five thousand dollars to fifty million, but

the amount usually seems large for whatever type of investor it is.

Investment decisions are big decisions.That combination—making big decisions about things they don't

understand—tends to make investors very skittish. VCs are notorious

for leading founders on. Some of the more unscrupulous do it

deliberately. But even the most well-intentioned investors can

behave in a way that would seem crazy in everyday life. One day

they're full of enthusiasm and seem ready to write you a check on

the spot; the next they won't return your phone calls. They're not

playing games with you. They just can't make up their minds.

[1]If that weren't bad enough, these wildly fluctuating nodes are all

linked together. Startup investors all know one another, and (though

they hate to admit it) the biggest factor in their opinion of you

is the opinion of other investors.

[2]

Talk about a recipe for

an unstable system. You get the opposite of the damping that the

fear/greed balance usually produces in markets. No one is interested

in a startup that's a "bargain" because everyone else hates it.So the inefficient market you get because there are so few players

is exacerbated by the fact that they act less than independently.

The result is a system like some kind of primitive, multi-celled

sea creature, where you irritate one extremity and the whole thing

contracts violently.Y Combinator is working to fix this. We're trying to increase the

number of investors just as we're increasing the number of startups.

We hope that as the number of both increases we'll get something

more like an efficient market. As t approaches infinity, Demo Day

approaches an auction.Unfortunately, t is still very far from infinity. What does a

startup do now, in the imperfect world we currently inhabit? The

most important thing is not to let fundraising get you down. Startups

live or die on morale. If you let the difficulty of raising money

destroy your morale, it will become a self-fulfilling prophecy.

Bootstrapping (= Consulting)Some would-be founders may by now be thinking, why deal with investors

at all? If raising money is so painful, why do it?One answer to that is obvious: because you need money to live on.

It's a fine idea in principle to finance your startup with its own

revenues, but you can't create instant customers. Whatever you

make, you have to sell a certain amount to break even. It will

take time to grow your sales to that point, and it's hard to predict,

till you try, how long it will take.We could not have bootstrapped Viaweb, for example. We charged

quite a lot for our software—about $140 per user per month—but

it was at least a year before our revenues would have covered even

our paltry costs. We didn't have enough saved to live on for a

year.If you factor out the "bootstrapped" companies that were actually

funded by their founders through savings or a day job, the remainder

either (a) got really lucky, which is hard to do on demand, or (b)

began life as consulting companies and gradually transformed

themselves into product companies.Consulting is the only option you can count on. But consulting is

far from free money. It's not as painful as raising money from

investors, perhaps, but the pain is spread over a longer period.

Years, probably. And for many types of startup, that delay could

be fatal. If you're working on something so unusual that no one

else is likely to think of it, you can take your time. Joshua

Schachter gradually built Delicious on the side while working on

Wall Street. He got away with it because no one else realized it

was a good idea. But if you were building something as obviously

necessary as online store software at about the same time as Viaweb,

and you were working on it on the side while spending most of your

time on client work, you were not in a good position.Bootstrapping sounds great in principle, but this apparently verdant

territory is one from which few startups emerge alive. The mere

fact that bootstrapped startups tend to be famous on that account

should set off alarm bells. If it worked so well, it would be the

norm.

[3]

Bootstrapping may get easier, because starting a company is getting

cheaper. But I don't think we'll ever reach the point where most

startups can do without outside funding. Technology tends to

get dramatically cheaper, but living expenses don't.The upshot is, you can choose your pain: either the short, sharp

pain of raising money, or the chronic ache of consulting. For a

given total amount of pain, raising money is the better choice,

because new technology is usually more valuable now than later.But although for most startups raising money will be the lesser

evil, it's still a pretty big evil—so big that it can easily kill

you. Not merely in the obvious sense that if you fail to raise

money you might have to shut the company down, but because the

process of raising money itself can kill you.To survive it you need a set of techniques mostly

orthogonal to the ones used in convincing investors, just as mountain

climbers need to know survival techniques that are mostly orthogonal

to those used in physically getting up and down mountains.

1. Have low expectations.The reason raising money destroys so many startups' morale is not

simply that it's hard, but that it's so much harder than they

expected. What kills you is the disappointment. And the lower

your expectations, the harder it is to be disappointed.Startup founders tend to be optimistic. This can work well in

technology, at least some of the time, but it's the wrong way to

approach raising money. Better to assume investors will always let

you down. Acquirers too, while we're at it. At YC one of our

secondary mantras is "Deals fall through." No matter what deal

you have going on, assume it will fall through. The predictive

power of this simple rule is amazing.There will be a tendency, as a deal progresses, to start to believe

it will happen, and then to depend on it happening. You must resist

this. Tie yourself to the mast. This is what kills you. Deals

do not have a trajectory like most other human interactions, where

shared plans solidify linearly over time. Deals often fall through

at the last moment. Often the other party doesn't really think

about what they want till the last moment. So you can't use your

everyday intuitions about shared plans as a guide. When it comes

to deals, you have to consciously turn them off and become

pathologically cynical.This is harder to do than it sounds. It's very flattering when

eminent investors seem interested in funding you. It's easy to

start to believe that raising money will be quick and straightforward.

But it hardly ever is.

2. Keep working on your startup.It sounds obvious to say that you should keep working on your startup

while raising money. Actually this is hard to do. Most startups

don't manage to.Raising money has a mysterious capacity to suck up all your attention.

Even if you only have one meeting a day with investors, somehow

that one meeting will burn up your whole day. It costs not just

the time of the actual meeting, but the time getting there and back,

and the time preparing for it beforehand and thinking about it

afterward.The best way to survive the distraction of meeting with investors

is probably to partition the company: to pick one founder to deal

with investors while the others keep the company going. This works

better when a startup has 3 founders than 2, and better when the

leader of the company is not also the lead developer. In the best

case, the company keeps moving forward at about half speed.That's the best case, though. More often than not the company comes

to a standstill while raising money. And that is dangerous for so

many reasons. Raising money always takes longer than you expect.

What seems like it's going to be a 2 week interruption turns into

a 4 month interruption. That can be very demoralizing. And worse

still, it can make you less attractive to investors. They want to

invest in companies that are dynamic. A company that hasn't done

anything new in 4 months doesn't seem dynamic, so they start to

lose interest. Investors rarely grasp this, but much of what

they're responding to when they lose interest in a startup is the

damage done by their own indecision.The solution: put the startup first. Fit meetings with investors

into the spare moments in your development schedule, rather than

doing development in the spare moments between meetings with

investors. If you keep the company moving forward—releasing new

features, increasing traffic, doing deals, getting written

about—those investor meetings are more likely to be productive. Not just

because your startup will seem more alive, but also because it will

be better for your own morale, which is one of the main ways investors

judge you.

3. Be conservative.As conditions get worse, the optimal strategy becomes more conservative.

When things go well you can take risks; when things are bad you

want to play it safe.I advise approaching fundraising as if it were always going badly.

The reason is that between your ability to delude yourself and the

wildly unstable nature of the system you're dealing with, things

probably either already are or could easily become much worse than

they seem.What I tell most startups we fund is that if someone reputable

offers you funding on reasonable terms, take it. There have been

startups that ignored this advice and got away with it—startups

that ignored a good offer in the hope of getting a better one, and

actually did. But in the same position I'd give the same advice

again. Who knows how many bullets were in the gun they were playing

Russian roulette with?Corollary: if an investor seems interested, don't just let them

sit. You can't assume someone interested in investing will stay

interested. In fact, you can't even tell (they can't even tell)

if they're really interested till you try to convert that interest

into money. So if you have hot prospect, either close them now or

write them off. And unless you already have enough funding, that

reduces to: close them now.Startups don't win by getting great funding rounds, but by making

great products. So finish raising money and get

back to work.

4. Be flexible.There are two questions VCs ask that you shouldn't answer: "Who

else are you talking to?" and "How much are you trying to raise?"VCs don't expect you to answer the first question. They ask it just

in case.

[4]

They do seem to expect an answer to the second. But

I don't think you should just tell them a number. Not as a way to

play games with them, but because you shouldn't have a fixed

amount you need to raise.The custom of a startup needing a fixed amount of funding is an

obsolete one left over from the days when startups were more

expensive. A company that needed to build a factory or hire 50

people obviously needed to raise a certain minimum amount. But few

technology startups are in that position today.We advise startups to tell investors there are several different

routes they could take depending on how much they raised. As little

as $50k could pay for food and rent for the founders for a year.

A couple hundred thousand would let them get office space and hire

some smart people they know from school. A couple million would

let them really blow this thing out. The message (and not just the

message, but the fact) should be: we're going to succeed no matter

what. Raising more money just lets us do it faster.If you're raising an angel round, the size of the round can even

change on the fly. In fact, it's just as well to make the round

small initially, then expand as needed, rather than trying to raise

a large round and risk losing the investors you already have if you

can't raise the full amount. You may even want to do a "rolling

close," where the round has no predetermined size, but instead you

sell stock to investors one at a time as they say yes. That helps

break deadlocks, because you can start as soon as the first one

is ready to buy.

[5]

5. Be independent.A startup with a couple founders in their early twenties can have

expenses so low that they could be profitable on

as little as $2000 per month. That's negligible as corporate

revenues go, but the effect on your morale and your bargaining

position is anything but. At YC we use the phrase "ramen profitable"

to describe the situation where you're making just enough to pay

your living expenses. Once you cross into ramen profitable,

everything changes. You may still need investment to make it big,

but you don't need it this month.You can't plan when you start a startup how long

it will take to become profitable. But if you find yourself in a

position where a little more effort expended on sales would carry

you over the threshold of ramen profitable, do it.Investors like it when you're ramen profitable. It shows you've

thought about making money, instead of just working on amusing

technical problems; it shows you have the discipline to keep your

expenses low; but above all, it means you don't need them.There is nothing investors like more than a startup that seems like

it's going to succeed even without them. Investors like it when

they can help a startup, but they don't like startups that would

die without that help.At YC we spend a lot of time trying to predict how the startups we've

funded will do, because we're trying to learn how to pick winners.

We've now watched the trajectories of so many startups that we're

getting better at predicting them. And when we're talking

about startups we think are likely to succeed, what we find ourselves

saying is things like "Oh, those guys can take care of themselves.

They'll be fine." Not "those guys are really smart" or

"those guys are working on a great idea."

[6]

When we predict good outcomes for startups, the qualities

that come up in the supporting arguments are toughness, adaptability,

determination. Which means to the extent we're correct, those are

the qualities you need to win.Investors know this, at least unconsciously. The reason they like

it when you don't need them is not simply that they like what they

can't have, but because that quality is what makes founders succeed.Sam Altman

has it. You could parachute him into an island full of

cannibals and come back in 5 years and he'd be the king. If you're

Sam Altman, you don't have to be profitable to convey to investors

that you'll succeed with or without them. (He wasn't, and he did.)

Not everyone has Sam's deal-making ability. I myself don't. But

if you don't, you can let the numbers speak for you.

6. Don't take rejection personally.Getting rejected by investors can make you start to doubt yourself.

After all, they're more experienced than you. If they think your

startup is lame, aren't they probably right?Maybe, maybe not. The way to handle rejection is with precision.

You shouldn't simply ignore rejection. It might mean something.

But you shouldn't automatically get demoralized either.To understand what rejection means, you have to understand first

of all how common it is. Statistically, the average VC is a rejection

machine. David Hornik, a partner at August, told me:

The numbers for me ended up being something like 500 to 800 plans

received and read, somewhere between 50 and 100 initial 1 hour

meetings held, about 20 companies that I got interested in, about

5 that I got serious about and did a bunch of work, 1 to 2 deals

done in a year. So the odds are against you. You

may be a great entrepreneur, working on interesting stuff, etc.

but it is still incredibly unlikely that you get funded.

This is less true with angels, but VCs reject practically everyone.

The structure of their business means a partner does at most 2 new

investments a year, no matter how many good startups approach him.In addition to the odds being terrible, the average investor is,

as I mentioned, a pretty bad judge of startups. It's harder to

judge startups than most other things, because great startup ideas

tend to seem wrong. A good startup idea has to be not just good but

novel. And to be both good and novel, an idea probably has to seem

bad to most people, or someone would already be doing it and it

wouldn't be novel.That makes judging startups harder than most other things one judges.

You have to be an intellectual contrarian to be a good startup

investor. That's a problem for VCs, most of whom are not particularly

imaginative. VCs are mostly money guys, not people who make things.

[7]

Angels are better at appreciating novel ideas, because most

were founders themselves.So when you get a rejection, use the data that's in it, and not what's

not. If an investor gives you specific reasons for not investing,

look at your startup and ask if they're right. If they're real

problems, fix them. But don't just take their word for it. You're

supposed to be the domain expert; you have to decide.Though a rejection doesn't necessarily tell you anything about your

startup, it does suggest your pitch could be improved. Figure out

what's not working and change it. Don't just think "investors are

stupid." Often they are, but figure out precisely where you lose

them.Don't let rejections pile up as a depressing, undifferentiated heap.

Sort them and analyze them, and then instead of thinking "no one

likes us," you'll know precisely how big a problem you have, and

what to do about it.

7. Be able to downshift into consulting (if appropriate).Consulting, as I mentioned, is a dangerous way to finance a startup.

But it's better than dying. It's a bit like anaerobic respiration:

not the optimum solution for the long term, but it can save you

from an immediate threat. If you're having trouble raising money

from investors at all, it could save you to be able to shift

toward consulting.This works better for some startups than others. It wouldn't have

been a natural fit for, say, Google, but if your company was making

software for building web sites, you could degrade fairly gracefully

into consulting by building sites for clients with it.So long as you were careful not to get sucked permanently into

consulting, this could even have advantages. You'd understand your

users well if you were using the software for them. Plus as a

consulting company you might be able to get big-name users using

your software that you wouldn't have gotten as a product company.At Viaweb we were forced to operate like a consulting company

initially, because we were so desperate for users that we'd offer

to build merchants' sites for them if they'd sign up.

But we never charged for such work, because we didn't want them to

start treating us like actual consultants, and calling us every

time they wanted something changed on their site. We knew we had

to stay a product company, because only

that scales.

8. Avoid inexperienced investors.Though novice investors seem unthreatening they can be the most

dangerous sort, because they're so nervous. Especially in

proportion to the amount they invest. Raising $20,000 from a first-time

angel investor can be as much work as raising $2 million from

a VC fund.Their lawyers are generally inexperienced too. But while the

investors can admit they don't know what they're doing, their lawyers

can't. One YC startup negotiated terms for a tiny round with

an angel, only to receive a 70-page agreement from his lawyer. And

since the lawyer could never admit, in front of his client, that

he'd screwed up, he instead had to insist on retaining all the

draconian terms in it, so the deal fell through.Of course, someone has to take money from novice investors, or there

would never be any experienced ones. But if you do, either (a)

drive the process yourself, including supplying the

paperwork, or

(b) use them only to fill up a larger round led by someone else.

9. Know where you stand.The most dangerous thing about investors is their indecisiveness.

The worst case scenario is the long no, the no that comes after

months of meetings. Rejections from investors are like design

flaws: inevitable, but much less costly if you discover them early.So while you're talking to investors, constantly look for signs of

where you stand. How likely are they to offer you a term sheet?

What do they have to be convinced of first? You shouldn't necessarily

always be asking these questions outright—that could get

annoying—but you should always be collecting data about them.Investors tend to resist committing except to the extent you push

them to. It's in their interest to collect the maximum amount of

information while making the minimum number of decisions. The best

way to force them to act is, of course, competing investors. But

you can also apply some force by focusing the discussion:

by asking what specific questions they need answered to make

up their minds, and then answering them. If you get through several

obstacles and they keep raising new ones, assume that ultimately

they're going to flake.You have to be disciplined when collecting data about investors'

intentions. Otherwise their desire to lead you on will combine

with your own desire to be led on to produce completely inaccurate

impressions.Use the data to weight your strategy.

You'll probably be talking to several investors. Focus on the ones

that are most likely to say yes. The value of a potential investor

is a combination of how good it would be if they said yes, and how

likely they are to say it. Put the most weight on the second factor.

Partly because the most important quality in an investor is simply

investing. But also because, as I mentioned, the biggest factor

in investors' opinion of you is other investors' opinion of you.

If you're talking to several investors and you manage to get one

over the threshold of saying yes, it will make the others much more

interested. So you're not sacrificing the lukewarm investors if

you focus on the hot ones; convincing the hot investors is the best

way to convince the lukewarm ones.

FutureI'm hopeful things won't always be so awkward. I hope that as startups

get cheaper and the number of investors increases, raising money

will become, if not easy, at least straightforward.In the meantime, the brokenness of the funding process offers a big

opportunity. Most investors have no idea how dangerous they are.

They'd be surprised to hear that raising money from them is something

that has to be treated as a threat to a company's survival. They

just think they need a little more information to make up their

minds. They don't get that there are 10 other investors who also

want a little more information, and that the process of talking to

them all can bring a startup to a standstill for months.Because investors don't understand the cost of dealing with them,

they don't realize how much room there is for a potential competitor

to undercut them. I know from my own experience how much faster

investors could decide, because we've brought our own time down to

20 minutes (5 minutes of reading an application plus a 10 minute

interview plus 5 minutes of discussion). If you were investing

more money you'd want to take longer, of course. But if we can

decide in 20 minutes, should it take anyone longer than a couple

days?Opportunities like this don't sit unexploited forever, even in an

industry as conservative as venture capital. So

either existing investors will start to make up their minds faster,

or new investors will emerge who do.In the meantime founders have to treat raising money as a dangerous

process. Fortunately, I can fix the biggest danger right here.

The biggest danger is surprise. It's that startups will underestimate

the difficulty of raising money—that they'll cruise through all

the initial steps, but when they turn to raising money they'll find

it surprisingly hard, get demoralized, and give up. So I'm telling

you in advance: raising money is hard.Notes[1]

When investors can't make up their minds, they sometimes

describe it as if it were a property of the startup. "You're too

early for us," they sometimes say. But which of them, if they were

taken back in a time machine to the hour Google was founded, wouldn't

offer to invest at any valuation the founders chose? An hour old

is not too early if it's the right startup. What "you're too early"

really means is "we can't figure out yet whether you'll succeed."

[2]

Investors influence one another both directly and indirectly.

They influence one another directly through the "buzz" that surrounds

a hot startup. But they also influence one another indirectly

through the founders. When a lot of investors are interested in

you, it increases your confidence in a way that makes you much more

attractive to investors.No VC will admit they're influenced by buzz. Some genuinely aren't.

But there are few who can say they're not influenced by confidence.[3]

One VC who read this essay wrote:"We try to avoid companies that got bootstrapped with consulting.

It creates very bad behaviors/instincts that are hard to erase

from a company's culture."[4]

The optimal way to answer the first question is to say that

it would be improper to name names, while simultaneously implying

that you're talking to a bunch of other VCs who are all about to

give you term sheets. If you're the sort of person who understands

how to do that, go ahead. If not, don't even try. Nothing annoys

VCs more than clumsy efforts to manipulate them.[5]

The disadvantage of expanding a round on the fly is that the

valuation is fixed at the start, so if you get a sudden rush of

interest, you may have to decide between turning some investors

away and selling more of the company than you meant to. That's a

good problem to have, however.[6]

I wouldn't say that intelligence doesn't matter in startups.

We're only comparing YC startups, who've already made it over a

certain threshold.[7]

But not all are. Though most VCs are suits at heart,

the most successful ones tend not to be. Oddly enough,

the best VCs tend to be the least VC-like.

Thanks to Trevor Blackwell, David Hornik, Jessica Livingston,

Robert Morris, and Fred Wilson for reading drafts of this.Russian Translation

The Pooled-Risk Company Management Company

July 2008At this year's startup school, David Heinemeier Hansson gave a

talk

in which he suggested that startup founders

should do things the old fashioned way. Instead of hoping to get

rich by building a valuable company and then selling stock in a

"liquidity event," founders should start companies that make money

and live off the revenues.Sounds like a good plan. Let's think about the optimal way to do

this.One disadvantage of living off the revenues of your company is that

you have to keep running it. And as anyone who runs their own

business can tell you, that requires your complete attention. You

can't just start a business and check out once things are going

well, or they stop going well surprisingly fast.The main economic motives of startup founders seem to be freedom

and security. They want enough money that (a) they don't have to

worry about running out of money and (b) they can spend their time

how they want. Running your own business offers neither. You

certainly don't have freedom: no boss is so demanding. Nor do you

have security, because if you stop paying attention to the company,

its revenues go away, and with them your income.The best case, for most people, would be if you could hire someone

to manage the company for you once you'd grown it to a certain size.

Suppose you could find a really good manager. Then you would have

both freedom and security. You could pay as little attention to

the business as you wanted, knowing that your manager would keep

things running smoothly. And that being so, revenues would continue

to flow in, so you'd have security as well.There will of course be some founders who wouldn't like that idea:

the ones who like running their company so much that there's nothing

else they'd rather do. But this group must be small. The way you

succeed in most businesses is to be fanatically attentive

to customers' needs. What are the odds that your own desires would

coincide exactly with the demands of this powerful, external force?Sure, running your own company can be fairly interesting. Viaweb

was more interesting than any job I'd had before. And since I made

much more money from it, it offered the highest ratio of income to

boringness of anything I'd done, by orders of magnitude. But was

it the most interesting work I could imagine doing? No.Whether the number of founders in the same position is asymptotic

or merely large, there are certainly a lot of them. For them the

right approach would be to hand the company over to a professional

manager eventually, if they could find one who was good enough.\_\_\_\_\_So far so good. But what if your manager was hit by a bus? What

you really want is a management company to run your company for

you. Then you don't depend on any one person.If you own rental property, there are companies you can hire to

manage it for you. Some will do everything, from finding tenants

to fixing leaks. Of course, running companies is a lot more

complicated than managing rental property, but let's suppose there

were management companies that could do it for you. They'd charge

a lot, but wouldn't it be worth it? I'd sacrifice a large percentage

of the income for the extra peace of mind.I realize what I'm describing already sounds too good to be true, but I

can think of a way to make it even more attractive. If

company management companies existed, there would be an additional

service they could offer clients: they could let them insure their

returns by pooling their risk. After all, even a perfect manager can't save a company

when, as sometimes happens, its whole market dies, just as property

managers can't save you from the building burning down. But a

company that managed a large enough number of companies could say

to all its clients: we'll combine the revenues from all your

companies, and pay you your proportionate share.If such management companies existed, they'd offer the maximum of

freedom and security. Someone would run your company for you, and

you'd be protected even if it happened to die.Let's think about how such a management company might be organized.

The simplest way would be to have a new kind of stock representing

the total pool of companies they were managing. When you signed

up, you'd trade your company's stock for shares of this pool, in

proportion to an estimate of your company's value that you'd both

agreed upon. Then you'd automatically get your share of the returns

of the whole pool.The catch is that because this kind of trade would be hard to undo,

you couldn't switch management companies. But there's a way they

could fix that: suppose all the company management companies got

together and agreed to allow their clients to exchange shares in

all their pools. Then you could, in effect, simultaneously choose

all the management companies to run yours for you, in whatever

proportion you wanted, and change your mind later as often as you

wanted.If such pooled-risk company management companies existed, signing

up with one would seem the ideal plan for most people following the

route David advocated.Good news: they do exist. What I've just

described is an acquisition by a public company.\_\_\_\_\_Unfortunately, though public acquirers are structurally identical

to pooled-risk company management companies, they don't think of

themselves that way. With a property management company, you can

just walk in whenever you want and say "manage my rental property

for me" and they'll do it. Whereas acquirers are, as of this

writing, extremely fickle. Sometimes they're in a buying mood and

they'll overpay enormously; other times they're not interested.

They're like property management companies run by madmen. Or more

precisely, by Benjamin Graham's Mr. Market.So while on average public acquirers behave like pooled-risk company

managers, you need a window of several years to get average case

performance. If you wait long enough (five years, say) you're

likely to hit an up cycle where some acquirer is hot to buy you.

But you can't choose when it happens.You can't assume investors will carry you for as long as you might

have to wait. Your company has to make money. Opinions are divided

about how early to focus on that.

Joe Kraus says you should try

charging customers right away. And yet some of the most successful

startups, including Google, ignored revenue at first and concentrated

exclusively on development. The answer probably depends on the

type of company you're starting. I can imagine some where trying

to make sales would be a good heuristic for product design, and

others where it would just be a distraction. The test is probably

whether it helps you to understand your users.You can choose whichever revenue strategy you think is best for the

type of company you're starting, so long as you're profitable.

Being profitable ensures you'll get at least the average of the

acquisition market—in which public companies do behave as pooled-risk

company management companies.David isn't mistaken in saying you should start a company to live

off its revenues. The mistake is thinking this is somehow opposed

to starting a company and selling it. In fact, for most people the

latter is merely the optimal case of the former.Thanks to Trevor Blackwell, Jessica Livingston, Michael

Mandel, Robert Morris, and Fred Wilson for reading drafts of this.

Russian Translation

Cities and Ambition

May 2008

Great cities attract ambitious people. You can sense it when you

walk around one. In a hundred subtle ways, the city sends you a

message: you could do more; you should try harder.The surprising thing is how different these messages can be. New

York tells you, above all: you should make more money. There are

other messages too, of course. You should be hipper. You should

be better looking. But the clearest message is that you should be

richer.What I like about Boston (or rather Cambridge) is that the message

there is: you should be smarter. You really should get around to

reading all those books you've been meaning to.When you ask what message a city sends, you sometimes get surprising

answers. As much as they respect brains in Silicon Valley, the

message the Valley sends is: you should be more powerful.That's not quite the same message New York sends. Power matters

in New York too of course, but New York is pretty impressed by a

billion dollars even if you merely inherited it. In Silicon Valley

no one would care except a few real estate agents. What matters

in Silicon Valley is how much effect you have on the world. The

reason people there care about Larry and Sergey is not their wealth

but the fact that they control Google, which affects practically

everyone.\_\_\_\_\_How much does it matter what message a city sends? Empirically,

the answer seems to be: a lot. You might think that if you had

enough strength of mind to do great things, you'd be able to transcend

your environment. Where you live should make at most a couple

percent difference. But if you look at the historical evidence,

it seems to matter more than that. Most people who did great things

were clumped together in a few places where that sort of thing was

done at the time.You can see how powerful cities are from something I wrote about

earlier: the case of the Milanese Leonardo.

Practically every

fifteenth century Italian painter you've heard of was from Florence,

even though Milan was just as big. People in Florence weren't

genetically different, so you have to assume there was someone born

in Milan with as much natural ability as Leonardo. What happened

to him?If even someone with the same natural ability as Leonardo

couldn't beat the force of environment, do you suppose you can?I don't. I'm fairly stubborn, but I wouldn't try to fight this

force. I'd rather use it. So I've thought a lot about where to

live.I'd always imagined Berkeley would be the ideal place — that

it would basically be Cambridge with good weather. But when I

finally tried living there a couple years ago, it turned out not

to be. The message Berkeley sends is: you should live better. Life

in Berkeley is very civilized. It's probably the place in America

where someone from Northern Europe would feel most at home. But

it's not humming with ambition.In retrospect it shouldn't have been surprising that a place so

pleasant would attract people interested above all in quality of

life. Cambridge with good weather, it turns out, is not Cambridge.

The people you find in Cambridge are not there by accident. You

have to make sacrifices to live there. It's expensive and somewhat

grubby, and the weather's often bad. So the kind of people you

find in Cambridge are the kind of people who want to live where the

smartest people are, even if that means living in an expensive,

grubby place with bad weather.As of this writing, Cambridge seems to be the intellectual capital

of the world. I realize that seems a preposterous claim. What

makes it true is that it's more preposterous to claim about anywhere

else. American universities currently seem to be the best, judging

from the flow of ambitious students. And what US city has a stronger

claim? New York? A fair number of smart people, but diluted by a

much larger number of neanderthals in suits. The Bay Area has a

lot of smart people too, but again, diluted; there are two great

universities, but they're far apart. Harvard and MIT are practically

adjacent by West Coast standards, and they're surrounded by about

20 other colleges and universities.

[1]Cambridge as a result feels like a town whose main industry is

ideas, while New York's is finance and Silicon Valley's is startups.\_\_\_\_\_When you talk about cities in the sense we are, what you're really

talking about is collections of people. For a long time cities

were the only large collections of people, so you could use the two

ideas interchangeably. But we can see how much things are changing

from the examples I've mentioned. New York is a classic great city.

But Cambridge is just part of a city, and Silicon Valley is not

even that. (San Jose is not, as it sometimes claims, the capital

of Silicon Valley. It's just 178 square miles at one end of it.)Maybe the Internet will change things further. Maybe one day the

most important community you belong to will be a virtual one, and

it won't matter where you live physically. But I wouldn't bet on

it. The physical world is very high bandwidth, and some of the

ways cities send you messages are quite subtle.One of the exhilarating things about coming back to Cambridge every

spring is walking through the streets at dusk, when you can see

into the houses. When you walk through Palo Alto in the evening,

you see nothing but the blue glow of TVs. In Cambridge you see

shelves full of promising-looking books. Palo Alto was probably

much like Cambridge in 1960, but you'd never guess now that there

was a university nearby. Now it's just one of the richer neighborhoods

in Silicon Valley.

[2]A city speaks to you mostly by accident — in things you see

through windows, in conversations you overhear. It's not something

you have to seek out, but something you can't turn off. One of the

occupational hazards of living in Cambridge is overhearing the

conversations of people who use interrogative intonation in declarative

sentences. But on average I'll take Cambridge conversations over

New York or Silicon Valley ones.A friend who moved to Silicon Valley in the late 90s said the worst

thing about living there was the low quality of the eavesdropping.

At the time I thought she was being deliberately eccentric. Sure,

it can be interesting to eavesdrop on people, but is good quality

eavesdropping so important that it would affect where you chose to

live? Now I understand what she meant. The conversations you

overhear tell you what sort of people you're among.\_\_\_\_\_No matter how determined you are, it's hard not to be influenced

by the people around you. It's not so much that you do whatever a

city expects of you, but that you get discouraged when no one around

you cares about the same things you do.There's an imbalance between encouragement and discouragement like

that between gaining and losing money. Most people overvalue

negative amounts of money: they'll work much harder to avoid losing

a dollar than to gain one. Similarly, although there are plenty of

people strong enough to resist doing something just because that's

what one is supposed to do where they happen to be, there are few

strong enough to keep working on something no one around them cares

about.Because ambitions are to some extent incompatible and admiration

is a zero-sum game, each city tends to focus on one type of ambition.

The reason Cambridge is the intellectual capital is not just that

there's a concentration of smart people there, but that there's

nothing else people there care about more. Professors in

New York and the Bay area are second class citizens — till they

start hedge funds or startups respectively.This suggests an answer to a question people in New York have

wondered about since the Bubble: whether New York could grow into

a startup hub to rival Silicon Valley. One reason that's unlikely

is that someone starting a startup in New York would feel like a

second class citizen.

[3]

There's already something else people in New York admire more.In the long term, that could be a bad thing for New York. The power

of an important new technology does eventually convert to money.

So by caring more about money and less about power than Silicon

Valley, New York is recognizing the same thing, but slower.

[4]

And in fact it has been losing to Silicon Valley at its own game:

the ratio of New York to California residents in the Forbes 400 has

decreased from 1.45 (81:56) when the list was first published in

1982 to .83 (73:88) in 2007.\_\_\_\_\_Not all cities send a message. Only those that are centers for

some type of ambition do. And it can be hard to tell exactly what

message a city sends without living there. I understand the messages

of New York, Cambridge, and Silicon Valley because I've lived for

several years in each of them. DC and LA seem to send messages

too, but I haven't spent long enough in either to say for sure what

they are.The big thing in LA seems to be fame. There's an A List of people

who are most in demand right now, and what's most admired is to be

on it, or friends with those who are. Beneath that, the message is

much like New York's, though perhaps with more emphasis on physical

attractiveness.In DC the message seems to be that the most important thing is who

you know. You want to be an insider. In practice this seems to

work much as in LA. There's an A List and you want to be on it or

close to those who are. The only difference is how the A List is

selected. And even that is not that different.At the moment, San Francisco's message seems to be the same as

Berkeley's: you should live better. But this will change if enough

startups choose SF over the Valley. During the Bubble that was a

predictor of failure — a self-indulgent choice, like buying

expensive office furniture. Even now I'm suspicious when startups

choose SF. But if enough good ones do, it stops being a self-indulgent

choice, because the center of gravity of Silicon Valley will shift

there.I haven't found anything like Cambridge for intellectual ambition.

Oxford and Cambridge (England) feel like Ithaca or Hanover: the

message is there, but not as strong.Paris was once a great intellectual center. If you went there in

1300, it might have sent the message Cambridge does now. But I

tried living there for a bit last year, and the ambitions of the

inhabitants are not intellectual ones. The message Paris sends now

is: do things with style. I liked that, actually. Paris is the

only city I've lived in where people genuinely cared about art. In

America only a few rich people buy original art, and even the more

sophisticated ones rarely get past judging it by the brand name of

the artist. But looking through windows at dusk in Paris you can

see that people there actually care what paintings look like.

Visually, Paris has the best eavesdropping I know.

[5]There's one more message I've heard from cities: in London you can

still (barely) hear the message that one should be more aristocratic.

If you listen for it you can also hear it in Paris, New York, and

Boston. But this message is everywhere very faint. It would have

been strong 100 years ago, but now I probably wouldn't have picked

it up at all if I hadn't deliberately tuned in to that wavelength

to see if there was any signal left.\_\_\_\_\_So far the complete list of messages I've picked up from cities is:

wealth, style, hipness, physical attractiveness, fame, political

power, economic power, intelligence, social class, and quality of

life.My immediate reaction to this list is that it makes me slightly

queasy. I'd always considered ambition a good thing, but I realize

now that was because I'd always implicitly understood it to mean

ambition in the areas I cared about. When you list everything

ambitious people are ambitious about, it's not so pretty.On closer examination I see a couple things on the list that are

surprising in the light of history. For example, physical

attractiveness wouldn't have been there 100 years ago (though it

might have been 2400 years ago). It has always mattered for women,

but in the late twentieth century it seems to have started to matter

for men as well. I'm not sure why — probably some combination

of the increasing power of women, the increasing influence of actors

as models, and the fact that so many people work in offices now:

you can't show off by wearing clothes too fancy to wear in a factory,

so you have to show off with your body instead.Hipness is another thing you wouldn't have seen on the list 100

years ago. Or wouldn't you? What it means is to know what's what.

So maybe it has simply replaced the component of social class that

consisted of being "au fait." That could explain why hipness seems

particularly admired in London: it's version 2 of the traditional

English delight in obscure codes that only insiders understand.Economic power would have been on the list 100 years ago, but what

we mean by it is changing. It used to mean the control of vast

human and material resources. But increasingly it means the ability

to direct the course of technology, and some of the people in a

position to do that are not even rich — leaders of important

open source projects, for example. The Captains of Industry of

times past had laboratories full of clever people cooking up new

technologies for them. The new breed are themselves those people.As this force gets more attention, another is dropping off the list:

social class. I think the two changes are related. Economic power,

wealth, and social class are just names for the same thing at

different stages in its life: economic power converts to wealth,

and wealth to social class. So the focus of admiration is simply

shifting upstream.\_\_\_\_\_Does anyone who wants to do great work have to live in a great city?

No; all great cities inspire some sort of ambition, but they aren't

the only places that do. For some kinds of work, all you need is

a handful of talented colleagues.What cities provide is an audience, and a funnel for peers. These

aren't so critical in something like math or physics, where no

audience matters except your peers, and judging ability is sufficiently

straightforward that hiring and admissions committees can do it

reliably. In a field like math or physics all you need is a

department with the right colleagues in it. It could be anywhere — in

Los Alamos, New Mexico, for example.It's in fields like the arts or writing or technology that the

larger environment matters. In these the best practitioners aren't

conveniently collected in a few top university departments and

research labs — partly because talent is harder to judge, and

partly because people pay for these things, so one doesn't need to

rely on teaching or research funding to support oneself. It's in

these more chaotic fields that it helps most to be in a great city:

you need the encouragement of feeling that people around you care

about the kind of work you do, and since you have to find peers for

yourself, you need the much larger intake mechanism of a great city.You don't have to live in a great city your whole life to benefit

from it. The critical years seem to be the early and middle ones

of your career. Clearly you don't have to grow up in a great city.

Nor does it seem to matter if you go to college in one. To most

college students a world of a few thousand people seems big enough.

Plus in college you don't yet have to face the hardest kind of

work — discovering new problems to solve.It's when you move on to the next and much harder step that it helps

most to be in a place where you can find peers and encouragement.

You seem to be able to leave, if you want, once you've found both.

The Impressionists show the typical pattern: they were born all

over France (Pissarro was born in the Carribbean) and died all over

France, but what defined them were the years they spent together

in Paris.\_\_\_\_\_Unless you're sure what you want to do and where the leading center

for it is, your best bet is probably to try living in several

places when you're young. You can never tell what message a city

sends till you live there, or even whether it still sends one.

Often your information will be wrong: I tried living in Florence

when I was 25, thinking it would be an art center, but it turned

out I was 450 years too late.Even when a city is still a live center of ambition, you won't know

for sure whether its message will resonate with you till you hear

it. When I moved to New York, I was very excited at first. It's

an exciting place. So it took me quite a while to realize I just

wasn't like the people there. I kept searching for the Cambridge

of New York. It turned out it was way, way uptown: an hour uptown

by air.Some people know at 16 what sort of work they're going to do, but

in most ambitious kids, ambition seems to precede anything specific

to be ambitious about. They know they want to do something great.

They just haven't decided yet whether they're going to be a rock

star or a brain surgeon. There's nothing wrong with that. But it

means if you have this most common type of ambition, you'll probably

have to figure out where to live by trial and error. You'll

probably have to find the city where you feel at home to know what sort of

ambition you have.Notes[1]

This is one of the advantages of not having the universities

in your country controlled by the government. When governments

decide how to allocate resources, political deal-making causes

things to be spread out geographically. No central goverment would

put its two best universities in the same town, unless it was the

capital (which would cause other problems). But scholars seem to

like to cluster together as much as people in any other field, and

when given the freedom to they derive the same advantages from it.[2]

There are still a few old professors in Palo Alto, but one by

one they die and their houses are transformed by developers into

McMansions and sold to VPs of Bus Dev.[3]

How many times have you read about startup founders who continued

to live inexpensively as their companies took off? Who continued

to dress in jeans and t-shirts, to drive the old car they had in

grad school, and so on? If you did that in New York, people would

treat you like shit. If you walk into a fancy restaurant in San

Francisco wearing a jeans and a t-shirt, they're nice to you; who

knows who you might be? Not in New York.One sign of a city's potential as a technology center is the number

of restaurants that still require jackets for men. According to

Zagat's there are none in San Francisco, LA, Boston, or Seattle,

4 in DC, 6 in Chicago, 8 in London, 13 in New York, and 20 in Paris.(Zagat's lists the Ritz Carlton Dining Room in SF as requiring jackets

but I couldn't believe it, so I called to check and in fact they

don't. Apparently there's only one restaurant left on the entire West

Coast that still requires jackets: The French Laundry in Napa Valley.)[4]

Ideas are one step upstream from economic power, so it's

conceivable that intellectual centers like Cambridge will one day

have an edge over Silicon Valley like the one the Valley has over

New York.This seems unlikely at the moment; if anything Boston is falling

further and further behind. The only reason I even mention the

possibility is that the path from ideas to startups has recently

been getting smoother. It's a lot easier now for a couple of hackers

with no business experience to start a startup than it was 10 years

ago. If you extrapolate another 20 years, maybe the balance of

power will start to shift back. I wouldn't bet on it, but I wouldn't

bet against it either.[5]

If Paris is where people care most about art, why is New York

the center of gravity of the art business? Because in the twentieth

century, art as brand split apart from art as stuff. New York is

where the richest buyers are, but all they demand from art is brand,

and since you can base brand on anything with a sufficiently

identifiable style, you may as well use the local stuff.Thanks to Trevor Blackwell, Sarah Harlin, Jessica Livingston,

Jackie McDonough, Robert Morris, and David Sloo for reading drafts

of this.Italian TranslationPortuguese TranslationChinese TranslationKorean Translation

Disconnecting Distraction

Note: The strategy described at the end of this essay didn't work.

It would work for a while, and then I'd gradually find myself

using the Internet on my work computer. I'm trying other

strategies now, but I think this time I'll wait till I'm sure

they work before writing about them.May 2008Procrastination feeds on distractions. Most people find it

uncomfortable just to sit and do nothing; you avoid work by doing

something else.So one way to beat procrastination is to starve it of distractions.

But that's not as straightforward as it sounds, because there are

people working hard to distract you. Distraction is not a static

obstacle that you avoid like you might avoid a rock in the road.

Distraction seeks you out.Chesterfield described dirt as matter out of place. Distracting

is, similarly, desirable at the wrong time. And technology is

continually being refined to produce more and more desirable things.

Which means that as we learn to avoid one class of distractions,

new ones constantly appear, like drug-resistant bacteria.Television, for example, has after 50 years of refinement reached

the point where it's like visual crack. I realized when I was 13

that TV was addictive, so I stopped watching it. But I read recently

that the average American watches

4 hours

of TV a day. A quarter

of their life.TV is in decline now, but only because people have found even more

addictive ways of wasting time. And what's especially dangerous

is that many happen at your computer. This is no accident. An

ever larger percentage of office workers sit in front of computers

connected to the Internet, and distractions always evolve toward

the procrastinators.I remember when computers were, for me at least, exclusively for

work. I might occasionally dial up a server to get mail or ftp

files, but most of the time I was offline. All I could do was write

and program. Now I feel as if someone snuck a television onto my

desk. Terribly addictive things are just a click away. Run into

an obstacle in what you're working on? Hmm, I wonder what's new

online. Better check.After years of carefully avoiding classic time sinks like TV, games,

and Usenet, I still managed to fall prey to distraction, because

I didn't realize that it evolves. Something that used to be safe,

using the Internet, gradually became more and more dangerous. Some

days I'd wake up, get a cup of tea and check the news, then check

email, then check the news again, then answer a few emails, then

suddenly notice it was almost lunchtime and I hadn't gotten any real

work done. And this started to happen more and more often.It took me surprisingly long to realize how distracting the Internet

had become, because the problem was intermittent. I ignored it the

way you let yourself ignore a bug that only appears intermittently. When

I was in the middle of a project, distractions weren't really a

problem. It was when I'd finished one project and was deciding

what to do next that they always bit me.Another reason it was hard to notice the danger of this new type

of distraction was that social customs hadn't yet caught up with

it. If I'd spent a whole morning sitting on a sofa watching TV,

I'd have noticed very quickly. That's a known danger sign, like

drinking alone. But using the Internet still looked and felt a

lot like work.Eventually, though, it became clear that the Internet had become so much

more distracting that I had to start treating it differently.

Basically, I had to add a new application to my list of known time

sinks: Firefox.\* \* \*The problem is a hard one to solve because most people still need

the Internet for some things. If you drink too much, you can solve

that problem by stopping entirely. But you can't solve the problem

of overeating by stopping eating. I couldn't simply avoid the

Internet entirely, as I'd done with previous time sinks.At first I tried rules. For example, I'd tell myself I was only

going to use the Internet twice a day. But these schemes never

worked for long. Eventually something would come up that required

me to use it more than that. And then I'd gradually slip back

into my old ways.Addictive things have to be treated as if they were sentient

adversaries—as if there were a little man in your head always

cooking up the most plausible arguments for doing whatever you're

trying to stop doing. If you leave a path to it, he'll find it.The key seems to be visibility. The biggest ingredient in most bad habits

is denial. So you have to make it so that you can't merely slip

into doing the thing you're trying to avoid. It has to set off

alarms.Maybe in the long term the right answer for dealing with Internet

distractions will be

software that watches and controls them. But

in the meantime I've found a more drastic solution that definitely

works: to set up a separate computer for using the Internet.I now leave wifi turned off on my main computer except when I need

to transfer a file or edit a web page, and I have a separate laptop

on the other side

of the room that I use to check mail or browse the web. (Irony of

ironies, it's the computer Steve Huffman wrote Reddit on. When

Steve and Alexis auctioned off their old laptops for charity, I

bought them for the Y Combinator museum.)My rule is that I can spend as much time online as I want, as long

as I do it on that computer. And this turns out to be enough. When

I have to sit on the other side of the room to check email or browse

the web, I become much more aware of it. Sufficiently aware, in

my case at least, that it's hard to spend more than about an hour

a day online.And my main computer is now freed for work. If you try this trick,

you'll probably be struck by how different it feels when your

computer is disconnected from the Internet. It was alarming to me

how foreign it felt to sit in front of a computer that could

only be used for work, because that showed how much time I must

have been wasting.Wow. All I can do at this computer is work. Ok, I better work

then.That's the good part. Your old bad habits now help you to work.

You're used to sitting in front of that computer for hours at a

time. But you can't browse the web or check email now. What are

you going to do? You can't just sit there. So you start working.Good and Bad ProcrastinationSpanish TranslationArabic TranslationCatalan TranslationRussian TranslationSpanish Translation

Lies We Tell Kids

May 2008Adults lie constantly to kids. I'm not saying we should stop, but

I think we should at least examine which lies we tell and why.There may also be a benefit to us. We were all lied to as kids,

and some of the lies we were told still affect us. So by studying

the ways adults lie to kids, we may be able to clear our heads of

lies we were told.I'm using the word "lie" in a very general sense: not just overt

falsehoods, but also all the more subtle ways we mislead kids.

Though "lie" has negative connotations, I don't mean to suggest we

should never do this—just that we should pay attention when

we do.

[1]One of the most remarkable things about the way we lie to kids is

how broad the conspiracy is. All adults know what their culture

lies to kids about: they're the questions you answer "Ask

your parents." If a kid asked who won the World Series in 1982

or what the atomic weight of carbon was, you could just tell him.

But if a kid asks you "Is there a God?" or "What's a prostitute?"

you'll probably say "Ask your parents."Since we all agree, kids see few cracks in the view of the world

presented to them. The biggest disagreements are between parents

and schools, but even those are small. Schools are careful what

they say about controversial topics, and if they do contradict what

parents want their kids to believe, parents either pressure the

school into keeping

quiet or move their kids to a new school.The conspiracy is so thorough that most kids who discover it do so

only by discovering internal contradictions in what they're told.

It can be traumatic for the ones who wake up during the operation.

Here's what happened to Einstein:

Through the reading of popular scientific books I soon reached

the conviction that much in the stories of the Bible could not

be true. The consequence was a positively fanatic freethinking

coupled with the impression that youth is intentionally being

deceived by the state through lies: it was a crushing impression.

[2]

I remember that feeling. By 15 I was convinced the world was corrupt

from end to end. That's why movies like The Matrix have such

resonance. Every kid grows up in a fake world. In a way it would

be easier if the forces behind it were as clearly differentiated

as a bunch of evil machines, and one could make a clean break just by

taking a pill.

ProtectionIf you ask adults why they lie to kids, the most common reason they

give is to protect them. And kids do need protecting. The environment

you want to create for a newborn child will be quite unlike the

streets of a big city.That seems so obvious it seems wrong to call it a lie. It's certainly

not a bad lie to tell, to give a baby the impression the world is

quiet and warm and safe. But this harmless type of lie can turn

sour if left unexamined.Imagine if you tried to keep someone in as protected an environment

as a newborn till age 18. To mislead someone so grossly about the

world would seem not protection but abuse. That's an extreme

example, of course; when parents do that sort of thing it becomes

national news. But you see the same problem on a smaller scale in

the malaise teenagers feel in suburbia.The main purpose of suburbia is to provide a protected environment

for children to grow up in. And it seems great for 10 year olds.

I liked living in suburbia when I was 10. I didn't notice how

sterile it was. My whole world was no bigger than a few friends'

houses I bicycled to and some woods I ran around in. On a log scale

I was midway between crib and globe. A suburban street was just

the right size. But as I grew older, suburbia started to feel

suffocatingly fake.Life can be pretty good at 10 or 20, but it's often frustrating at

15. This is too big a problem to solve here, but certainly one

reason life sucks at 15 is that kids are trapped in a world designed

for 10 year olds.What do parents hope to protect their children from by raising them

in suburbia? A friend who moved out of Manhattan said merely that

her 3 year old daughter "saw too much." Off the top of my head,

that might include: people who are high or drunk, poverty, madness,

gruesome medical conditions, sexual behavior of various degrees of

oddness, and violent anger.I think it's the anger that would worry me most if I had a 3 year

old. I was 29 when I moved to New York and I was surprised even

then. I wouldn't want a 3 year old to see some of the disputes I

saw. It would be too frightening. A lot of the things adults

conceal from smaller children, they conceal because they'd be

frightening, not because they want to conceal the existence of such

things. Misleading the child is just a byproduct.This seems one of the most justifiable types of lying adults do to

kids. But because the lies are indirect we don't keep a very strict

accounting of them. Parents know they've concealed the facts about

sex, and many at some point sit their kids down and explain more.

But few tell their kids about the differences between the real world

and the cocoon they grew up in. Combine this with the confidence

parents try to instill in their kids, and every year you get a new

crop of 18 year olds who think they know how to run the world.Don't all 18 year olds think they know how to run the world? Actually

this seems to be a recent innovation, no more than about 100 years old.

In preindustrial times teenage kids were junior members of the adult

world and comparatively well aware of their shortcomings. They

could see they weren't as strong or skillful as the village smith.

In past times people lied to kids about some things more than we

do now, but the lies implicit in an artificial, protected environment

are a recent invention. Like a lot of new inventions, the rich got

this first. Children of kings and great magnates were the first

to grow up out of touch with the world. Suburbia means half the

population can live like kings in that respect.

Sex (and Drugs)I'd have different worries about raising teenage kids in New York.

I'd worry less about what they'd see, and more about what they'd

do. I went to college with a lot of kids who grew up in Manhattan,

and as a rule they seemed pretty jaded. They seemed to have lost

their virginity at an average of about 14 and by college had tried

more drugs than I'd even heard of.The reasons parents don't want their teenage kids having sex are

complex. There are some obvious dangers: pregnancy and sexually

transmitted diseases. But those aren't the only reasons parents

don't want their kids having sex. The average parents of a 14 year

old girl would hate the idea of her having sex even if there were

zero risk of pregnancy or sexually transmitted diseases.Kids can probably sense they aren't being told the whole story.

After all, pregnancy and sexually transmitted diseases are just as

much a problem for adults, and they have sex.What really bothers parents about their teenage kids having sex?

Their dislike of the idea is so visceral it's probably inborn. But

if it's inborn it should be universal, and there are plenty of

societies where parents don't mind if their teenage kids have

sex—indeed, where it's normal for 14 year olds to become

mothers. So what's going on? There does seem to be a universal

taboo against sex with prepubescent children. One can imagine

evolutionary reasons for that. And I think this is the main reason

parents in industrialized societies dislike teenage kids having

sex. They still think of them as children, even though biologically

they're not, so the taboo against child sex still has force.One thing adults conceal about sex they also conceal about drugs:

that it can cause great pleasure. That's what makes sex and drugs

so dangerous. The desire for them can cloud one's judgement—which

is especially frightening when the judgement being clouded is the

already wretched judgement of a teenage kid.Here parents' desires conflict. Older societies told kids they had

bad judgement, but modern parents want their children to be confident.

This may well be a better plan than the old one of putting them in

their place, but it has the side effect that after having implicitly

lied to kids about how good their judgement is, we then have to lie

again about all the things they might get into trouble with if they

believed us.If parents told their kids the truth about sex and drugs, it would

be: the reason you should avoid these things is that you have lousy

judgement. People with twice your experience still get burned by

them. But this may be one of those cases where the truth wouldn't

be convincing, because one of the symptoms of bad judgement is

believing you have good judgement. When you're too weak to lift

something, you can tell, but when you're making a decision impetuously,

you're all the more sure of it.

InnocenceAnother reason parents don't want their kids having sex is that

they want to keep them innocent. Adults have a certain model of

how kids are supposed to behave, and it's different from what they

expect of other adults.One of the most obvious differences is the words kids are allowed

to use. Most parents use words when talking to other adults that

they wouldn't want their kids using. They try to hide even the

existence of these words for as long as they can. And this is

another of those conspiracies everyone participates in: everyone

knows you're not supposed to swear in front of kids.I've never heard more different explanations for anything parents

tell kids than why they shouldn't swear. Every parent I know forbids

their children to swear, and yet no two of them have the same

justification. It's clear most start with not wanting kids to

swear, then make up the reason afterward.So my theory about what's going on is that the function of

swearwords is to mark the speaker as an adult. There's no difference

in the meaning of "shit" and "poopoo." So why should one be ok for

kids to say and one forbidden? The only explanation is: by definition.

[3]Why does it bother adults so much when kids do things reserved for

adults? The idea of a foul-mouthed, cynical 10 year old leaning

against a lamppost with a cigarette hanging out of the corner of

his mouth is very disconcerting. But why?One reason we want kids to be innocent is that we're programmed to

like certain kinds of helplessness. I've several times heard mothers

say they deliberately refrained from correcting their young children's

mispronunciations because they were so cute. And if you think about

it, cuteness is helplessness. Toys and cartoon characters meant to

be cute always have clueless expressions and stubby, ineffectual

limbs.It's not surprising we'd have an inborn desire to love and protect

helpless creatures, considering human offspring are so helpless for

so long. Without the helplessness that makes kids cute, they'd be

very annoying. They'd merely seem like incompetent adults. But

there's more to it than that. The reason our hypothetical jaded

10 year old bothers me so much is not just that he'd be annoying,

but that he'd have cut off his prospects for growth so early. To

be jaded you have to think you know how the world works, and any

theory a 10 year old had about that would probably be a pretty

narrow one.Innocence is also open-mindedness. We want kids to be innocent so

they can continue to learn. Paradoxical as it sounds, there are

some kinds of knowledge that get in the way of other kinds of

knowledge. If you're going to learn that the world is a brutal

place full of people trying to take advantage of one another, you're

better off learning it last. Otherwise you won't bother learning

much more.Very smart adults often seem unusually innocent, and I don't think

this is a coincidence. I think they've deliberately avoided learning

about certain things. Certainly I do. I used to think I wanted

to know everything. Now I know I don't.

DeathAfter sex, death is the topic adults lie most conspicuously about

to kids. Sex I believe they conceal because of deep taboos. But

why do we conceal death from kids? Probably because small children

are particularly horrified by it. They want to feel safe, and death

is the ultimate threat.One of the most spectacular lies our parents told us was about the

death of our first cat. Over the years, as we asked for more

details, they were compelled to invent more, so the story grew quite

elaborate. The cat had died at the vet's office. Of what? Of the

anaesthesia itself. Why was the cat at the vet's office? To be

fixed. And why had such a routine operation killed it? It wasn't

the vet's fault; the cat had a congenitally weak heart; the anaesthesia

was too much for it; but there was no way anyone could have

known this in advance. It was not till we were in our twenties

that the truth came out: my sister, then about three, had accidentally

stepped on the cat and broken its back.They didn't feel the need to tell us the cat was now happily in cat

heaven. My parents never claimed that people or animals who died

had "gone to a better place," or that we'd meet them again. It

didn't seem to harm us.My grandmother told us an edited version of the death of my

grandfather. She said they'd been sitting reading one day, and

when she said something to him, he didn't answer. He seemed to be

asleep, but when she tried to rouse him, she couldn't. "He was

gone." Having a heart attack sounded like falling asleep. Later I

learned it hadn't been so neat, and the heart attack had taken most

of a day to kill him.Along with such outright lies, there must have been a lot of changing

the subject when death came up. I can't remember that, of course,

but I can infer it from the fact that I didn't really grasp I was

going to die till I was about 19. How could I have missed something

so obvious for so long? Now that I've seen parents managing the

subject, I can see how: questions about death are gently but firmly

turned aside.On this topic, especially, they're met half-way by kids. Kids often

want to be lied to. They want to believe they're living in a

comfortable, safe world as much as their parents want them to believe

it.

[4]

IdentitySome parents feel a strong adherence to an ethnic or religious group

and want their kids to feel it too. This usually requires two

different kinds of lying: the first is to tell the child that he

or she is an X, and the second is whatever specific lies Xes

differentiate themselves by believing.

[5]Telling a child they have a particular ethnic or religious identity

is one of the stickiest things you can tell them. Almost anything

else you tell a kid, they can change their mind about later when

they start to think for themselves. But if you tell a kid they're

a member of a certain group, that seems nearly impossible to shake.This despite the fact that it can be one of the most premeditated

lies parents tell. When parents are of different religions, they'll

often agree between themselves that their children will be "raised

as Xes." And it works. The kids obligingly grow up considering

themselves as Xes, despite the fact that if their parents had chosen

the other way, they'd have grown up considering themselves as Ys.One reason this works so well is the second kind of lie involved.

The truth is common property. You can't distinguish your group by

doing things that are rational, and believing things that are true.

If you want to set yourself apart from other people, you have to

do things that are arbitrary, and believe things that are false.

And after having spent their whole lives doing things that are arbitrary

and believing things that are false, and being regarded as odd by

"outsiders" on that account, the cognitive dissonance pushing

children to regard themselves as Xes must be enormous. If they

aren't an X, why are they attached to all these arbitrary beliefs

and customs? If they aren't an X, why do all the non-Xes call them

one?This form of lie is not without its uses. You can use it to carry

a payload of beneficial beliefs, and they will also become part of

the child's identity. You can tell the child that in addition to

never wearing the color yellow, believing the world was created by

a giant rabbit, and always snapping their fingers before eating

fish, Xes are also particularly honest and industrious. Then X

children will grow up feeling it's part of their identity to be

honest and industrious.This probably accounts for a lot of the spread of modern religions,

and explains why their doctrines are a combination of the useful

and the bizarre. The bizarre half is what makes the religion stick,

and the useful half is the payload.

[6]

AuthorityOne of the least excusable reasons adults lie to kids is to maintain

power over them. Sometimes these lies are truly sinister, like a

child molester telling his victims they'll get in trouble if they

tell anyone what happened to them. Others seem more innocent; it

depends how badly adults lie to maintain their power, and what they

use it for.Most adults make some effort to conceal their flaws from children.

Usually their motives are mixed. For example, a father who has an

affair generally conceals it from his children. His motive is

partly that it would worry them, partly that this would introduce

the topic of sex, and partly (a larger part than he would admit)

that he doesn't want to tarnish himself in their eyes.If you want to learn what lies are told to kids, read almost any

book written to teach them about "issues."

[7]

Peter Mayle wrote

one called Why Are We Getting a Divorce? It begins with the three

most important things to remember about divorce, one of which is:

You shouldn't put the blame on one parent, because divorce is

never only one person's fault.

[8]

Really? When a man runs off with his secretary, is it always partly

his wife's fault? But I can see why Mayle might have said this.

Maybe it's more important for kids to respect their parents than

to know the truth about them.But because adults conceal their flaws, and at the same time insist

on high standards of behavior for kids, a lot of kids grow up feeling

they fall hopelessly short. They walk around feeling horribly evil

for having used a swearword, while in fact most of the adults around

them are doing much worse things.This happens in intellectual as well as moral questions. The more

confident people are, the more willing they seem to be to answer a

question "I don't know." Less confident people feel they have to

have an answer or they'll look bad. My parents were pretty good

about admitting when they didn't know things, but I must have been

told a lot of lies of this type by teachers, because I rarely heard

a teacher say "I don't know" till I got to college. I remember

because it was so surprising to hear someone say that in front of

a class.The first hint I had that teachers weren't omniscient came in sixth

grade, after my father contradicted something I'd learned in school.

When I protested that the teacher had said the opposite, my father

replied that the guy had no idea what he was talking about—that

he was just an elementary school teacher, after all.Just a teacher? The phrase seemed almost grammatically ill-formed.

Didn't teachers know everything about the subjects they taught?

And if not, why were they the ones teaching us?The sad fact is, US public school teachers don't generally understand

the stuff they're teaching very well. There are some sterling

exceptions, but as a rule people planning to go into teaching rank

academically near the bottom of the college population. So the

fact that I still thought at age 11 that teachers were infallible

shows what a job the system must have done on my brain.

SchoolWhat kids get taught in school is a complex mix of lies. The most

excusable are those told to simplify ideas to make them easy to

learn. The problem is, a lot of propaganda gets slipped into the

curriculum in the name of simplification.Public school textbooks represent a compromise between what various

powerful groups want kids to be told. The lies are rarely overt.

Usually they consist either of omissions or of over-emphasizing

certain topics at the expense of others. The view of history we

got in elementary school was a crude hagiography, with at least one

representative of each powerful group.The famous scientists I remember were Einstein, Marie Curie, and

George Washington Carver. Einstein was a big deal because his

work led to the atom bomb. Marie Curie was involved with X-rays.

But I was mystified about Carver. He seemed to have done stuff

with peanuts.It's obvious now that he was on the list because he was black (and

for that matter that Marie Curie was on it because she was a woman),

but as a kid I was confused for years about him. I wonder if it

wouldn't have been better just to tell us the truth: that there

weren't any famous black scientists. Ranking George Washington

Carver with Einstein misled us not only about science, but about

the obstacles blacks faced in his time.As subjects got softer, the lies got more frequent. By the time

you got to politics and recent history, what we were taught was

pretty much pure propaganda. For example, we were taught to regard

political leaders as saints—especially the recently martyred

Kennedy and King. It was astonishing to learn later that they'd

both been serial womanizers, and that Kennedy was a speed freak to

boot. (By the time King's plagiarism emerged, I'd lost the ability

to be surprised by the misdeeds of famous people.)I doubt you could teach kids recent history without teaching them

lies, because practically everyone who has anything to say about

it has some kind of spin to put on it. Much recent history consists

of spin. It would probably be better just to teach them metafacts

like that.Probably the biggest lie told in schools, though, is that the way

to succeed is through following "the rules." In fact most such

rules are just hacks to manage large groups efficiently.

PeaceOf all the reasons we lie to kids, the most powerful is probably

the same mundane reason they lie to us.Often when we lie to people it's not part of any conscious strategy,

but because they'd react violently to the truth. Kids, almost by

definition, lack self-control. They react violently to things—and

so they get lied to a lot.

[9]A few Thanksgivings ago, a friend of mine found himself in a situation

that perfectly illustrates the complex motives we have when we lie

to kids. As the roast turkey appeared on the table, his alarmingly

perceptive 5 year old son suddenly asked if the turkey had wanted

to die. Foreseeing disaster, my friend and his wife rapidly

improvised: yes, the turkey had wanted to die, and in fact had lived

its whole life with the aim of being their Thanksgiving dinner.

And that (phew) was the end of that.Whenever we lie to kids to protect them, we're usually also lying

to keep the peace.One consequence of this sort of calming lie is that we grow up

thinking horrible things are normal. It's hard for us to feel a

sense of urgency as adults over something we've literally been

trained not to worry about. When I was about 10 I saw a documentary

on pollution that put me into a panic. It seemed the planet was

being irretrievably ruined. I went to my mother afterward to ask

if this was so. I don't remember what she said, but she made me

feel better, so I stopped worrying about it.That was probably the best way to handle a frightened 10 year old.

But we should understand the price. This sort of lie is one of the

main reasons bad things persist: we're all trained to ignore them.

DetoxA sprinter in a race almost immediately enters a state called "oxygen

debt." His body switches to an emergency source of energy that's

faster than regular aerobic respiration. But this process builds

up waste products that ultimately require extra oxygen to break

down, so at the end of the race he has to stop and pant for a while

to recover.We arrive at adulthood with a kind of truth debt. We were told a

lot of lies to get us (and our parents) through our childhood. Some

may have been necessary. Some probably weren't. But we all arrive

at adulthood with heads full of lies.There's never a point where the adults sit you down and explain all

the lies they told you. They've forgotten most of them. So if

you're going to clear these lies out of your head, you're going to

have to do it yourself.Few do. Most people go through life with bits of packing material

adhering to their minds and never know it. You probably never can

completely undo the effects of lies you were told as a kid, but

it's worth trying. I've found that whenever I've been able to undo

a lie I was told, a lot of other things fell into place.Fortunately, once you arrive at adulthood you get a valuable new

resource you can use to figure out what lies you were told. You're

now one of the liars. You get to watch behind the scenes as adults

spin the world for the next generation of kids.The first step in clearing your head is to realize how far you are

from a neutral observer. When I left high school I was, I thought,

a complete skeptic. I'd realized high school was crap. I thought

I was ready to question everything I knew. But among the many other

things I was ignorant of was how much debris there already was in

my head. It's not enough to consider your mind a blank slate. You

have to consciously erase it.

Notes[1]

One reason I stuck with such a brutally simple word is that

the lies we tell kids are probably not quite as harmless as we

think. If you look at what adults told children in the past, it's

shocking how much they lied to them. Like us, they did it with the

best intentions. So if we think we're as open as one could reasonably

be with children, we're probably fooling ourselves. Odds are people

in 100 years will be as shocked at some of the lies we tell as we

are at some of the lies people told 100 years ago.I can't predict which these will be, and I don't want to write an

essay that will seem dumb in 100 years. So instead of using special

euphemisms for lies that seem excusable according to present fashions,

I'm just going to call all our lies lies.(I have omitted one type: lies told to play games with kids'

credulity. These range from "make-believe," which is not really a

lie because it's told with a wink, to the frightening lies told by

older siblings. There's not much to say about these: I wouldn't

want the first type to go away, and wouldn't expect the second type

to.)[2]

Calaprice, Alice (ed.), The Quotable Einstein, Princeton

University Press, 1996.[3]

If you ask parents why kids shouldn't swear, the less educated

ones usually reply with some question-begging answer like "it's

inappropriate," while the more educated ones come up with elaborate

rationalizations. In fact the less educated parents seem closer

to the truth.[4]

As a friend with small children pointed out, it's easy for small

children to consider themselves immortal, because time seems to

pass so slowly for them. To a 3 year old, a day feels like a month

might to an adult. So 80 years sounds to him like 2400 years would

to us.[5]

I realize I'm going to get endless grief for classifying religion

as a type of lie. Usually people skirt that issue with some

equivocation implying that lies believed for a sufficiently long

time by sufficiently large numbers of people are immune to the usual

standards for truth. But because I can't predict which lies future

generations will consider inexcusable, I can't safely omit any type

we tell. Yes, it seems unlikely that religion will be out of fashion

in 100 years, but no more unlikely than it would have seemed to

someone in 1880 that schoolchildren in 1980 would be taught that

masturbation was perfectly normal and not to feel guilty about it.[6]

Unfortunately the payload can consist of bad customs as well

as good ones. For example, there are certain qualities that some

groups in America consider "acting white." In fact most of them

could as accurately be called "acting Japanese." There's nothing

specifically white about such customs. They're common to all cultures

with long traditions of living in cities. So it is probably a

losing bet for a group to consider behaving the opposite way as

part of its identity.[7]

In this context, "issues" basically means "things we're going

to lie to them about." That's why there's a special name for these

topics.[8]

Mayle, Peter, Why Are We Getting a Divorce?, Harmony, 1988.[9]

The ironic thing is, this is also the main reason kids lie to

adults. If you freak out when people tell you alarming things,

they won't tell you them. Teenagers don't tell their parents what

happened that night they were supposed to be staying at a friend's

house for the same reason parents don't tell 5 year olds the truth

about the Thanksgiving turkey. They'd freak if they knew.

Thanks to Sam Altman, Marc Andreessen, Trevor Blackwell,

Patrick Collison, Jessica Livingston, Jackie McDonough, Robert

Morris, and David Sloo for reading drafts of this. And since there

are some controversial ideas here, I should add that none of them

agreed with everything in it.German TranslationFrench TranslationRussian Translation

Be Good

April 2008(This essay is derived from a talk at the 2008 Startup School.)About a month after we started Y Combinator we came up with the

phrase that became our motto: Make something people want. We've

learned a lot since then, but if I were choosing now that's still

the one I'd pick.Another thing we tell founders is not to worry too much about the

business model, at least at first. Not because making money is

unimportant, but because it's so much easier than building something

great.A couple weeks ago I realized that if you put those two ideas

together, you get something surprising. Make something people want.

Don't worry too much about making money. What you've got is a

description of a charity.When you get an unexpected result like this, it could either be a

bug or a new discovery. Either businesses aren't supposed to be

like charities, and we've proven by reductio ad absurdum that one

or both of the principles we began with is false. Or we have a new

idea.I suspect it's the latter, because as soon as this thought occurred

to me, a whole bunch of other things fell into place.ExamplesFor example, Craigslist. It's not a charity, but they run it like

one. And they're astoundingly successful. When you scan down the

list of most popular web sites, the number of employees at Craigslist

looks like a misprint. Their revenues aren't as high as they could

be, but most startups would be happy to trade places with them.In Patrick O'Brian's novels, his captains always try to get upwind

of their opponents. If you're upwind, you decide when and if to

engage the other ship. Craigslist is effectively upwind of enormous

revenues. They'd face some challenges if they wanted to make more,

but not the sort you face when you're tacking upwind, trying to

force a crappy product on ambivalent users by spending ten times

as much on sales as on development. [1]I'm not saying startups should aim to end up like Craigslist.

They're a product of unusual circumstances. But they're a good

model for the early phases.Google looked a lot like a charity in the beginning. They didn't

have ads for over a year. At year 1, Google was indistinguishable

from a nonprofit. If a nonprofit or government organization had

started a project to index the web, Google at year 1 is the limit

of what they'd have produced.Back when I was working on spam filters I thought it would be a

good idea to have a web-based email service with good spam filtering.

I wasn't thinking of it as a company. I just wanted to keep people

from getting spammed. But as I thought more about this project, I

realized it would probably have to be a company. It would cost

something to run, and it would be a pain to fund with grants and

donations.That was a surprising realization. Companies often claim to be

benevolent, but it was surprising to realize there were purely

benevolent projects that had to be embodied as companies to work.I didn't want to start another company, so I didn't do it. But if

someone had, they'd probably be quite rich now. There was a window

of about two years when spam was increasing rapidly but all the big

email services had terrible filters. If someone had launched a

new, spam-free mail service, users would have flocked to it.Notice the pattern here? From either direction we get to the same

spot. If you start from successful startups, you find they often

behaved like nonprofits. And if you start from ideas for nonprofits,

you find they'd often make good startups.PowerHow wide is this territory? Would all good nonprofits be good

companies? Possibly not. What makes Google so valuable is that

their users have money. If you make people with money love you,

you can probably get some of it. But could you also base a successful

startup on behaving like a nonprofit to people who don't have money?

Could you, for example, grow a successful startup out of curing an

unfashionable but deadly disease like malaria?I'm not sure, but I suspect that if you pushed this idea, you'd be

surprised how far it would go. For example, people who apply to Y

Combinator don't generally have much money, and yet we can profit

by helping them, because with our help they could make money. Maybe

the situation is similar with malaria. Maybe an organization that

helped lift its weight off a country could benefit from the resulting

growth.I'm not proposing this is a serious idea. I don't know anything

about malaria. But I've been kicking ideas around long enough to

know when I come across a powerful one.One way to guess how far an idea extends is to ask yourself at what

point you'd bet against it. The thought of betting against benevolence

is alarming in the same way as saying that something is technically

impossible. You're just asking to be made a fool of, because these

are such powerful forces. [2]For example, initially I thought maybe this principle only applied

to Internet startups. Obviously it worked for Google, but what

about Microsoft? Surely Microsoft isn't benevolent? But when I

think back to the beginning, they were. Compared to IBM they were

like Robin Hood. When IBM introduced the PC, they thought they

were going to make money selling hardware at high prices. But by

gaining control of the PC standard, Microsoft opened up the market

to any manufacturer. Hardware prices plummeted, and lots of people

got to have computers who couldn't otherwise have afforded them.

It's the sort of thing you'd expect Google to do.Microsoft isn't so benevolent now. Now when one thinks of what

Microsoft does to users, all the verbs that come to mind begin with

F. [3] And yet it doesn't seem to pay.

Their stock price has been flat for years. Back when they were

Robin Hood, their stock price rose like Google's. Could there be

a connection?You can see how there would be. When you're small, you can't bully

customers, so you have to charm them. Whereas when you're big you

can maltreat them at will, and you tend to, because it's easier

than satisfying them. You grow big by being nice, but you can stay

big by being mean.You get away with it till the underlying conditions change, and

then all your victims escape. So "Don't be evil" may be the most

valuable thing Paul Buchheit made for Google, because it may turn

out to be an elixir of corporate youth. I'm sure they find it

constraining, but think how valuable it will be if it saves them

from lapsing into the fatal laziness that afflicted Microsoft and

IBM.The curious thing is, this elixir is freely available to any other

company. Anyone can adopt "Don't be evil." The catch is that

people will hold you to it. So I don't think you're going to see

record labels or tobacco companies using this discovery.MoraleThere's a lot of external evidence that benevolence works. But how

does it work? One advantage of investing in a large number of

startups is that you get a lot of data about how they work. From

what we've seen, being good seems to help startups in three ways:

it improves their morale, it makes other people want to help them,

and above all, it helps them be decisive.Morale is tremendously important to a startup—so important

that morale alone is almost enough to determine success. Startups

are often described as emotional roller-coasters. One minute you're

going to take over the world, and the next you're doomed. The

problem with feeling you're doomed is not just that it makes you

unhappy, but that it makes you stop working. So the downhills

of the roller-coaster are more of a self fulfilling prophecy than

the uphills. If feeling you're going to succeed makes you work

harder, that probably improves your chances of succeeding, but if

feeling you're going to fail makes you stop working, that practically

guarantees you'll fail.Here's where benevolence comes in. If you feel you're really helping

people, you'll keep working even when it seems like your startup

is doomed. Most of us have some amount of natural benevolence.

The mere fact that someone needs you makes you want to help them.

So if you start the kind of startup where users come back each day,

you've basically built yourself a giant tamagotchi. You've made

something you need to take care of.Blogger is a famous example of a startup that went through really

low lows and survived. At one point they ran out of money and

everyone left. Evan Williams came in to work the next day, and there

was no one but him. What kept him going? Partly that users needed

him. He was hosting thousands of people's blogs. He couldn't just

let the site die.There are many advantages of launching quickly, but the most important

may be that once you have users, the tamagotchi effect kicks in.

Once you have users to take care of, you're forced to figure out

what will make them happy, and that's actually very valuable

information.The added confidence that comes from trying to help people can

also help you with investors. One of the founders of

Chatterous told

me recently that he and his cofounder had decided that this service

was something the world needed, so they were going to keep working

on it no matter what, even if they had to move back to Canada and live

in their parents' basements.Once they realized this, they stopped caring so much what investors thought

about them. They still met with them, but they weren't going to

die if they didn't get their money. And you know what? The investors

got a lot more interested. They could sense that the Chatterouses

were going to do this startup with or without them.If you're really committed and your startup is cheap to run, you

become very hard to kill. And practically all startups, even the

most successful, come close to death at some point. So if doing

good for people gives you a sense of mission that makes you harder

to kill, that alone more than compensates for whatever you lose by

not choosing a more selfish project.HelpAnother advantage of being good is that it makes other people want

to help you. This too seems to be an inborn trait in humans.One of the startups we've funded, Octopart, is currently locked in

a classic battle of good versus evil. They're a search site for

industrial components. A lot of people need to search for components,

and before Octopart there was no good way to do it. That, it turned

out, was no coincidence.Octopart built the right way to search for components. Users like

it and they've been growing rapidly. And yet for most of Octopart's

life, the biggest distributor, Digi-Key, has been trying to force

them take their prices off the site. Octopart is sending them

customers for free, and yet Digi-Key is trying to make that traffic

stop. Why? Because their current business model depends on

overcharging people who have incomplete information about prices.

They don't want search to work.The Octoparts are the nicest guys in the world. They dropped out

of the PhD program in physics at Berkeley to do this. They just

wanted to fix a problem they encountered in their research. Imagine

how much time you could save the world's engineers if they could

do searches online. So when I hear that a big, evil company is

trying to stop them in order to keep search broken, it makes me

really want to help them. It makes me spend more time on the Octoparts

than I do with most of the other startups we've funded. It just

made me spend several minutes telling you how great they are. Why?

Because they're good guys and they're trying to help the world.If you're benevolent, people will rally around you: investors,

customers, other companies, and potential employees. In the long

term the most important may be the potential employees. I think

everyone knows now that

good hackers are much better than mediocre

ones. If you can attract the best hackers to work for you, as

Google has, you have a big advantage. And the very best hackers

tend to be idealistic. They're not desperate for a job. They can

work wherever they want. So most want to work on things that will

make the world better.CompassBut the most important advantage of being good is that it acts as

a compass. One of the hardest parts of doing a startup is that you

have so many choices. There are just two or three of you, and a

thousand things you could do. How do you decide?Here's the answer: Do whatever's best for your users. You can hold

onto this like a rope in a hurricane, and it will save you if

anything can. Follow it and it will take you through everything

you need to do.It's even the answer to questions that seem unrelated, like how to

convince investors to give you money. If you're a good salesman,

you could try to just talk them into it. But the more reliable

route is to convince them through your users: if you make something

users love enough to tell their friends, you grow exponentially,

and that will convince any investor.Being good is a particularly useful strategy for making decisions

in complex situations because it's stateless. It's like telling

the truth. The trouble with lying is that you have to remember

everything you've said in the past to make sure you don't contradict

yourself. If you tell the truth you don't have to remember anything,

and that's a really useful property in domains where things happen

fast.For example, Y Combinator has now invested in 80 startups, 57 of

which are still alive. (The rest have died or merged or been

acquired.) When you're trying to advise 57 startups, it turns out

you have to have a stateless algorithm. You can't have ulterior

motives when you have 57 things going on at once, because you can't

remember them. So our rule is just to do whatever's best for the

founders. Not because we're particularly benevolent, but because

it's the only algorithm that works on that scale.When you write something telling people to be good, you seem to be

claiming to be good yourself. So I want to say explicitly that I

am not a particularly good person. When I was a kid I was firmly

in the camp of bad. The way adults used the word good, it seemed

to be synonymous with quiet, so I grew up very suspicious of it.You know how there are some people whose names come up in conversation

and everyone says "He's such a great guy?" People never say

that about me. The best I get is "he means well." I am not claiming

to be good. At best I speak good as a second language.So I'm not suggesting you be good in the usual sanctimonious way.

I'm suggesting it because it works. It will work not just as a

statement of "values," but as a guide to strategy,

and even a design spec for software. Don't just not be evil. Be

good.Notes[1] Fifty years ago

it would have seemed shocking for a public company not to pay

dividends. Now many tech companies don't. The markets seem to

have figured out how to value potential dividends. Maybe that isn't

the last step in this evolution. Maybe markets will eventually get

comfortable with potential earnings. (VCs already are, and at least

some of them consistently make money.)I realize this sounds like the stuff one used to hear about the

"new economy" during the Bubble. Believe me, I was not drinking

that kool-aid at the time. But I'm convinced there were some

good

ideas buried in Bubble thinking. For example, it's ok to focus on

growth instead of profits—but only if the growth is genuine.

You can't be buying users; that's a pyramid scheme. But a company

with rapid, genuine growth is valuable, and eventually markets learn

how to value valuable things.[2] The idea of starting

a company with benevolent aims is currently undervalued, because

the kind of people who currently make that their explicit goal don't

usually do a very good job.It's one of the standard career paths of trustafarians to start

some vaguely benevolent business. The problem with most of them

is that they either have a bogus political agenda or are feebly

executed. The trustafarians' ancestors didn't get rich by preserving

their traditional culture; maybe people in Bolivia don't want to

either. And starting an organic farm, though it's at least

straightforwardly benevolent, doesn't help people on the scale that

Google does.Most explicitly benevolent projects don't hold themselves sufficiently

accountable. They act as if having good intentions were enough to

guarantee good effects.[3] Users dislike their

new operating system so much that they're starting petitions to

save the old one. And the old one was nothing special. The hackers

within Microsoft must know in their hearts that if the company

really cared about users they'd just advise them to switch to OSX.Thanks to Trevor Blackwell, Paul Buchheit, Jessica Livingston,

and Robert Morris for reading drafts of this.Russian TranslationGerman Translation

Why There Aren't More Googles

Want to start a startup? Get funded by

Y Combinator.

April 2008Umair Haque

wrote recently that the reason there aren't more Googles is

that most startups get bought before they can change the world.

Google, despite serious interest from Microsoft and Yahoo—what

must have seemed like lucrative interest at the time—didn't

sell out. Google might simply have been nothing but Yahoo's or

MSN's search box.Why isn't it? Because Google had a deeply felt sense of purpose:

a conviction to change the world for the better.

This has a nice sound to it, but it isn't true.

Google's founders were willing to sell early on.

They just wanted more than acquirers were willing to pay.It was the same with Facebook. They would have sold, but Yahoo blew it

by offering too little.Tip for acquirers: when a startup turns you down, consider raising

your offer, because there's a good chance the outrageous price they

want will later seem a bargain.

[1]From the evidence I've seen so far,

startups that turn down acquisition offers usually end up doing better.

Not always, but usually there's a bigger offer coming, or

perhaps even an IPO.Of course, the reason startups do better when they turn down

acquisition offers is not necessarily that all such offers undervalue

startups. More likely the reason is that the kind of founders who

have the balls to turn down a big offer also tend to be very

successful. That spirit is exactly what you want in a startup.While I'm sure Larry and Sergey do want to change the world, at

least now, the reason Google survived to become a big, independent

company is the same reason Facebook has so far remained independent:

acquirers underestimated them.Corporate M&A is a strange business in that respect. They consistently

lose the best deals, because turning down reasonable offers is the

most reliable test you could invent for whether a startup will make

it big.VCsSo what's the real reason there aren't more Googles? Curiously

enough, it's the same reason Google and Facebook have remained

independent: money guys undervalue the most innovative startups.The reason there aren't more Googles is not that investors encourage

innovative startups to sell out, but that they won't even fund them.

I've learned a lot about VCs during the 3 years we've been doing Y

Combinator, because we often have to work quite closely with them.

The most surprising thing I've learned is how conservative they

are. VC firms present an image of boldly encouraging innovation.

Only a handful actually do, and even they are more conservative in

reality than you'd guess from reading their sites.I used to think of VCs as piratical: bold but unscrupulous. On

closer acquaintance they turn out to be more like bureaucrats.

They're more upstanding than I used to think (the good ones, at

least), but less bold. Maybe the VC industry has changed. Maybe

they used to be bolder.

But I suspect it's the startup world that has

changed, not them. The low cost of starting a startup means the

average good bet is a riskier one, but most existing VC firms still

operate as if they were investing in hardware startups in 1985.Howard Aiken said "Don't worry about people stealing your ideas.

If your ideas are any good, you'll have to ram them down people's

throats." I have a similar feeling when I'm trying to convince VCs

to invest in startups Y Combinator has funded. They're terrified

of really novel ideas, unless the founders are good enough salesmen

to compensate.But it's the bold ideas that generate the biggest returns. Any

really good new idea will seem bad to most people; otherwise someone

would already be doing it. And

yet most VCs are driven by consensus, not just within their firms,

but within the VC community. The biggest factor determining how a

VC will feel about your startup is how other VCs feel about it. I

doubt they realize it, but this algorithm guarantees they'll miss

all the very best ideas. The more people who have to like a new

idea, the more outliers you lose.Whoever the next Google is, they're probably being told right now

by VCs to come back when they have more "traction."Why are VCs so conservative? It's probably a combination of factors.

The large size of their investments makes them conservative.

Plus they're investing other people's money, which makes

them worry they'll get in trouble if they do something risky and

it fails. Plus most of them are money guys rather than technical

guys, so they don't understand what the startups they're investing

in do.What's NextThe exciting thing about market economies is that stupidity equals

opportunity. And so it is in this case. There is a huge, unexploited

opportunity in startup investing. Y Combinator funds startups at

the very beginning. VCs will fund them once they're already starting

to succeed. But between the two there is a substantial gap.There are companies that will give $20k to a startup that has nothing

more than the founders, and there are companies that will give $2

million to a startup that's already taking off,

but there aren't enough investors who will give $200k to a startup

that seems very promising but still has some things to figure out.

This territory is occupied mostly by

individual angel investors—people like Andy Bechtolsheim, who

gave Google $100k when they seemed promising but still had some

things to figure out. I like angels, but there just aren't enough

of them, and investing is for most of them a part time job.And yet as it gets cheaper to start startups, this sparsely occupied

territory is becoming more and more valuable. Nowadays a lot of

startups don't want to raise multi-million dollar series A rounds.

They don't need that much money, and they don't want the hassles

that come with it. The median startup coming out of Y Combinator

wants to raise $250-500k. When they go to VC firms they have to

ask for more because they know VCs aren't interested in such small

deals.VCs are money managers. They're looking for ways to put large sums

to work. But the startup world is evolving away from their current

model.Startups have gotten cheaper. That means they want less money, but

also that there are more of them. So you can still get large returns

on large amounts of money; you just have to spread it more broadly.I've tried to explain this to VC firms. Instead of making one $2

million investment, make five $400k investments. Would that mean

sitting on too many boards? Don't sit on their boards. Would that

mean too much due diligence? Do less. If you're investing at a

tenth the valuation, you only have to be a tenth as sure.It seems obvious. But I've proposed to several VC firms that they

set aside some money and designate one partner to make more, smaller

bets, and they react as if I'd proposed the partners all get nose

rings. It's remarkable how wedded they are to their standard m.o.But there is a big opportunity here, and one way or the other it's

going to get filled. Either VCs will evolve down into this gap or,

more likely, new investors will appear to fill it. That will be a

good thing when it happens, because these new investors will be

compelled by the structure of the investments they make to be ten

times bolder than present day VCs. And that will get us a lot more

Googles. At least, as long as acquirers remain stupid.

Notes[1]

Another tip: If you want to get all that value, don't destroy the

startup after you buy it. Give the founders enough autonomy that

they can grow the acquisition into what it would have become.Thanks to Sam Altman, Paul Buchheit, David Hornik, Jessica

Livingston, Robert Morris, and Fred Wilson for reading drafts of this.Russian Translation

Some Heroes

April 2008There are some topics I save up because they'll be so much fun to

write about. This is one of them: a list of my heroes.I'm not claiming this is a list of the n most admirable people.

Who could make such a list, even if they wanted to?Einstein isn't on the list, for example, even though he probably

deserves to be on any shortlist of admirable people. I once asked

a physicist friend if Einstein was really as smart as his fame

implies, and she said that yes, he was. So why isn't he on the

list? Because I had to ask. This is a list of people who've

influenced me, not people who would have if I understood their work.My test was to think of someone and ask "is this person my

hero?" It often returned surprising answers. For example,

it returned false for Montaigne, who was arguably the inventor of

the essay. Why? When I thought

about what it meant to call someone a hero, it meant I'd decide what

to do by asking what they'd do in the same situation. That's a

stricter standard than admiration.After I made the list, I looked to see if there was a pattern, and

there was, a very clear one. Everyone on the list had two qualities:

they cared almost excessively about their work, and they were

absolutely honest. By honest I don't mean trustworthy so much as

that they never pander: they never say or do something because

that's what the audience wants. They are all fundamentally subversive

for this reason, though they conceal it to varying degrees.

Jack LambertI grew up in Pittsburgh in the 1970s. Unless you were there it's

hard to imagine how that town felt about the Steelers. Locally,

all the news was bad. The steel industry was dying. But the

Steelers were the best team in football — and moreover, in a

way that seemed to reflect the personality of the city. They didn't

do anything fancy. They just got the job done.Other players were more famous: Terry Bradshaw, Franco Harris, Lynn

Swann. But they played offense, and you always get more attention

for that. It seemed to me as a twelve year old football expert

that the best of them all was

Jack Lambert. And what made him so

good was that he was utterly relentless. He didn't just care about

playing well; he cared almost too much. He seemed to regard it as

a personal insult when someone from the other team had possession

of the ball on his side of the line of scrimmage.The suburbs of Pittsburgh in the 1970s were a pretty dull place.

School was boring. All the adults around were bored with their

jobs working for big companies. Everything that came to us through

the mass media was (a) blandly uniform and (b) produced elsewhere.

Jack Lambert was the exception. He was like nothing else I'd seen.

Kenneth ClarkKenneth Clark is the best nonfiction writer I know of, on any

subject. Most people who write about art history don't really like

art; you can tell from a thousand little signs. But Clark did, and

not just intellectually, but the way one anticipates a delicious

dinner.What really makes him stand out, though, is the quality of his

ideas. His style is deceptively casual, but there is more in

his books than in a library

of art monographs. Reading

The Nude is like a ride in a

Ferrari. Just as you're getting settled, you're slammed back in

your seat by the acceleration. Before you can adjust, you're thrown

sideways as the car screeches into the first turn. His brain throws

off ideas almost too fast to grasp them. Finally at the end of the

chapter you come to a halt, with your eyes wide and a big smile on

your face.Kenneth Clark was a star in his day, thanks to the documentary

series

Civilisation. And if you read only one book about

art history,

Civilisation is the one I'd recommend. It's

much better than the drab Sears Catalogs of art that undergraduates

are forced to buy for Art History 101.

Larry MihalkoA lot of people have a great teacher at some point in their childhood.

Larry Mihalko was mine. When I look back it's like there's a line

drawn between third and fourth grade. After Mr. Mihalko, everything

was different.Why? First of all, he was intellectually curious. I had a few

other teachers who were smart, but I wouldn't describe them as

intellectually curious. In retrospect, he was out of place as an

elementary school teacher, and I think he knew it. That must have

been hard for him, but it was wonderful for us, his students. His

class was a constant adventure. I used to like going to school

every day.The other thing that made him different was that he liked us. Kids

are good at telling that. The other teachers were at best benevolently

indifferent. But Mr. Mihalko seemed like he actually wanted to

be our friend. On the last day of fourth grade, he got out one of

the heavy school record players and played James Taylor's "You've

Got a Friend" to us. Just call out my name, and you know wherever

I am, I'll come running. He died at 59 of lung cancer. I've never

cried like I cried at his funeral.

LeonardoOne of the things I've learned about making things that I didn't

realize when I was a kid is that much of the best stuff isn't made

for audiences, but for oneself. You see paintings and drawings in

museums and imagine they were made for you to look at. Actually a

lot of the best ones were made as a way of exploring the world, not

as a way to please other people. The best of these explorations

are sometimes more pleasing than stuff made explicitly to please.Leonardo did a lot of things. One of his most admirable qualities

was that he did so many different things that were admirable. What

people know of him now is his paintings and his more flamboyant

inventions, like flying machines. That makes him seem like some

kind of dreamer who sketched artists' conceptions of rocket ships

on the side. In fact he made a large number of far more practical

technical discoveries. He was as good an engineer as a painter.His most impressive work, to me, is his

drawings. They're clearly

made more as a way of studying the world than producing something

beautiful. And yet they can hold their own with any work of art

ever made. No one else, before or since, was that good when no one

was looking.

Robert MorrisRobert Morris has a very unusual quality: he's never wrong. It

might seem this would require you to be omniscient, but actually

it's surprisingly easy. Don't say anything unless you're fairly

sure of it. If you're not omniscient, you just don't end up saying

much.More precisely, the trick is to pay careful attention to how you

qualify what you say. By using this trick, Robert has, as far as

I know, managed to be mistaken only once, and that was when he was

an undergrad. When the Mac came out, he said that little desktop

computers would never be suitable for real hacking.It's wrong to call it a trick in his case, though. If it were a

conscious trick, he would have slipped in a moment of excitement.

With Robert this quality is wired-in. He has an almost superhuman

integrity. He's not just generally correct, but also correct about

how correct he is.You'd think it would be such a great thing never to be wrong that

everyone would do this. It doesn't seem like that much extra work

to pay as much attention to the error on an idea as to the idea

itself. And yet practically no one does. I know how hard it is,

because since meeting Robert I've tried to do in software what he

seems to do in hardware.

P. G. WodehousePeople are finally starting to admit that Wodehouse was a great

writer. If you want to be thought a great novelist in your own

time, you have to sound intellectual. If what you write is popular,

or entertaining, or funny, you're ipso facto suspect. That makes

Wodehouse doubly impressive, because it meant that to write as he

wanted to, he had to commit to being despised in his own lifetime.Evelyn Waugh called him a great writer, but to most people at the

time that would have read as a chivalrous or deliberately perverse

gesture. At the time any random autobiographical novel by a recent

college grad could count on more respectful treatment from the

literary establishment.Wodehouse may have begun with simple atoms, but the way he composed

them into molecules was near faultless. His rhythm in particular.

It makes me self-conscious to write about it. I can think of only

two other writers who came near him for style: Evelyn Waugh and

Nancy Mitford. Those three used the English language like they

owned it.But Wodehouse has something neither of them did. He's at ease.

Evelyn Waugh and Nancy Mitford cared what other people thought of

them: he wanted to seem aristocratic; she was afraid she wasn't

smart enough. But Wodehouse didn't give a damn what anyone thought

of him. He wrote exactly what he wanted.

Alexander CalderCalder's on this list because he makes me happy. Can his work stand

up to Leonardo's? Probably not. There might not be anything from

the 20th Century that can. But what was good about Modernism,

Calder had, and had in a way that he made seem effortless.What was good about Modernism was its freshness. Art became stuffy

in the nineteenth century. The paintings that were popular at the

time were mostly the art equivalent of McMansions—big,

pretentious, and fake. Modernism meant starting over, making things

with the same earnest motives that children might. The artists who

benefited most from this were the ones who had preserved a child's

confidence, like Klee and Calder.Klee was impressive because he could work in so many different

styles. But between the two I like Calder better, because his work

seemed happier. Ultimately the point of art is to engage the viewer.

It's hard to predict what will; often something that seems interesting

at first will bore you after a month. Calder's

sculptures never

get boring. They just sit there quietly radiating optimism, like

a battery that never runs out. As far as I can tell from books and

photographs, the happiness of Calder's work is his own happiness

showing through.

Jane AustenEveryone admires Jane Austen. Add my name to the list. To me she

seems the best novelist of all time.I'm interested in how things work. When I read most novels, I pay

as much attention to the author's choices as to the story. But in

her novels I can't see the gears at work. Though I'd really like

to know how she does what she does, I can't figure it out, because

she's so good that her stories don't seem made up. I feel like I'm

reading a description of something that actually happened.I used to read a lot of novels when I was younger. I can't read

most anymore, because they don't have enough information in them.

Novels seem so impoverished compared to history and biography. But

reading Austen is like reading

nonfiction. She writes so well you don't even notice her.

John McCarthyJohn McCarthy invented Lisp, the field of (or at least the term)

artificial intelligence, and was an early member of both of the top

two computer science departments, MIT and Stanford. No one would

dispute that he's one of the greats, but he's an especial hero to

me because of

Lisp.It's hard for us now to understand what a conceptual leap that was

at the time. Paradoxically, one of the reasons his achievement is

hard to appreciate is that it was so successful. Practically every

programming language invented in the last 20 years includes ideas

from Lisp, and each year the median language gets more Lisplike.In 1958 these ideas were anything but obvious. In 1958 there seem

to have been two ways of thinking about programming. Some people

thought of it as math, and proved things about Turing Machines.

Others thought of it as a way to get things done, and designed

languages all too influenced by the technology of the day. McCarthy

alone bridged the gap. He designed a language that was math. But

designed is not really the word; discovered is more like it.

The SpitfireAs I was making this list I found myself thinking of people like

Douglas Bader

and

R.J. Mitchell

and

Jeffrey Quill and I realized

that though all of them had done many things in their lives, there

was one factor above all that connected them: the Spitfire.This is supposed to be a list of heroes. How can a machine be on

it? Because that machine was not just a machine. It was a lens

of heroes. Extraordinary devotion went into it, and extraordinary

courage came out.It's a cliche to call World War II a contest between good and evil,

but between fighter designs, it really was. The Spitfire's original

nemesis, the ME 109, was a brutally practical plane. It was a

killing machine. The Spitfire was optimism embodied. And not just

in its beautiful lines: it was at the edge of what could be

manufactured. But taking the high road worked. In the air, beauty

had the edge, just.

Steve JobsPeople alive when Kennedy was killed usually remember exactly where

they were when they heard about it. I remember exactly where I was

when a friend asked if I'd heard Steve Jobs had cancer. It was

like the floor dropped out. A few seconds later she told me that

it was a rare operable type, and that he'd be ok. But those seconds

seemed long.I wasn't sure whether to include Jobs on this list. A lot of people

at Apple seem to be afraid of him, which is a bad sign. But he

compels admiration.There's no name for what Steve Jobs is, because there hasn't been

anyone quite like him before. He doesn't design Apple's products

himself. Historically the closest analogy to what he does are the

great Renaissance patrons of the arts. As the CEO of a company,

that makes him unique.Most CEOs delegate

taste to a subordinate.

The

design paradox

means they're choosing more or less at random. But Steve

Jobs actually has taste himself — such good taste that he's shown

the world how much more important taste is than they realized.

Isaac NewtonNewton has a strange role in my pantheon of heroes: he's the one I

reproach myself with. He worked on big things, at least for part

of his life. It's so easy to get distracted working on small stuff.

The questions you're answering are pleasantly familiar. You get

immediate rewards — in fact, you get bigger rewards in your

time if you work on matters of passing importance. But I'm

uncomfortably aware that this is the route to well-deserved obscurity.To do really great things, you have to seek out questions people

didn't even realize were questions. There have probably been other

people who did this as well as Newton, for their time, but Newton

is my model of this kind of thought. I can just begin to understand

what it must have felt like for him.You only get one life. Why not do something huge? The phrase "paradigm

shift" is overused now, but Kuhn was onto something. And you know

more are out there, separated from us by what will later seem a

surprisingly thin wall of laziness and stupidity. If we work like

Newton.Thanks to Trevor Blackwell, Jessica Livingston, and Jackie McDonough for reading drafts of this.Japanese Translation

How to Disagree

March 2008The web is turning writing into a conversation. Twenty years ago,

writers wrote and readers read. The web lets readers respond, and

increasingly they do—in comment threads, on forums, and in their

own blog posts.Many who respond to something disagree with it. That's to be

expected. Agreeing tends to motivate people less than disagreeing.

And when you agree there's less to say. You could expand on something

the author said, but he has probably already explored the

most interesting implications. When you disagree you're entering

territory he may not have explored.The result is there's a lot more disagreeing going on, especially

measured by the word. That doesn't mean people are getting angrier.

The structural change in the way we communicate is enough to account

for it. But though it's not anger that's driving the increase in

disagreement, there's a danger that the increase in disagreement

will make people angrier. Particularly online, where it's easy to

say things you'd never say face to face.If we're all going to be disagreeing more, we should be careful to

do it well. What does it mean to disagree well? Most readers can

tell the difference between mere name-calling and a carefully

reasoned refutation, but I think it would help to put names on the

intermediate stages. So here's an attempt at a disagreement

hierarchy:

DH0. Name-calling.This is the lowest form of disagreement, and probably also the most

common. We've all seen comments like this:

u r a fag!!!!!!!!!!

But it's important to realize that more articulate name-calling has

just as little weight. A comment like

The author is a self-important dilettante.

is really nothing more than a pretentious version of "u r a fag."

DH1. Ad Hominem.An ad hominem attack is not quite as weak as mere name-calling. It

might actually carry some weight. For example, if a senator wrote

an article saying senators' salaries should be increased, one could

respond:

Of course he would say that. He's a senator.

This wouldn't refute the author's argument, but it may at least be

relevant to the case. It's still a very weak form of disagreement,

though. If there's something wrong with the senator's argument,

you should say what it is; and if there isn't, what difference does

it make that he's a senator?Saying that an author lacks the authority to write about a topic

is a variant of ad hominem—and a particularly useless sort, because

good ideas often come from outsiders. The question is whether the

author is correct or not. If his lack of authority caused him to

make mistakes, point those out. And if it didn't, it's not a

problem.

DH2. Responding to Tone.The next level up we start to see responses to the writing, rather

than the writer. The lowest form of these is to disagree with the

author's tone. E.g.

I can't believe the author dismisses intelligent design in such

a cavalier fashion.

Though better than attacking the author, this is still a weak form

of disagreement. It matters much more whether the author is wrong

or right than what his tone is. Especially since tone is so hard

to judge. Someone who has a chip on their shoulder about some topic

might be offended by a tone that to other readers seemed neutral.So if the worst thing you can say about something is to criticize

its tone, you're not saying much. Is the author flippant, but

correct? Better that than grave and wrong. And if the author is

incorrect somewhere, say where.DH3. Contradiction.In this stage we finally get responses to what was said, rather

than how or by whom. The lowest form of response to an argument

is simply to state the opposing case, with little or no supporting

evidence.This is often combined with DH2 statements, as in:

I can't believe the author dismisses intelligent design in such

a cavalier fashion. Intelligent design is a legitimate scientific

theory.

Contradiction can sometimes have some weight. Sometimes merely

seeing the opposing case stated explicitly is enough to see that

it's right. But usually evidence will help.DH4. Counterargument.At level 4 we reach the first form of convincing disagreement:

counterargument. Forms up to this point can usually be ignored as

proving nothing. Counterargument might prove something. The problem

is, it's hard to say exactly what.Counterargument is contradiction plus reasoning and/or evidence.

When aimed squarely at the original argument, it can be convincing.

But unfortunately it's common for counterarguments to be aimed at

something slightly different. More often than not, two people

arguing passionately about something are actually arguing about two

different things. Sometimes they even agree with one another, but

are so caught up in their squabble they don't realize it.There could be a legitimate reason for arguing against something

slightly different from what the original author said: when you

feel they missed the heart of the matter. But when you do that,

you should say explicitly you're doing it.DH5. Refutation.The most convincing form of disagreement is refutation. It's also

the rarest, because it's the most work. Indeed, the disagreement

hierarchy forms a kind of pyramid, in the sense that the higher you

go the fewer instances you find.To refute someone you probably have to quote them. You have to

find a "smoking gun," a passage in whatever you disagree with that

you feel is mistaken, and then explain why it's mistaken. If you

can't find an actual quote to disagree with, you may be arguing

with a straw man.While refutation generally entails quoting, quoting doesn't necessarily

imply refutation. Some writers quote parts of things they disagree

with to give the appearance of legitimate refutation, then follow

with a response as low as DH3 or even DH0.DH6. Refuting the Central Point.The force of a refutation depends on what you refute. The most

powerful form of disagreement is to refute someone's central point.Even as high as DH5 we still sometimes see deliberate dishonesty,

as when someone picks out minor points of an argument and refutes

those. Sometimes the spirit in which this is done makes it more

of a sophisticated form of ad hominem than actual refutation. For

example, correcting someone's grammar, or harping on minor mistakes

in names or numbers. Unless the opposing argument actually depends

on such things, the only purpose of correcting them is to

discredit one's opponent.Truly refuting something requires one to refute its central point,

or at least one of them. And that means one has to commit explicitly

to what the central point is. So a truly effective refutation would

look like:

The author's main point seems to be x. As he says:

<quotation>

But this is wrong for the following reasons...

The quotation you point out as mistaken need not be the actual

statement of the author's main point. It's enough to refute something

it depends upon.

What It MeansNow we have a way of classifying forms of disagreement. What good

is it? One thing the disagreement hierarchy doesn't give us is

a way of picking a winner. DH levels merely describe the form of

a statement, not whether it's correct. A DH6 response could still

be completely mistaken.But while DH levels don't set a lower bound on the convincingness

of a reply, they do set an upper bound. A DH6 response might be

unconvincing, but a DH2 or lower response is always unconvincing.The most obvious advantage of classifying the forms of disagreement

is that it will help people to evaluate what they read. In particular,

it will help them to see through intellectually dishonest arguments.

An eloquent speaker or writer can give the impression of vanquishing

an opponent merely by using forceful words. In fact that is probably

the defining quality of a demagogue. By giving names to the different

forms of disagreement, we give critical readers a pin for popping

such balloons.Such labels may help writers too. Most intellectual dishonesty is

unintentional. Someone arguing against the tone of something he

disagrees with may believe he's really saying something. Zooming

out and seeing his current position on the disagreement hierarchy

may inspire him to try moving up to counterargument or refutation.But the greatest benefit of disagreeing well is not just that it

will make conversations better, but that it will make the people

who have them happier. If you study conversations, you find there

is a lot more meanness down in DH1 than up in DH6. You don't have

to be mean when you have a real point to make. In fact, you don't

want to. If you have something real to say, being mean just gets

in the way.If moving up the disagreement hierarchy makes people less mean,

that will make most of them happier. Most people don't really enjoy

being mean; they do it because they can't help it.

Thanks to Trevor Blackwell and Jessica Livingston for reading

drafts of this.Related:What You Can't SayThe Age of the EssayItalian TranslationRussian TranslationSwedish TranslationSpanish TranslationGerman TranslationFrench TranslationArabic TranslationFinnish TranslationItalian Translation

You Weren't Meant to Have a Boss

Want to start a startup? Get funded by

Y Combinator.

March 2008, rev. June 2008Technology tends to separate normal from natural. Our bodies

weren't designed to eat the foods that people in rich countries eat, or

to get so little exercise.

There may be a similar problem with the way we work:

a normal job may be as bad for us intellectually as white flour

or sugar is for us physically.I began to suspect this after spending several years working

with startup founders. I've now worked with over 200 of them, and I've

noticed a definite difference between programmers working on their

own startups and those working for large organizations.

I wouldn't say founders seem happier, necessarily;

starting a startup can be very stressful. Maybe the best way to put

it is to say that they're happier in the sense that your body is

happier during a long run than sitting on a sofa eating

doughnuts.Though they're statistically abnormal, startup founders seem to be

working in a way that's more natural for humans.I was in Africa last year and saw a lot of animals in the wild that

I'd only seen in zoos before. It was remarkable how different they

seemed. Particularly lions. Lions in the wild seem about ten times

more alive. They're like different animals. I suspect that working

for oneself feels better to humans in much the same way that living

in the wild must feel better to a wide-ranging predator like a lion.

Life in a zoo is easier, but it isn't the life they were designed

for.

TreesWhat's so unnatural about working for a big company? The root of

the problem is that humans weren't meant to work in such large

groups.Another thing you notice when you see animals in the wild is that

each species thrives in groups of a certain size. A herd of impalas

might have 100 adults; baboons maybe 20; lions rarely 10. Humans

also seem designed to work in groups, and what I've read about

hunter-gatherers accords with research on organizations and my own

experience to suggest roughly what the ideal size is: groups of 8

work well; by 20 they're getting hard to manage; and a group of 50

is really unwieldy.

[1]

Whatever the upper limit is, we are clearly not meant to work in

groups of several hundred. And yet—for reasons having more

to do with technology than human nature—a great many people

work for companies with hundreds or thousands of employees.Companies know groups that large wouldn't work, so they divide

themselves into units small enough to work together. But to

coordinate these they have to introduce something new: bosses.These smaller groups are always arranged in a tree structure. Your

boss is the point where your group attaches to the tree. But when

you use this trick for dividing a large group into smaller ones,

something strange happens that I've never heard anyone mention

explicitly. In the group one level up from yours, your boss

represents your entire group. A group of 10 managers is not merely

a group of 10 people working together in the usual way. It's really

a group of groups. Which means for a group of 10 managers to work

together as if they were simply a group of 10 individuals, the group

working for each manager would have to work as if they were a single

person—the workers and manager would each share only one

person's worth of freedom between them.In practice a group of people are never able to act as if they were

one person. But in a large organization divided into groups in

this way, the pressure is always in that direction. Each group

tries its best to work as if it were the small group of individuals

that humans were designed to work in. That was the point of creating

it. And when you propagate that constraint, the result is that

each person gets freedom of action in inverse proportion to the

size of the entire tree.

[2]Anyone who's worked for a large organization has felt this. You

can feel the difference between working for a company with 100

employees and one with 10,000, even if your group has only 10 people.

Corn SyrupA group of 10 people within a large organization is a kind of fake

tribe. The number of people you interact with is about right. But

something is missing: individual initiative. Tribes of hunter-gatherers

have much more freedom. The leaders have a little more power than other

members of the tribe, but they don't generally tell them what to

do and when the way a boss can.It's not your boss's fault. The real problem is that in the group

above you in the hierarchy, your entire group is one virtual person.

Your boss is just the way that constraint is imparted to you.So working in a group of 10 people within a large organization feels

both right and wrong at the same time. On the surface it feels

like the kind of group you're meant to work in, but something major

is missing. A job at a big company is like high fructose corn

syrup: it has some of the qualities of things you're meant to like,

but is disastrously lacking in others.Indeed, food is an excellent metaphor to explain what's wrong with

the usual sort of job.For example, working for a big company is the default thing to do,

at least for programmers. How bad could it be? Well, food shows

that pretty clearly. If you were dropped at a random point in

America today, nearly all the food around you would be bad for you.

Humans were not designed to eat white flour, refined sugar, high

fructose corn syrup, and hydrogenated vegetable oil. And yet if

you analyzed the contents of the average grocery store you'd probably

find these four ingredients accounted for most of the calories.

"Normal" food is terribly bad for you. The only people who eat

what humans were actually designed to eat are a few Birkenstock-wearing

weirdos in Berkeley.If "normal" food is so bad for us, why is it so common? There are

two main reasons. One is that it has more immediate appeal. You

may feel lousy an hour after eating that pizza, but eating the first

couple bites feels great. The other is economies of scale.

Producing junk food scales; producing fresh vegetables doesn't.

Which means (a) junk food can be very cheap, and (b) it's worth

spending a lot to market it.If people have to choose between something that's cheap, heavily

marketed, and appealing in the short term, and something that's

expensive, obscure, and appealing in the long term, which do you

think most will choose?It's the same with work. The average MIT graduate wants to work

at Google or Microsoft, because it's a recognized brand, it's safe,

and they'll get paid a good salary right away. It's the job

equivalent of the pizza they had for lunch. The drawbacks will

only become apparent later, and then only in a vague sense of

malaise.And founders and early employees of startups, meanwhile, are like

the Birkenstock-wearing weirdos of Berkeley: though a tiny minority

of the population, they're the ones living as humans are meant to.

In an artificial world, only extremists live naturally.

ProgrammersThe restrictiveness of big company jobs is particularly hard on

programmers, because the essence of programming is to build new

things. Sales people make much the same pitches every day; support

people answer much the same questions; but once you've written a

piece of code you don't need to write it again. So a programmer

working as programmers are meant to is always making new things.

And when you're part of an organization whose structure gives each

person freedom in inverse proportion to the size of the tree, you're

going to face resistance when you do something new.This seems an inevitable consequence of bigness. It's true even

in the smartest companies. I was talking recently to a founder who

considered starting a startup right out of college, but went to

work for Google instead because he thought he'd learn more there.

He didn't learn as much as he expected. Programmers learn by doing,

and most of the things he wanted to do, he couldn't—sometimes

because the company wouldn't let him, but often because the company's

code wouldn't let him. Between the drag of legacy code, the overhead

of doing development in such a large organization, and the restrictions

imposed by interfaces owned by other groups, he could only try a

fraction of the things he would have liked to. He said he has

learned much more in his own startup, despite the fact that he has

to do all the company's errands as well as programming, because at

least when he's programming he can do whatever he wants.An obstacle downstream propagates upstream. If you're not allowed

to implement new ideas, you stop having them. And vice versa: when

you can do whatever you want, you have more ideas about what to do.

So working for yourself makes your brain more powerful in the same

way a low-restriction exhaust system makes an engine more powerful.Working for yourself doesn't have to mean starting a startup, of

course. But a programmer deciding between a regular job at a big

company and their own startup is probably going to learn more doing

the startup.You can adjust the amount of freedom you get by scaling the size

of company you work for. If you start the company, you'll have the

most freedom. If you become one of the first 10 employees you'll

have almost as much freedom as the founders. Even a company with

100 people will feel different from one with 1000.Working for a small company doesn't ensure freedom. The tree

structure of large organizations sets an upper bound on freedom,

not a lower bound. The head of a small company may still choose

to be a tyrant. The point is that a large organization is compelled

by its structure to be one.

ConsequencesThat has real consequences for both organizations and individuals.

One is that companies will inevitably slow down as they grow larger,

no matter how hard they try to keep their startup mojo. It's a

consequence of the tree structure that every large organization is

forced to adopt.Or rather, a large organization could only avoid slowing down if

they avoided tree structure. And since human nature limits the

size of group that can work together, the only way I can imagine

for larger groups to avoid tree structure would be to have no

structure: to have each group actually be independent, and to work

together the way components of a market economy do.That might be worth exploring. I suspect there are already some

highly partitionable businesses that lean this way. But I don't

know any technology companies that have done it.There is one thing companies can do short of structuring themselves

as sponges: they can stay small. If I'm right, then it really

pays to keep a company as small as it can be at every stage.

Particularly a technology company. Which means it's doubly important

to hire the best people. Mediocre hires hurt you twice: they get

less done, but they also make you big, because you need more of

them to solve a given problem.For individuals the upshot is the same: aim small. It will always

suck to work for large organizations, and the larger the organization,

the more it will suck.In an essay I wrote a couple years ago

I advised graduating seniors

to work for a couple years for another company before starting their

own. I'd modify that now. Work for another company if you want

to, but only for a small one, and if you want to start your own

startup, go ahead.The reason I suggested college graduates not start startups immediately

was that I felt most would fail. And they will. But ambitious

programmers are better off doing their own thing and failing than

going to work at a big company. Certainly they'll learn more. They

might even be better off financially. A lot of people in their

early twenties get into debt, because their expenses grow even

faster than the salary that seemed so high when they left school.

At least if you start a startup and fail your net worth will be

zero rather than negative.

[3]We've now funded so many different types of founders that we have

enough data to see patterns, and there seems to be no benefit from

working for a big company. The people who've worked for a few years

do seem better than the ones straight out of college, but only

because they're that much older.The people who come to us from big companies often seem kind of

conservative. It's hard to say how much is because big companies

made them that way, and how much is the natural conservatism that

made them work for the big companies in the first place. But

certainly a large part of it is learned. I know because I've seen

it burn off.Having seen that happen so many times is one of the things that

convinces me that working for oneself, or at least for a small

group, is the natural way for programmers to live. Founders arriving

at Y Combinator often have the downtrodden air of refugees. Three

months later they're transformed: they have so much more

confidence

that they seem as if they've grown several inches taller.

[4]

Strange as this sounds, they seem both more worried and happier at the same

time. Which is exactly how I'd describe the way lions seem in the

wild.Watching employees get transformed into founders makes it clear

that the difference between the two is due mostly to environment—and

in particular that the environment in big companies is toxic to

programmers. In the first couple weeks of working on their own

startup they seem to come to life, because finally they're working

the way people are meant to.Notes[1]

When I talk about humans being meant or designed to live a

certain way, I mean by evolution.[2]

It's not only the leaves who suffer. The constraint propagates

up as well as down. So managers are constrained too; instead of

just doing things, they have to act through subordinates.[3]

Do not finance your startup with credit cards. Financing a

startup with debt is usually a stupid move, and credit card debt

stupidest of all. Credit card debt is a bad idea, period. It is

a trap set by evil companies for the desperate and the foolish.[4]

The founders we fund used to be younger (initially we encouraged

undergrads to apply), and the first couple times I saw this I used

to wonder if they were actually getting physically taller.Thanks to Trevor Blackwell, Ross Boucher, Aaron Iba, Abby

Kirigin, Ivan Kirigin, Jessica Livingston, and Robert Morris for

reading drafts of this.French TranslationRussian Translation

A New Venture Animal

March 2008, rev May 2013(This essay grew out of something I wrote for myself to figure

out what we do. Even though Y Combinator is now 3 years old, we're still

trying to understand its implications.)

I was annoyed recently to read a description of Y Combinator that

said "Y Combinator does seed funding for startups." What was

especially annoying about it was that I wrote it. This doesn't

really convey what we do. And the reason it's inaccurate is that,

paradoxically, funding very early stage startups is not mainly about

funding.Saying YC does seed funding for startups is a description in terms

of earlier models. It's like calling a car a horseless carriage.When you scale animals you can't just keep everything in proportion.

For example, volume grows as the cube of linear dimension, but

surface area only as the square. So as animals get bigger they

have trouble radiating heat. That's why mice and rabbits are furry

and elephants and hippos aren't. You can't make a mouse by scaling

down an elephant.YC represents a new, smaller kind of animal—so much smaller

that all the rules are different.Before us, most companies in the startup funding business were

venture capital funds. VCs generally fund later stage companies

than we do. And they supply so much money that, even though the

other things they do may be very valuable, it's not that inaccurate

to regard VCs as sources of money. Good VCs are "smart money," but

they're still money.All good investors supply a combination of money and help. But

these scale differently, just as volume and surface area do. Late

stage investors supply huge amounts of money and

comparatively little help: when a company about to go public gets

a mezzanine round of $50 million, the deal tends to be almost

entirely about money. As you move earlier in the venture

funding process, the ratio of help to money increases, because

earlier stage companies have different needs. Early stage companies

need less money because they're smaller and cheaper to run, but

they need more help because life is so precarious for them. So

when VCs do a series A round for, say, $2 million, they generally

expect to offer a significant amount of help along with the money.Y Combinator occupies the earliest end of the spectrum. We're at

least one and generally two steps before VC funding. (Though some

startups go straight from YC to VC, the most common trajectory is

to do an angel round first.) And what happens at Y Combinator is

as different from what happens in a series A round as a series A

round is from a mezzanine financing.At our end, money is almost a negligible factor. The startup usually

consists of just the founders. Their living expenses are the

company's main expense, and since most founders are under 30, their

living expenses are low. But at this early stage companies need a

lot of help. Practically every question is still unanswered. Some

companies we've funded have been working on their software for a

year or more, but others haven't decided what to work on, or even

who the founders should be.When PR people and journalists recount the histories of startups

after they've become big, they always underestimate how uncertain

things were at first. They're not being deliberately misleading.

When you look at a company like Google, it's hard to imagine they

could once have been small and helpless. Sure, at one point they

were a just a couple guys in a garage—but even then their

greatness was assured, and all they had to do was roll forward along

the railroad tracks of destiny.Far from it. A lot of startups with just as promising beginnings

end up failing. Google has such momentum now that it would be hard

for anyone to stop them. But all it would have taken in the beginning

would have been for two Google employees to focus on the wrong

things for six months, and the company could have died.We know, because we've been there, just how vulnerable startups are

in the earliest phases. Curiously enough, that's why founders tend

to get so rich from them. Reward is always proportionate to risk,

and very early stage startups are insanely risky.What we really do at Y Combinator is get startups launched straight.

One of many metaphors you could use for YC is a steam catapult on

an aircraft carrier. We get startups airborne. Barely airborne,

but enough that they can accelerate fast.When you're launching planes they have to be set up properly or

you're just launching projectiles. They have to be pointed straight

down the deck; the wings have to be trimmed properly; the engines

have to be at full power; the pilot has to be ready. These are the

kind of problems we deal with. After we fund startups we work

closely with them for three months—so closely in fact that

we insist they move to where we are. And what we do in those three

months is make sure everything is set up for launch. If there are

tensions between cofounders we help sort them out. We get all the

paperwork set up properly so there are no nasty surprises later.

If the founders aren't sure what to focus on first, we try to figure

that out. If there is some obstacle right in front of them, we

either try to remove it, or shift the startup sideways. The goal

is to get every distraction out of the way so the founders can use

that time to build (or finish building) something impressive. And

then near the end of the three months we push the button on the

steam catapult in the form of Demo Day, where the current group of

startups present to pretty much every investor in Silicon Valley.Launching companies isn't identical with launching products. Though

we do spend a lot of time on launch strategies for products, there

are some things that take too long to build for a startup to launch

them before raising their next round of funding. Several of the

most promising startups we've funded haven't launched their products

yet, but are definitely launched as companies.In the earliest stage, startups not only have more questions to

answer, but they tend to be different kinds of questions. In later

stage startups the questions are about deals, or hiring, or

organization. In the earliest phase they tend to be about technology

and design. What do you make? That's the first problem to solve.

That's why our motto is "Make something people want." This is

always a good thing for companies to do, but it's even more important

early on, because it sets the bounds for every other question. Who

you hire, how much money you raise, how you market yourself—they

all depend on what you're making.Because the early problems are so much about technology and design,

you probably need to be hackers to do what we do. While some VCs

have technical backgrounds, I don't know any who still write code.

Their expertise is mostly in business—as it should be, because

that's the kind of expertise you need in the phase between series

A and (if you're lucky) IPO.We're so different from VCs that we're really a different kind of

animal. Can we claim founders are better off as a result of this

new type of venture firm? I'm pretty sure the answer is yes, because

YC is an improved version of what happened to our startup, and our

case was not atypical. We started Viaweb with $10,000 in seed money

from our friend Julian. He was a lawyer and arranged all our

paperwork, so we could just code. We spent three months building

a version 1, which we then presented to investors to raise more

money. Sounds familiar, doesn't it? But YC improves on that

significantly. Julian knew a lot about law and business, but his

advice ended there; he was not a startup guy. So we made some basic

mistakes early on. And when we presented to investors, we presented

to only 2, because that was all we knew. If we'd had our later

selves to encourage and advise us, and Demo Day to present at, we

would have been in much better shape. We probably could have raised

money at 3 to 5 times the valuation we did.If we take 7% of a company we fund, the founders only have to do

7.5% better in their next round of funding

to end up net ahead. We certainly manage that.So who is our 7% coming out of? If the founders end up net ahead

it's not coming out of them. So is it coming out of later stage

investors? Well, they do end up paying more. But I think they pay

more because the company is actually more valuable. And later stage

investors have no problem with that. The returns of a VC fund

depend on the quality of the companies they invest in, not how

cheaply they can buy stock in them.If what we do is useful, why wasn't anyone doing it before? There

are two answers to that. One is that people were doing it before,

just haphazardly on a smaller scale. Before us, seed funding came

primarily from individual angel investors. Larry and Sergey, for

example, got their seed funding from Andy Bechtolsheim, one of the

founders of Sun. And because he was a startup guy he probably gave

them useful advice. But raising money from angel investors is a

hit or miss thing. It's a sideline for most of them, so they only

do a handful of deals a year and they don't spend a lot of time on

the startups they invest in. And they're hard to reach, because

they don't want random startups pestering them with business plans.

The Google guys were lucky because they knew someone who knew

Bechtolsheim. It generally takes a personal introduction with

angels.The other reason no one was doing quite what we do is that till

recently it was a lot more expensive to start a startup. You'll

notice we haven't funded any biotech startups. That's still

expensive. But advancing technology has made web startups so cheap

that you really can get a company airborne for $15,000. If you

understand how to operate a steam catapult, at least.So in effect what's happened is that a new ecological niche has

opened up, and Y Combinator is the new kind of animal that has moved

into it. We're not a replacement for venture capital funds. We

occupy a new, adjacent niche. And conditions in our niche are

really quite different. It's not just that the problems we face

are different; the whole structure of the business is different.

VCs are playing a zero-sum game. They're all competing for a slice

of a fixed amount of "deal flow," and that explains a lot of their

behavior. Whereas our m.o. is to create new deal flow, by encouraging

hackers who would have gotten jobs to start their own startups

instead. We compete more with employers than VCs.It's not surprising something like this would happen. Most fields

become more specialized—more articulated—as they develop,

and startups are certainly an area in which there has been a lot

of development over the past couple decades. The venture business

in its present form is only about forty years old. It stands to

reason it would evolve.And it's natural that the new niche would at first be described,

even by its inhabitants, in terms of the old one. But really Y

Combinator is not in the startup funding business. Really we're

more of a small, furry steam catapult.Thanks to Trevor Blackwell, Jessica Livingston, and Robert Morris

for reading drafts of this.

Comment on this essay.

Trolls

February 2008A user on Hacker News recently posted a

comment

that set me thinking:

Something about hacker culture that never really set well with

me was this the nastiness. ... I just don't understand why people

troll like they do.

I've thought a lot over the last couple years about the problem of

trolls. It's an old one, as old as forums, but

we're still just learning what the causes are and how to address

them.There are two senses of the word "troll." In the original sense

it meant someone, usually an outsider, who deliberately stirred up

fights in a forum by saying controversial things.

[1]

For example,

someone who didn't use a certain programming language might go to

a forum for users of that language and make disparaging remarks

about it, then sit back and watch as people rose to the bait. This

sort of trolling was in the nature of a practical joke, like letting

a bat loose in a room full of people.The definition then spread to people who behaved like assholes in

forums, whether intentionally or not. Now when people talk about

trolls they usually mean this broader sense of the word. Though

in a sense this is historically inaccurate, it is in other ways

more accurate, because when someone is being an asshole it's usually

uncertain even in their own mind how much is deliberate.

That is arguably one of the defining qualities of an asshole.I think trolling in the broader sense has four causes. The most

important is distance. People will say things in anonymous forums

that they'd never dare say to someone's face, just as they'll do

things in cars that they'd never do as pedestrians like tailgate

people, or honk at them, or cut them off.Trolling tends to be particularly bad in forums related to computers,

and I think that's due to the kind of people you find there. Most

of them (myself included) are more comfortable dealing with abstract

ideas than with people. Hackers can be abrupt even in person. Put

them on an anonymous forum, and the problem gets worse.The third cause of trolling is incompetence. If you disagree with

something, it's easier to say "you suck" than to figure out and

explain exactly what you disagree with. You're also safe that way

from refutation. In this respect trolling is a lot like graffiti.

Graffiti happens at the intersection of ambition and incompetence:

people want to make their mark on the world, but have no other way

to do it than literally making a mark on the world.

[2]The final contributing factor is the culture of the forum. Trolls

are like children (many are children) in that they're capable of

a wide range of behavior depending on what they think will be

tolerated. In a place where rudeness isn't tolerated, most can be

polite. But vice versa as well.There's a sort of Gresham's Law of trolls: trolls are willing to

use a forum with a lot of thoughtful people in it, but thoughtful

people aren't willing to use a forum with a lot of trolls in it.

Which means that once trolling takes hold, it tends to become the

dominant culture. That had already happened to Slashdot and Digg by

the time I paid attention to comment threads there, but I watched

it happen to Reddit.News.YC is, among other things, an experiment to see if this fate

can be avoided. The sites's guidelines

explicitly ask people not to say things they wouldn't say face to

face. If someone starts being rude, other users will step in and

tell them to stop. And when people seem to be deliberately trolling,

we ban them ruthlessly.Technical tweaks may also help. On Reddit, votes on your comments

don't affect your karma score, but they do on News.YC. And it does

seem to influence people when they can see their reputation in the

eyes of their peers drain away after making an asshole remark.

Often users have second thoughts and delete such comments.One might worry this would prevent people from expressing controversial

ideas, but empirically that doesn't seem to be what happens. When

people say something substantial that gets modded down, they

stubbornly leave it up. What people delete are wisecracks, because

they have less invested in them.So far the experiment seems to be working. The level of conversation

on News.YC is as high as on any forum I've seen. But we still only

have about 8,000 uniques a day. The conversations on Reddit were

good when it was that small. The challenge is whether we can keep

things this way.I'm optimistic we will. We're not depending just on technical

tricks. The core users of News.YC are mostly refugees from other

sites that were overrun by trolls. They feel about trolls roughly

the way refugees from Cuba or Eastern Europe feel about dictatorships.

So there are a lot of people working to keep this from happening

again.

Notes[1]

I mean forum in the general sense of a place to exchange views.

The original Internet forums were not web sites but Usenet newsgroups.[2]

I'm talking here about everyday tagging. Some graffiti is

quite impressive (anything becomes art if you do it well enough)

but the median tag is just visual spam.Russian Translation

Six Principles for Making New Things

February 2008The fiery reaction to the release of Arc had

an unexpected consequence: it made me realize I had a design

philosophy. The main complaint of the more articulate critics was

that Arc seemed so flimsy. After years of working on it, all I had

to show for myself were a few thousand lines of macros? Why hadn't

I worked on more substantial problems?As I was mulling over these remarks it struck me how familiar they

seemed. This was exactly the kind of thing people said at first

about Viaweb, and Y Combinator, and most of my essays.When we launched Viaweb, it seemed laughable to VCs and e-commerce

"experts." We were just a couple guys in an apartment,

which did not seem cool in 1995 the way it does now. And the thing

we'd built, as far as they could tell, wasn't even software.

Software, to them, equalled big, honking Windows apps. Since Viaweb

was the first web-based app

they'd seen, it seemed to be nothing

more than a website. They were even more contemptuous when they

discovered that Viaweb didn't process credit card transactions (we

didn't for the whole first year). Transaction processing seemed

to them what e-commerce was all about. It sounded serious and

difficult.And yet, mysteriously, Viaweb ended up crushing all its competitors.The initial reaction to

Y Combinator was almost identical. It

seemed laughably lightweight. Startup funding meant series A rounds:

millions of dollars given to a small number of startups founded by

people with established credentials after months of serious,

businesslike meetings, on terms described in a document a foot

thick. Y Combinator seemed inconsequential. It's too early to say

yet whether Y Combinator will turn out like Viaweb, but judging

from the number of imitations, a lot of people seem to think we're

on to something.I can't measure whether my essays are successful, except in page

views, but the reaction to them is at least different from when I

started. At first the default reaction of the Slashdot trolls was

(translated into articulate terms): "Who is this guy and what

authority does he have to write about these topics? I haven't read

the essay, but there's no way anything so short and written in such

an informal style could have anything useful to say about such and

such topic, when people with degrees in the subject have already

written many thick books about it." Now there's a new generation

of trolls on a new generation of sites, but they have at least

started to omit the initial "Who is this guy?"Now people are saying the same things about Arc that they said at

first about Viaweb and Y Combinator and most of my essays. Why the

pattern? The answer, I realized, is that my m.o. for all four has

been the same.Here it is: I like to find (a) simple solutions (b) to overlooked

problems (c) that actually need to be solved, and (d) deliver them

as informally as possible, (e) starting with a very crude version

1, then (f) iterating rapidly.When I first laid out these principles explicitly, I noticed something

striking: this is practically a recipe for generating a contemptuous

initial reaction. Though simple solutions are better, they don't

seem as impressive as complex ones. Overlooked problems are by

definition problems that most people think don't matter. Delivering

solutions in an informal way means that instead of judging something

by the way it's presented, people have to actually understand it,

which is more work. And starting with a crude version 1 means your

initial effort is always small and incomplete.I'd noticed, of course, that people never seemed to grasp new ideas

at first. I thought it was just because most people were stupid.

Now I see there's more to it than that. Like a

contrarian investment fund, someone following this strategy will

almost always be doing things that seem wrong to the average person.As with contrarian investment strategies, that's exactly the point.

This technique is successful (in the long term) because it gives you

all the advantages other people forgo by trying to seem legit. If

you work on overlooked problems, you're more likely to discover new

things, because you have less competition. If you deliver solutions

informally, you (a) save all the effort you would have had to expend

to make them look impressive, and (b) avoid the danger of fooling

yourself as well as your audience. And if you release a crude

version 1 then iterate, your solution can benefit from the imagination

of nature, which, as Feynman pointed out, is more powerful than

your own.In the case of Viaweb, the simple solution was to make the software

run on the server. The overlooked problem was to generate web sites

automatically; in 1995, online stores were all made by hand by human

designers, but we knew this wouldn't scale. The part that actually

mattered was graphic design, not transaction processing.

The informal delivery mechanism was me, showing up in jeans and a

t-shirt at some retailer's office. And the crude version 1 was,

if I remember correctly, less than 10,000 lines of code when we

launched.The power of this technique extends beyond startups and programming

languages and essays. It probably extends to any kind of creative

work. Certainly it can be used in painting: this is exactly

what Cezanne and Klee did.At Y Combinator we bet money on it, in the sense that we encourage

the startups we fund to work this way. There are always new ideas

right under your nose. So look for simple things that other people

have overlooked—things people will later claim were

"obvious"—especially when they've been led astray by obsolete

conventions,

or by trying to do things that are superficially impressive. Figure

out what the real problem is, and make sure you solve that. Don't

worry about trying to look corporate; the product is what wins in

the long term. And launch as soon as you can, so you start learning

from users what you should have been making.Reddit is a classic example of

this approach. When Reddit first

launched, it seemed like there was nothing to it. To the graphically

unsophisticated its deliberately minimal design seemed like no

design at all. But Reddit solved the real problem, which was to

tell people what was new and otherwise stay out of the way. As a

result it became massively successful. Now that conventional ideas

have caught up with it, it seems obvious. People look at Reddit

and think the founders were lucky. Like all such things, it was

harder than it looked. The Reddits pushed so hard against the

current that they reversed it; now it looks like they're merely

floating downstream.So when you look at something like Reddit and think "I wish I could

think of an idea like that," remember: ideas like that are all

around you. But you ignore them because they look wrong.

Why to Move to a Startup Hub

October 2007After the last

talk I gave, one of the organizers

got up on the

stage to deliver an impromptu rebuttal. That never happened before.

I only heard the first few sentences, but that was enough to tell

what I said that upset him: that startups would do better if they

moved to Silicon Valley.This conference was in London, and most of the audience seemed to

be from the UK. So saying startups should move to Silicon Valley

seemed like a nationalistic remark: an obnoxious American telling

them that if they wanted to do things right they should all just

move to America.Actually I'm less American than I seem. I didn't say so, but I'm

British by birth. And just as Jews are ex officio allowed to tell

Jewish jokes, I don't feel like I have to bother being diplomatic

with a British audience.The idea that startups would do better to move to Silicon Valley

is not even a nationalistic one.

[1]

It's the same thing I say to

startups in the US. Y Combinator alternates between coasts every

6 months. Every other funding cycle is in Boston. And even though

Boston is the second biggest startup hub in the US (and the world),

we tell the startups from those cycles that their best bet is to

move to Silicon Valley. If that's true of Boston, it's even more

true of every other city.This is about cities, not countries.And I think I can prove I'm right. You can easily reduce the

opposing argument ad what most people would agree was absurdum.

Few would be willing to claim that it doesn't matter at all where

a startup is—that a startup operating out of a small agricultural

town wouldn't benefit from moving to a startup hub. Most people

could see how it might be helpful to be in a place where there was

infrastructure for startups, accumulated knowledge about how to

make them work, and other people trying to do it. And yet whatever

argument you use to prove that startups don't need to move from

London to Silicon Valley could equally well be used to prove startups

don't need to move from smaller towns to London.The difference between cities is a matter of degree. And if, as

nearly everyone who knows agrees, startups are better off in Silicon

Valley than Boston, then they're better off in Silicon Valley than

everywhere else too.I realize I might seem to have a vested interest in this conclusion,

because startups that move to the US might do it through Y Combinator.

But the American startups we've funded will attest that I say the

same thing to them.I'm not claiming of course that every startup has to go to Silicon

Valley to succeed. Just that all other things being equal, the

more of a startup hub a place is, the better startups will do there.

But other considerations can outweigh the advantages of moving.

I'm not saying founders with families should uproot them to move

halfway around the world; that might be too much of a distraction.Immigration difficulties might be another reason to stay put.

Dealing with immigration problems is like raising money: for some

reason it seems to consume all your attention. A startup can't

afford much of that. One Canadian startup we funded spent about 6

months working on moving to the US. Eventually they just gave up,

because they couldn't afford to take so much time away from working

on their software.(If another country wanted to establish a rival to Silicon Valley,

the single best thing they could do might be to create a special

visa for startup founders. US immigration policy is one of Silicon

Valley's biggest weaknesses.)If your startup is connected to a specific industry, you may be

better off in one of its centers. A startup doing something related

to entertainment might want to be in New York or LA.And finally, if a good investor has committed to fund

you if you stay where you are, you should probably stay. Finding

investors is hard. You generally shouldn't pass up a definite

funding offer to move.

[2]In fact, the quality of the investors may be the main advantage of

startup hubs. Silicon Valley investors are noticeably more aggressive

than Boston ones. Over and over, I've seen startups we've funded

snatched by west coast investors out from under the noses of Boston

investors who saw them first but acted too slowly. At this year's

Boston Demo Day, I told the audience that this happened every year,

so if they saw a startup they liked, they should make them an offer.

And yet within a month it had happened again: an aggressive west

coast VC who had met the founder of a YC-funded startup a week

before beat out a Boston VC who had known him for years. By the

time the Boston VC grasped what was happening, the deal was already

gone.Boston investors will admit they're more conservative. Some want

to believe this comes from the city's prudent Yankee character.

But Occam's razor suggests the truth is less flattering. Boston

investors are probably more conservative than Silicon Valley investors

for the same reason Chicago investors are more conservative than

Boston ones. They don't understand startups as well.West coast investors aren't bolder because they're irresponsible

cowboys, or because the good weather makes them optimistic. They're

bolder because they know what they're doing. They're the skiers

who ski on the diamond slopes. Boldness is the essence of venture

investing. The way you get big returns is not by trying to avoid

losses, but by trying to ensure you get some of the big hits. And

the big hits often look risky at first.Like Facebook. Facebook was started in Boston. Boston VCs had the

first shot at them. But they said no, so Facebook moved to Silicon

Valley and raised money there. The partner who turned them down

now says that "may turn out to have been a mistake."Empirically, boldness wins. If the aggressive ways of west coast

investors are going to come back to bite them, it has been a long

time coming. Silicon Valley has been pulling ahead of Boston since

the 1970s. If there was going to be a comeuppance for the west

coast investors, the bursting of the Bubble would have been it.

But since then the west coast has just pulled further ahead.West coast investors are confident enough of their judgement to act

boldly; east coast investors, not so much; but anyone who thinks

east coast investors act that way out of prudence should see the

frantic reactions of an east coast VC in the process of losing a

deal to a west coast one.In addition to the concentration that comes from specialization,

startup hubs are also markets. And markets are usually centralized.

Even now, when traders could be anywhere, they cluster in a few

cities. It's hard to say exactly what it is about face to face

contact that makes deals happen, but whatever it is, it hasn't yet

been duplicated by technology.Walk down University Ave at the right time, and you might overhear

five different people talking on the phone about deals. In fact,

this is part of the reason Y Combinator is in Boston half the time:

it's hard to stand that year round. But though it can sometimes

be annoying to be surrounded by people who only think about one

thing, it's the place to be if that one thing is what you're trying

to do.I was talking recently to someone who works on search at Google.

He knew a lot of people at Yahoo, so he was in a good position to

compare the two companies. I asked him why Google was better at

search. He said it wasn't anything specific Google did, but simply

that they understood search so much better.And that's why startups thrive in startup hubs like Silicon Valley.

Startups are a very specialized business, as specialized as diamond

cutting. And in startup hubs they understand it.

Notes[1]

The nationalistic idea is the converse: that startups should

stay in a certain city because of the country it's in. If you

really have a "one world" viewpoint, deciding to move from London

to Silicon Valley is no different from deciding to move from Chicago

to Silicon Valley.[2]

An investor who merely seems like he will fund you, however,

you can ignore. Seeming like they will fund you one day is the way

investors say No.Thanks to Sam Altman, Jessica Livingston, Harjeet Taggar, and Kulveer

Taggar for reading drafts of this.

Comment on this essay.Japanese Translation

The Future of Web Startups

Want to start a startup? Get funded by

Y Combinator.

October 2007(This essay is derived from a keynote at FOWA in October 2007.)There's something interesting happening right now. Startups are

undergoing the same transformation that technology does when it becomes

cheaper.It's a pattern we see over and over in technology. Initially

there's some device that's very expensive and made

in small quantities. Then someone discovers how to make them cheaply;

many more get built; and as a result they can be used in new ways.Computers are a familiar example. When I was a kid, computers were

big, expensive machines built one at a time. Now they're a commodity.

Now we can stick computers in everything.This pattern is very old. Most of the turning

points in economic history are instances of it. It happened to

steel in the 1850s, and to power in the 1780s.

It happened to cloth manufacture in the thirteenth century, generating

the wealth that later brought about the Renaissance. Agriculture

itself was an instance of this pattern.Now as well as being produced by startups, this pattern

is happening to startups. It's so cheap to start web startups

that orders of magnitudes more will be started. If the pattern

holds true, that should cause dramatic changes.1. Lots of StartupsSo my first prediction about the future of web startups is pretty

straightforward: there will be a lot of them. When starting a

startup was expensive, you had to get the permission of investors

to do it. Now the only threshold is courage.Even that threshold is getting lower, as people watch others take

the plunge and survive. In the last batch of startups we funded,

we had several founders who said they'd thought of applying before,

but weren't sure and got jobs instead. It was only after hearing

reports of friends who'd done it that they decided to try it

themselves.Starting a startup is hard, but having a 9 to 5 job is hard too,

and in some ways a worse kind of hard. In a startup you have lots

of worries, but you don't have that feeling that your life is flying

by like you do in a big company. Plus in a startup you could make

much more money.As word spreads that startups work, the number may grow

to a point that would now seem surprising.We now think of it as normal to have a job at a company, but this

is the thinnest of historical veneers. Just two or three

lifetimes ago, most people in what are now called industrialized

countries lived by farming. So while it may seem surprising to

propose that large numbers of people will change the way they make

a living, it would be more surprising if they didn't.2. StandardizationWhen technology makes something dramatically cheaper, standardization

always follows. When you make things in large volumes you tend

to standardize everything that doesn't need to change.At Y Combinator we still only have four people, so we try to

standardize everything. We could hire employees, but we want to be

forced to figure out how to scale investing.We often tell startups to release a minimal version one quickly,

then let the needs of the users determine what to do

next. In essense, let the market design the product. We've

done the same thing ourselves. We think of the techniques we're

developing for dealing with large numbers of startups as like

software. Sometimes it literally is software, like

Hacker News and

our application system.One of the most important things we've been working on standardizing

are investment terms. Till now investment terms have been

individually negotiated.

This is a problem for founders, because it makes raising money

take longer and cost more in legal fees. So as well as using the

same paperwork for every deal we do, we've commissioned generic

angel paperwork that all the startups we fund can use for future

rounds.Some investors will still want to cook up their own deal terms.

Series A rounds, where you raise a million dollars or more, will

be custom deals for the forseeable future. But I think angel rounds

will start to be done mostly with standardized agreements. An angel

who wants to insert a bunch of complicated terms into the agreement

is probably not one you want anyway.3. New Attitude to AcquisitionAnother thing I see starting to get standardized is acquisitions.

As the volume of startups increases, big companies will start to

develop standardized procedures that make acquisitions little

more work than hiring someone.Google is the leader here, as in so many areas of technology. They

buy a lot of startups— more than most people realize, because they

only announce a fraction of them. And being Google, they're

figuring out how to do it efficiently.One problem they've solved is how to think about acquisitions. For

most companies, acquisitions still carry some stigma of inadequacy.

Companies do them because they have to, but there's usually some

feeling they shouldn't have to—that their own programmers should

be able to build everything they need.Google's example should cure the rest of the world of this idea.

Google has by far the best programmers of any public technology

company. If they don't have a problem doing acquisitions, the

others should have even less problem. However many Google does,

Microsoft should do ten times as many.One reason Google doesn't have a problem with acquisitions

is that they know first-hand the quality of the people they can get

that way. Larry and Sergey only started Google after making the

rounds of the search engines trying to sell their idea and finding

no takers. They've been the guys coming in to visit the big

company, so they know who might be sitting across that conference

table from them.4. Riskier Strategies are PossibleRisk is always proportionate to reward. The way to get really big

returns is to do things that seem crazy, like starting a new search

engine in 1998, or turning down a billion dollar acquisition offer.This has traditionally been a problem in venture funding. Founders

and investors have different attitudes to risk. Knowing that risk

is on average proportionate to reward, investors like risky strategies,

while founders, who don't have a big enough sample size to care

what's true on average, tend to be more conservative.If startups are easy to start, this conflict goes away, because

founders can start them younger, when it's rational to take more

risk, and can start more startups total in their careers. When

founders can do lots of startups, they can start to look at the

world in the same portfolio-optimizing way as investors. And that

means the overall amount of wealth created can be greater, because

strategies can be riskier.5. Younger, Nerdier FoundersIf startups become a cheap commodity, more people will be able to

have them, just as more people could have computers once microprocessors

made them cheap. And in particular, younger and more technical

founders will be able to start startups than could before.Back when it cost a lot to start a startup, you had to convince

investors to let you do it. And that required very different skills

from actually doing the startup. If investors were perfect judges,

the two would require exactly the same skills. But unfortunately

most investors are terrible judges. I know because I see behind

the scenes what an enormous amount of work it takes to raise money,

and the amount of selling required in an industry is always inversely

proportional to the judgement of the buyers.Fortunately, if startups get cheaper to start, there's another way

to convince investors. Instead of going to venture capitalists

with a business plan and trying to convince them to fund it, you

can get a product launched on a few tens of thousands of dollars

of seed money from us or your uncle, and approach them with a

working company instead of a plan for one. Then instead of

having to seem smooth and confident, you can just point them to

Alexa.This way of convincing investors is better suited to hackers, who

often went into technology in part because they felt uncomfortable

with the amount of fakeness required in other fields.6. Startup Hubs Will PersistIt might seem that if startups get cheap to start, it will mean the

end of startup hubs like Silicon Valley. If all you need to start

a startup is rent money, you should be able to do it anywhere.This is kind of true and kind of false. It's true that you can now

start a startup anywhere. But you have to do more with a

startup than just start it. You have to make it succeed. And that

is more likely to happen in a startup hub.I've thought a lot about this question, and it seems to me the

increasing cheapness of web startups will if anything increase the

importance of startup hubs. The value of startup hubs, like centers

for any kind of business, lies in something very old-fashioned:

face to face meetings. No technology in the immediate future will

replace walking down University Ave and running into a friend who

tells you how to fix a bug that's been bothering you all weekend,

or visiting a friend's startup down the street and ending up in a

conversation with one of their investors.The question of whether to be in a startup hub is like the question

of whether to take outside investment. The question is not whether

you need it, but whether it brings any advantage at all.

Because anything that brings an advantage will give your competitors

an advantage over you if they do it and you don't. So if you hear

someone saying "we don't need to be in Silicon Valley," that use

of the word "need" is a sign they're not even thinking about the

question right.And while startup hubs are as powerful magnets as ever, the increasing

cheapness of starting a startup means the particles they're attracting

are getting lighter. A startup now can be just a pair of 22 year

old guys. A company like that can move much more easily than one

with 10 people, half of whom have kids.We know because we make people move for Y Combinator, and it doesn't

seem to be a problem. The advantage of being able to work together

face to face for three months outweighs the inconvenience of moving.

Ask anyone who's done it.The mobility of seed-stage startups means that seed funding is a

national business. One of the most common emails we get is from

people asking if we can help them set up a local clone of Y Combinator.

But this just wouldn't work. Seed funding isn't regional, just as

big research universities aren't.Is seed funding not merely national, but international? Interesting

question. There are signs it may be. We've had an ongoing

stream of founders from outside the US, and they tend to do

particularly well, because they're all people who were so determined

to succeed that they were willing to move to another country to do

it.The more mobile startups get, the harder it would be to start new

silicon valleys. If startups are mobile, the best local talent

will go to the real Silicon Valley,

and all they'll get at the local one will be the people who didn't

have the energy to move.This is not a nationalistic idea, incidentally. It's cities that

compete, not countries. Atlanta is just as hosed as Munich.7. Better Judgement NeededIf the number of startups increases dramatically, then the people

whose job is to judge them are going to have to get better at

it. I'm thinking particularly of investors and acquirers. We now

get on the order of 1000 applications a year. What are we going

to do if we get 10,000?That's actually an alarming idea. But we'll figure out some kind

of answer. We'll have to. It will probably involve writing some

software, but fortunately we can do that.Acquirers will also have to get better at picking winners.

They generally do better than investors, because they pick

later, when there's more performance to measure. But even at the

most advanced acquirers, identifying companies to

buy is extremely ad hoc, and completing the acquisition often

involves a great deal of unneccessary friction.I think acquirers may eventually have chief acquisition officers

who will both identify good acquisitions and make the deals happen.

At the moment those two functions are separate. Promising new

startups are often discovered by developers. If someone powerful

enough wants to buy them, the deal is handed over to corp dev guys

to negotiate. It would be better if both were combined in

one group, headed by someone with a technical background and some

vision of what they wanted to accomplish. Maybe in the future big

companies will have both a VP of Engineering responsible for

technology developed in-house, and a CAO responsible for bringing

technology in from outside.At the moment, there is no one within big companies who gets in

trouble when they buy a startup for $200 million that they could

have bought earlier for $20 million. There should start to be

someone who gets in trouble for that.8. College Will ChangeIf the best hackers start their own companies after college

instead of getting jobs, that will change what happens in college.

Most of these changes will be for the better. I think the experience

of college is warped in a bad way by the expectation that afterward

you'll be judged by potential employers.One change will be in the meaning of "after

college," which will switch from when one graduates from college

to when one leaves it. If you're starting your own company, why

do you need a degree? We don't encourage people to start startups

during college, but the best founders are certainly

capable of it. Some of the most successful companies we've funded

were started by undergrads.I grew up in a time where college degrees seemed really important,

so I'm alarmed to be saying things like this, but there's nothing

magical about a degree. There's nothing that magically changes

after you take that last exam. The importance of degrees is due

solely to the administrative needs of large organizations. These

can certainly affect your life—it's hard to get into grad

school, or to get a work visa in the US, without an undergraduate

degree—but tests like this will matter less and

less.As well as mattering less whether students get degrees, it will

also start to matter less where they go to college. In a startup

you're judged by users, and they don't care where you went to

college. So in a world of startups, elite universities will play

less of a role as gatekeepers. In the US it's a national scandal

how easily children of rich parents game college admissions.

But the way this problem ultimately gets solved may not be by

reforming the universities but by going around them. We in the

technology world are used to that sort of solution: you don't beat

the incumbents; you redefine the problem to make them irrelevant.The greatest value of universities is not the brand name or perhaps

even the classes so much as the people you meet. If

it becomes common to start a startup after college, students may start

trying to maximize this. Instead of focusing on getting

internships at companies they want to work for, they may start

to focus on working with other students they want as cofounders.What students do in their classes will change too. Instead of

trying to get good grades to impress future employers, students

will try to learn things. We're talking about some pretty dramatic

changes here.9. Lots of CompetitorsIf it gets easier to start a startup, it's easier for competitors too.

That doesn't erase the advantage of

increased cheapness, however. You're not all playing a zero-sum

game. There's not some fixed number of startups that can succeed,

regardless of how many are started.In fact, I don't think there's any limit to the number of startups

that could succeed. Startups succeed by creating wealth, which is

the satisfaction of people's desires. And people's desires seem

to be effectively infinite, at least in the short term.What the increasing number of startups does mean is that you won't

be able to sit on a good idea. Other people have your idea, and

they'll be increasingly likely to do something about it.10. Faster AdvancesThere's a good side to that, at least for consumers of

technology. If people get right to work implementing ideas instead

of sitting on them, technology will evolve faster.Some kinds of innovations happen a company at a time, like the

punctuated equilibrium model of evolution. There are some kinds

of ideas that are so threatening that it's hard for big companies

even to think of them. Look at what a hard time Microsoft is

having discovering web apps. They're like a character in a movie

that everyone in the audience can see something bad is about to

happen to, but who can't see it himself. The big innovations

that happen a company at a time will obviously happen faster if

the rate of new companies increases.But in fact there will be a double speed increase. People won't

wait as long to act on new ideas, but also those ideas will

increasingly be developed within startups rather than big companies.

Which means technology will evolve faster per company as well.Big companies are just not a good place to make things happen fast.

I talked recently to a founder whose startup had been acquired by

a big company. He was a precise sort of guy, so he'd measured their

productivity before and after. He counted lines of code, which can

be a dubious measure, but in this case was meaningful because it

was the same group of programmers. He found they were one thirteenth

as productive after the acquisition.The company that bought them was not a particularly stupid one.

I think what he was measuring was mostly the cost of bigness. I

experienced this myself, and his number sounds about right. There's

something about big companies that just sucks the energy out of

you.Imagine what all that energy could do if it were put to use. There

is an enormous latent capacity in the world's hackers that most

people don't even realize is there. That's the main reason we do

Y Combinator: to let loose all this energy by making it easy for

hackers to start their own startups.A Series of TubesThe process of starting startups is currently like the plumbing in

an old house. The pipes are narrow and twisty, and there are leaks

in every joint. In the future this mess will gradually be replaced

by a single, huge pipe. The water will still have to get from A

to B, but it will get there faster and without the risk of spraying

out through some random leak.This will change a lot of things for the better. In a big, straight

pipe like that, the force of being measured by one's performance

will propagate back through the whole system. Performance is always

the ultimate test, but there are so many kinks in the plumbing now

that most people are insulated from it most of the time. So you

end up with a world in which high school students think they need

to get good grades to get into elite colleges, and college students

think they need to get good grades to impress employers, within

which the employees waste most of their time in political battles,

and from which consumers have to buy anyway because there are so

few choices. Imagine if that sequence became a big, straight pipe.

Then the effects of being measured by performance would propagate

all the way back to high school, flushing out all the arbitrary

stuff people are measured by now. That is the future of web startups.Thanks to Brian Oberkirch and Simon Willison for inviting me to

speak, and the crew at Carson Systems for making everything run smoothly.Japanese Translation

How to Do Philosophy

September 2007In high school I decided I was going to study philosophy in college.

I had several motives, some more honorable than others. One of the

less honorable was to shock people. College was regarded as job

training where I grew up, so studying philosophy seemed an impressively

impractical thing to do. Sort of like slashing holes in your clothes

or putting a safety pin through your ear, which were other forms

of impressive impracticality then just coming into fashion.But I had some more honest motives as well. I thought studying

philosophy would be a shortcut straight to wisdom. All the people

majoring in other things would just end up with a bunch of domain

knowledge. I would be learning what was really what.I'd tried to read a few philosophy books. Not recent ones; you

wouldn't find those in our high school library. But I tried to

read Plato and Aristotle. I doubt I believed I understood them,

but they sounded like they were talking about something important.

I assumed I'd learn what in college.The summer before senior year I took some college classes. I learned

a lot in the calculus class, but I didn't learn much in Philosophy

101. And yet my plan to study philosophy remained intact. It was

my fault I hadn't learned anything. I hadn't read the books we

were assigned carefully enough. I'd give Berkeley's Principles

of Human Knowledge another shot in college. Anything so admired

and so difficult to read must have something in it, if one could

only figure out what.Twenty-six years later, I still don't understand Berkeley. I have

a nice edition of his collected works. Will I ever read it? Seems

unlikely.The difference between then and now is that now I understand why

Berkeley is probably not worth trying to understand. I think I see

now what went wrong with philosophy, and how we might fix it.WordsI did end up being a philosophy major for most of college. It

didn't work out as I'd hoped. I didn't learn any magical truths

compared to which everything else was mere domain knowledge. But

I do at least know now why I didn't. Philosophy doesn't really

have a subject matter in the way math or history or most other

university subjects do. There is no core of knowledge one must

master. The closest you come to that is a knowledge of what various

individual philosophers have said about different topics over the

years. Few were sufficiently correct that people have forgotten

who discovered what they discovered.Formal logic has some subject matter. I took several classes in

logic. I don't know if I learned anything from them.

[1]

It does seem to me very important to be able to flip ideas around in

one's head: to see when two ideas don't fully cover the space of

possibilities, or when one idea is the same as another but with a

couple things changed. But did studying logic teach me the importance

of thinking this way, or make me any better at it? I don't know.There are things I know I learned from studying philosophy. The

most dramatic I learned immediately, in the first semester of

freshman year, in a class taught by Sydney Shoemaker. I learned

that I don't exist. I am (and you are) a collection of cells that

lurches around driven by various forces, and calls itself I. But

there's no central, indivisible thing that your identity goes with.

You could conceivably lose half your brain and live. Which means

your brain could conceivably be split into two halves and each

transplanted into different bodies. Imagine waking up after such

an operation. You have to imagine being two people.The real lesson here is that the concepts we use in everyday life

are fuzzy, and break down if pushed too hard. Even a concept as

dear to us as I. It took me a while to grasp this, but when I

did it was fairly sudden, like someone in the nineteenth century

grasping evolution and realizing the story of creation they'd been

told as a child was all wrong.

[2]

Outside of math there's a limit

to how far you can push words; in fact, it would not be a bad

definition of math to call it the study of terms that have precise

meanings. Everyday words are inherently imprecise. They work well

enough in everyday life that you don't notice. Words seem to work,

just as Newtonian physics seems to. But you can always make them

break if you push them far enough.I would say that this has been, unfortunately for philosophy, the

central fact of philosophy. Most philosophical debates are not

merely afflicted by but driven by confusions over words. Do we

have free will? Depends what you mean by "free." Do abstract ideas

exist? Depends what you mean by "exist."Wittgenstein is popularly credited with the idea that most philosophical

controversies are due to confusions over language. I'm not sure

how much credit to give him. I suspect a lot of people realized

this, but reacted simply by not studying philosophy, rather than

becoming philosophy professors.How did things get this way? Can something people have spent

thousands of years studying really be a waste of time? Those are

interesting questions. In fact, some of the most interesting

questions you can ask about philosophy. The most valuable way to

approach the current philosophical tradition may be neither to get

lost in pointless speculations like Berkeley, nor to shut them down

like Wittgenstein, but to study it as an example of reason gone

wrong.HistoryWestern philosophy really begins with Socrates, Plato, and Aristotle.

What we know of their predecessors comes from fragments and references

in later works; their doctrines could be described as speculative

cosmology that occasionally strays into analysis. Presumably they

were driven by whatever makes people in every other society invent

cosmologies.

[3]With Socrates, Plato, and particularly Aristotle, this tradition

turned a corner. There started to be a lot more analysis. I suspect

Plato and Aristotle were encouraged in this by progress in math.

Mathematicians had by then shown that you could figure things out

in a much more conclusive way than by making up fine sounding stories

about them.

[4]People talk so much about abstractions now that we don't realize

what a leap it must have been when they first started to. It was

presumably many thousands of years between when people first started

describing things as hot or cold and when someone asked "what is

heat?" No doubt it was a very gradual process. We don't know if

Plato or Aristotle were the first to ask any of the questions they

did. But their works are the oldest we have that do this on a large

scale, and there is a freshness (not to say naivete) about them

that suggests some of the questions they asked were new to them,

at least.Aristotle in particular reminds me of the phenomenon that happens

when people discover something new, and are so excited by it that

they race through a huge percentage of the newly discovered territory

in one lifetime. If so, that's evidence of how new this kind of

thinking was.

[5]This is all to explain how Plato and Aristotle can be very impressive

and yet naive and mistaken. It was impressive even to ask the

questions they did. That doesn't mean they always came up with

good answers. It's not considered insulting to say that ancient

Greek mathematicians were naive in some respects, or at least lacked

some concepts that would have made their lives easier. So I hope

people will not be too offended if I propose that ancient philosophers

were similarly naive. In particular, they don't seem to have fully

grasped what I earlier called the central fact of philosophy: that

words break if you push them too far."Much to the surprise of the builders of the first digital computers,"

Rod Brooks wrote, "programs written for them usually did not work."

[6]

Something similar happened when people first started trying

to talk about abstractions. Much to their surprise, they didn't

arrive at answers they agreed upon. In fact, they rarely seemed

to arrive at answers at all.They were in effect arguing about artifacts induced by sampling at

too low a resolution.The proof of how useless some of their answers turned out to be is

how little effect they have. No one after reading Aristotle's

Metaphysics does anything differently as a result.

[7]Surely I'm not claiming that ideas have to have practical applications

to be interesting? No, they may not have to. Hardy's boast that

number theory had no use whatsoever wouldn't disqualify it. But

he turned out to be mistaken. In fact, it's suspiciously hard to

find a field of math that truly has no practical use. And Aristotle's

explanation of the ultimate goal of philosophy in Book A of the

Metaphysics implies that philosophy should be useful too.Theoretical KnowledgeAristotle's goal was to find the most general of general principles.

The examples he gives are convincing: an ordinary worker builds

things a certain way out of habit; a master craftsman can do more

because he grasps the underlying principles. The trend is clear:

the more general the knowledge, the more admirable it is. But then

he makes a mistake—possibly the most important mistake in the

history of philosophy. He has noticed that theoretical knowledge

is often acquired for its own sake, out of curiosity, rather than

for any practical need. So he proposes there are two kinds of

theoretical knowledge: some that's useful in practical matters and

some that isn't. Since people interested in the latter are interested

in it for its own sake, it must be more noble. So he sets as his

goal in the Metaphysics the exploration of knowledge that has no

practical use. Which means no alarms go off when he takes on grand

but vaguely understood questions and ends up getting lost in a sea

of words.His mistake was to confuse motive and result. Certainly, people

who want a deep understanding of something are often driven by

curiosity rather than any practical need. But that doesn't mean

what they end up learning is useless. It's very valuable in practice

to have a deep understanding of what you're doing; even if you're

never called on to solve advanced problems, you can see shortcuts

in the solution of simple ones, and your knowledge won't break down

in edge cases, as it would if you were relying on formulas you

didn't understand. Knowledge is power. That's what makes theoretical

knowledge prestigious. It's also what causes smart people to be

curious about certain things and not others; our DNA is not so

disinterested as we might think.So while ideas don't have to have immediate practical applications

to be interesting, the kinds of things we find interesting will

surprisingly often turn out to have practical applications.The reason Aristotle didn't get anywhere in the Metaphysics was

partly that he set off with contradictory aims: to explore the most

abstract ideas, guided by the assumption that they were useless.

He was like an explorer looking for a territory to the north of

him, starting with the assumption that it was located to the south.And since his work became the map used by generations of future

explorers, he sent them off in the wrong direction as well.

[8]

Perhaps worst of all, he protected them from both the criticism of

outsiders and the promptings of their own inner compass by establishing

the principle that the most noble sort of theoretical knowledge had

to be useless.The Metaphysics is mostly a failed experiment. A few ideas from

it turned out to be worth keeping; the bulk of it has had no effect

at all. The Metaphysics is among the least read of all famous

books. It's not hard to understand the way Newton's Principia

is, but the way a garbled message is.Arguably it's an interesting failed experiment. But unfortunately

that was not the conclusion Aristotle's successors derived from

works like the Metaphysics.

[9]

Soon after, the western world

fell on intellectual hard times. Instead of version 1s to be

superseded, the works of Plato and Aristotle became revered texts

to be mastered and discussed. And so things remained for a shockingly

long time. It was not till around 1600 (in Europe, where the center

of gravity had shifted by then) that one found people confident

enough to treat Aristotle's work as a catalog of mistakes. And

even then they rarely said so outright.If it seems surprising that the gap was so long, consider how little

progress there was in math between Hellenistic times and the

Renaissance.In the intervening years an unfortunate idea took hold: that it

was not only acceptable to produce works like the Metaphysics,

but that it was a particularly prestigious line of work, done by a

class of people called philosophers. No one thought to go back and

debug Aristotle's motivating argument. And so instead of correcting

the problem Aristotle discovered by falling into it—that you can

easily get lost if you talk too loosely about very abstract ideas—they

continued to fall into it.The SingularityCuriously, however, the works they produced continued to attract

new readers. Traditional philosophy occupies a kind of singularity

in this respect. If you write in an unclear way about big ideas,

you produce something that seems tantalizingly attractive to

inexperienced but intellectually ambitious students. Till one knows

better, it's hard to distinguish something that's hard to understand

because the writer was unclear in his own mind from something like

a mathematical proof that's hard to understand because the ideas

it represents are hard to understand. To someone who hasn't learned

the difference, traditional philosophy seems extremely attractive:

as hard (and therefore impressive) as math, yet broader in scope.

That was what lured me in as a high school student.This singularity is even more singular in having its own defense

built in. When things are hard to understand, people who suspect

they're nonsense generally keep quiet. There's no way to prove a

text is meaningless. The closest you can get is to show that the

official judges of some class of texts can't distinguish them from

placebos.

[10]And so instead of denouncing philosophy, most people who suspected

it was a waste of time just studied other things. That alone is

fairly damning evidence, considering philosophy's claims. It's

supposed to be about the ultimate truths. Surely all smart people

would be interested in it, if it delivered on that promise.Because philosophy's flaws turned away the sort of people who might

have corrected them, they tended to be self-perpetuating. Bertrand

Russell wrote in a letter in 1912:

Hitherto the people attracted to philosophy have been mostly those

who loved the big generalizations, which were all wrong, so that

few people with exact minds have taken up the subject.

[11]

His response was to launch Wittgenstein at it, with dramatic results.I think Wittgenstein deserves to be famous not for the discovery

that most previous philosophy was a waste of time, which judging

from the circumstantial evidence must have been made by every smart

person who studied a little philosophy and declined to pursue it

further, but for how he acted in response.

[12]

Instead of quietly

switching to another field, he made a fuss, from inside. He was

Gorbachev.The field of philosophy is still shaken from the fright Wittgenstein

gave it.

[13]

Later in life he spent a lot of time talking about

how words worked. Since that seems to be allowed, that's what a

lot of philosophers do now. Meanwhile, sensing a vacuum in the

metaphysical speculation department, the people who used to do

literary criticism have been edging Kantward, under new names like

"literary theory," "critical theory," and when they're feeling

ambitious, plain "theory." The writing is the familiar word salad:

Gender is not like some of the other grammatical modes which

express precisely a mode of conception without any reality that

corresponds to the conceptual mode, and consequently do not express

precisely something in reality by which the intellect could be

moved to conceive a thing the way it does, even where that motive

is not something in the thing as such.

[14]

The singularity I've described is not going away. There's a market

for writing that sounds impressive and can't be disproven. There

will always be both supply and demand. So if one group abandons

this territory, there will always be others ready to occupy it.A ProposalWe may be able to do better. Here's an intriguing possibility.

Perhaps we should do what Aristotle meant to do, instead of what

he did. The goal he announces in the Metaphysics seems one worth

pursuing: to discover the most general truths. That sounds good.

But instead of trying to discover them because they're useless,

let's try to discover them because they're useful.I propose we try again, but that we use that heretofore despised

criterion, applicability, as a guide to keep us from wondering

off into a swamp of abstractions. Instead of trying to answer the

question:

What are the most general truths?

let's try to answer the question

Of all the useful things we can say, which are the most general?

The test of utility I propose is whether we cause people who read

what we've written to do anything differently afterward. Knowing

we have to give definite (if implicit) advice will keep us from

straying beyond the resolution of the words we're using.The goal is the same as Aristotle's; we just approach it from a

different direction.As an example of a useful, general idea, consider that of the

controlled experiment. There's an idea that has turned out to be

widely applicable. Some might say it's part of science, but it's

not part of any specific science; it's literally meta-physics (in

our sense of "meta"). The idea of evolution is another. It turns

out to have quite broad applications—for example, in genetic

algorithms and even product design. Frankfurt's distinction between

lying and bullshitting seems a promising recent example.

[15]These seem to me what philosophy should look like: quite general

observations that would cause someone who understood them to do

something differently.Such observations will necessarily be about things that are imprecisely

defined. Once you start using words with precise meanings, you're

doing math. So starting from utility won't entirely solve the

problem I described above—it won't flush out the metaphysical

singularity. But it should help. It gives people with good

intentions a new roadmap into abstraction. And they may thereby

produce things that make the writing of the people with bad intentions

look bad by comparison.One drawback of this approach is that it won't produce the sort of

writing that gets you tenure. And not just because it's not currently

the fashion. In order to get tenure in any field you must not

arrive at conclusions that members of tenure committees can disagree

with. In practice there are two kinds of solutions to this problem.

In math and the sciences, you can prove what you're saying, or at

any rate adjust your conclusions so you're not claiming anything

false ("6 of 8 subjects had lower blood pressure after the treatment").

In the humanities you can either avoid drawing any definite conclusions

(e.g. conclude that an issue is a complex one), or draw conclusions

so narrow that no one cares enough to disagree with you.The kind of philosophy I'm advocating won't be able to take either

of these routes. At best you'll be able to achieve the essayist's

standard of proof, not the mathematician's or the experimentalist's.

And yet you won't be able to meet the usefulness test without

implying definite and fairly broadly applicable conclusions. Worse

still, the usefulness test will tend to produce results that annoy

people: there's no use in telling people things they already believe,

and people are often upset to be told things they don't.Here's the exciting thing, though. Anyone can do this. Getting

to general plus useful by starting with useful and cranking up the

generality may be unsuitable for junior professors trying to get

tenure, but it's better for everyone else, including professors who

already have it. This side of the mountain is a nice gradual slope.

You can start by writing things that are useful but very specific,

and then gradually make them more general. Joe's has good burritos.

What makes a good burrito? What makes good food? What makes

anything good? You can take as long as you want. You don't have

to get all the way to the top of the mountain. You don't have to

tell anyone you're doing philosophy.If it seems like a daunting task to do philosophy, here's an

encouraging thought. The field is a lot younger than it seems.

Though the first philosophers in the western tradition lived about

2500 years ago, it would be misleading to say the field is 2500

years old, because for most of that time the leading practitioners

weren't doing much more than writing commentaries on Plato or

Aristotle while watching over their shoulders for the next invading

army. In the times when they weren't, philosophy was hopelessly

intermingled with religion. It didn't shake itself free till a

couple hundred years ago, and even then was afflicted by the

structural problems I've described above. If I say this, some will

say it's a ridiculously overbroad and uncharitable generalization,

and others will say it's old news, but here goes: judging from their

works, most philosophers up to the present have been wasting their

time. So in a sense the field is still at the first step.

[16]That sounds a preposterous claim to make. It won't seem so

preposterous in 10,000 years. Civilization always seems old, because

it's always the oldest it's ever been. The only way to say whether

something is really old or not is by looking at structural evidence,

and structurally philosophy is young; it's still reeling from the

unexpected breakdown of words.Philosophy is as young now as math was in 1500. There is a lot

more to discover.Notes

[1]

In practice formal logic is not much use, because despite

some progress in the last 150 years we're still only able to formalize

a small percentage of statements. We may never do that much better,

for the same reason 1980s-style "knowledge representation" could

never have worked; many statements may have no representation more

concise than a huge, analog brain state.[2]

It was harder for Darwin's contemporaries to grasp this than

we can easily imagine. The story of creation in the Bible is not

just a Judeo-Christian concept; it's roughly what everyone must

have believed since before people were people. The hard part of

grasping evolution was to realize that species weren't, as they

seem to be, unchanging, but had instead evolved from different,

simpler organisms over unimaginably long periods of time.Now we don't have to make that leap. No one in an industrialized

country encounters the idea of evolution for the first time as an

adult. Everyone's taught about it as a child, either as truth or

heresy.[3]

Greek philosophers before Plato wrote in verse. This must

have affected what they said. If you try to write about the nature

of the world in verse, it inevitably turns into incantation. Prose

lets you be more precise, and more tentative.[4]

Philosophy is like math's

ne'er-do-well brother. It was born when Plato and Aristotle looked

at the works of their predecessors and said in effect "why can't

you be more like your brother?" Russell was still saying the same

thing 2300 years later.Math is the precise half of the most abstract ideas, and philosophy

the imprecise half. It's probably inevitable that philosophy will

suffer by comparison, because there's no lower bound to its precision.

Bad math is merely boring, whereas bad philosophy is nonsense. And

yet there are some good ideas in the imprecise half.[5]

Aristotle's best work was in logic and zoology, both of which

he can be said to have invented. But the most dramatic departure

from his predecessors was a new, much more analytical style of

thinking. He was arguably the first scientist.[6]

Brooks, Rodney, Programming in Common Lisp, Wiley, 1985, p.

94.[7]

Some would say we depend on Aristotle more than we realize,

because his ideas were one of the ingredients in our common culture.

Certainly a lot of the words we use have a connection with Aristotle,

but it seems a bit much to suggest that we wouldn't have the concept

of the essence of something or the distinction between matter and

form if Aristotle hadn't written about them.One way to see how much we really depend on Aristotle would be to

diff European culture with Chinese: what ideas did European culture

have in 1800 that Chinese culture didn't, in virtue of Aristotle's

contribution?[8]

The meaning of the word "philosophy" has changed over time.

In ancient times it covered a broad range of topics, comparable in

scope to our "scholarship" (though without the methodological

implications). Even as late as Newton's time it included what we

now call "science." But core of the subject today is still what

seemed to Aristotle the core: the attempt to discover the most

general truths.Aristotle didn't call this "metaphysics." That name got assigned

to it because the books we now call the Metaphysics came after

(meta = after) the Physics in the standard edition of Aristotle's

works compiled by Andronicus of Rhodes three centuries later. What

we call "metaphysics" Aristotle called "first philosophy."[9]

Some of Aristotle's immediate successors may have realized

this, but it's hard to say because most of their works are lost.[10]

Sokal, Alan, "Transgressing the Boundaries: Toward a Transformative

Hermeneutics of Quantum Gravity," Social Text 46/47, pp. 217-252.Abstract-sounding nonsense seems to be most attractive when it's

aligned with some axe the audience already has to grind. If this

is so we should find it's most popular with groups that are (or

feel) weak. The powerful don't need its reassurance.[11]

Letter to Ottoline Morrell, December 1912. Quoted in:Monk, Ray, Ludwig Wittgenstein: The Duty of Genius, Penguin, 1991,

p. 75.[12]

A preliminary result, that all metaphysics between Aristotle

and 1783 had been a waste of time, is due to I. Kant.[13]

Wittgenstein asserted a sort of mastery to which the inhabitants

of early 20th century Cambridge seem to have been peculiarly

vulnerable—perhaps partly because so many had been raised religious

and then stopped believing, so had a vacant space in their heads

for someone to tell them what to do (others chose Marx or Cardinal

Newman), and partly because a quiet, earnest place like Cambridge

in that era had no natural immunity to messianic figures, just as

European politics then had no natural immunity to dictators.[14]

This is actually from the Ordinatio of Duns Scotus (ca.

1300), with "number" replaced by "gender." Plus ca change.Wolter, Allan (trans), Duns Scotus: Philosophical Writings, Nelson,

1963, p. 92.[15]

Frankfurt, Harry, On Bullshit, Princeton University Press,

2005.[16]

Some introductions to philosophy now take the line that

philosophy is worth studying as a process rather than for any

particular truths you'll learn. The philosophers whose works they

cover would be rolling in their graves at that. They hoped they

were doing more than serving as examples of how to argue: they hoped

they were getting results. Most were wrong, but it doesn't seem

an impossible hope.This argument seems to me like someone in 1500 looking at the lack

of results achieved by alchemy and saying its value was as a process.

No, they were going about it wrong. It turns out it is possible

to transmute lead into gold (though not economically at current

energy prices), but the route to that knowledge was to

backtrack and try another approach.Thanks to Trevor Blackwell, Paul Buchheit, Jessica Livingston,

Robert Morris, Mark Nitzberg, and Peter Norvig for reading drafts of this.French Translation

News from the Front

September 2007A few weeks ago I had a thought so heretical that it really surprised

me. It may not matter all that much where you go to college.For me, as for a lot of middle class kids, getting into a good

college was more or less the meaning of life when I was growing up.

What was I? A student. To do that well meant to get good grades.

Why did one have to get good grades? To get into a good college.

And why did one want to do that? There seemed to be several reasons:

you'd learn more, get better jobs, make more money. But it didn't

matter exactly what the benefits would be. College was a bottleneck

through which all your future prospects passed; everything would

be better if you went to a better college.A few weeks ago I realized that somewhere along the line I had

stopped believing that.What first set me thinking about this was the new trend of worrying

obsessively about what

kindergarten

your kids go to. It seemed to

me this couldn't possibly matter. Either it won't help your kid

get into Harvard, or if it does, getting into Harvard won't mean

much anymore. And then I thought: how much does it mean even now?It turns out I have a lot of data about that. My three partners

and I run a seed stage investment firm called

Y Combinator. We

invest when the company is just a couple guys and an idea. The

idea doesn't matter much; it will change anyway. Most of our

decision is based on the founders. The average founder is three

years out of college. Many have just graduated; a few are still

in school. So we're in much the same position as a graduate program,

or a company hiring people right out of college. Except our choices

are immediately and visibly tested. There are two possible outcomes

for a startup: success or failure—and usually you know within a

year which it will be.The test applied to a startup is among the purest of real world

tests. A startup succeeds or fails depending almost entirely on

the efforts of the founders. Success is decided by the market: you

only succeed if users like what you've built. And users don't care

where you went to college.As well as having precisely measurable results, we have a lot of

them. Instead of doing a small number of large deals like a

traditional venture capital fund, we do a large number of small

ones. We currently fund about 40 companies a year, selected from

about 900 applications representing a total of about 2000 people.

[1]Between the volume of people we judge and the rapid, unequivocal

test that's applied to our choices, Y Combinator has been an

unprecedented opportunity for learning how to pick winners. One

of the most surprising things we've learned is how little it matters

where people went to college.I thought I'd already been cured of caring about that. There's

nothing like going to grad school at Harvard to cure you of any

illusions you might have about the average Harvard undergrad. And

yet Y Combinator showed us we were still overestimating people who'd

been to elite colleges. We'd interview people from MIT or Harvard

or Stanford and sometimes find ourselves thinking: they must be

smarter than they seem. It took us a few iterations to learn to

trust our senses.Practically everyone thinks that someone who went to MIT or Harvard

or Stanford must be smart. Even people who hate you for it believe

it.But when you think about what it means to have gone to an elite

college, how could this be true? We're talking about a decision

made by admissions officers—basically, HR people—based on a

cursory examination of a huge pile of depressingly similar applications

submitted by seventeen year olds. And what do they have to go on?

An easily gamed standardized test; a short essay telling you what

the kid thinks you want to hear; an interview with a random alum;

a high school record that's largely an index of obedience. Who

would rely on such a test?And yet a lot of companies do. A lot of companies are very much

influenced by where applicants went to college. How could they be?

I think I know the answer to that.There used to be a saying in the corporate world: "No one ever got

fired for buying IBM." You no longer hear this about IBM specifically,

but the idea is very much alive; there is a whole category of

"enterprise" software companies that exist to take advantage of it.

People buying technology for large organizations don't care if they

pay a fortune for mediocre software. It's not their money. They

just want to buy from a supplier who seems safe—a company with

an established name, confident salesmen, impressive offices, and

software that conforms to all the current fashions. Not necessarily

a company that will deliver so much as one that, if they do let you

down, will still seem to have been a prudent choice. So companies

have evolved to fill that niche.A recruiter at a big company is in much the same position as someone

buying technology for one. If someone went to Stanford and is not

obviously insane, they're probably a safe bet. And a safe bet is

enough. No one ever measures recruiters by the later performance

of people they turn down.

[2]I'm not saying, of course, that elite colleges have evolved to prey

upon the weaknesses of large organizations the way enterprise

software companies have. But they work as if they had. In addition

to the power of the brand name, graduates of elite colleges have

two critical qualities that plug right into the way large organizations

work. They're good at doing what they're asked, since that's what

it takes to please the adults who judge you at seventeen. And

having been to an elite college makes them more confident.Back in the days when people might spend their whole career at one

big company, these qualities must have been very valuable. Graduates

of elite colleges would have been capable, yet amenable to authority.

And since individual performance is so hard to measure in large

organizations, their own confidence would have been the starting

point for their reputation.Things are very different in the new world of startups. We couldn't

save someone from the market's judgement even if we wanted to. And

being charming and confident counts for nothing with users. All

users care about is whether you make something they like. If you

don't, you're dead.Knowing that test is coming makes us work a lot harder to get the

right answers than anyone would if they were merely hiring people.

We can't afford to have any illusions about the predictors of

success. And what we've found is that the variation between schools

is so much smaller than the variation between individuals that it's

negligible by comparison. We can learn more about someone in the

first minute of talking to them than by knowing where they went to

school.It seems obvious when you put it that way. Look at the individual,

not where they went to college. But that's a weaker statement than

the idea I began with, that it doesn't matter much where a given

individual goes to college. Don't you learn things at the best

schools that you wouldn't learn at lesser places?Apparently not. Obviously you can't prove this in the case of a

single individual, but you can tell from aggregate evidence: you

can't, without asking them, distinguish people who went to one

school from those who went to another three times as far down the

US News list.

[3]

Try it and see.How can this be? Because how much you learn in college depends a

lot more on you than the college. A determined party animal can

get through the best school without learning anything. And someone

with a real thirst for knowledge will be able to find a few smart

people to learn from at a school that isn't prestigious at all.

The other students are the biggest advantage of going to an elite

college; you learn more from them than the professors. But

you should be able to reproduce this at most colleges if you make

a conscious effort to find smart friends. At

most colleges you can find at least a handful of other smart students,

and most people have only a handful of close friends in college

anyway.

[4]

The odds of finding smart professors are even better.

The curve for faculty is a lot flatter than for students, especially

in math and the hard sciences; you have to go pretty far down the

list of colleges before you stop finding smart professors in the

math department.So it's not surprising that we've found the relative prestige of

different colleges useless in judging individuals. There's a lot

of randomness in how colleges select people, and what they learn

there depends much more on them than the college. Between these

two sources of variation, the college someone went to doesn't mean

a lot. It is to some degree a predictor of ability, but so weak

that we regard it mainly as a source of error and try consciously

to ignore it.I doubt what we've discovered is an anomaly specific to startups.

Probably people have always overestimated the importance of where

one goes to college. We're just finally able to measure it.The unfortunate thing is not just that people are judged by such a

superficial test, but that so many judge themselves by it. A lot

of people, probably the majority of people in America, have

some amount of insecurity about where, or whether, they went to

college. The tragedy of the situation is that by far the greatest

liability of not having gone to the college you'd have liked is

your own feeling that you're thereby lacking something. Colleges

are a bit like exclusive clubs in this respect. There is only one

real advantage to being a member of most exclusive clubs: you know

you wouldn't be missing much if you weren't. When you're excluded,

you can only imagine the advantages of being an insider. But

invariably they're larger in your imagination than in real life.So it is with colleges. Colleges differ, but they're nothing like

the stamp of destiny so many imagine them to be. People aren't

what some admissions officer decides about them at seventeen.

They're what they make themselves.Indeed, the great advantage of not caring where people went to

college is not just that you can stop judging them (and yourself)

by superficial measures, but that you can focus instead on what

really matters. What matters is what you make of yourself.

I think that's what we

should tell kids. Their job isn't to get good grades so they can

get into a good college, but to learn and do. And not just because

that's more rewarding than worldly success. That will increasingly

be the route to worldly success.

Notes[1]

Is what we measure worth measuring? I think so. You can get

rich simply by being energetic and unscrupulous, but getting rich

from a technology startup takes some amount of brains. It is just

the kind of work the upper middle class values; it has about the

same intellectual component as being a doctor.[2]

Actually, someone did, once. Mitch Kapor's wife Freada was

in charge of HR at Lotus in the early years. (As he is at pains

to point out, they did not become romantically involved till

afterward.) At one point they worried Lotus was losing its startup

edge and turning into a big company. So as an experiment she sent

their recruiters the resumes of the first 40 employees, with

identifying details changed. These were the people who had made

Lotus into the star it was. Not one got an interview.[3]

The US News list? Surely no one trusts that. Even if the

statistics they consider are useful, how do they decide on the

relative weights? The reason the US News list is meaningful is

precisely because they are so intellectually dishonest in that

respect. There is no external source they can use to calibrate the

weighting of the statistics they use; if there were, we could just

use that instead. What they must do is adjust the weights till the

top schools are the usual suspects in about the right order. So

in effect what the US News list tells us is what the editors think

the top schools are, which is probably not far from the conventional

wisdom on the matter. The amusing thing is, because some schools

work hard to game the system, the editors will have to keep tweaking

their algorithm to get the rankings they want.[4]

Possible doesn't mean easy, of course. A smart student at a party school

will inevitably be something of an outcast, just as he or

she would be in most high schools.

Thanks to Trevor Blackwell, Sarah Harlin, Jessica Livingston, Jackie

McDonough, Peter Norvig, and Robert Morris for reading drafts of

this.French Translation

How Not to Die

Want to start a startup? Get funded by

Y Combinator.

August 2007(This is a talk I gave at the last

Y Combinator dinner of the summer.

Usually we don't have a speaker at the last dinner; it's more of

a party. But it seemed worth spoiling the atmosphere if I could

save some of the startups from

preventable deaths. So at the last minute I cooked up this rather

grim talk. I didn't mean this as an essay; I wrote it down

because I only had two hours before dinner and think fastest while

writing.)

A couple days ago I told a reporter that we expected about a third

of the companies we funded to succeed. Actually I was being

conservative. I'm hoping it might be as much as a half. Wouldn't

it be amazing if we could achieve a 50% success rate?Another way of saying that is that half of you are going to die. Phrased

that way, it doesn't sound good at all. In fact, it's kind of weird

when you think about it, because our definition of success is that

the founders get rich. If half the startups we fund succeed, then

half of you are going to get rich and the other half are going to

get nothing.If you can just avoid dying, you get rich. That sounds like a joke,

but it's actually a pretty good description of what happens in a

typical startup. It certainly describes what happened in Viaweb.

We avoided dying till we got rich.It was really close, too. When we were visiting Yahoo to talk about

being acquired, we had to interrupt everything and borrow one of

their conference rooms to talk down an investor who was about to

back out of a new funding round we needed to stay alive. So even

in the middle of getting rich we were fighting off the grim reaper.You may have heard that quote about luck consisting of opportunity

meeting preparation. You've now done the preparation. The work

you've done so far has, in effect, put you in a position to get

lucky: you can now get rich by not letting your company die. That's

more than most people have. So let's talk about how not to die.We've done this five times now, and we've seen a bunch of startups

die. About 10 of them so far. We don't know exactly what happens

when they die, because they generally don't die loudly and heroically.

Mostly they crawl off somewhere and die.For us the main indication of impending doom is when we don't hear

from you. When we haven't heard from, or about, a startup for a

couple months, that's a bad sign. If we send them an email asking

what's up, and they don't reply, that's a really bad sign. So far

that is a 100% accurate predictor of death.Whereas if a startup regularly does new deals and releases and

either sends us mail or shows up at YC events, they're probably

going to live.I realize this will sound naive, but maybe the linkage works in

both directions. Maybe if you can arrange that we keep hearing

from you, you won't die.That may not be so naive as it sounds. You've probably noticed

that having dinners every Tuesday with us and the other founders

causes you to get more done than you would otherwise, because every

dinner is a mini Demo Day. Every dinner is a kind of a deadline.

So the mere constraint of staying in regular contact with us will

push you to make things happen, because otherwise you'll be embarrassed

to tell us that you haven't done anything new since the last time

we talked.If this works, it would be an amazing hack. It would be pretty

cool if merely by staying in regular contact with us you could get

rich. It sounds crazy, but there's a good chance that would work.A variant is to stay in touch with other YC-funded startups. There

is now a whole neighborhood of them in San Francisco. If you move

there, the peer pressure that made you work harder all summer will

continue to operate.When startups die, the official cause of death is always either

running out of money or a critical founder bailing. Often the two

occur simultaneously. But I think the underlying cause is usually

that they've become demoralized. You rarely hear of a startup

that's working around the clock doing deals and pumping out new

features, and dies because they can't pay their bills and their ISP

unplugs their server.Startups rarely die in mid keystroke. So keep typing!If so many startups get demoralized and fail when merely by hanging

on they could get rich, you have to assume that running a startup

can be demoralizing. That is certainly true. I've been there, and

that's why I've never done another startup. The low points in a

startup are just unbelievably low. I bet even Google had moments

where things seemed hopeless.Knowing that should help. If you know it's going to feel terrible

sometimes, then when it feels terrible you won't think "ouch, this

feels terrible, I give up." It feels that way for everyone. And

if you just hang on, things will probably get better. The metaphor

people use to describe the way a startup feels is at least a roller

coaster and not drowning. You don't just sink and sink; there are

ups after the downs.Another feeling that seems alarming but is in fact normal in a

startup is the feeling that what you're doing isn't working. The

reason you can expect to feel this is that what you do probably

won't work. Startups almost never get it right the first time.

Much more commonly you launch something, and no one cares. Don't

assume when this happens that you've failed. That's normal for

startups. But don't sit around doing nothing. Iterate.I like Paul Buchheit's suggestion of trying to make something that

at least someone really loves. As long as you've made something

that a few users are ecstatic about, you're on the right track. It

will be good for your morale to have even a handful of users who

really love you, and startups run on morale. But also it

will tell you what to focus on. What is it about you that they

love? Can you do more of that? Where can you find more people who

love that sort of thing? As long as you have some core of users

who love you, all you have to do is expand it. It may take a while,

but as long as you keep plugging away, you'll win in the end. Both

Blogger and Delicious did that. Both took years to succeed. But

both began with a core of fanatically devoted users, and all Evan

and Joshua had to do was grow that core incrementally.

Wufoo is

on the same trajectory now.So when you release something and it seems like no one cares, look

more closely. Are there zero users who really love you, or is there

at least some little group that does? It's quite possible there

will be zero. In that case, tweak your product and try again.

Every one of you is working on a space that contains at least one

winning permutation somewhere in it. If you just keep trying,

you'll find it.Let me mention some things not to do. The number one thing not to

do is other things. If you find yourself saying a sentence that

ends with "but we're going to keep working on the startup," you are

in big trouble. Bob's going to grad school, but we're going to

keep working on the startup. We're moving back to Minnesota, but

we're going to keep working on the startup. We're taking on some

consulting projects, but we're going to keep working on the startup.

You may as well just translate these to "we're giving up on the

startup, but we're not willing to admit that to ourselves," because

that's what it means most of the time. A startup is so hard that

working on it can't be preceded by "but."In particular, don't go to graduate school, and don't start other

projects. Distraction is fatal to startups. Going to (or back to)

school is a huge predictor of death because in addition to the

distraction it gives you something to say you're doing. If you're

only doing a startup, then if the startup fails, you fail. If

you're in grad school and your startup fails, you can say later "Oh

yeah, we had this startup on the side when I was in grad school,

but it didn't go anywhere."You can't use euphemisms like "didn't go anywhere" for something

that's your only occupation. People won't let you.One of the most interesting things we've discovered from working

on Y Combinator is that founders are more motivated by the fear of

looking bad than by the hope of getting millions of dollars. So

if you want to get millions of dollars, put yourself in a position

where failure will be public and humiliating.When we first met the founders of

Octopart, they seemed very smart,

but not a great bet to succeed, because they didn't seem especially

committed. One of the two founders was still in grad school. It

was the usual story: he'd drop out if it looked like the startup

was taking off. Since then he has not only dropped out of grad

school, but appeared full length in

Newsweek

with the word "Billionaire"

printed across his chest. He just cannot fail now. Everyone he

knows has seen that picture. Girls who dissed him in high school

have seen it. His mom probably has it on the fridge. It would be

unthinkably humiliating to fail now. At this point he is committed

to fight to the death.I wish every startup we funded could appear in a Newsweek article

describing them as the next generation of billionaires, because

then none of them would be able to give up. The success rate would

be 90%. I'm not kidding.When we first knew the Octoparts they were lighthearted, cheery

guys. Now when we talk to them they seem grimly determined. The

electronic parts distributors are trying to squash them to keep

their monopoly pricing. (If it strikes you as odd that people still

order electronic parts out of thick paper catalogs in 2007, there's

a reason for that. The distributors want to prevent the transparency

that comes from having prices online.) I feel kind of bad that

we've transformed these guys from lighthearted to grimly determined.

But that comes with the territory. If a startup succeeds, you get

millions of dollars, and you don't get that kind of money just by

asking for it. You have to assume it takes some amount of pain.And however tough things get for the Octoparts, I predict they'll

succeed. They may have to morph themselves into something totally

different, but they won't just crawl off and die. They're smart;

they're working in a promising field; and they just cannot give up.All of you guys already have the first two. You're all smart and

working on promising ideas. Whether you end up among the living

or the dead comes down to the third ingredient, not giving up.So I'll tell you now: bad shit is coming. It always is in a startup.

The odds of getting from launch to liquidity without some kind of

disaster happening are one in a thousand. So don't get demoralized.

When the disaster strikes, just say to yourself, ok, this was what

Paul was talking about. What did he say to do? Oh, yeah. Don't

give up.Japanese TranslationArabic Translation

Holding a Program in One's Head

August 2007A good programmer working intensively on his own code can hold it

in his mind the way a mathematician holds a problem he's working

on. Mathematicians don't answer questions by working them out on

paper the way schoolchildren are taught to. They do more in their

heads: they try to understand a problem space well enough that they

can walk around it the way you can walk around the memory of the

house you grew up in. At its best programming is the same. You

hold the whole program in your head, and you can manipulate it at

will.That's particularly valuable at the start of a project, because

initially the most important thing is to be able to change what

you're doing. Not just to solve the problem in a different way,

but to change the problem you're solving.Your code is your understanding of the problem you're exploring.

So it's only when you have your code in your head that you really

understand the problem.It's not easy to get a program into your head. If you leave a

project for a few months, it can take days to really understand it

again when you return to it. Even when you're actively working on

a program it can take half an hour to load into your head when you

start work each day. And that's in the best case. Ordinary

programmers working in typical office conditions never enter this

mode. Or to put it more dramatically, ordinary programmers working

in typical office conditions never really understand the problems

they're solving.Even the best programmers don't always have the whole program they're

working on loaded into their heads. But there are things you can

do to help:

Avoid distractions. Distractions are bad for many types of work,

but especially bad for programming, because programmers tend to

operate at the limit of the detail they can handle.The danger of a distraction depends not on how long it is, but

on how much it scrambles your brain. A programmer can leave the

office and go and get a sandwich without losing the code in his

head. But the wrong kind of interruption can wipe your brain

in 30 seconds.Oddly enough, scheduled distractions may be worse than unscheduled

ones. If you know you have a meeting in an hour, you don't even

start working on something hard. Work in long stretches. Since there's a fixed cost each time

you start working on a program, it's more efficient to work in

a few long sessions than many short ones. There will of course

come a point where you get stupid because you're tired. This

varies from person to person. I've heard of people hacking for

36 hours straight, but the most I've ever been able to manage

is about 18, and I work best in chunks of no more than 12.The optimum is not the limit you can physically endure. There's

an advantage as well as a cost of breaking up a project. Sometimes

when you return to a problem after a rest, you find your unconscious

mind has left an answer waiting for you. Use succinct languages. More

powerful programming languages

make programs shorter. And programmers seem to think of programs

at least partially in the language they're using to write them.

The more succinct the language, the shorter the program, and the

easier it is to load and keep in your head.You can magnify the effect of a powerful language by using a

style called bottom-up programming, where you write programs in

multiple layers, the lower ones acting as programming languages

for those above. If you do this right, you only have to keep

the topmost layer in your head. Keep rewriting your program. Rewriting a program often yields

a cleaner design. But it would have advantages even if it didn't:

you have to understand a program completely to rewrite it, so

there is no better way to get one loaded into your head. Write rereadable code. All programmers know it's good to write

readable code. But you yourself are the most important reader.

Especially in the beginning; a prototype is a conversation with

yourself. And when writing for yourself you have different

priorities. If you're writing for other people, you may not

want to make code too dense. Some parts of a program may be

easiest to read if you spread things out, like an introductory

textbook. Whereas if you're writing code to make it easy to reload

into your head, it may be best to go for brevity. Work in small groups. When you manipulate a program in your

head, your vision tends to stop at the edge of the code you own.

Other parts you don't understand as well, and more importantly,

can't take liberties with. So the smaller the number of

programmers, the more completely a project can mutate. If there's

just one programmer, as there often is at first, you can do

all-encompassing redesigns. Don't have multiple people editing the same piece of code. You

never understand other people's code as well as your own. No

matter how thoroughly you've read it, you've only read it, not

written it. So if a piece of code is written by multiple authors,

none of them understand it as well as a single author would.And of course you can't safely redesign something other people

are working on. It's not just that you'd have to ask permission.

You don't even let yourself think of such things. Redesigning

code with several authors is like changing laws; redesigning

code you alone control is like seeing the other interpretation

of an ambiguous image.If you want to put several people to work on a project, divide

it into components and give each to one person. Start small. A program gets easier to hold in your head as you

become familiar with it. You can start to treat parts as black

boxes once you feel confident you've fully explored them. But

when you first start working on a project, you're forced to see

everything. If you start with too big a problem, you may never

quite be able to encompass it. So if you need to write a big,

complex program, the best way to begin may not be to write a

spec for it, but to write a prototype that solves a subset of

the problem. Whatever the advantages of planning, they're often

outweighed by the advantages of being able to keep a program in

your head.

It's striking how often programmers manage to hit all eight points

by accident. Someone has an idea for a new project, but because

it's not officially sanctioned, he has to do it in off hours—which

turn out to be more productive because there are no distractions.

Driven by his enthusiasm for the new project he works on it for

many hours at a stretch. Because it's initially just an

experiment, instead of a "production" language he uses a mere

"scripting" language—which is in fact far more powerful. He

completely rewrites the program several times; that wouldn't be

justifiable for an official project, but this is a labor of love

and he wants it to be perfect. And since no one is going to see

it except him, he omits any comments except the note-to-self variety.

He works in a small group perforce, because he either hasn't told

anyone else about the idea yet, or it seems so unpromising that no

one else is allowed to work on it. Even if there is a group, they

couldn't have multiple people editing the same code, because it

changes too fast for that to be possible. And the project starts

small because the idea is small at first; he just has some cool

hack he wants to try out.Even more striking are the number of officially sanctioned projects

that manage to do all eight things wrong. In fact, if you look at

the way software gets written in most organizations, it's almost

as if they were deliberately trying to do things wrong. In a sense,

they are. One of the defining qualities of organizations since

there have been such a thing is to treat individuals as interchangeable

parts. This works well for more parallelizable tasks, like fighting

wars. For most of history a well-drilled army of professional

soldiers could be counted on to beat an army of individual warriors,

no matter how valorous. But having ideas is not very parallelizable.

And that's what programs are: ideas.It's not merely true that organizations dislike the idea of depending

on individual genius, it's a tautology. It's part of the definition

of an organization not to. Of our current concept of an organization,

at least.Maybe we could define a new kind of organization that combined the

efforts of individuals without requiring them to be interchangeable.

Arguably a market is such a form of organization, though it may be

more accurate to describe a market as a degenerate case—as what

you get by default when organization isn't possible.Probably the best we'll do is some kind of hack, like making the

programming parts of an organization work differently from the rest.

Perhaps the optimal solution is for big companies not even to try

to develop ideas in house, but simply to

buy them. But regardless

of what the solution turns out to be, the first step is to realize

there's a problem. There is a contradiction in the very phrase

"software company." The two words are pulling in opposite directions.

Any good programmer in a large organization is going to be at odds

with it, because organizations are designed to prevent what

programmers strive for.Good programmers manage to get a lot done anyway.

But often it

requires practically an act of rebellion against the organizations

that employ them. Perhaps it will help if more people understand that the way

programmers behave is driven by the demands of the work they do.

It's not because they're irresponsible that they work in long binges

during which they blow off all other obligations, plunge straight into

programming instead of writing specs first, and rewrite code that

already works. It's not because they're unfriendly that they prefer

to work alone, or growl at people who pop their head in the door

to say hello. This apparently random collection of annoying habits

has a single explanation: the power of holding a program in one's

head.Whether or not understanding this can help large organizations, it

can certainly help their competitors. The weakest point in big

companies is that they don't let individual programmers do great

work. So if you're a little startup, this is the place to attack

them. Take on the kind of problems that have to be solved in one

big brain.

Thanks to Sam Altman, David Greenspan, Aaron Iba, Jessica Livingston,

Robert Morris, Peter Norvig, Lisa Randall, Emmett Shear, Sergei Tsarev,

and Stephen Wolfram for reading drafts of this.Japanese TranslationSimplified Chinese TranslationPortuguese TranslationBulgarian TranslationRussian Translation

Stuff

July 2007I have too much stuff. Most people in America do. In fact, the

poorer people are, the more stuff they seem to have. Hardly anyone

is so poor that they can't afford a front yard full of old cars.It wasn't always this way. Stuff used to be rare and valuable.

You can still see evidence of that if you look for it. For example,

in my house in Cambridge, which was built in 1876, the bedrooms

don't have closets. In those days people's stuff fit in a chest

of drawers. Even as recently as a few decades ago there was a lot

less stuff. When I look back at photos from the 1970s, I'm surprised

how empty houses look. As a kid I had what I thought was a huge

fleet of toy cars, but they'd be dwarfed by the number of toys my

nephews have. All together my Matchboxes and Corgis took up about

a third of the surface of my bed. In my nephews' rooms the bed is

the only clear space.Stuff has gotten a lot cheaper, but our attitudes toward it haven't

changed correspondingly. We overvalue stuff.That was a big problem

for me when I had no money. I felt poor, and stuff seemed valuable,

so almost instinctively I accumulated it. Friends would leave

something behind when they moved, or I'd see something as I was

walking down the street on trash night (beware of anything you find

yourself describing as "perfectly good"), or I'd find something in

almost new condition for a tenth its retail price at a garage sale.

And pow, more stuff.In fact these free or nearly free things weren't bargains, because

they were worth even less than they cost. Most of the stuff I

accumulated was worthless, because I didn't need it.What I didn't understand was that the value of some new acquisition

wasn't the difference between its retail price and what I paid for

it. It was the value I derived from it. Stuff is an extremely

illiquid asset. Unless you have some plan for selling that valuable

thing you got so cheaply, what difference does it make what it's

"worth?" The only way you're ever going to extract any value from

it is to use it. And if you don't have any immediate use for it,

you probably never will.Companies that sell stuff have spent huge sums training us to think

stuff is still valuable. But it would be closer to the truth to

treat stuff as worthless.In fact, worse than worthless, because once you've accumulated a

certain amount of stuff, it starts to own you rather than the other

way around. I know of one couple who couldn't retire to the town

they preferred because they couldn't afford a place there big enough

for all their stuff. Their house isn't theirs; it's their stuff's.And unless you're extremely organized, a house full of stuff can

be very depressing. A cluttered room saps one's spirits. One

reason, obviously, is that there's less room for people in a room

full of stuff. But there's more going on than that. I think humans

constantly scan their environment to build a mental model of what's

around them. And the harder a scene is to parse, the less energy

you have left for conscious thoughts. A cluttered room is literally

exhausting.(This could explain why clutter doesn't seem to bother kids as much

as adults. Kids are less perceptive. They build a coarser model

of their surroundings, and this consumes less energy.)I first realized the worthlessness of stuff when I lived in Italy

for a year. All I took with me was one large backpack of stuff.

The rest of my stuff I left in my landlady's attic back in the US.

And you know what? All I missed were some of the books. By the

end of the year I couldn't even remember what else I had stored in

that attic.And yet when I got back I didn't discard so much as a box of it.

Throw away a perfectly good rotary telephone? I might need that

one day.The really painful thing to recall is not just that I accumulated

all this useless stuff, but that I often spent money I desperately

needed on stuff that I didn't.Why would I do that? Because the people whose job is to sell you

stuff are really, really good at it. The average 25 year old is

no match for companies that have spent years figuring out how to

get you to spend money on stuff. They make the experience of buying

stuff so pleasant that "shopping" becomes a leisure activity.How do you protect yourself from these people? It can't be easy.

I'm a fairly skeptical person, and their tricks worked on me well

into my thirties. But one thing that might work is to ask yourself,

before buying something, "is this going to make my life noticeably

better?"A friend of mine cured herself of a clothes buying habit by asking

herself before she bought anything "Am I going to wear this all the

time?" If she couldn't convince herself that something she was

thinking of buying would become one of those few things she wore

all the time, she wouldn't buy it. I think that would work for any

kind of purchase. Before you buy anything, ask yourself: will this

be something I use constantly? Or is it just something nice? Or

worse still, a mere bargain?The worst stuff in this respect may be stuff you don't use much

because it's too good. Nothing owns you like fragile stuff. For

example, the "good china" so many households have, and whose defining

quality is not so much that it's fun to use, but that one must be

especially careful not to break it.Another way to resist acquiring stuff is to think of the overall

cost of owning it. The purchase price is just the beginning. You're

going to have to think about that thing for years—perhaps for

the rest of your life. Every thing you own takes energy away from

you. Some give more than they take. Those are the only things

worth having.I've now stopped accumulating stuff. Except books—but books are

different. Books are more like a fluid than individual objects.

It's not especially inconvenient to own several thousand books,

whereas if you owned several thousand random possessions you'd be

a local celebrity. But except for books, I now actively avoid

stuff. If I want to spend money on some kind of treat, I'll take

services over

goods any day.I'm not claiming this is because I've achieved some kind of zenlike

detachment from material things. I'm talking about something more

mundane. A historical change has taken place, and I've now realized

it. Stuff used to be valuable, and now it's not.In industrialized countries the same thing happened with food in

the middle of the twentieth century. As food got cheaper (or we

got richer; they're indistinguishable), eating too much started to

be a bigger danger than eating too little. We've now reached that

point with stuff. For most people, rich or poor, stuff has become

a burden.The good news is, if you're carrying a burden without knowing it,

your life could be better than you realize. Imagine walking around

for years with five pound ankle weights, then suddenly having them

removed.

Spanish TranslationRussian TranslationItalian TranslationPolish TranslationTurkish TranslationFrench TranslationSlovak TranslationRomanian TranslationGerman Translation

The Equity Equation

July 2007An investor wants to give you money for a certain percentage of

your startup. Should you take it? You're about to hire your first

employee. How much stock should you give him?These are some of the hardest questions founders face. And yet

both have the same answer:1/(1 - n)Whenever you're trading stock in your company for anything, whether

it's money or an employee or a deal with another company, the test

for whether to do it is the same. You should give up n% of your

company if what you trade it for improves your average outcome

enough that the (100 - n)% you have left is worth more than the

whole company was before.For example, if an investor wants to buy half your company, how

much does that investment have to improve your average outcome for

you to break even? Obviously it has to double: if you trade half

your company for something that more than doubles the company's

average outcome, you're net ahead. You have half as big a share

of something worth more than twice as much.In the general case, if n is the fraction of the company you're

giving up, the deal is a good one if it makes the company worth

more than 1/(1 - n).For example, suppose Y Combinator offers to fund you in return for

7% of your company. In this case, n is .07 and 1/(1 - n) is 1.075.

So you should take the deal if you believe we can improve your

average outcome by more than 7.5%. If we improve your outcome by

10%, you're net ahead, because the remaining .93 you hold is worth

.93 x 1.1 = 1.023.

[1]One of the things the equity equation shows us is that, financially

at least, taking money from a top VC firm can be a really good deal.

Greg Mcadoo from Sequoia recently said at a YC dinner that when

Sequoia invests alone they like to take about 30% of a company.

1/.7 = 1.43, meaning that deal is worth taking if they can improve

your outcome by more than 43%. For the average startup, that would

be an extraordinary bargain. It would improve the average startup's

prospects by more than 43% just to be able to say they were funded

by Sequoia, even if they never actually got the money.The reason Sequoia is such a good deal is that the percentage of

the company they take is artificially low. They don't even try to

get market price for their investment; they limit their holdings

to leave the founders enough stock to feel the company is still

theirs.The catch is that Sequoia gets about 6000 business plans a year and

funds about 20 of them, so the odds of getting this great deal are

1 in 300. The companies that make it through are not average startups.Of course, there are other factors to consider in a VC deal. It's

never just a straight trade of money for stock. But if it were,

taking money from a top firm would generally be a bargain.You can use the same formula when giving stock to employees, but

it works in the other direction. If i is the average outcome for

the company with the addition of some new person, then they're worth

n such that i = 1/(1 - n). Which means n = (i - 1)/i.For example, suppose you're just two founders and you want to hire

an additional hacker who's so good you feel he'll increase the

average outcome of the whole company by 20%. n = (1.2 - 1)/1.2 =

.167. So you'll break even if you trade 16.7% of the company

for him.That doesn't mean 16.7% is the right amount of stock to give him.

Stock is not the only cost of hiring someone: there's usually salary

and overhead as well. And if the company merely breaks even on the

deal, there's no reason to do it.I think to translate salary and overhead into stock you should

multiply the annual rate by about 1.5. Most startups grow fast or

die; if you die you don't have to pay the guy, and if you grow fast

you'll be paying next year's salary out of next year's valuation,

which should be 3x this year's. If your valuation grows 3x a year,

the total cost in stock of a new hire's salary and overhead is 1.5

years' cost at the present valuation. [2]How much of an additional margin should the company need as the

"activation energy" for the deal? Since this is in effect the

company's profit on a hire, the market will determine that: if

you're a hot opportunity, you can charge more.Let's run through an example. Suppose the company wants to make a

"profit" of 50% on the new hire mentioned above. So subtract a

third from 16.7% and we have 11.1% as his "retail" price. Suppose

further that he's going to cost $60k a year in salary and overhead,

x 1.5 = $90k total. If the company's valuation is $2 million, $90k

is 4.5%. 11.1% - 4.5% = an offer of 6.6%.Incidentally, notice how important it is for early employees to

take little salary. It comes right out of stock that could otherwise

be given to them.Obviously there is a great deal of play in these numbers. I'm not

claiming that stock grants can now be reduced to a formula. Ultimately

you always have to guess. But at least know what you're guessing.

If you choose a number based on your gut feel, or a table of typical

grant sizes supplied by a VC firm, understand what those are estimates

of.And more generally, when you make any decision involving equity,

run it through 1/(1 - n) to see if it makes sense. You should

always feel richer after trading equity. If the trade didn't

increase the value of your remaining shares enough to put you net

ahead, you wouldn't have (or shouldn't have) done it.Notes[1] This is why we

can't believe anyone would think Y Combinator was a bad deal. Does

anyone really think we're so useless that in three months we can't

improve a startup's prospects by 6.4%?

[2] The obvious choice

for your present valuation is the post-money valuation of your last

funding round. This probably undervalues the company, though,

because (a) unless your last round just happened, the company is

presumably worth more, and (b) the valuation of an early funding

round usually reflects some other contribution by the investors.Thanks to Sam Altman, Trevor Blackwell, Paul Buchheit,

Hutch Fishman, David Hornik, Paul Kedrosky, Jessica Livingston, Gary Sabot, and

Joshua Schachter for reading drafts of this.

An Alternative Theory of Unions

May 2007People who worry about the increasing gap between rich and poor

generally look back on the mid twentieth century as a golden age.

In those days we had a large number of high-paying union manufacturing

jobs that boosted the median income. I wouldn't quite call the

high-paying union job a myth, but I think people who dwell on it

are reading too much into it.Oddly enough, it was working with startups that made me realize

where the high-paying union job came from. In a rapidly growing

market, you don't worry too much about efficiency. It's more

important to grow fast. If there's some mundane problem getting

in your way, and there's a simple solution that's somewhat expensive,

just take it and get on with more important things. EBay didn't

win by paying less for servers than their competitors.Difficult though it may be to imagine now, manufacturing was a

growth industry in the mid twentieth century. This was an era when

small firms making everything from cars to candy were getting

consolidated into a new kind of corporation with national reach and

huge economies of scale. You had to grow fast or die. Workers

were for these companies what servers are for an Internet startup.

A reliable supply was more important than low cost.If you looked in the head of a 1950s auto executive, the attitude

must have been: sure, give 'em whatever they ask for, so long as

the new model isn't delayed.In other words, those workers were not paid what their work was

worth. Circumstances being what they were, companies would have

been stupid to insist on paying them so little.If you want a less controversial example of this phenomenon, ask

anyone who worked as a consultant building web sites during the

Internet Bubble. In the late nineties you could get paid huge sums

of money for building the most trivial things. And yet does anyone

who was there have any expectation those days will ever return? I

doubt it. Surely everyone realizes that was just a temporary

aberration.The era of labor unions seems to have been the same kind of aberration,

just spread

over a longer period, and mixed together with a lot of ideology

that prevents people from viewing it with as cold an eye as they

would something like consulting during the Bubble.Basically, unions were just Razorfish.People who think the labor movement was the creation of heroic union

organizers have a problem to explain: why are unions shrinking now?

The best they can do is fall back on the default explanation of

people living in fallen civilizations. Our ancestors were giants.

The workers of the early twentieth century must have had a moral

courage that's lacking today.In fact there's a simpler explanation. The early twentieth century

was just a fast-growing startup overpaying for infrastructure. And

we in the present are not a fallen people, who have abandoned

whatever mysterious high-minded principles produced the high-paying

union job. We simply live in a time when the fast-growing companies

overspend on different things.

The Hacker's Guide to Investors

April 2007(This essay is derived from a keynote talk at the 2007 ASES Summit

at Stanford.)The world of investors is a foreign one to most hackers—partly

because investors are so unlike hackers, and partly because they

tend to operate in secret. I've been dealing with this world for

many years, both as a founder and an investor, and I still don't

fully understand it.In this essay I'm going to list some of the more surprising things

I've learned about investors. Some I only learned in the past year.Teaching hackers how to deal with investors is probably the second

most important thing we do at Y Combinator. The most important

thing for a startup is to make something good. But everyone knows

that's important. The dangerous thing about investors is that

hackers don't know how little they know about this strange world.1. The investors are what make a startup hub.About a year ago I tried to figure out what you'd need to reproduce

Silicon Valley. I decided the

critical ingredients were rich people

and nerds—investors and founders. People are all you need to

make technology, and all the other people will move.If I had to narrow that down, I'd say investors are the limiting

factor. Not because they contribute more to the startup, but simply

because they're least willing to move. They're rich. They're not

going to move to Albuquerque just because there are some smart

hackers there they could invest in. Whereas hackers will move to

the Bay Area to find investors.2. Angel investors are the most critical.There are several types of investors. The two main categories are

angels and VCs: VCs invest other people's money, and angels invest

their own.Though they're less well known, the angel investors are probably

the more critical ingredient in creating a silicon valley. Most

companies that VCs invest in would never have made it that far if angels

hadn't invested first. VCs say between half and three quarters of

companies that raise series A rounds have taken some outside

investment already.

[1]Angels are willing to fund riskier projects than VCs. They also

give valuable advice, because (unlike VCs) many have been startup

founders themselves.Google's story shows the key role angels play. A lot of people know

Google raised money from Kleiner and Sequoia. What most don't realize

is how late. That VC round was a series B round; the premoney

valuation was $75 million. Google was already a successful company

at that point. Really, Google was funded with angel money.It may seem odd that the canonical Silicon Valley startup was funded

by angels, but this is not so surprising. Risk is always proportionate

to reward. So the most successful startup of all is likely to have

seemed an extremely risky bet at first, and that is exactly the

kind VCs won't touch.Where do angel investors come from? From other startups. So startup

hubs like Silicon Valley benefit from something like the marketplace

effect, but shifted in time: startups are there because startups

were there.3. Angels don't like publicity.If angels are so important, why do we hear more about VCs? Because

VCs like publicity. They need to market themselves to the investors

who are their "customers"—the endowments and pension funds and

rich families whose money they invest—and also to founders who

might come to them for funding.Angels don't need to market themselves to investors because they

invest their own money. Nor do they want to market themselves to

founders: they don't want random people pestering them with business

plans. Actually, neither do VCs. Both angels and VCs get deals

almost exclusively through personal introductions.

[2]The reason VCs want a strong brand is not to draw in more business

plans over the transom, but so they win deals when competing

against other VCs. Whereas angels are rarely in direct competition,

because (a) they do fewer deals, (b) they're happy to split them,

and (c) they invest at a point where the stream is broader.4. Most investors, especially VCs, are not like founders.Some angels are, or were, hackers. But most VCs are a different

type of people: they're dealmakers.If you're a hacker, here's a thought experiment you can run to

understand why there are basically no hacker VCs: How would you

like a job where you never got to make anything, but instead spent

all your time listening to other people pitch (mostly terrible)

projects, deciding whether to fund them, and sitting on their boards

if you did? That would not be fun for most hackers. Hackers like

to make things. This would be like being an administrator.Because most VCs are a different species of people from

founders, it's hard to know what they're thinking. If you're a

hacker, the last time you had to deal with these guys was in high

school. Maybe in college you walked past their fraternity on your

way to the lab. But don't underestimate them. They're as expert

in their world as you are in yours. What they're good at is reading

people, and making deals work to their advantage. Think twice

before you try to beat them at that.5. Most investors are momentum investors.Because most investors are dealmakers rather than technology people,

they generally don't understand what you're doing. I knew as a

founder that most VCs didn't get technology. I also knew some made

a lot of money. And yet it never occurred to me till recently to

put those two ideas together and ask "How can VCs make money by

investing in stuff they don't understand?"The answer is that they're like momentum investors. You can (or

could once) make a lot of money by noticing sudden changes in stock

prices. When a stock jumps upward, you buy, and when it suddenly

drops, you sell. In effect you're insider trading, without knowing

what you know. You just know someone knows something, and that's

making the stock move.This is how most venture investors operate. They don't try to look

at something and predict whether it will take off. They win by

noticing that something is taking off a little sooner than everyone

else. That generates almost as good returns as actually being able

to pick winners. They may have to pay a little more than they would

if they got in at the very beginning, but only a little.Investors always say what they really care about is the team.

Actually what they care most about is your traffic, then what other

investors think, then the team. If you don't yet have any traffic,

they fall back on number 2, what other investors think. And this,

as you can imagine, produces wild oscillations in the "stock price"

of a startup. One week everyone wants you, and they're begging not

to be cut out of the deal. But all it takes is for one big investor

to cool on you, and the next week no one will return your phone

calls. We regularly have startups go from hot to cold or cold to

hot in a matter of days, and literally nothing has changed.There are two ways to deal with this phenomenon. If you're feeling

really confident, you can try to ride it. You can start by asking

a comparatively lowly VC for a small amount of money, and then after

generating interest there, ask more prestigious VCs for larger

amounts, stirring up a crescendo of buzz, and then "sell" at the

top. This is extremely risky, and takes months even if you succeed.

I wouldn't try it myself. My advice is to err on the side of safety:

when someone offers you a decent deal, just take it and get on with

building the company. Startups win or lose based on the quality

of their product, not the quality of their funding deals.6. Most investors are looking for big hits.Venture investors like companies that could go public. That's where

the big returns are. They know the odds of any individual startup

going public are small, but they want to invest in those that at

least have a chance of going public.Currently the way VCs seem to operate is to invest in a bunch of

companies, most of which fail, and one of which is Google. Those

few big wins compensate for losses on their other investments. What this

means is that most VCs will only invest in you if you're a potential

Google. They don't care about companies that are a safe bet to be

acquired for $20 million. There needs to be a chance, however

small, of the company becoming really big.Angels are different in this respect. They're happy to invest in

a company where the most likely outcome is a $20 million acquisition

if they can do it at a low enough valuation. But of course they

like companies that could go public too. So having an ambitious

long-term plan pleases everyone.If you take VC money, you have to mean it, because the structure

of VC deals prevents early acquisitions. If you take VC money,

they won't let you sell early.7. VCs want to invest large amounts.The fact that they're running investment funds makes VCs want to

invest large amounts. A typical VC fund is now hundreds of millions

of dollars. If $400 million has to be invested by 10 partners,

they have to invest $40 million each. VCs usually sit on the boards

of companies they fund. If the average deal size was $1 million,

each partner would have to sit on 40 boards, which would not be

fun. So they prefer bigger deals, where they can put a lot of money

to work at once.VCs don't regard you as a bargain if you don't need a lot of money.

That may even make you less attractive, because it means their

investment creates less of a barrier to entry for competitors.Angels are in a different position because they're investing their

own money. They're happy to invest small amounts—sometimes as

little as $20,000—as long as the potential returns look good

enough. So if you're doing something inexpensive, go to angels.8. Valuations are fiction.VCs admit that valuations are an artifact. They decide how much

money you need and how much of the company they want, and those two

constraints yield a valuation.Valuations increase as the size of the investment does. A company

that an angel is willing to put $50,000 into at a valuation of a

million can't take $6 million from VCs at that valuation. That

would leave the founders less than a seventh of the company between

them (since the option pool would also come out of that seventh).

Most VCs wouldn't want that, which is why you never hear of deals

where a VC invests $6 million at a premoney valuation of $1 million.If valuations change depending on the amount invested, that shows

how far they are from reflecting any kind of value of the company.Since valuations are made up, founders shouldn't care too much about

them. That's not the part to focus on. In fact, a high valuation

can be a bad thing. If you take funding at a premoney valuation

of $10 million, you won't be selling the company for 20. You'll

have to sell for over 50 for the VCs to get even a 5x return, which

is low to them. More likely they'll want you to hold out for 100.

But needing to get a high price decreases the chance of getting

bought at all; many companies can buy you for $10 million, but only

a handful for 100. And since a startup is like a pass/fail course

for the founders, what you want to optimize is your chance of a

good outcome, not the percentage of the company you keep.So why do founders chase high valuations? They're tricked by

misplaced ambition. They feel they've achieved more if they get a

higher valuation. They usually know other founders, and if they

get a higher valuation they can say "mine is bigger than yours."

But funding is not the real test. The real test is the final outcome

for the founder, and getting too high a valuation may just make a

good outcome less likely.The one advantage of a high valuation is that you get less dilution.

But there is another less sexy way to achieve that: just take less

money.9. Investors look for founders like the current stars.Ten years ago investors were looking for the next Bill Gates. This

was a mistake, because Microsoft was a very anomalous startup. They

started almost as a contract programming operation, and the reason

they became huge was that IBM happened to drop the PC standard in

their lap.Now all the VCs are looking for the next Larry and Sergey. This

is a good trend, because Larry and Sergey are closer to the ideal

startup founders.Historically investors thought it was important for a founder to

be an expert in business. So they were willing to fund teams of

MBAs who planned to use the money to pay programmers to build their

product for them. This is like funding Steve Ballmer in the hope

that the programmer he'll hire is Bill Gates—kind of backward,

as the events of the Bubble showed. Now most VCs know they should

be funding technical guys. This is more pronounced among the very

top funds; the lamer ones still want to fund MBAs.If you're a hacker, it's good news that investors are looking for

Larry and Sergey. The bad news is, the only investors who can do

it right are the ones who knew them when they were a

couple of CS grad students, not the confident media stars they are

today. What investors still don't get is how clueless and tentative

great founders can seem at the very beginning.10. The contribution of investors tends to be underestimated.Investors do more for startups than give them money. They're helpful

in doing deals and arranging introductions, and some of the smarter

ones, particularly angels, can give good advice about the product.In fact, I'd say what separates the great investors from the mediocre

ones is the quality of their advice. Most investors give advice,

but the top ones give good advice.Whatever help investors give a startup tends to be underestimated.

It's to everyone's advantage to let the world think the founders

thought of everything. The goal of the investors is for the company

to become valuable, and the company seems more valuable if it seems

like all the good ideas came from within.This trend is compounded by the obsession that the press has with

founders. In a company founded by two people, 10% of the ideas

might come from the first guy they hire. Arguably they've done a

bad job of hiring otherwise. And yet this guy will be almost

entirely overlooked by the press.I say this as a founder: the contribution of founders is always

overestimated. The danger here is that new founders, looking at

existing founders, will think that they're supermen that one couldn't

possibly equal oneself. Actually they have a hundred different

types of support people just offscreen making the whole show possible.

[3]11. VCs are afraid of looking bad.I've been very surprised to discover how timid most VCs are. They

seem to be afraid of looking bad to their partners, and perhaps

also to the limited partners—the people whose money they invest.You can measure this fear in how much less risk VCs are willing to

take. You can tell they won't make investments for their fund that

they might be willing to make themselves as angels. Though it's

not quite accurate to say that VCs are less willing to take risks.

They're less willing to do things that might look bad. That's not

the same thing.For example, most VCs would be very reluctant to invest in a startup

founded by a pair of 18 year old hackers, no matter how brilliant,

because if the startup failed their partners could turn on them and

say "What, you invested $x million of our money in a pair of 18

year olds?" Whereas if a VC invested in a startup founded by

three former banking executives in their 40s who planned to outsource

their product development—which to my mind is actually a lot

riskier than investing in a pair of really smart 18 year olds—he

couldn't be faulted, if it failed, for making such an apparently

prudent investment.As a friend of mine said, "Most VCs can't do anything that would

sound bad to the kind of doofuses who run pension funds." Angels

can take greater risks because they don't have to answer to anyone.12. Being turned down by investors doesn't mean much.Some founders are quite dejected when they get turned down by

investors. They shouldn't take it so much to heart. To start with,

investors are often wrong. It's hard to think of a successful

startup that wasn't turned down by investors at some point. Lots

of VCs rejected Google. So obviously the reaction of investors is

not a very meaningful test.Investors will often reject you for what seem to be superficial

reasons. I read of one VC who

turned

down a startup simply because

they'd given away so many little bits of stock that the deal required

too many signatures to close.

[4]

The reason investors can get away

with this is that they see so many deals. It doesn't matter if

they underestimate you because of some surface imperfection, because

the next best deal will be almost as good.

Imagine picking out

apples at a grocery store. You grab one with a little bruise.

Maybe it's just a surface bruise, but why even bother checking when

there are so many other unbruised apples to choose from?Investors would be the first to admit they're often wrong. So when

you get rejected by investors, don't think "we suck," but instead

ask "do we suck?" Rejection is a question, not an answer.13. Investors are emotional.I've been surprised to discover how emotional investors can be.

You'd expect them to be cold and calculating, or at least businesslike,

but often they're not. I'm not sure if it's their position of power

that makes them this way, or the large sums of money involved, but

investment negotiations can easily turn personal. If you offend

investors, they'll leave in a huff.A while ago an eminent VC firm offered a series A round to a startup

we'd seed funded. Then they heard a rival VC firm was also interested.

They were so afraid that they'd be rejected in favor of this other

firm that they gave the startup what's known as an "exploding

termsheet." They had, I think, 24 hours to say yes or no, or the

deal was off. Exploding termsheets are a somewhat dubious device,

but not uncommon. What surprised me was their reaction when I

called to talk about it. I asked if they'd still be interested in

the startup if the rival VC didn't end up making an offer, and they

said no. What rational basis could they have had for saying that?

If they thought the startup was worth investing in, what difference

should it make what some other VC thought? Surely it was their

duty to their limited partners simply to invest in the best

opportunities they found; they should be delighted if the other VC

said no, because it would mean they'd overlooked a good opportunity.

But of course there was no rational basis for their decision. They

just couldn't stand the idea of taking this rival firm's rejects.In this case the exploding termsheet was not (or not only) a tactic

to pressure the startup. It was more like the high school trick

of breaking up with someone before they can break up with you. In

an earlier essay I said that VCs were a lot like high school girls.

A few VCs have joked about that characterization, but none have

disputed it.14. The negotiation never stops till the closing.Most deals, for investment or acquisition, happen in two phases.

There's an initial phase of negotiation about the big questions.

If this succeeds you get a termsheet, so called because it outlines

the key terms of a deal. A termsheet is not legally binding,

but it is a definite step. It's supposed to mean that a

deal is going to happen, once the lawyers work out all the details.

In theory these details are minor ones; by definition all the

important points are supposed to be covered in the termsheet.Inexperience and wishful thinking combine to make founders feel

that when they have a termsheet, they have a deal. They want there

to be a deal; everyone acts like they have a deal; so there must

be a deal. But there isn't and may not be for several months. A

lot can change for a startup in several months. It's not uncommon

for investors and acquirers to get buyer's remorse. So you have

to keep pushing, keep selling, all the way to the close. Otherwise

all the "minor" details left unspecified in the termsheet will be

interpreted to your disadvantage. The other side may even break

the deal; if they do that, they'll usually seize on some technicality

or claim you misled them, rather than admitting they changed their

minds.It can be hard to keep the pressure on an investor or acquirer all

the way to the closing, because the most effective pressure is

competition from other investors or acquirers, and these tend to

drop away when you get a termsheet. You should try to stay as close

friends as you can with these rivals, but the most important thing

is just to keep up the momentum in your startup. The investors or

acquirers chose you because you seemed hot. Keep doing whatever

made you seem hot. Keep releasing new features; keep getting new

users; keep getting mentioned in the press and in blogs.15. Investors like to co-invest.I've been surprised how willing investors are to split deals. You

might think that if they found a good deal they'd want it all to

themselves, but they seem positively eager to syndicate. This is

understandable with angels; they invest on a smaller scale and don't

like to have too much money tied up in any one deal. But VCs also

share deals a lot. Why?Partly I think this is an artifact of the rule I quoted earlier:

after traffic, VCs care most what other VCs think. A deal that has

multiple VCs interested in it is more likely to close, so of deals

that close, more will have multiple investors.There is one rational reason to want multiple VCs in a deal: Any

investor who co-invests with you is one less investor who could

fund a competitor. Apparently Kleiner and Sequoia didn't like

splitting the Google deal, but it did at least have the advantage,

from each one's point of view, that there probably wouldn't be a

competitor funded by the other. Splitting deals thus has similar

advantages to confusing paternity.But I think the main reason VCs like splitting deals is the fear

of looking bad. If another firm shares the deal, then in the event

of failure it will seem to have been a prudent choice—a consensus

decision, rather than just the whim of an individual partner.16. Investors collude.Investing is not covered by antitrust law. At least, it better not

be, because investors regularly do things that would be illegal

otherwise. I know personally of cases where one investor has talked

another out of making a competitive offer, using the promise of

sharing future deals.In principle investors are all competing for the same deals, but

the spirit of cooperation is stronger than the spirit of competition.

The reason, again, is that there are so many deals. Though a

professional investor may have a closer relationship with a founder

he invests in than with other investors, his relationship with the

founder is only going to last a couple years, whereas his relationship

with other firms will last his whole career. There isn't so much

at stake in his interactions with other investors, but there will

be a lot of them. Professional investors are constantly trading

little favors.Another reason investors stick together is to preserve the power

of investors as a whole. So you will not, as of this writing, be

able to get investors into an auction for your series A round.

They'd rather lose the deal than establish a precedent of VCs

competitively bidding against one another. An efficient startup

funding market may be coming in the distant future; things tend to

move in that direction; but it's certainly not here now.

17. Large-scale investors care about their portfolio, not any

individual company.The reason startups work so well is that everyone with power also

has equity. The only way any of them can succeed is if they all

do. This makes everyone naturally pull in the same direction,

subject to differences of opinion about tactics.The problem is, larger scale investors don't have exactly the same

motivation. Close, but not identical. They don't need any given

startup to succeed, like founders do, just their portfolio as a

whole to. So in borderline cases the rational thing for them to

do is to sacrifice unpromising startups.Large-scale investors tend to put startups in three categories:

successes, failures, and the "living dead"—companies that are

plugging along but don't seem likely in the immediate future to get

bought or go public. To the founders, "living dead" sounds harsh.

These companies may be far from failures by ordinary standards. But

they might as well be from a venture investor's point of view, and

they suck up just as much time and attention as the successes. So

if such a company has two possible strategies, a conservative one

that's slightly more likely to work in the end, or a risky one that

within a short time will either yield a giant success or kill the

company, VCs will push for the kill-or-cure option. To them the

company is already a write-off. Better to have resolution, one way

or the other, as soon as possible.If a startup gets into real trouble, instead of trying to save it

VCs may just sell it at a low price to another of their portfolio

companies. Philip Greenspun said in Founders at Work that Ars Digita's VCs did this to them.18. Investors have different risk profiles from founders.Most people would rather a 100% chance of $1 million than a 20%

chance of $10 million. Investors are rich enough to be rational

and prefer the latter. So they'll always tend to encourage founders

to keep rolling the dice. If a company is doing well, investors

will want founders to turn down most acquisition offers. And indeed,

most startups that turn down acquisition offers ultimately do better.

But it's still hair-raising for the founders, because they might

end up with nothing. When someone's offering to buy you for a price

at which your stock is worth $5 million, saying no is equivalent

to having $5 million and betting it all on one spin of the roulette

wheel.Investors will tell you the company is worth more. And they may

be right. But that doesn't mean it's wrong to sell. Any financial

advisor who put all his client's assets in the stock of a single,

private company would probably lose his license for it.More and more, investors are letting founders cash out partially.

That should correct the problem. Most founders have such low standards

that they'll feel rich with a sum that doesn't seem huge to investors.

But this custom is spreading too slowly, because VCs are afraid of

seeming irresponsible. No one wants to be the first VC to give

someone fuck-you money and then actually get told "fuck you." But

until this does start to happen, we know VCs are being too conservative.19. Investors vary greatly.Back when I was a founder I used to think all VCs were the same.

And in fact they do all look

the same. They're all what hackers

call "suits." But since I've been dealing with VCs more I've learned

that some suits are smarter than others.They're also in a business where winners tend to keep winning and

losers to keep losing. When a VC firm has been successful in the

past, everyone wants funding from them, so they get the pick of all

the new deals. The self-reinforcing nature of the venture funding

market means that the top ten firms live in a completely different

world from, say, the hundredth. As well as being smarter, they

tend to be calmer and more upstanding; they don't need to do iffy

things to get an edge, and don't want to because they have more

brand to protect.There are only two kinds of VCs you want to take money from, if you

have the luxury of choosing: the "top tier" VCs, meaning about the

top 20 or so firms, plus a few new ones that are not among the top

20 only because they haven't been around long enough.It's particularly important to raise money from a top firm if you're

a hacker, because they're more confident. That means they're less

likely to stick you with a business guy as CEO, like VCs used to

do in the 90s. If you seem smart and want to do it, they'll let

you run the company.20. Investors don't realize how much it costs to raise money from

them.Raising money is a huge time suck at just the point where startups

can least afford it. It's not unusual for it to take five or six

months to close a funding round. Six weeks is fast. And raising

money is not just something you can leave running as a background

process. When you're raising money, it's inevitably the main focus

of the company. Which means building the product isn't.Suppose a Y Combinator company starts talking to VCs after demo

day, and is successful in raising money from them, closing the deal

after a comparatively short 8 weeks. Since demo day occurs after

10 weeks, the company is now 18 weeks old. Raising money, rather

than working on the product, has been the company's main focus for

44% of its existence. And mind you, this an example where things

turned out well.When a startup does return to working on the product after a funding

round finally closes, it's as if they were returning to work after

a months-long illness. They've lost most of their momentum.Investors have no idea how much they damage the companies they

invest in by taking so long to do it. But companies do. So there

is a big opportunity here for a new kind of venture fund that invests

smaller amounts at lower valuations, but promises to either close

or say no very quickly. If there were such a firm, I'd recommend

it to startups in preference to any other, no matter how prestigious.

Startups live on speed and momentum.21. Investors don't like to say no.The reason funding deals take so long to close is mainly that

investors can't make up their minds. VCs are not big companies;

they can do a deal in 24 hours if they need to. But they usually

let the initial meetings stretch out over a couple weeks. The

reason is the selection algorithm I mentioned earlier. Most don't

try to predict whether a startup will win, but to notice quickly

that it already is winning. They care what the market thinks of

you and what other VCs think of you, and they can't judge those

just from meeting you.Because they're investing in things that (a) change fast and (b)

they don't understand, a lot of investors will reject you in a way

that can later be claimed not to have been a rejection. Unless you

know this world, you may not even realize you've been rejected.

Here's a VC saying no:

We're really excited about your project, and we want to keep in

close touch as you develop it further.

Translated into more straightforward language, this means: We're

not investing in you, but we may change our minds if it looks like

you're taking off. Sometimes they're more candid and say explicitly

that they need to "see some traction." They'll invest in you if

you start to get lots of users. But so would any VC. So all they're

saying is that you're still at square 1.Here's a test for deciding whether a VC's response was yes or no.

Look down at your hands. Are you holding a termsheet?22. You need investors.Some founders say "Who needs investors?" Empirically the answer

seems to be: everyone who wants to succeed. Practically every

successful startup takes outside investment at some point.Why? What the people who think they don't need investors forget is

that they will have competitors. The question is not whether you

need outside investment, but whether it could help you at all.

If the answer is yes, and you don't take investment, then competitors

who do will have an advantage over you. And in the startup world

a little advantage can expand into a lot.Mike Moritz famously said that he invested in Yahoo because he

thought they had a few weeks' lead over their competitors. That

may not have mattered quite so much as he thought, because Google

came along three years later and kicked Yahoo's ass. But there is

something in what he said. Sometimes a small lead can grow into

the yes half of a binary choice.Maybe as it gets cheaper to start a startup, it will start to be

possible to succeed in a competitive market without outside funding.

There are certainly

costs to raising money. But as of this writing the empirical

evidence says it's a net win.23. Investors like it when you don't need them.A lot of founders approach investors as if they needed their

permission to start a company—as if it were like getting into

college. But you don't need investors to start most companies;

they just make it easier.And in fact, investors greatly prefer it if you don't need them.

What excites them, both consciously and unconsciously, is the sort

of startup that approaches them saying "the train's leaving the

station; are you in or out?" not the one saying "please can we have

some money to start a company?"Most investors are "bottoms" in the sense that the startups they

like most are those that are rough with them. When Google stuck

Kleiner and Sequoia with a $75 million premoney valuation, their

reaction was probably "Ouch! That feels so good." And they were

right, weren't they? That deal probably made them more than any

other they've done.The thing is, VCs are pretty good at reading people. So don't try

to act tough with them unless you really are the next Google, or

they'll see through you in a second. Instead of acting tough, what

most startups should do is simply always have a backup plan. Always

have some alternative plan for getting started if any given investor

says no. Having one is the best insurance against needing one.So you shouldn't start a startup that's expensive to start, because

then you'll be at the mercy of investors. If you ultimately want

to do something that will cost a lot, start by doing a cheaper

subset of it, and expand your ambitions when and if you raise more

money.Apparently the most likely animals to be left alive after a nuclear

war are cockroaches, because they're so hard to kill. That's what

you want to be as a startup, initially. Instead of a beautiful

but fragile flower that needs to have its stem in a plastic tube

to support itself, better to be small, ugly, and indestructible.Notes[1]

I may be underestimating VCs. They may play some behind the scenes

role in IPOs, which you ultimately need if you want to create a silicon

valley.[2]

A few VCs have an email address you can send your business

plan to, but the number of startups that get funded this way is

basically zero. You should always get a personal introduction—and

to a partner, not an associate.[3]

Several people have told us that the most valuable thing about

startup school

was that they got to see famous startup founders and realized

they were just ordinary guys. Though we're happy to provide this

service, this is not generally the way we pitch startup school to

potential speakers.[4]

Actually this sounds to me like a VC who got buyer's remorse,

then used a technicality to get out of the deal. But it's telling

that it even seemed a plausible excuse.Thanks to Sam Altman, Paul Buchheit, Hutch Fishman, and Robert

Morris for reading drafts of

this, and to Kenneth King of ASES for inviting me to speak.

Comment on this essay.

Two Kinds of Judgement

April 2007There are two different ways people judge you. Sometimes judging

you correctly is the end goal. But there's a second much more

common type of judgement where it isn't. We tend to regard all

judgements of us as the first type. We'd probably be happier if

we realized which are and which aren't.The first type of judgement, the type where judging you is the end

goal, include court cases, grades in classes, and most competitions.

Such judgements can of course be mistaken, but because the goal is

to judge you correctly, there's usually some kind of appeals process.

If you feel you've been misjudged, you can protest that you've been

treated unfairly.Nearly all the judgements made on children are of this type, so we

get into the habit early in life of thinking that all judgements

are.But in fact there is a second much larger class of judgements where

judging you is only a means to something else. These include college

admissions, hiring and investment decisions, and of course the

judgements made in dating. This kind of judgement is not really

about you.Put yourself in the position of someone selecting players for a

national team. Suppose for the sake of simplicity that this is a

game with no positions, and that you have to select 20 players.

There will be a few stars who clearly should make the team, and

many players who clearly shouldn't. The only place your judgement

makes a difference is in the borderline cases. Suppose you screw

up and underestimate the 20th best player, causing him not to make

the team, and his place to be taken by the 21st best. You've still

picked a good team. If the players have the usual distribution of

ability, the 21st best player will be only slightly worse than the

20th best. Probably the difference between them will be less than

the measurement error.The 20th best player may feel he has been misjudged. But your goal

here wasn't to provide a service estimating people's ability. It

was to pick a team, and if the difference between the 20th and 21st

best players is less than the measurement error, you've still done

that optimally.It's a false analogy even to use the word unfair to describe this

kind of misjudgement. It's not aimed at producing a correct estimate

of any given individual, but at selecting a reasonably optimal set.One thing that leads us astray here is that the selector seems to

be in a position of power. That makes him seem like a judge. If

you regard someone judging you as a customer instead of a judge,

the expectation of fairness goes away. The author of a good novel

wouldn't complain that readers were unfair for preferring a

potboiler with a racy cover. Stupid, perhaps, but not unfair.Our early training and our self-centeredness combine to make us

believe that every judgement of us is about us. In fact most aren't.

This is a rare case where being less self-centered will make people

more confident. Once you realize how little most people judging

you care about judging you accurately—once you realize that because

of the normal distribution of most applicant pools, it matters least

to judge accurately in precisely the cases where judgement has the

most effect—you won't take rejection so personally.And curiously enough, taking rejection less personally may help you

to get rejected less often. If you think someone judging you will

work hard to judge you correctly, you can afford to be passive.

But the more you realize that most judgements are greatly influenced

by random, extraneous factors—that most people judging you are

more like a fickle novel buyer than a wise and perceptive

magistrate—the more you realize you can do things to influence the

outcome.One good place to apply this principle is in college applications.

Most high school students applying to college do it with the usual

child's mix of inferiority and self-centeredness: inferiority in

that they assume that admissions committees must be all-seeing;

self-centeredness in that they assume admissions committees care

enough about them to dig down into their application and figure out

whether they're good or not. These combine to make applicants

passive in applying and hurt when they're rejected. If college

applicants realized how quick and impersonal most selection processes

are, they'd make more effort to sell themselves, and take the outcome

less personally.Spanish TranslationRussian TranslationArabic Translation

Microsoft is Dead

April 2007A few days ago I suddenly realized Microsoft was dead. I was talking

to a young startup founder about how Google was different from

Yahoo. I said that Yahoo had been warped from the start by

their fear of Microsoft. That was why they'd positioned themselves

as a "media company" instead of a technology company. Then I looked

at his face and realized he didn't understand. It was as if I'd

told him how much girls liked Barry Manilow in the mid

80s. Barry who?Microsoft? He didn't say anything, but I could tell he didn't quite

believe anyone would be frightened of them.Microsoft cast

a shadow over the software world for almost 20 years

starting in the late 80s.

I can remember when it was IBM before them. I mostly ignored this

shadow. I never used Microsoft software, so it only affected me

indirectly—for example, in the spam I got from botnets. And

because I wasn't paying attention, I didn't notice when the shadow

disappeared.But it's gone now. I can sense that. No one is even afraid of

Microsoft anymore. They still make a lot of money—so does IBM,

for that matter. But they're not dangerous.When did Microsoft die, and of what? I know they seemed dangerous

as late as 2001, because I wrote an essay then

about how they were

less dangerous than they seemed. I'd guess they were dead by 2005.

I know when we started Y Combinator we didn't worry about Microsoft

as competition for the startups we funded. In fact, we've never

even invited them to the demo days we organize for startups to

present to investors. We invite Yahoo and Google and some other

Internet companies, but we've never bothered to invite Microsoft.

Nor has anyone there ever even sent us an email. They're in a

different world.What killed them? Four things, I think, all of them occurring

simultaneously in the mid 2000s.The most obvious is Google. There can only be one big man in town,

and they're clearly it. Google is the most dangerous company

now by far, in both the good and bad senses of the word. Microsoft

can at best limp along afterward.When did Google take the lead? There will be a tendency to push

it back to their IPO in August 2004, but they weren't setting the

terms of the debate then. I'd say they took the lead in

2005. Gmail was one of the things that put them over the edge.

Gmail showed they could do more than search.Gmail also showed how much you could do with web-based software,

if you took advantage of what later came to be called "Ajax." And

that was the second cause of Microsoft's death: everyone can see the

desktop is over. It now seems inevitable that applications will

live on the web—not just email, but everything, right up to

Photoshop. Even Microsoft sees that now.Ironically, Microsoft unintentionally helped create Ajax. The x

in Ajax is from the XMLHttpRequest object, which lets the browser

communicate with the server in the background while displaying a page.

(Originally the only way to communicate with the server was to

ask for a new page.) XMLHttpRequest was created by Microsoft in the late 90s

because they needed it for Outlook. What they didn't realize was

that it would be useful to a lot of other people too—in fact, to

anyone who wanted to make web apps work like desktop ones.The other critical component of Ajax is Javascript, the programming

language that runs in the browser. Microsoft saw the danger of

Javascript and tried to keep it broken for as long as they could.

[1]

But eventually the open source world won, by producing

Javascript libraries that grew over the brokenness of Explorer

the way a tree grows over barbed wire.The third cause of Microsoft's death was broadband Internet. Anyone

who cares can have fast Internet access

now. And the bigger the pipe to the server, the less you need the

desktop.The last nail in the coffin came, of all places, from Apple.

Thanks to OS X, Apple has come back from the dead in a way

that is extremely rare in technology.

[2]

Their victory is so complete that I'm now surprised when I come across

a computer running Windows. Nearly all the people we fund at Y

Combinator use Apple laptops. It was the same in the audience at

startup

school. All the computer people use Macs or Linux now. Windows is for

grandmas, like Macs used to be in the 90s. So not only does the

desktop no longer matter, no one who cares about computers uses

Microsoft's anyway.And of course Apple has Microsoft on the run in music

too, with TV and phones on the way.I'm glad Microsoft is dead. They were like Nero or

Commodus—evil

in the way only inherited power can make you. Because remember,

the Microsoft monopoly didn't begin with Microsoft. They got it

from IBM. The software business was overhung by a

monopoly from about the mid-1950s to about 2005. For practically

its whole existence, that is. One of the reasons "Web 2.0" has

such an air of euphoria about it is the feeling, conscious or not,

that this era of monopoly may finally be over.Of course, as a hacker I can't help thinking about how something

broken could be fixed. Is there some way Microsoft could come back?

In principle, yes. To see how, envision two things: (a) the amount

of cash Microsoft now has on hand, and (b) Larry and Sergey making

the rounds of all the search engines ten years ago trying to sell

the idea for Google for a million dollars, and being turned down

by everyone.The surprising fact is, brilliant hackers—dangerously brilliant

hackers—can be had very cheaply, by the standards of a

company as rich as Microsoft. They can't

hire smart people anymore,

but they could buy as many as they wanted for only an order of magnitude

more. So if they wanted to be a contender

again, this is how they could do it:

Buy all the good "Web 2.0" startups. They could get substantially

all of them for less than they'd have to pay for Facebook. Put them all in a building in Silicon Valley, surrounded by

lead shielding to protect them from any contact with Redmond.

I feel safe suggesting this, because they'd never do it. Microsoft's

biggest weakness is that they still don't realize how much they

suck. They still think they can write software in house. Maybe they

can, by the standards of the desktop world. But that world ended

a few years ago.I already know what the reaction to this essay will be. Half the

readers will say that Microsoft is still an enormously profitable

company, and that I should be more

careful about drawing conclusions based on what a few people think

in our insular little "Web 2.0" bubble. The other half, the younger

half, will complain that this is old news.See also: Microsoft is Dead: the Cliffs NotesNotes[1]

It doesn't take a conscious effort to make software incompatible.

All you have to do is not work too hard at fixing bugs—which, if

you're a big company, you produce in copious quantities. The

situation is analogous to the writing of "literary

theorists." Most don't try to be obscure; they just don't make an

effort to be clear. It wouldn't pay.[2]

In part because Steve Jobs got pushed out by John Sculley in

a way that's rare among technology companies. If Apple's board

hadn't made that blunder, they wouldn't have had to bounce back.Portuguese TranslationSimplified Chinese TranslationKorean Translation

Why to Not Not Start a Startup

Want to start a startup? Get funded by

Y Combinator.

March 2007(This essay is derived from talks at the 2007

Startup School and the Berkeley CSUA.)We've now been doing Y Combinator long enough to have some data

about success rates. Our first batch, in the summer of 2005, had

eight startups in it. Of those eight, it now looks as if at least

four succeeded. Three have been acquired:

Reddit was a merger of

two, Reddit and Infogami, and a third was acquired that we can't

talk about yet. Another from that batch was

Loopt, which is doing

so well they could probably be acquired in about ten minutes if

they wanted to.So about half the founders from that first summer, less than two

years ago, are now rich, at least by their standards. (One thing

you learn when you get rich is that there are many degrees of it.)I'm not ready to predict our success rate will stay as high as 50%.

That first batch could have been an anomaly. But we should be able

to do better than the oft-quoted (and probably made

up) standard figure of 10%. I'd feel safe aiming at 25%.Even the founders who fail don't seem to have such a bad time. Of

those first eight startups, three are now probably dead. In two

cases the founders just went on to do other things at the end of

the summer. I don't think they were traumatized by the experience.

The closest to a traumatic failure was Kiko, whose founders kept

working on their startup for a whole year before being squashed by

Google Calendar. But they ended up happy. They sold their software

on eBay for a quarter of a million dollars. After they paid back

their angel investors, they had about a year's salary each.

[1]

Then they immediately went on to start a new and much more exciting

startup, Justin.TV.So here is an even more striking statistic: 0% of that first batch

had a terrible experience. They had ups and downs, like every

startup, but I don't think any would have traded it for a job in a

cubicle. And that statistic is probably not an anomaly. Whatever

our long-term success rate ends up being, I think the rate of people

who wish they'd gotten a regular job will stay close to 0%.The big mystery to me is: why don't more people start startups? If

nearly everyone who does it prefers it to a regular job, and a

significant percentage get rich, why doesn't everyone want to do

this? A lot of people think we get thousands of applications for

each funding cycle. In fact we usually only get several hundred.

Why don't more people apply? And while it must seem to anyone

watching this world that startups are popping up like crazy, the

number is small compared to the number of people with the necessary

skills. The great majority of programmers still go straight from

college to cubicle, and stay there.It seems like people are not acting in their own interest. What's

going on? Well, I can answer that. Because of Y Combinator's

position at the very start of the venture funding process, we're

probably the world's leading experts on the psychology of people

who aren't sure if they want to start a company.There's nothing wrong with being unsure. If you're a hacker thinking

about starting a startup and hesitating before taking the leap,

you're part of a grand tradition. Larry and Sergey seem to have

felt the same before they started Google, and so did Jerry and Filo

before they started Yahoo. In fact, I'd guess the most successful

startups are the ones started by uncertain hackers rather than

gung-ho business guys.We have some evidence to support this. Several of the most successful

startups we've funded told us later that they only decided to apply

at the last moment. Some decided only hours before the deadline.The way to deal with uncertainty is to analyze it into components.

Most people who are reluctant to do something have about eight

different reasons mixed together in their heads, and don't know

themselves which are biggest. Some will be justified and some

bogus, but unless you know the relative proportion of each, you

don't know whether your overall uncertainty is mostly justified or

mostly bogus.So I'm going to list all the components of people's reluctance to

start startups, and explain which are real. Then would-be founders

can use this as a checklist to examine their own feelings.I admit my goal is to increase your self-confidence. But there are

two things different here from the usual confidence-building exercise.

One is that I'm motivated to be honest. Most people in the

confidence-building business have already achieved their goal when

you buy the book or pay to attend the seminar where they tell you

how great you are. Whereas if I encourage people to start startups

who shouldn't, I make my own life worse. If I encourage too many

people to apply to Y Combinator, it just means more work for me,

because I have to read all the applications.The other thing that's going to be different is my approach. Instead

of being positive, I'm going to be negative. Instead of telling

you "come on, you can do it" I'm going to consider all the reasons

you aren't doing it, and show why most (but not all) should be

ignored. We'll start with the one everyone's born with.1. Too youngA lot of people think they're too young to start a startup. Many

are right. The median age worldwide is about 27, so probably a

third of the population can truthfully say they're too young.What's too young? One of our goals with Y Combinator was to discover

the lower bound on the age of startup founders. It always seemed

to us that investors were too conservative here—that they wanted

to fund professors, when really they should be funding grad students

or even undergrads.The main thing we've discovered from pushing the edge of this

envelope is not where the edge is, but how fuzzy it is. The outer

limit may be as low as 16. We don't look beyond 18 because people

younger than that can't legally enter into contracts. But the most

successful founder we've funded so far, Sam Altman, was 19 at the

time.Sam Altman, however, is an outlying data point. When he was 19,

he seemed like he had a 40 year old inside him. There are other

19 year olds who are 12 inside.There's a reason we have a distinct word "adult" for people over a

certain age. There is a threshold you cross. It's conventionally

fixed at 21, but different people cross it at greatly varying ages.

You're old enough to start a startup if you've crossed this threshold,

whatever your age.How do you tell? There are a couple tests adults use. I realized

these tests existed after meeting Sam Altman, actually. I noticed

that I felt like I was talking to someone much older. Afterward I

wondered, what am I even measuring? What made him seem older?One test adults use is whether you still have the kid flake reflex.

When you're a little kid and you're asked to do something hard, you

can cry and say "I can't do it" and the adults will probably let

you off. As a kid there's a magic button you can press by saying

"I'm just a kid" that will get you out of most difficult situations.

Whereas adults, by definition, are not allowed to flake. They still

do, of course, but when they do they're ruthlessly pruned.The other way to tell an adult is by how they react to a challenge.

Someone who's not yet an adult will tend to respond to a challenge

from an adult in a way that acknowledges their dominance. If an

adult says "that's a stupid idea," a kid will either crawl away

with his tail between his legs, or rebel. But rebelling presumes

inferiority as much as submission. The adult response to

"that's a stupid idea," is simply to look the other person in the

eye and say "Really? Why do you think so?"There are a lot of adults who still react childishly to challenges,

of course. What you don't often find are kids who react to challenges

like adults. When you do, you've found an adult, whatever their

age.2. Too inexperiencedI once wrote that startup founders should be at least 23, and that

people should work for another company for a few years before

starting their own. I no longer believe that, and what changed my

mind is the example of the startups we've funded.I still think 23 is a better age than 21. But the best way to get

experience if you're 21 is to start a startup. So, paradoxically,

if you're too inexperienced to start a startup, what you should do

is start one. That's a way more efficient cure for inexperience

than a normal job. In fact, getting a normal job may actually make

you less able to start a startup, by turning you into a tame animal

who thinks he needs an office to work in and a product manager to

tell him what software to write.What really convinced me of this was the Kikos. They started a

startup right out of college. Their inexperience caused them to

make a lot of mistakes. But by the time we funded their second

startup, a year later, they had become extremely formidable. They

were certainly not tame animals. And there is no way they'd have

grown so much if they'd spent that year working at Microsoft, or

even Google. They'd still have been diffident junior programmers.So now I'd advise people to go ahead and start startups right out

of college. There's no better time to take risks than when you're

young. Sure, you'll probably fail. But even failure will get you

to the ultimate goal faster than getting a job.It worries me a bit to be saying this, because in effect we're

advising people to educate themselves by failing at our expense,

but it's the truth.3. Not determined enoughYou need a lot of determination to succeed as a startup founder.

It's probably the single best predictor of success.Some people may not be determined enough to make it. It's

hard for me to say for sure, because I'm so determined that I can't

imagine what's going on in the heads of people who aren't. But I

know they exist.Most hackers probably underestimate their determination. I've seen

a lot become visibly more determined as they get used to running a

startup. I can think of

several we've funded who would have been delighted at first to be

bought for $2 million, but are now set on world domination.How can you tell if you're determined enough, when Larry and Sergey

themselves were unsure at first about starting a company? I'm

guessing here, but I'd say the test is whether you're sufficiently

driven to work on your own projects. Though they may have been

unsure whether they wanted to start a company, it doesn't seem as

if Larry and Sergey were meek little research assistants, obediently

doing their advisors' bidding. They started projects of their own.

4. Not smart enoughYou may need to be moderately smart to succeed as a startup founder.

But if you're worried about this, you're probably mistaken. If

you're smart enough to worry that you might not be smart enough to

start a startup, you probably are.And in any case, starting a startup just doesn't require that much

intelligence. Some startups do. You have to be good at math to

write Mathematica. But most companies do more mundane stuff where

the decisive factor is effort, not brains. Silicon Valley can warp

your perspective on this, because there's a cult of smartness here.

People who aren't smart at least try to act that way. But if you

think it takes a lot of intelligence to get rich, try spending a

couple days in some of the fancier bits of New York or LA.If you don't think you're smart enough to start a startup doing

something technically difficult, just write enterprise software.

Enterprise software companies aren't technology companies, they're

sales companies, and sales depends mostly on effort.5. Know nothing about businessThis is another variable whose coefficient should be zero. You

don't need to know anything about business to start a startup. The

initial focus should be the product. All you need to know in this

phase is how to build things people want. If you succeed, you'll

have to think about how to make money from it. But this is so easy

you can pick it up on the fly.I get a fair amount of flak for telling founders just to make

something great and not worry too much about making money. And yet

all the empirical evidence points that way: pretty much 100% of

startups that make something popular manage to make money from it.

And acquirers tell me privately that revenue is not what they buy

startups for, but their strategic value. Which means, because they

made something people want. Acquirers know the rule holds for them

too: if users love you, you can always make money from that somehow,

and if they don't, the cleverest business model in the world won't

save you.So why do so many people argue with me? I think one reason is that

they hate the idea that a bunch of twenty year olds could get rich

from building something cool that doesn't make any money. They

just don't want that to be possible. But how possible it is doesn't

depend on how much they want it to be.For a while it annoyed me to hear myself described as some kind of

irresponsible pied piper, leading impressionable young hackers down

the road to ruin. But now I realize this kind of controversy is a

sign of a good idea.The most valuable truths are the ones most people don't believe.

They're like undervalued stocks. If you start with them, you'll

have the whole field to yourself. So when you find an idea you

know is good but most people disagree with, you should not

merely ignore their objections, but push aggressively in that

direction. In this case, that means you should seek out ideas that

would be popular but seem hard to make money from.We'll bet a seed round you can't make something popular that we

can't figure out how to make money from.6. No cofounderNot having a cofounder is a real problem. A startup is too much

for one person to bear. And though we differ from other investors

on a lot of questions, we all agree on this. All investors, without

exception, are more likely to fund you with a cofounder than without.We've funded two single founders, but in both cases we suggested

their first priority should be to find a cofounder. Both did. But

we'd have preferred them to have cofounders before they applied.

It's not super hard to get a cofounder for a project that's just

been funded, and we'd rather have cofounders committed enough to

sign up for something super hard.If you don't have a cofounder, what should you do? Get one. It's

more important than anything else. If there's no one where you

live who wants to start a startup with you, move where there are

people who do. If no one wants to work with you on your current

idea, switch to an idea people want to work on.If you're still in school, you're surrounded by potential cofounders.

A few years out it gets harder to find them. Not only do you have

a smaller pool to draw from, but most already have jobs, and perhaps

even families to support. So if you had friends in college you

used to scheme about startups with, stay in touch with them as well

as you can. That may help keep the dream alive.It's possible you could meet a cofounder through something like a

user's group or a conference. But I wouldn't be too optimistic.

You need to work with someone to know whether you want them as a

cofounder.

[2]The real lesson to draw from this is not how to find a cofounder,

but that you should start startups when you're young and there are

lots of them around.7. No ideaIn a sense, it's not a problem if you don't have a good idea, because

most startups change their idea anyway. In the average Y Combinator

startup, I'd guess 70% of the idea is new at the end of the

first three months. Sometimes it's 100%.In fact, we're so sure the founders are more important than the

initial idea that we're going to try something new this funding

cycle. We're going to let people apply with no idea at all. If you

want, you can answer the question on the application form that asks

what you're going to do with "We have no idea." If you seem really

good we'll accept you anyway. We're confident we can sit down with

you and cook up some promising project.Really this just codifies what we do already. We put little weight

on the idea. We ask mainly out of politeness. The kind of question

on the application form that we really care about is the one where

we ask what cool things you've made. If what you've made is version

one of a promising startup, so much the better, but the main thing

we care about is whether you're good at making things. Being lead

developer of a popular open source project counts almost as much.That solves the problem if you get funded by Y Combinator. What

about in the general case? Because in another sense, it is a problem

if you don't have an idea. If you start a startup with no idea,

what do you do next?So here's the brief recipe for getting startup ideas. Find something

that's missing in your own life, and supply that need—no matter

how specific to you it seems. Steve Wozniak built himself a computer;

who knew so many other people would want them? A need that's narrow

but genuine is a better starting point than one that's broad but

hypothetical. So even if the problem is simply that you don't have

a date on Saturday night, if you can think of a way to fix that by

writing software, you're onto something, because a lot of other

people have the same problem.8. No room for more startupsA lot of people look at the ever-increasing number of startups and

think "this can't continue." Implicit in their thinking is a

fallacy: that there is some limit on the number of startups there

could be. But this is false. No one claims there's any limit on

the number of people who can work for salary at 1000-person companies.

Why should there be any limit on the number who can work for equity

at 5-person companies?

[3]Nearly everyone who works is satisfying some kind of need. Breaking

up companies into smaller units doesn't make those needs go away.

Existing needs would probably get satisfied more efficiently by a

network of startups than by a few giant, hierarchical organizations,

but I don't think that would mean less opportunity, because satisfying

current needs would lead to more. Certainly this tends to be the

case in individuals. Nor is there anything wrong with that. We

take for granted things that medieval kings would have considered

effeminate luxuries, like whole buildings heated to spring temperatures

year round. And if things go well, our descendants will take for

granted things we would consider shockingly luxurious. There is

no absolute standard for material wealth. Health care is a component

of it, and that alone is a black hole. For the foreseeable future,

people will want ever more material wealth, so there is no limit

to the amount of work available for companies, and for startups in

particular.Usually the limited-room fallacy is not expressed directly. Usually

it's implicit in statements like "there are only so many startups

Google, Microsoft, and Yahoo can buy." Maybe, though the list of

acquirers is a lot longer than that. And whatever you think of

other acquirers, Google is not stupid. The reason big companies

buy startups is that they've created something valuable. And why

should there be any limit to the number of valuable startups companies

can acquire, any more than there is a limit to the amount of wealth

individual people want? Maybe there would be practical limits on

the number of startups any one acquirer could assimilate, but if

there is value to be had, in the form of upside that founders are

willing to forgo in return for an immediate payment, acquirers will

evolve to consume it. Markets are pretty smart that way.9. Family to supportThis one is real. I wouldn't advise anyone with a family to start

a startup. I'm not saying it's a bad idea, just that I don't want

to take responsibility for advising it. I'm willing to take

responsibility for telling 22 year olds to start startups. So what

if they fail? They'll learn a lot, and that job at Microsoft will

still be waiting for them if they need it. But I'm not prepared

to cross moms.What you can do, if you have a family and want to start a startup,

is start a consulting business you can then gradually turn into a

product business. Empirically the chances of pulling that off seem

very small. You're never going to produce Google this way. But at

least you'll never be without an income.Another way to decrease the risk is to join an existing startup

instead of starting your own. Being one of the first employees of

a startup is a lot like being a founder, in both the good ways and

the bad. You'll be roughly 1/n^2 founder, where n is your employee

number.As with the question of cofounders, the real lesson here is to start

startups when you're young.10. Independently wealthyThis is my excuse for not starting a startup. Startups are stressful.

Why do it if you don't need the money? For every "serial entrepreneur,"

there are probably twenty sane ones who think "Start another

company? Are you crazy?"I've come close to starting new startups a couple times, but I

always pull back because I don't want four years of my life to be

consumed by random schleps. I know this business well enough to

know you can't do it half-heartedly. What makes a good startup

founder so dangerous is his willingness to endure infinite schleps.There is a bit of a problem with retirement, though. Like a lot

of people, I like to work. And one of the many weird little problems

you discover when you get rich is that a lot of the interesting

people you'd like to work with are not rich. They need to work at

something that pays the bills. Which means if you want to have

them as colleagues, you have to work at something that pays the

bills too, even though you don't need to. I think this is what

drives a lot of serial entrepreneurs, actually.That's why I love working on Y Combinator so much. It's an excuse

to work on something interesting with people I like.11. Not ready for commitmentThis was my reason for not starting a startup for most of my twenties.

Like a lot of people that age, I valued freedom most of all. I was

reluctant to do anything that required a commitment of more than a

few months. Nor would I have wanted to do anything that completely

took over my life the way a startup does. And that's fine. If you

want to spend your time travelling around, or playing in a band,

or whatever, that's a perfectly legitimate reason not to start a

company.If you start a startup that succeeds, it's going to consume at least

three or four years. (If it fails, you'll be done a lot quicker.)

So you shouldn't do it if you're not ready for commitments on that

scale. Be aware, though, that if you get a regular job, you'll

probably end up working there for as long as a startup would take,

and you'll find you have much less spare time than you might expect.

So if you're ready to clip on that ID badge and go to that orientation

session, you may also be ready to start that startup.12. Need for structureI'm told there are people who need structure in their lives. This

seems to be a nice way of saying they need someone to tell them

what to do. I believe such people exist. There's plenty of empirical

evidence: armies, religious cults, and so on. They may even be the

majority.If you're one of these people, you probably shouldn't start a

startup. In fact, you probably shouldn't even go to work for one.

In a good startup, you don't get told what to do very much. There

may be one person whose job title is CEO, but till the company has

about twelve people no one should be telling anyone what to do.

That's too inefficient. Each person should just do what they need

to without anyone telling them.If that sounds like a recipe for chaos, think about a soccer team.

Eleven people manage to work together in quite complicated ways,

and yet only in occasional emergencies does anyone tell anyone else

what to do. A reporter once asked David Beckham if there were any

language problems at Real Madrid, since the players were from about

eight different countries. He said it was never an issue, because

everyone was so good they never had to talk. They all just did the

right thing.How do you tell if you're independent-minded enough to start a

startup? If you'd bristle at the suggestion that you aren't, then

you probably are.13. Fear of uncertaintyPerhaps some people are deterred from starting startups because

they don't like the uncertainty. If you go to work for Microsoft,

you can predict fairly accurately what the next few years will be

like—all too accurately, in fact. If you start a startup, anything

might happen.Well, if you're troubled by uncertainty, I can solve that problem

for you: if you start a startup, it will probably fail. Seriously,

though, this is not a bad way to think

about the whole experience. Hope for the best, but expect the

worst. In the worst case, it will at least be interesting. In the

best case you might get rich.No one will blame you if the startup tanks, so long as you made a

serious effort. There may once have been a time when employers

would regard that as a mark against you, but they wouldn't now. I

asked managers at big companies, and they all said they'd prefer

to hire someone who'd tried to start a startup and failed over

someone who'd spent the same time working at a big company.Nor will investors hold it against you, as long as you didn't fail

out of laziness or incurable stupidity. I'm told there's a lot

of stigma attached to failing in other places—in Europe, for

example. Not here. In America, companies, like practically

everything else, are disposable.14. Don't realize what you're avoidingOne reason people who've been out in the world for a year or two

make better founders than people straight from college is that they

know what they're avoiding. If their startup fails, they'll have

to get a job, and they know how much jobs suck.If you've had summer jobs in college, you may think you know what

jobs are like, but you probably don't. Summer jobs at technology

companies are not real jobs. If you get a summer job as a waiter,

that's a real job. Then you have to carry your weight. But software

companies don't hire students for the summer as a source of cheap

labor. They do it in the hope of recruiting them when they graduate.

So while they're happy if you produce, they don't expect you to.That will change if you get a real job after you graduate. Then

you'll have to earn your keep. And since most of what big companies

do is boring, you're going to have to work on boring stuff. Easy,

compared to college, but boring. At first it may seem cool to get

paid for doing easy stuff, after paying to do hard stuff in college.

But that wears off after a few months. Eventually it gets demoralizing

to work on dumb stuff, even if it's easy and you get paid a lot.And that's not the worst of it. The thing that really sucks about

having a regular job is the expectation that you're supposed to be

there at certain times. Even Google is afflicted with this,

apparently. And what this means, as everyone who's had a regular

job can tell you, is that there are going to be times when you have

absolutely no desire to work on anything, and you're going to have

to go to work anyway and sit in front of your screen and pretend

to. To someone who likes work, as most good hackers do, this is

torture.In a startup, you skip all that. There's no concept of office hours

in most startups. Work and life just get mixed together. But the

good thing about that is that no one minds if you have a life at

work. In a startup you can do whatever you want most of the time.

If you're a founder, what you want to do most of the time is work.

But you never have to pretend to.If you took a nap in your office in a big company, it would seem

unprofessional. But if you're starting a startup and you fall

asleep in the middle of the day, your cofounders will just assume

you were tired.15. Parents want you to be a doctorA significant number of would-be startup founders are probably

dissuaded from doing it by their parents. I'm not going to say you

shouldn't listen to them. Families are entitled to their own

traditions, and who am I to argue with them? But I will give you

a couple reasons why a safe career might not be what your parents

really want for you.One is that parents tend to be more conservative for their kids

than they would be for themselves. This is actually a rational

response to their situation. Parents end up sharing more of their

kids' ill fortune than good fortune. Most parents don't mind this;

it's part of the job; but it does tend to make them excessively

conservative. And erring on the side of conservatism is still

erring. In almost everything, reward is proportionate to risk. So

by protecting their kids from risk, parents are, without realizing

it, also protecting them from rewards. If they saw that, they'd

want you to take more risks.The other reason parents may be mistaken is that, like generals,

they're always fighting the last war. If they want you to be a

doctor, odds are it's not just because they want you to help the

sick, but also because it's a prestigious and lucrative career.

[4]

But not so lucrative or prestigious as it was when their

opinions were formed. When I was a kid in the seventies, a doctor

was the thing to be. There was a sort of golden triangle involving

doctors, Mercedes 450SLs, and tennis. All three vertices now seem

pretty dated.The parents who want you to be a doctor may simply not realize how

much things have changed. Would they be that unhappy if you were

Steve Jobs instead? So I think the way to deal with your parents'

opinions about what you should do is to treat them like feature

requests. Even if your only goal is to please them, the way to do

that is not simply to give them what they ask for. Instead think

about why they're asking for something, and see if there's a better

way to give them what they need.16. A job is the defaultThis leads us to the last and probably most powerful reason people

get regular jobs: it's the default thing to do. Defaults are

enormously powerful, precisely because they operate without any

conscious choice.To almost everyone except criminals, it seems an axiom that if you

need money, you should get a job. Actually this tradition is not

much more than a hundred years old. Before that, the default way

to make a living was by farming. It's a bad plan to treat something

only a hundred years old as an axiom. By historical standards,

that's something that's changing pretty rapidly.We may be seeing another such change right now. I've read a lot

of economic history, and I understand the startup world pretty well,

and it now seems to me fairly likely that we're seeing the beginning

of a change like the one from farming to manufacturing.And you know what? If you'd been around when that change began

(around 1000 in Europe) it would have seemed to nearly everyone

that running off to the city to make your fortune was a crazy thing

to do. Though serfs were in principle forbidden to leave their

manors, it can't have been that hard to run away to a city. There

were no guards patrolling the perimeter of the village. What

prevented most serfs from leaving was that it seemed insanely risky.

Leave one's plot of land? Leave the people you'd spent your whole

life with, to live in a giant city of three or four thousand complete

strangers? How would you live? How would you get food, if you

didn't grow it?Frightening as it seemed to them, it's now the default with us to

live by our wits. So if it seems risky to you to start a startup,

think how risky it once seemed to your ancestors to live as we do

now. Oddly enough, the people who know this best are the very ones

trying to get you to stick to the old model. How can Larry and

Sergey say you should come work as their employee, when they didn't

get jobs themselves?Now we look back on medieval peasants and wonder how they stood it.

How grim it must have been to till the same fields your whole life

with no hope of anything better, under the thumb of lords and priests

you had to give all your surplus to and acknowledge as your masters.

I wouldn't be surprised if one day people look back on what we

consider a normal job in the same way. How grim it would be to

commute every day to a cubicle in some soulless office complex, and

be told what to do by someone you had to acknowledge as a boss—someone

who could call you into their office and say "take a seat,"

and you'd sit! Imagine having to ask permission to release

software to users. Imagine being sad on Sunday afternoons because

the weekend was almost over, and tomorrow you'd have to get up and

go to work. How did they stand it?It's exciting to think we may be on the cusp of another shift like

the one from farming to manufacturing. That's why I care about

startups. Startups aren't interesting just because they're a way

to make a lot of money. I couldn't care less about other ways to

do that, like speculating in securities. At most those are interesting

the way puzzles are. There's more going on with startups. They

may represent one of those rare, historic shifts in the way

wealth is created.That's ultimately what drives us to work on Y Combinator. We want

to make money, if only so we don't have to stop doing it, but that's

not the main goal. There have only been a handful of these great

economic shifts in human history. It would be an amazing hack to

make one happen faster.

Notes[1]

The only people who lost were us. The angels had convertible

debt, so they had first claim on the proceeds of the auction. Y

Combinator only got 38 cents on the dollar.[2]

The best kind of organization for that might be an open source

project, but those don't involve a lot of face to face meetings.

Maybe it would be worth starting one that did.[3]

There need to be some number of big companies to acquire the

startups, so the number of big companies couldn't decrease to zero.[4]

Thought experiment: If doctors did the same work, but as

impoverished outcasts, which parents would still want their kids

to be doctors?Thanks to Trevor Blackwell, Jessica Livingston, and Robert

Morris for reading drafts of this, to the founders of Zenter

for letting me use their web-based PowerPoint killer even though

it isn't launched yet, and to Ming-Hay Luk

of the Berkeley CSUA for inviting me to speak.

Comment on this essay.Russian TranslationJapanese TranslationKorean Translation

Is It Worth Being Wise?

February 2007A few days ago I finally figured out something I've wondered about

for 25 years: the relationship between wisdom and intelligence.

Anyone can see they're not the same by the number of people who are

smart, but not very wise. And yet intelligence and wisdom do seem

related. How?What is wisdom? I'd say it's knowing what to do in a lot of

situations. I'm not trying to make a deep point here about the

true nature of wisdom, just to figure out how we use the word. A

wise person is someone who usually knows the right thing to do.And yet isn't being smart also knowing what to do in certain

situations? For example, knowing what to do when the teacher tells

your elementary school class to add all the numbers from 1 to 100?

[1]Some say wisdom and intelligence apply to different types of

problems—wisdom to human problems and intelligence to abstract

ones. But that isn't true. Some wisdom has nothing to do with

people: for example, the wisdom of the engineer who knows certain

structures are less prone to failure than others. And certainly

smart people can find clever solutions to human problems as well

as abstract ones.

[2]Another popular explanation is that wisdom comes from experience

while intelligence is innate. But people are not simply wise in

proportion to how much experience they have. Other things must

contribute to wisdom besides experience, and some may be innate: a

reflective disposition, for example.Neither of the conventional explanations of the difference between

wisdom and intelligence stands up to scrutiny. So what is the

difference? If we look at how people use the words "wise" and

"smart," what they seem to mean is different shapes of performance.Curve"Wise" and "smart" are both ways of saying someone knows what to

do. The difference is that "wise" means one has a high average

outcome across all situations, and "smart" means one does spectacularly

well in a few. That is, if you had a graph in which the x axis

represented situations and the y axis the outcome, the graph of the

wise person would be high overall, and the graph of the smart person

would have high peaks.The distinction is similar to the rule that one should judge talent

at its best and character at its worst. Except you judge intelligence

at its best, and wisdom by its average. That's how the two are

related: they're the two different senses in which the same curve

can be high.So a wise person knows what to do in most situations, while a smart

person knows what to do in situations where few others could. We

need to add one more qualification: we should ignore cases where

someone knows what to do because they have inside information.

[3]

But aside from that, I don't think we can get much more specific

without starting to be mistaken.Nor do we need to. Simple as it is, this explanation predicts, or

at least accords with, both of the conventional stories about the

distinction between wisdom and intelligence. Human problems are

the most common type, so being good at solving those is key in

achieving a high average outcome. And it seems natural that a

high average outcome depends mostly on experience, but that dramatic

peaks can only be achieved by people with certain rare, innate

qualities; nearly anyone can learn to be a good swimmer, but to be

an Olympic swimmer you need a certain body type.This explanation also suggests why wisdom is such an elusive concept:

there's no such thing. "Wise" means something—that one is

on average good at making the right choice. But giving the name

"wisdom" to the supposed quality that enables one to do that doesn't

mean such a thing exists. To the extent "wisdom" means anything,

it refers to a grab-bag of qualities as various as self-discipline,

experience, and empathy.

[4]Likewise, though "intelligent" means something, we're asking for

trouble if we insist on looking for a single thing called "intelligence."

And whatever its components, they're not all innate. We use the

word "intelligent" as an indication of ability: a smart person can

grasp things few others could. It does seem likely there's some

inborn predisposition to intelligence (and wisdom too), but this

predisposition is not itself intelligence.One reason we tend to think of intelligence as inborn is that people

trying to measure it have concentrated on the aspects of it that

are most measurable. A quality that's inborn will obviously be

more convenient to work with than one that's influenced by experience,

and thus might vary in the course of a study. The problem comes

when we drag the word "intelligence" over onto what they're measuring.

If they're measuring something inborn, they can't be measuring

intelligence. Three year olds aren't smart. When we describe one

as smart, it's shorthand for "smarter than other three year olds."SplitPerhaps it's a technicality to point out that a predisposition to

intelligence is not the same as intelligence. But it's an important

technicality, because it reminds us that we can become smarter,

just as we can become wiser.The alarming thing is that we may have to choose between the two.If wisdom and intelligence are the average and peaks of the same

curve, then they converge as the number of points on the curve

decreases. If there's just one point, they're identical: the average

and maximum are the same. But as the number of points increases,

wisdom and intelligence diverge. And historically the number of

points on the curve seems to have been increasing: our ability is

tested in an ever wider range of situations.In the time of Confucius and Socrates, people seem to have regarded

wisdom, learning, and intelligence as more closely related than we

do. Distinguishing between "wise" and "smart" is a modern habit.

[5]

And the reason we do is that they've been diverging. As knowledge

gets more specialized, there are more points on the curve, and the

distinction between the spikes and the average becomes sharper,

like a digital image rendered with more pixels.One consequence is that some old recipes may have become obsolete.

At the very least we have to go back and figure out if they were

really recipes for wisdom or intelligence. But the really striking

change, as intelligence and wisdom drift apart, is that we may have

to decide which we prefer. We may not be able to optimize for both

simultaneously.Society seems to have voted for intelligence. We no longer admire

the sage—not the way people did two thousand years ago. Now

we admire the genius. Because in fact the distinction we began

with has a rather brutal converse: just as you can be smart without

being very wise, you can be wise without being very smart. That

doesn't sound especially admirable. That gets you James Bond, who

knows what to do in a lot of situations, but has to rely on Q for

the ones involving math.Intelligence and wisdom are obviously not mutually exclusive. In

fact, a high average may help support high peaks. But there are

reasons to believe that at some point you have to choose between

them. One is the example of very smart people, who are so often

unwise that in popular culture this now seems to be regarded as the

rule rather than the exception. Perhaps the absent-minded professor

is wise in his way, or wiser than he seems, but he's not wise in

the way Confucius or Socrates wanted people to be.

[6]NewFor both Confucius and Socrates, wisdom, virtue, and happiness were

necessarily related. The wise man was someone who knew what the

right choice was and always made it; to be the right choice, it had

to be morally right; he was therefore always happy, knowing he'd

done the best he could. I can't think of many ancient philosophers

who would have disagreed with that, so far as it goes."The superior man is always happy; the small man sad," said Confucius.

[7]Whereas a few years ago I read an interview with a mathematician

who said that most nights he went to bed discontented, feeling he

hadn't made enough progress.

[8]

The Chinese and Greek words we

translate as "happy" didn't mean exactly what we do by it, but

there's enough overlap that this remark contradicts them.Is the mathematician a small man because he's discontented? No;

he's just doing a kind of work that wasn't very common in Confucius's

day.Human knowledge seems to grow fractally. Time after time, something

that seemed a small and uninteresting area—experimental error,

even—turns out, when examined up close, to have as much in

it as all knowledge up to that point. Several of the fractal buds

that have exploded since ancient times involve inventing and

discovering new things. Math, for example, used to be something a

handful of people did part-time. Now it's the career of thousands.

And in work that involves making new things, some old rules don't

apply.Recently I've spent some time advising people, and there I find the

ancient rule still works: try to understand the situation as well

as you can, give the best advice you can based on your experience,

and then don't worry about it, knowing you did all you could. But

I don't have anything like this serenity when I'm writing an essay.

Then I'm worried. What if I run out of ideas? And when I'm writing,

four nights out of five I go to bed discontented, feeling I didn't

get enough done.Advising people and writing are fundamentally different types of

work. When people come to you with a problem and you have to figure

out the right thing to do, you don't (usually) have to invent

anything. You just weigh the alternatives and try to judge which

is the prudent choice. But prudence can't tell me what sentence

to write next. The search space is too big.Someone like a judge or a military officer can in much of his work

be guided by duty, but duty is no guide in making things. Makers

depend on something more precarious: inspiration. And like most

people who lead a precarious existence, they tend to be worried,

not contented. In that respect they're more like the small man of

Confucius's day, always one bad harvest (or ruler) away from

starvation. Except instead of being at the mercy of weather and

officials, they're at the mercy of their own imagination.LimitsTo me it was a relief just to realize it might be ok to be discontented.

The idea that a successful person should be happy has thousands of

years of momentum behind it. If I was any good, why didn't I have

the easy confidence winners are supposed to have? But that, I now

believe, is like a runner asking "If I'm such a good athlete, why

do I feel so tired?" Good runners still get tired; they just get

tired at higher speeds.People whose work is to invent or discover things are in the same

position as the runner. There's no way for them to do the best

they can, because there's no limit to what they could do. The

closest you can come is to compare yourself to other people. But

the better you do, the less this matters. An undergrad who gets

something published feels like a star. But for someone at the top

of the field, what's the test of doing well? Runners can at least

compare themselves to others doing exactly the same thing; if you

win an Olympic gold medal, you can be fairly content, even if you

think you could have run a bit faster. But what is a novelist to

do?Whereas if you're doing the kind of work in which problems are

presented to you and you have to choose between several alternatives,

there's an upper bound on your performance: choosing the best every

time. In ancient societies, nearly all work seems to have been of

this type. The peasant had to decide whether a garment was worth

mending, and the king whether or not to invade his neighbor, but

neither was expected to invent anything. In principle they could

have; the king could have invented firearms, then invaded his

neighbor. But in practice innovations were so rare that they weren't

expected of you, any more than goalkeepers are expected to score

goals.

[9]

In practice, it seemed as if there was a correct decision

in every situation, and if you made it you'd done your job perfectly,

just as a goalkeeper who prevents the other team from scoring is

considered to have played a perfect game.In this world, wisdom seemed paramount.

[10]

Even now, most people

do work in which problems are put before them and they have to

choose the best alternative. But as knowledge has grown more

specialized, there are more and more types of work in which people

have to make up new things, and in which performance is therefore

unbounded. Intelligence has become increasingly important relative

to wisdom because there is more room for spikes.RecipesAnother sign we may have to choose between intelligence and wisdom

is how different their recipes are. Wisdom seems to come largely

from curing childish qualities, and intelligence largely from

cultivating them.Recipes for wisdom, particularly ancient ones, tend to have a

remedial character. To achieve wisdom one must cut away all the

debris that fills one's head on emergence from childhood, leaving

only the important stuff. Both self-control and experience have

this effect: to eliminate the random biases that come from your own

nature and from the circumstances of your upbringing respectively.

That's not all wisdom is, but it's a large part of it. Much of

what's in the sage's head is also in the head of every twelve year

old. The difference is that in the head of the twelve year old

it's mixed together with a lot of random junk.The path to intelligence seems to be through working on hard problems.

You develop intelligence as you might develop muscles, through

exercise. But there can't be too much compulsion here. No amount

of discipline can replace genuine curiosity. So cultivating

intelligence seems to be a matter of identifying some bias in one's

character—some tendency to be interested in certain types of

things—and nurturing it. Instead of obliterating your

idiosyncrasies in an effort to make yourself a neutral vessel for

the truth, you select one and try to grow it from a seedling into

a tree.The wise are all much alike in their wisdom, but very smart people

tend to be smart in distinctive ways.Most of our educational traditions aim at wisdom. So perhaps one

reason schools work badly is that they're trying to make intelligence

using recipes for wisdom. Most recipes for wisdom have an element

of subjection. At the very least, you're supposed to do what the

teacher says. The more extreme recipes aim to break down your

individuality the way basic training does. But that's not the route

to intelligence. Whereas wisdom comes through humility, it may

actually help, in cultivating intelligence, to have a mistakenly

high opinion of your abilities, because that encourages you to keep

working. Ideally till you realize how mistaken you were.(The reason it's hard to learn new skills late in life is not just

that one's brain is less malleable. Another probably even worse

obstacle is that one has higher standards.)I realize we're on dangerous ground here. I'm not proposing the

primary goal of education should be to increase students' "self-esteem."

That just breeds laziness. And in any case, it doesn't really fool

the kids, not the smart ones. They can tell at a young age that a

contest where everyone wins is a fraud.A teacher has to walk a narrow path: you want to encourage kids to

come up with things on their own, but you can't simply applaud

everything they produce. You have to be a good audience: appreciative,

but not too easily impressed. And that's a lot of work. You have

to have a good enough grasp of kids' capacities at different ages

to know when to be surprised.That's the opposite of traditional recipes for education. Traditionally

the student is the audience, not the teacher; the student's job is

not to invent, but to absorb some prescribed body of material. (The

use of the term "recitation" for sections in some colleges is a

fossil of this.) The problem with these old traditions is that

they're too much influenced by recipes for wisdom.DifferentI deliberately gave this essay a provocative title; of course it's

worth being wise. But I think it's important to understand the

relationship between intelligence and wisdom, and particularly what

seems to be the growing gap between them. That way we can avoid

applying rules and standards to intelligence that are really meant

for wisdom. These two senses of "knowing what to do" are more

different than most people realize. The path to wisdom is through

discipline, and the path to intelligence through carefully selected

self-indulgence. Wisdom is universal, and intelligence idiosyncratic.

And while wisdom yields calmness, intelligence much of the time

leads to discontentment.That's particularly worth remembering. A physicist friend recently

told me half his department was on Prozac. Perhaps if we acknowledge

that some amount of frustration is inevitable in certain kinds

of work, we can mitigate its effects. Perhaps we can box it up and

put it away some of the time, instead of letting it flow together

with everyday sadness to produce what seems an alarmingly large

pool. At the very least, we can avoid being discontented about

being discontented.If you feel exhausted, it's not necessarily because there's something

wrong with you. Maybe you're just running fast.Notes[1]

Gauss was supposedly asked this when he was 10. Instead of

laboriously adding together the numbers like the other students,

he saw that they consisted of 50 pairs that each summed to 101 (100

+ 1, 99 + 2, etc), and that he could just multiply 101 by 50 to get

the answer, 5050.[2]

A variant is that intelligence is the ability to solve problems,

and wisdom the judgement to know how to use those solutions. But

while this is certainly an important relationship between wisdom

and intelligence, it's not the distinction between them. Wisdom

is useful in solving problems too, and intelligence can help in

deciding what to do with the solutions.[3]

In judging both intelligence and wisdom we have to factor out

some knowledge. People who know the combination of a safe will be

better at opening it than people who don't, but no one would say

that was a test of intelligence or wisdom.But knowledge overlaps with wisdom and probably also intelligence.

A knowledge of human nature is certainly part of wisdom. So where

do we draw the line?Perhaps the solution is to discount knowledge that at some point

has a sharp drop in utility. For example, understanding French

will help you in a large number of situations, but its value drops

sharply as soon as no one else involved knows French. Whereas the

value of understanding vanity would decline more gradually.The knowledge whose utility drops sharply is the kind that has

little relation to other knowledge. This includes mere conventions,

like languages and safe combinations, and also what we'd call

"random" facts, like movie stars' birthdays, or how to distinguish

1956 from 1957 Studebakers.[4]

People seeking some single thing called "wisdom" have been

fooled by grammar. Wisdom is just knowing the right thing to do,

and there are a hundred and one different qualities that help in

that. Some, like selflessness, might come from meditating in an

empty room, and others, like a knowledge of human nature, might

come from going to drunken parties.Perhaps realizing this will help dispel the cloud of semi-sacred

mystery that surrounds wisdom in so many people's eyes. The mystery

comes mostly from looking for something that doesn't exist. And

the reason there have historically been so many different schools

of thought about how to achieve wisdom is that they've focused on

different components of it.When I use the word "wisdom" in this essay, I mean no more than

whatever collection of qualities helps people make the right choice

in a wide variety of situations.[5]

Even in English, our sense of the word "intelligence" is

surprisingly recent. Predecessors like "understanding" seem to

have had a broader meaning.[6]

There is of course some uncertainty about how closely the remarks

attributed to Confucius and Socrates resemble their actual opinions.

I'm using these names as we use the name "Homer," to mean the

hypothetical people who said the things attributed to them.[7]

Analects VII:36, Fung trans.Some translators use "calm" instead of "happy." One source of

difficulty here is that present-day English speakers have a different

idea of happiness from many older societies. Every language probably

has a word meaning "how one feels when things are going well," but

different cultures react differently when things go well. We react

like children, with smiles and laughter. But in a more reserved

society, or in one where life was tougher, the reaction might be a

quiet contentment.[8]

It may have been Andrew Wiles, but I'm not sure. If anyone

remembers such an interview, I'd appreciate hearing from you.[9]

Confucius claimed proudly that he had never invented

anything—that he had simply passed on an accurate account of

ancient traditions. [Analects VII:1] It's hard for us now to

appreciate how important a duty it must have been in preliterate

societies to remember and pass on the group's accumulated knowledge.

Even in Confucius's time it still seems to have been the first duty

of the scholar.[10]

The bias toward wisdom in ancient philosophy may be exaggerated

by the fact that, in both Greece and China, many of the first

philosophers (including Confucius and Plato) saw themselves as

teachers of administrators, and so thought disproportionately about

such matters. The few people who did invent things, like storytellers,

must have seemed an outlying data point that could be ignored.Thanks to Trevor Blackwell, Sarah Harlin, Jessica Livingston,

and Robert Morris for reading drafts of this.Polish TranslationFrench TranslationRussian TranslationRussian Translation

Learning from Founders

January 2007(Foreword to Jessica Livingston's

Founders at Work.)Apparently sprinters reach their highest speed right out of the

blocks, and spend the rest of the race slowing down. The winners

slow down the least. It's that way with most startups too. The

earliest phase is usually the most productive. That's when they

have the really big ideas. Imagine what Apple was like when 100%

of its employees were either Steve Jobs or Steve Wozniak.The striking thing about this phase is that it's completely different

from most people's idea of what business is like. If you looked

in people's heads (or stock photo collections) for images representing

"business," you'd get images of people dressed up in suits, groups

sitting around conference tables looking serious, Powerpoint

presentations, people producing thick reports for one another to

read. Early stage startups are the exact opposite of this. And

yet they're probably the most productive part of the whole economy.Why the disconnect? I think there's a general principle at work

here: the less energy people expend on performance, the more they

expend on appearances to compensate. More often than not the energy

they expend on seeming impressive makes their actual performance

worse. A few years ago I read an article in which a car magazine

modified the "sports" model of some production car to get the fastest

possible standing quarter mile. You know how they did it? They

cut off all the crap the manufacturer had bolted onto the car to

make it look fast.Business is broken the same way that car was. The effort that goes

into looking productive is not merely wasted, but actually makes

organizations less productive. Suits, for example. Suits do not

help people to think better. I bet most executives at big companies

do their best thinking when they wake up on Sunday morning and go

downstairs in their bathrobe to make a cup of coffee. That's when

you have ideas. Just imagine what a company would be like if people

could think that well at work. People do in startups, at least

some of the time. (Half the time you're in a panic because your

servers are on fire, but the other half you're thinking as deeply

as most people only get to sitting alone on a Sunday morning.)Ditto for most of the other differences between startups and what

passes for productivity in big companies. And yet conventional

ideas of professionalism have such an iron grip on our minds that

even startup founders are affected by them. In our startup, when

outsiders came to visit we tried hard to seem "professional." We'd

clean up our offices, wear better clothes, try to arrange that a

lot of people were there during conventional office hours. In fact,

programming didn't get done by well-dressed people at clean desks

during office hours. It got done by badly dressed people (I was

notorious for programmming wearing just a towel) in offices strewn

with junk at 2 in the morning. But no visitor would understand

that. Not even investors, who are supposed to be able to recognize

real productivity when they see it. Even we were affected by the

conventional wisdom. We thought of ourselves as impostors, succeeding

despite being totally unprofessional. It was as if we'd created a

Formula 1 car but felt sheepish because it didn't look like a car

was supposed to look.In the car world, there are at least some people who know that a

high performance car looks like a Formula 1 racecar, not a sedan

with giant rims and a fake spoiler bolted to the trunk. Why not

in business? Probably because startups are so small. The really

dramatic growth happens when a startup only has three or four people,

so only three or four people see that, whereas tens of thousands

see business as it's practiced by Boeing or Philip Morris.This book can help fix that problem, by showing everyone what, till

now, only a handful people got to see: what happens in the first

year of a startup. This is what real productivity looks like. This

is the Formula 1 racecar. It looks weird, but it goes fast.Of course, big companies won't be able to do everything these

startups do. In big companies there's always going to be more

politics, and less scope for individual decisions. But seeing what

startups are really like will at least show other organizations

what to aim for. The time may soon be coming when instead of

startups trying to seem more corporate, corporations will try to

seem more like startups. That would be a good thing.

Japanese

Translation

Founders at WorkThere can't be more than a couple thousand

people who know first-hand what happens in the first month of a

successful startup. Jessica Livingston got them to tell us.

So despite the interview format, this is

really a how-to book. It is probably the single most valuable

book a startup founder could read.

How Art Can Be Good

December 2006I grew up believing that taste is just a matter of personal preference.

Each person has things they like, but no one's preferences are any

better than anyone else's. There is no such thing as good taste.Like a lot of things I grew up believing, this turns out to be

false, and I'm going to try to explain why.One problem with saying there's no such thing as good taste is that

it also means there's no such thing as good art. If there were

good art, then people who liked it would have better taste than

people who didn't. So if you discard taste, you also have to discard

the idea of art being good, and artists being good at making it.It was pulling on that thread that unravelled my childhood faith

in relativism. When you're trying to make things, taste becomes a

practical matter. You have to decide what to do next. Would it

make the painting better if I changed that part? If there's no

such thing as better, it doesn't matter what you do. In fact, it

doesn't matter if you paint at all. You could just go out and buy

a ready-made blank canvas. If there's no such thing as good, that

would be just as great an achievement as the ceiling of the Sistine

Chapel. Less laborious, certainly, but if you can achieve the same

level of performance with less effort, surely that's more impressive,

not less.Yet that doesn't seem quite right, does it?AudienceI think the key to this puzzle is to remember that art has an

audience. Art has a purpose, which is to interest its audience.

Good art (like good anything) is art that achieves its purpose

particularly well. The meaning of "interest" can vary. Some works

of art are meant to shock, and others to please; some are meant to

jump out at you, and others to sit quietly in the background. But

all art has to work on an audience, and—here's the critical

point—members of the audience share things in common.For example, nearly all humans find human faces engaging. It seems

to be wired into us. Babies can recognize faces practically from

birth. In fact, faces seem to have co-evolved with our interest

in them; the face is the body's billboard. So all other things

being equal, a painting with faces in it will interest people more

than one without.

[1]One reason it's easy to believe that taste is merely personal

preference is that, if it isn't, how do you pick out the people

with better taste? There are billions of people, each with their

own opinion; on what grounds can you prefer one to another?

[2]But if audiences have a lot in common, you're not in a position of

having to choose one out of a random set of individual biases,

because the set isn't random. All humans find faces

engaging—practically by definition: face recognition is

in our DNA. And so

having a notion of good art, in the sense of art that does its job

well, doesn't require you to pick out a few individuals and label

their opinions as correct. No matter who you pick, they'll find

faces engaging.Of course, space aliens probably wouldn't find human faces engaging.

But there might be other things they shared in common with us. The

most likely source of examples is math. I expect space aliens would

agree with us most of the time about which of two proofs was better.

Erdos thought so. He called a maximally elegant proof one out of

God's book, and presumably God's book is universal.

[3]Once you start talking about audiences, you don't have to argue

simply that there are or aren't standards of taste. Instead tastes

are a series of concentric rings, like ripples in a pond. There

are some things that will appeal to you and your friends, others

that will appeal to most people your age, others that will appeal

to most humans, and perhaps others that would appeal to most sentient

beings (whatever that means).The picture is slightly more complicated than that, because in the

middle of the pond there are overlapping sets of ripples. For

example, there might be things that appealed particularly to men,

or to people from a certain culture.If good art is art that interests its audience, then when you talk

about art being good, you also have to say for what audience. So

is it meaningless to talk about art simply being good or bad? No,

because one audience is the set of all possible humans. I think

that's the audience people are implicitly talking about when they

say a work of art is good: they mean it would engage any human.

[4]And that is a meaningful test, because although, like any everyday

concept, "human" is fuzzy around the edges, there are a lot of

things practically all humans have in common. In addition to our

interest in faces, there's something special about primary colors

for nearly all of us, because it's an artifact of the way our eyes

work. Most humans will also find images of 3D objects engaging,

because that also seems to be built into our visual perception.

[5]

And beneath that there's edge-finding, which makes images

with definite shapes more engaging than mere blur.Humans have a lot more in common than this, of course. My goal is

not to compile a complete list, just to show that there's some solid

ground here. People's preferences aren't random. So an artist

working on a painting and trying to decide whether to change some

part of it doesn't have to think "Why bother? I might as well flip

a coin." Instead he can ask "What would make the painting more

interesting to people?" And the reason you can't equal Michelangelo

by going out and buying a blank canvas is that the ceiling of the

Sistine Chapel is more interesting to people.A lot of philosophers have had a hard time believing it was possible

for there to be objective standards for art. It seemed obvious that

beauty, for example, was something that happened in the head of the observer,

not something that was a property of objects. It was thus

"subjective" rather than "objective." But in fact if you narrow the

definition of beauty to something that works a certain way on

humans, and you observe how much humans have in common, it turns out

to be a property of objects after all. You don't

have to choose between something being a property of the

subject or the object if subjects all react similarly.

Being good art is thus a property of objects as much as, say, being

toxic to humans is: it's good art if it consistently affects humans

in a certain way.

ErrorSo could we figure out what the best art is by taking a vote? After

all, if appealing to humans is the test, we should be able to just

ask them, right?Well, not quite. For products of nature that might work. I'd be

willing to eat the apple the world's population had voted most

delicious, and I'd probably be willing to visit the beach they voted

most beautiful, but having to look at the painting they voted the

best would be a crapshoot.Man-made stuff is different. For one thing, artists, unlike apple

trees, often deliberately try to trick us. Some tricks are quite

subtle. For example, any work of art sets expectations by its level

of finish. You don't expect photographic accuracy in something

that looks like a quick sketch. So one widely used trick, especially

among illustrators, is to intentionally make a painting or drawing

look like it was done faster than it was. The average person looks

at it and thinks: how amazingly skillful. It's like saying something

clever in a conversation as if you'd thought of it on the spur of

the moment, when in fact you'd worked it out the day before.Another much less subtle influence is brand. If you go to see the

Mona Lisa, you'll probably be disappointed, because it's hidden

behind a thick glass wall and surrounded by a frenzied crowd taking

pictures of themselves in front of it. At best you can see it the

way you see a friend across the room at a crowded party. The Louvre

might as well replace it with copy; no one would be able to tell.

And yet the Mona Lisa is a small, dark painting. If you found

people who'd never seen an image of it and sent them to a museum

in which it was hanging among other paintings with a tag labelling

it as a portrait by an unknown fifteenth century artist, most would

walk by without giving it a second look.For the average person, brand dominates all other factors in the

judgement of art. Seeing a painting they recognize from reproductions

is so overwhelming that their response to it as a painting is drowned

out.And then of course there are the tricks people play on themselves.

Most adults looking at art worry that if they don't like what they're

supposed to, they'll be thought uncultured. This doesn't just

affect what they claim to like; they actually make themselves like

things they're supposed to.That's why you can't just take a vote. Though appeal to people is

a meaningful test, in practice you can't measure it, just as you

can't find north using a compass with a magnet sitting next to it.

There are sources of error so powerful that if you take a vote, all

you're measuring is the error.We can, however, approach our goal from another direction, by using

ourselves as guinea pigs. You're human. If you want to know what

the basic human reaction to a piece of art would be, you can at

least approach that by getting rid of the sources of error in your

own judgements.For example, while anyone's reaction to a famous painting will be

warped at first by its fame, there are ways to decrease its effects.

One is to come back to the painting over and over. After a few

days the fame wears off, and you can start to see it as a painting.

Another is to stand close. A painting familiar from reproductions

looks more familiar from ten feet away; close in you see details

that get lost in reproductions, and which you're therefore seeing

for the first time.There are two main kinds of error that get in the way of seeing a

work of art: biases you bring from your own circumstances, and

tricks played by the artist. Tricks are straightforward to correct

for. Merely being aware of them usually prevents them from working.

For example, when I was ten I used to be very impressed by airbrushed

lettering that looked like shiny metal. But once you study how

it's done, you see that it's a pretty cheesy trick—one of the

sort that relies on pushing a few visual buttons really hard to

temporarily overwhelm the viewer. It's like trying to convince

someone by shouting at them.The way not to be vulnerable to tricks is to explicitly seek out

and catalog them. When you notice a whiff of dishonesty coming

from some kind of art, stop and figure out what's going on. When

someone is obviously pandering to an audience that's easily fooled,

whether it's someone making shiny stuff to impress ten year olds,

or someone making conspicuously avant-garde stuff to impress would-be

intellectuals, learn how they do it. Once you've seen enough

examples of specific types of tricks, you start to become a connoisseur

of trickery in general, just as professional magicians are.What counts as a trick? Roughly, it's something done with contempt

for the audience. For example, the guys designing Ferraris in the

1950s were probably designing cars that they themselves admired.

Whereas I suspect over at General Motors the marketing people are

telling the designers, "Most people who buy SUVs do it to seem

manly, not to drive off-road. So don't worry about the suspension;

just make that sucker as big and tough-looking as you can."

[6]I think with some effort you can make yourself nearly immune to

tricks. It's harder to escape the influence of your own circumstances,

but you can at least move in that direction. The way to do it is

to travel widely, in both time and space. If you go and see all

the different kinds of things people like in other cultures, and

learn about all the different things people have liked in the past,

you'll probably find it changes what you like. I doubt you could

ever make yourself into a completely universal person, if only

because you can only travel in one direction in time. But if you

find a work of art that would appeal equally to your friends, to

people in Nepal, and to the ancient Greeks, you're probably onto

something.My main point here is not how to have good taste, but that there

can even be such a thing. And I think I've shown that. There is

such a thing as good art. It's art that interests its human audience,

and since humans have a lot in common, what interests them is not

random. Since there's such a thing as good art, there's

also such a thing as good taste, which is the ability to recognize

it.If we were talking about the taste of apples, I'd agree that taste

is just personal preference. Some people like certain kinds of

apples and others like other kinds, but how can you say that one

is right and the other wrong?

[7]The thing is, art isn't apples. Art is man-made. It comes with a

lot of cultural baggage, and in addition the people who make it

often try to trick us. Most people's judgement of art is dominated

by these extraneous factors; they're like someone trying to judge

the taste of apples in a dish made of equal parts apples and jalapeno

peppers. All they're tasting is the peppers. So it turns out you

can pick out some people and say that they have better taste than

others: they're the ones who actually taste art like apples.Or to put it more prosaically, they're the people who (a) are hard

to trick, and (b) don't just like whatever they grew up with. If

you could find people who'd eliminated all such influences on their

judgement, you'd probably still see variation in what they liked.

But because humans have so much in common, you'd also find they

agreed on a lot. They'd nearly all prefer the ceiling of the Sistine

Chapel to a blank canvas.Making ItI wrote this essay because I was tired of hearing "taste is subjective"

and wanted to kill it once and for all. Anyone who makes things

knows intuitively that's not true. When you're trying to make art,

the temptation to be lazy is as great as in any other kind of work.

Of course it matters to do a good job. And yet you can see how

great a hold "taste is subjective" has even in the art world by how

nervous it makes people to talk about art being good or bad. Those

whose jobs require them to judge art, like curators, mostly resort

to euphemisms like "significant" or "important" or (getting dangerously

close) "realized."

[8]I don't have any illusions that being able to talk about art being

good or bad will cause the people who talk about it to have anything

more useful to say. Indeed, one of the reasons "taste is subjective"

found such a receptive audience is that, historically, the things

people have said about good taste have generally been such nonsense.It's not for the people who talk about art that I want to free the

idea of good art, but for those who

make it. Right now, ambitious

kids going to art school run smack into a brick wall. They arrive

hoping one day to be as good as the famous artists they've seen in

books, and the first thing they learn is that the concept of good

has been retired. Instead everyone is just supposed to explore

their own personal vision.

[9]When I was in art school, we were looking one day at a slide of

some great fifteenth century painting, and one of the students asked

"Why don't artists paint like that now?" The room suddenly got

quiet. Though rarely asked out loud, this question lurks uncomfortably

in the back of every art student's mind. It was as if someone had

brought up the topic of lung cancer in a meeting within Philip

Morris."Well," the professor replied, "we're interested in different

questions now." He was a pretty nice guy, but at the time I couldn't

help wishing I could send him back to fifteenth century Florence

to explain in person to Leonardo & Co. how we had moved beyond their

early, limited concept of art. Just imagine that conversation.In fact, one of the reasons artists in fifteenth century Florence made

such great things was that they believed you could make great things.

[10]

They were intensely competitive and were always trying to outdo

one another, like mathematicians or physicists today—maybe like

anyone who has ever done anything really well.The idea that you could make great things was not just a useful

illusion. They were actually right. So the most important consequence

of realizing there can be good art is that it frees artists to try

to make it. To the ambitious kids arriving at art school this year

hoping one day to make great things, I say: don't believe it when

they tell you this is a naive and outdated ambition. There is such

a thing as good art, and if you try to make it, there are people

who will notice.Notes[1]

This is not to say, of course, that good paintings must

have faces in them, just that everyone's visual piano has that key

on it. There are situations in which you want to avoid faces,

precisely because they attract so much attention. But you can see

how universally faces work by their prevalence in

advertising.[2]

The other reason it's easy to believe is that it makes people

feel good. To a kid, this idea is crack. In every other respect

they're constantly being told that they have a lot to learn. But

in this they're perfect. Their opinion carries the same weight as

any adult's. You should probably question anything you believed

as a kid that you'd want to believe this much.[3]

It's conceivable that the elegance of proofs is quantifiable,

in the sense that there may be some formal measure that turns out

to coincide with mathematicians' judgements. Perhaps it would be

worth trying to make a formal language for proofs in which those

considered more elegant consistently came out shorter (perhaps after

being macroexpanded or compiled).[4]

Maybe it would be possible to make art that would appeal to

space aliens, but I'm not going to get into that because (a) it's

too hard to answer, and (b) I'm satisfied if I can establish that

good art is a meaningful idea for human audiences.[5]

If early abstract paintings seem more interesting than later

ones, it may be because the first abstract painters were trained

to paint from life, and their hands thus tended to make the kind

of gestures you use in representing physical things. In effect

they were saying "scaramara" instead of "uebfgbsb."[6]

It's a bit more complicated, because sometimes artists

unconsciously use tricks by imitating art that does.[7]

I phrased this in terms of the taste of apples because if

people can see the apples, they can be fooled. When I was a kid

most apples were a variety called Red Delicious that had been bred

to look appealing in stores, but which didn't taste very good.[8]

To be fair, curators are in a difficult position. If they're

dealing with recent art, they have to include things in shows that

they think are bad. That's because the test for what gets included

in shows is basically the market price, and for recent art that is

largely determined by successful businessmen and their wives. So

it's not always intellectual dishonesty that makes curators and

dealers use neutral-sounding language.[9]

What happens in practice is that everyone gets really good at

talking about art. As the art itself gets more random, the effort

that would have gone into the work goes instead into the intellectual

sounding theory behind it. "My work represents an exploration of

gender and sexuality in an urban context," etc. Different people

win at that game.[10]

There were several other reasons, including that Florence was

then the richest and most sophisticated city in the world, and that

they lived in a time before photography had (a) killed portraiture

as a source of income and (b) made brand the dominant factor in the

sale of art.Incidentally, I'm not saying that good art = fifteenth century

European art. I'm not saying we should make what they made, but

that we should work like they worked. There are fields now in which

many people work with the same energy and honesty that fifteenth

century artists did, but art is not one of them.Thanks to Trevor Blackwell, Jessica Livingston, and Robert Morris

for reading drafts of this, and to Paul Watson for permission to

use the image at the top.

Comment on this essay.Japanese Translation

The 18 Mistakes That Kill Startups

Want to start a startup? Get funded by

Y Combinator.

October 2006In the Q & A period after a recent talk, someone asked what made

startups fail. After standing there gaping for a few seconds I

realized this was kind of a trick question. It's equivalent to

asking how to make a startup succeed — if you avoid every cause of

failure, you succeed — and that's too big a question to answer on

the fly.Afterwards I realized it could be helpful to look at the problem

from this direction. If you have a list of all the things you

shouldn't do, you can turn that into a recipe for succeeding just

by negating. And this form of list may be more useful in practice.

It's easier to catch yourself doing something you shouldn't than

always to remember to do something you should.

[1]In a sense there's just one mistake that kills startups: not making

something users want. If you make something users want, you'll

probably be fine, whatever else you do or don't do. And if you

don't make something users want, then you're dead, whatever else

you do or don't do. So really this is a list of 18 things that

cause startups not to make something users want. Nearly all failure

funnels through that.1. Single FounderHave you ever noticed how few successful startups were founded by

just one person? Even companies you think of as having one founder,

like Oracle, usually turn out to have more. It seems unlikely this

is a coincidence.What's wrong with having one founder? To start with, it's a vote

of no confidence. It probably means the founder couldn't talk any

of his friends into starting the company with him. That's pretty

alarming, because his friends are the ones who know him best.But even if the founder's friends were all wrong and the company

is a good bet, he's still at a disadvantage. Starting a startup

is too hard for one person. Even if you could do all the work

yourself, you need colleagues to brainstorm with, to talk you out

of stupid decisions, and to cheer you up when things go wrong.The last one might be the most important. The low points in a

startup are so low that few could bear them alone. When you have

multiple founders, esprit de corps binds them together in a way

that seems to violate conservation laws. Each thinks "I can't let

my friends down." This is one of the most powerful forces in human

nature, and it's missing when there's just one founder.2. Bad LocationStartups prosper in some places and not others. Silicon Valley

dominates, then Boston, then Seattle, Austin, Denver, and New York. After

that there's not much. Even in New York the number of startups per

capita is probably a 20th of what it is in Silicon Valley. In towns

like Houston and Chicago and Detroit it's too small to measure.Why is the falloff so sharp? Probably for the same reason it is

in other industries. What's the sixth largest fashion center in

the US? The sixth largest center for oil, or finance, or publishing?

Whatever they are they're probably so far from the top that it would

be misleading even to call them centers.It's an interesting question why cities

become startup hubs, but

the reason startups prosper in them is probably the same as it is

for any industry: that's where the experts are. Standards are

higher; people are more sympathetic to what you're doing; the kind

of people you want to hire want to live there; supporting industries

are there; the people you run into in chance meetings are in the

same business. Who knows exactly how these factors combine to boost

startups in Silicon Valley and squish them in Detroit, but it's

clear they do from the number of startups per capita in each.3. Marginal NicheMost of the groups that apply to Y Combinator suffer from a common

problem: choosing a small, obscure niche in the hope of avoiding

competition.If you watch little kids playing sports, you notice that below a

certain age they're afraid of the ball. When the ball comes near

them their instinct is to avoid it. I didn't make a lot of catches

as an eight year old outfielder, because whenever a fly ball came

my way, I used to close my eyes and hold my glove up more for

protection than in the hope of catching it.Choosing a marginal project is the startup equivalent of my eight

year old strategy for dealing with fly balls. If you make anything

good, you're going to have competitors, so you may as well face

that. You can only avoid competition by avoiding good ideas.I think this shrinking from big problems is mostly unconscious.

It's not that people think of grand ideas but decide to pursue

smaller ones because they seem safer. Your unconscious won't even

let you think of grand ideas. So the solution may be to think about

ideas without involving yourself. What would be a great idea for

someone else to do as a startup?4. Derivative IdeaMany of the applications we get are imitations of some existing

company. That's one source of ideas, but not the best. If you

look at the origins of successful startups, few were started in

imitation of some other startup. Where did they get their ideas?

Usually from some specific, unsolved problem the founders identified.Our startup made software for making online stores. When we started

it, there wasn't any; the few sites you could order from were

hand-made at great expense by web consultants. We knew that if

online shopping ever took off, these sites would have to be generated

by software, so we wrote some. Pretty straightforward.It seems like the best problems to solve are ones that affect you

personally. Apple happened because Steve Wozniak wanted a computer,

Google because Larry and Sergey couldn't find stuff online, Hotmail

because Sabeer Bhatia and Jack Smith couldn't exchange email at

work.So instead of copying the Facebook, with some variation that the

Facebook rightly ignored, look for ideas from the other direction.

Instead of starting from companies and working back to the problems

they solved, look for problems and imagine the company that might

solve them.

[2]

What do people complain about? What do you wish there was?5. ObstinacyIn some fields the way to succeed is to have a vision of what you

want to achieve, and to hold true to it no matter what setbacks you

encounter. Starting startups is not one of them. The stick-to-your-vision

approach works for something like winning an Olympic gold medal,

where the problem is well-defined. Startups are more like science,

where you need to follow the trail wherever it leads.So don't get too attached to your original plan, because it's

probably wrong. Most successful startups end up doing something

different than they originally intended — often so different that

it doesn't even seem like the same company. You have to be prepared

to see the better idea when it arrives. And the hardest part of

that is often discarding your old idea.But openness to new ideas has to be tuned just right. Switching

to a new idea every week will be equally fatal. Is there some kind

of external test you can use? One is to ask whether the ideas

represent some kind of progression. If in each new idea you're

able to re-use most of what you built for the previous ones, then

you're probably in a process that converges. Whereas if you keep

restarting from scratch, that's a bad sign.Fortunately there's someone you can ask for advice: your users. If

you're thinking about turning in some new direction and your users

seem excited about it, it's probably a good bet.6. Hiring Bad ProgrammersI forgot to include this in the early versions of the list,

because nearly all the founders I know are programmers. This is

not a serious problem for them. They might accidentally hire someone

bad, but it's not going to kill the company. In a pinch they can

do whatever's required themselves.But when I think about what killed most of the startups in the

e-commerce business back in the 90s, it was bad programmers. A lot

of those companies were started by business guys who thought the

way startups worked was that you had some clever idea and then hired

programmers to implement it. That's actually much harder than it

sounds — almost impossibly hard in fact — because business guys

can't tell which are the good programmers. They don't even get a

shot at the best ones, because no one really good wants a job

implementing the vision of a business guy.In practice what happens is that the business guys choose people

they think are good programmers (it says here on his resume that

he's a Microsoft Certified Developer) but who aren't. Then they're

mystified to find that their startup lumbers along like a World War

II bomber while their competitors scream past like jet fighters.

This kind of startup is in the same position as a big company,

but without the advantages.So how do you pick good programmers if you're not a programmer? I

don't think there's an answer. I was about to say you'd have to

find a good programmer to help you hire people. But if you can't

recognize good programmers, how would you even do that?7. Choosing the Wrong PlatformA related problem (since it tends to be done by bad programmers)

is choosing the wrong platform. For example, I think a lot of

startups during the Bubble killed themselves by deciding to build

server-based applications on Windows. Hotmail was still running

on FreeBSD for years after Microsoft bought it, presumably because

Windows couldn't handle the load. If Hotmail's founders

had chosen to use Windows, they would have been swamped.PayPal only just dodged this bullet. After they merged with X.com,

the new CEO wanted to switch to Windows — even after PayPal cofounder

Max Levchin showed that their software scaled only 1% as well on

Windows as Unix. Fortunately for PayPal they switched CEOs instead.Platform is a vague word. It could mean an operating system, or a

programming language, or a "framework" built on top of a programming

language. It implies something that both supports and limits, like

the foundation of a house.The scary thing about platforms is that there are always some that

seem to outsiders to be fine, responsible choices and yet, like

Windows in the 90s, will destroy you if you choose them. Java

applets were probably the most spectacular example. This was

supposed to be the new way of delivering applications. Presumably

it killed just about 100% of the startups who believed that.How do you pick the right platforms? The usual way is to hire good

programmers and let them choose. But there is a trick you could

use if you're not a programmer: visit a top computer science

department and see what they use in research projects.8. Slowness in LaunchingCompanies of all sizes have a hard time getting software done. It's

intrinsic to the medium; software is always 85% done. It takes an

effort of will to push through this and get something released to

users.

[3]Startups make all kinds of excuses for delaying their launch. Most

are equivalent to the ones people use for procrastinating in everyday

life. There's something that needs to happen first. Maybe. But

if the software were 100% finished and ready to launch at the push

of a button, would they still be waiting?One reason to launch quickly is that it forces you to actually

finish some quantum of work. Nothing is truly finished till it's

released; you can see that from the rush of work that's always

involved in releasing anything, no matter how finished you thought

it was. The other reason you need to launch is that it's only by

bouncing your idea off users that you fully understand it.Several distinct problems manifest themselves as delays in launching:

working too slowly; not truly understanding the problem; fear of

having to deal with users; fear of being judged; working on too

many different things; excessive perfectionism. Fortunately you

can combat all of them by the simple expedient of forcing yourself

to launch something fairly quickly.9. Launching Too EarlyLaunching too slowly has probably killed a hundred times more

startups than launching too fast, but it is possible to launch too

fast. The danger here is that you ruin your reputation. You launch

something, the early adopters try it out, and if it's no good they

may never come back.So what's the minimum you need to launch? We suggest startups think

about what they plan to do, identify a core that's both (a) useful

on its own and (b) something that can be incrementally expanded

into the whole project, and then get that done as soon as possible.This is the same approach I (and many other programmers) use for

writing software. Think about the overall goal, then start by

writing the smallest subset of it that does anything useful. If

it's a subset, you'll have to write it anyway, so in the worst case

you won't be wasting your time. But more likely you'll find that

implementing a working subset is both good for morale and helps you

see more clearly what the rest should do.The early adopters you need to impress are fairly tolerant. They

don't expect a newly launched product to do everything; it just has

to do something.10. Having No Specific User in MindYou can't build things users like without understanding them. I

mentioned earlier that the most successful startups seem to have

begun by trying to solve a problem their founders had. Perhaps

there's a rule here: perhaps you create wealth in proportion to how

well you understand the problem you're solving, and the problems

you understand best are your own.

[4]That's just a theory. What's not a theory is the converse: if

you're trying to solve problems you don't understand, you're hosed.And yet a surprising number of founders seem willing to

assume that someone, they're not sure exactly who, will want what

they're building. Do the founders want it? No, they're not the

target market. Who is? Teenagers. People interested in local

events (that one is a perennial tarpit). Or "business" users. What

business users? Gas stations? Movie studios? Defense contractors?You can of course build something for users other than yourself.

We did. But you should realize you're stepping into dangerous

territory. You're flying on instruments, in effect, so you should

(a) consciously shift gears, instead of assuming you can rely on

your intuitions as you ordinarily would, and (b) look at the

instruments.In this case the instruments are the users. When designing for

other people you have to be empirical. You can no longer guess

what will work; you have to find users and measure their responses.

So if you're going to make something for teenagers or "business"

users or some other group that doesn't include you, you have to be

able to talk some specific ones into using what you're making. If

you can't, you're on the wrong track.11. Raising Too Little MoneyMost successful startups take funding at some point. Like having

more than one founder, it seems a good bet statistically. How much

should you take, though?Startup funding is measured in time. Every startup that isn't

profitable (meaning nearly all of them, initially) has a certain

amount of time left before the money runs out and they have to stop.

This is sometimes referred to as runway, as in "How much runway do

you have left?" It's a good metaphor because it reminds you that

when the money runs out you're going to be airborne or dead.Too little money means not enough to get airborne. What airborne

means depends on the situation. Usually you have to advance to a

visibly higher level: if all you have is an idea, a working prototype;

if you have a prototype, launching; if you're launched, significant

growth. It depends on investors, because until you're profitable

that's who you have to convince.So if you take money from investors, you have to take enough to get

to the next step, whatever that is.

[5]

Fortunately you have some

control over both how much you spend and what the next step is. We

advise startups to set both low, initially: spend practically

nothing, and make your initial goal simply to build a solid prototype.

This gives you maximum flexibility.12. Spending Too MuchIt's hard to distinguish spending too much from raising too little.

If you run out of money, you could say either was the cause. The

only way to decide which to call it is by comparison with other

startups. If you raised five million and ran out of money, you

probably spent too much.Burning through too much money is not as common as it used to be.

Founders seem to have learned that lesson. Plus it keeps getting

cheaper to start a startup. So as of this writing few startups

spend too much. None of the ones we've funded have. (And not just

because we make small investments; many have gone on to raise further

rounds.)The classic way to burn through cash is by hiring a lot of people.

This bites you twice: in addition to increasing your costs, it slows

you down—so money that's getting consumed faster has to last

longer. Most hackers understand why that happens; Fred Brooks

explained it in The Mythical Man-Month.We have three general suggestions about hiring: (a) don't do it if

you can avoid it, (b) pay people with equity rather than salary,

not just to save money, but because you want the kind of people who

are committed enough to prefer that, and (c) only hire people who

are either going to write code or go out and get users, because

those are the only things you need at first.13. Raising Too Much MoneyIt's obvious how too little money could kill you, but is there such

a thing as having too much?Yes and no. The problem is not so much the money itself as what

comes with it. As one VC who spoke at Y Combinator said, "Once you

take several million dollars of my money, the clock is ticking."

If VCs fund you, they're not going to let you just put the money

in the bank and keep operating as two guys living on ramen. They

want that money to go to work.

[6]

At the very least you'll move

into proper office space and hire more people. That will change

the atmosphere, and not entirely for the better. Now most of your

people will be employees rather than founders. They won't be as

committed; they'll need to be told what to do; they'll start to

engage in office politics.When you raise a lot of money, your company moves to the suburbs

and has kids.Perhaps more dangerously, once you take a lot of money it gets

harder to change direction. Suppose your initial plan was to sell

something to companies. After taking VC money you hire a sales

force to do that. What happens now if you realize you should be

making this for consumers instead of businesses? That's a completely

different kind of selling. What happens, in practice, is that you

don't realize that. The more people you have, the more you stay

pointed in the same direction.Another drawback of large investments is the time they take. The

time required to raise money grows with the amount.

[7]

When the

amount rises into the millions, investors get very cautious. VCs

never quite say yes or no; they just engage you in an apparently

endless conversation. Raising VC scale investments is thus a huge

time sink — more work, probably, than the startup itself. And you

don't want to be spending all your time talking to investors while

your competitors are spending theirs building things.We advise founders who go on to seek VC money to take the first

reasonable deal they get. If you get an offer from a reputable

firm at a reasonable valuation with no unusually onerous terms,

just take it and get on with building the company.

[8]

Who cares

if you could get a 30% better deal elsewhere? Economically, startups

are an all-or-nothing game. Bargain-hunting among investors is a

waste of time.14. Poor Investor ManagementAs a founder, you have to manage your investors. You shouldn't

ignore them, because they may have useful insights. But neither

should you let them run the company. That's supposed to be your

job. If investors had sufficient vision to run the companies

they fund, why didn't they start them?Pissing off investors by ignoring them is probably less dangerous

than caving in to them. In our startup, we erred on the ignoring

side. A lot of our energy got drained

away in disputes with investors instead of going into the product.

But this was less costly than giving in, which would probably have

destroyed the company. If the founders know what they're doing,

it's better to have half their attention focused on the product

than the full attention of investors who don't.How hard you have to work on managing investors usually depends on

how much money you've taken. When you raise VC-scale money, the

investors get a great deal of control. If they have a board majority,

they're literally your bosses. In the more common case, where

founders and investors are equally represented and the deciding

vote is cast by neutral outside directors, all the investors have

to do is convince the outside directors and they control the company.If things go well, this shouldn't matter. So long as you seem to

be advancing rapidly, most investors will leave you alone. But

things don't always go smoothly in startups. Investors have made

trouble even for the most successful companies. One of the most

famous examples is Apple, whose board made a nearly fatal blunder

in firing Steve Jobs. Apparently even Google got a lot of grief

from their investors early on.15. Sacrificing Users to (Supposed) ProfitWhen I said at the beginning that if you make something users want,

you'll be fine, you may have noticed I didn't mention anything about

having the right business model. That's not because making money

is unimportant. I'm not suggesting that founders start companies

with no chance of making money in the hope of unloading them before

they tank. The reason we tell founders not to worry about the

business model initially is that making something people want is

so much harder.I don't know why it's so hard to make something people want. It

seems like it should be straightforward. But you can tell it must

be hard by how few startups do it.Because making something people want is so much harder than making

money from it, you should leave business models for later, just as

you'd leave some trivial but messy feature for version 2. In version

1, solve the core problem. And the core problem in a startup is

how to create wealth

(= how much people want something x the number

who want it), not how to convert that wealth into money.The companies that win are the ones that put users first. Google,

for example. They made search work, then worried about how to make

money from it. And yet some startup founders still think it's

irresponsible not to focus on the business model from the beginning.

They're often encouraged in this by investors whose experience comes

from less malleable industries.It is irresponsible not to think about business models. It's

just ten times more irresponsible not to think about the product.16. Not Wanting to Get Your Hands DirtyNearly all programmers would rather spend their time writing code

and have someone else handle the messy business of extracting money

from it. And not just the lazy ones. Larry and Sergey apparently

felt this way too at first. After developing their new search

algorithm, the first thing they tried was to get some other company

to buy it.Start a company? Yech. Most hackers would rather just have ideas.

But as Larry and Sergey found, there's not much of a market for

ideas. No one trusts an idea till you embody it in a product and

use that to grow a user base. Then they'll pay big time.Maybe this will change, but I doubt it will change much. There's

nothing like users for convincing acquirers. It's not just that

the risk is decreased. The acquirers are human, and they have a

hard time paying a bunch of young guys millions of dollars just for

being clever. When the idea is embodied in a company with a lot

of users, they can tell themselves they're buying the users rather

than the cleverness, and this is easier for them to swallow.

[9]If you're going to attract users, you'll probably have to get up

from your computer and go find some. It's unpleasant work, but if

you can make yourself do it you have a much greater chance of

succeeding. In the first batch of startups we funded, in the summer

of 2005, most of the founders spent all their time building their

applications. But there was one who was away half the time talking

to executives at cell phone companies, trying to arrange deals.

Can you imagine anything more painful for a hacker?

[10]

But it

paid off, because this startup seems the most successful of that

group by an order of magnitude.If you want to start a startup, you have to face the fact that you

can't just hack. At least one hacker will have to spend some of

the time doing business stuff.17. Fights Between FoundersFights between founders are surprisingly common. About 20% of the

startups we've funded have had a founder leave. It happens so often

that we've reversed our attitude to vesting. We still don't require

it, but now we advise founders to vest so there will be an orderly

way for people to quit.A founder leaving doesn't necessarily kill a startup, though. Plenty

of successful startups have had that happen.

[11]

Fortunately it's

usually the least committed founder who leaves. If there are three

founders and one who was lukewarm leaves, big deal. If you have

two and one leaves, or a guy with critical technical skills leaves,

that's more of a problem. But even that is survivable. Blogger

got down to one person, and they bounced back.Most of the disputes I've seen between founders could have been

avoided if they'd been more careful about who they started a company

with. Most disputes are not due to the situation but the people.

Which means they're inevitable. And most founders who've been

burned by such disputes probably had misgivings, which they suppressed,

when they started the company. Don't suppress misgivings. It's

much easier to fix problems before the company is started than

after. So don't include your housemate in your startup because

he'd feel left out otherwise. Don't start a company with someone

you dislike because they have some skill you need and you worry you

won't find anyone else. The people are the most important ingredient

in a startup, so don't compromise there.18. A Half-Hearted EffortThe failed startups you hear most about are the spectacular

flameouts. Those are actually the elite of failures. The most

common type is not the one that makes spectacular mistakes, but the

one that doesn't do much of anything — the one we never even hear

about, because it was some project a couple guys started on the

side while working on their day jobs, but which never got anywhere

and was gradually abandoned.Statistically, if you want to avoid failure, it would seem like the

most important thing is to quit your day job. Most founders of

failed startups don't quit their day jobs, and most founders of

successful ones do. If startup failure were a disease, the CDC

would be issuing bulletins warning people to avoid day jobs.Does that mean you should quit your day job? Not necessarily. I'm

guessing here, but I'd guess that many of these would-be founders

may not have the kind of determination it takes to start a company,

and that in the back of their minds, they know it. The reason they

don't invest more time in their startup is that they know it's a

bad investment.

[12]I'd also guess there's some band of people who could have succeeded

if they'd taken the leap and done it full-time, but didn't. I have

no idea how wide this band is, but if the winner/borderline/hopeless

progression has the sort of distribution you'd expect, the number

of people who could have made it, if they'd quit their day job, is

probably an order of magnitude larger than the number who do make

it.

[13]If that's true, most startups that could succeed fail because the

founders don't devote their whole efforts to them. That certainly

accords with what I see out in the world. Most startups fail because

they don't make something people want, and the reason most don't

is that they don't try hard enough.In other words, starting startups is just like everything else.

The biggest mistake you can make is not to try hard enough. To the

extent there's a secret to success, it's not to be in denial about

that.

Notes[1]

This is not a complete list of the causes of failure,

just those you can control. There are also several you can't,

notably ineptitude and bad luck.[2]

Ironically, one variant of the Facebook that might work is a

facebook exclusively for college students.[3]

Steve Jobs tried to motivate people by saying "Real artists

ship." This is a fine sentence, but unfortunately not true. Many

famous works of art are unfinished. It's true in fields that have

hard deadlines, like architecture and filmmaking, but even there

people tend to be tweaking stuff till it's yanked out of their

hands.[4]

There's probably also a second factor: startup founders tend

to be at the leading edge of technology, so problems they face are

probably especially valuable.[5]

You should take more than you think you'll need, maybe 50% to

100% more, because software takes longer to write and deals longer

to close than you expect.[6]

Since people sometimes call us VCs, I should add that we're

not. VCs invest large amounts of other people's money. We invest

small amounts of our own, like angel investors.[7]

Not linearly of course, or it would take forever to raise five

million dollars. In practice it just feels like it takes forever.Though if you include the cases where VCs don't invest, it would

literally take forever in the median case. And maybe we should,

because the danger of chasing large investments is not just that

they take a long time. That's the best case. The real danger

is that you'll expend a lot of time and get nothing.[8]

Some VCs will offer you an artificially low valuation to see

if you have the balls to ask for more. It's lame that VCs play

such games, but some do. If you're dealing with one of those you

should push back on the valuation a bit.[9]

Suppose YouTube's founders had gone to Google in 2005 and told

them "Google Video is badly designed. Give us $10 million and we'll

tell you all the mistakes you made." They would have gotten

the royal raspberry. Eighteen months later Google paid $1.6 billion

for the same lesson, partly because they could then tell themselves

that they were buying a phenomenon, or a community, or some vague

thing like that.I don't mean to be hard on Google. They did better than their

competitors, who may have now missed the video boat entirely.[10]

Yes, actually: dealing with the government. But phone companies

are up there.[11]

Many more than most people realize, because companies don't advertise

this. Did you know Apple originally had three founders?[12]

I'm not dissing these people. I don't have the determination

myself. I've twice come close to starting startups since Viaweb,

and both times I bailed because I realized that without the spur

of poverty I just wasn't willing to endure the stress of a startup.[13]

So how do you know whether you're in the category of people

who should quit their day job, or the presumably larger one who

shouldn't? I got to the point of saying that this was hard to judge

for yourself and that you should seek outside advice, before realizing

that that's what we do. We think of ourselves as investors, but

viewed from the other direction Y Combinator is a service for

advising people whether or not to quit their day job. We could be

mistaken, and no doubt often are, but we do at least bet money on

our conclusions.Thanks to Sam Altman, Jessica Livingston, Greg McAdoo, and Robert Morris

for reading drafts of this.

Japanese TranslationSpanish TranslationRomanian TranslationChinese TranslationArabic Translation

A Student's Guide to Startups

Want to start a startup? Get funded by

Y Combinator.

October 2006(This essay is derived from a talk at MIT.)Till recently graduating seniors had two choices: get a job or go

to grad school. I think there will increasingly be a third option:

to start your own startup. But how common will that be?I'm sure the default will always be to get a job, but starting a

startup could well become as popular as grad school. In the late

90s my professor friends used to complain that they couldn't get

grad students, because all the undergrads were going to work for

startups. I wouldn't be surprised if that situation returns, but

with one difference: this time they'll be starting their own

instead of going to work for other people's.The most ambitious students will at this point be asking: Why wait

till you graduate? Why not start a startup while you're in college?

In fact, why go to college at all? Why not start a startup instead?A year and a half ago I gave a talk

where I said that the average age of the founders of

Yahoo, Google, and Microsoft was 24, and that if grad students could

start startups, why not undergrads? I'm glad I phrased that as a

question, because now I can pretend it wasn't merely a rhetorical

one. At the time I couldn't imagine why there should be any lower

limit for the age of startup founders. Graduation is a bureaucratic

change, not a biological one. And certainly there are undergrads

as competent technically as most grad students. So why shouldn't

undergrads be able to start startups as well as grad students?I now realize that something does change at graduation: you lose a

huge excuse for failing. Regardless of how complex your life is,

you'll find that everyone else, including your family and friends,

will discard all the low bits and regard you as having a single

occupation at any given time. If you're in college and have a

summer job writing software, you still read as a student. Whereas

if you graduate and get a job programming, you'll be instantly

regarded by everyone as a programmer.The problem with starting a startup while you're still in school

is that there's a built-in escape hatch. If you start a startup

in the summer between your junior and senior year, it reads to

everyone as a summer job.

So if it goes nowhere, big deal; you return to school in the

fall with all the other seniors; no one regards you as a failure,

because your occupation is student, and you didn't fail at that.

Whereas if you start a startup just one year later, after you

graduate, as long as you're not accepted to grad school in the fall

the startup reads to everyone as your occupation. You're

now a startup founder, so you have to do well at that.For nearly everyone, the opinion of one's peers is the most powerful

motivator of all—more powerful even than the nominal goal of most

startup founders, getting rich.

[1]

About a month into each funding

cycle we have an event called Prototype Day where each startup

presents to the others what they've got so far. You might think

they wouldn't need any more motivation. They're working on their

cool new idea; they have funding for the immediate future; and

they're playing a game with only two outcomes: wealth or failure.

You'd think that would be motivation enough. And yet the prospect

of a demo pushes most of them into a

rush of activity.Even if you start a startup explicitly to get rich, the money you

might get seems pretty theoretical most of the time. What drives

you day to day is not wanting to look bad.You probably can't change that. Even if you could, I don't think

you'd want to; someone who really, truly doesn't care what his peers

think of him is probably a psychopath. So the best you can do is

consider this force like a wind, and set up your boat accordingly.

If you know your peers are going to push you in some direction,

choose good peers, and position yourself so they push you in a

direction you like.Graduation changes the prevailing winds, and those make a difference.

Starting a startup is so hard

that it's a close call even for the ones that succeed. However

high a startup may be flying now, it probably has a few leaves stuck

in the landing gear from those trees it barely cleared at the end

of the runway. In such a close game, the smallest increase in the

forces against you can be enough to flick you over the edge into

failure.When we first started Y Combinator

we encouraged people to start

startups while they were still in college. That's partly because

Y Combinator began as a kind of summer program. We've kept the

program shape—all of us having dinner together once a week turns

out to be a good idea—but we've decided now

that the party line should be to tell people to wait till they

graduate.Does that mean you can't start a startup in college? Not at all.

Sam Altman, the co-founder of Loopt,

had just finished his sophomore year when we funded them, and Loopt

is probably the most promising of all the startups we've funded so

far. But Sam Altman is a very unusual guy. Within about three

minutes of meeting him, I remember thinking "Ah, so this is what

Bill Gates must have been like when he was 19."If it can work to start a startup during college, why do

we tell people not to? For the same reason that the probably

apocryphal violinist, whenever he was asked to judge someone's

playing, would always say they didn't have enough talent to make

it as a pro. Succeeding as a musician takes determination as well

as talent, so this answer works out to be the right advice for

everyone. The ones who are uncertain believe it and give up, and

the ones who are sufficiently determined think "screw that, I'll

succeed anyway."So our official policy now is only to fund undergrads we can't talk

out of it. And frankly, if you're not certain, you should wait.

It's not as if all the opportunities to start companies are going

to be gone if you don't do it now. Maybe the window will close on

some idea you're working on, but that won't be the last idea you'll

have. For every idea that times out, new ones become feasible.

Historically the opportunities to start startups have only increased

with time.In that case, you might ask, why not wait longer? Why not go work

for a while, or go to grad school, and then start a startup? And

indeed, that might be a good idea. If I had to pick the sweet spot

for startup founders, based on who we're most excited to see

applications from, I'd say it's probably the mid-twenties. Why?

What advantages does someone in their mid-twenties have over someone

who's 21? And why isn't it older? What can 25 year olds do that

32 year olds can't? Those turn out to be questions worth examining.PlusIf you start a startup soon after college, you'll be a young founder

by present standards, so you should know what the relative advantages

of young founders are. They're not what you might think. As a

young founder your strengths are: stamina, poverty, rootlessness,

colleagues, and ignorance.The importance of stamina shouldn't be surprising. If you've heard

anything about startups you've probably heard about the long hours.

As far as I can tell these are universal. I can't think of any

successful startups whose founders worked 9 to 5. And it's

particularly necessary for younger founders to work long hours

because they're probably not as efficient as they'll be later.Your second advantage, poverty, might not sound like an advantage,

but it is a huge one. Poverty implies you can live cheaply,

and this is critically important for startups. Nearly every startup

that fails, fails by running out of money. It's a little misleading

to put it this way, because there's usually some other underlying

cause. But regardless of the source of your problems, a low burn

rate gives you more opportunity to recover from them. And since

most startups make all kinds of mistakes at first, room to recover

from mistakes is a valuable thing to have.Most startups end up doing something different than they planned.

The way the successful ones find something that works is by trying

things that don't. So the worst thing you can do in a startup is

to have a rigid, pre-ordained plan and then start spending a lot

of money to implement it. Better to operate cheaply and give your

ideas time to evolve.Recent grads can live on practically nothing, and this gives you

an edge over older founders, because the main cost in software

startups is people. The guys with kids and mortgages are at a

real disadvantage. This is one reason I'd bet on the 25 year old

over the 32 year old. The 32 year old probably is a better programmer,

but probably also has a much more expensive life. Whereas a 25

year old has some work experience (more on that later) but can live

as cheaply as an undergrad.Robert Morris and I were 29 and 30 respectively when we started

Viaweb, but fortunately we still lived like 23 year olds. We both had

roughly zero assets. I would have loved to have a mortgage,

since that would have meant I had a house. But in retrospect

having nothing turned out to be convenient. I wasn't tied down and

I was used to living cheaply.Even more important than living cheaply, though, is thinking cheaply.

One reason the Apple II was so popular was that it was cheap. The

computer itself was cheap, and it used cheap, off-the-shelf peripherals

like a cassette tape recorder for data storage and a TV as a monitor.

And you know why? Because Woz designed this computer for himself,

and he couldn't afford anything more.We benefitted from the same phenomenon. Our prices were

daringly low for the time. The top level of service was

$300 a month, which was an order of magnitude below the norm. In

retrospect this was a smart move, but we didn't do it because we

were smart. $300 a month seemed like a lot of money to us. Like

Apple, we created something inexpensive, and therefore popular,

simply because we were poor.A lot of startups have that form: someone comes along and makes

something for a tenth or a hundredth of what it used to cost, and

the existing players can't follow because they don't even want to

think about a world in which that's possible. Traditional long

distance carriers, for example, didn't even want to think about

VoIP. (It was coming, all the same.) Being poor helps in this

game, because your own personal bias points in the same direction

technology evolves in.The advantages of rootlessness are similar to those of poverty.

When you're young you're more mobile—not just because you don't

have a house or much stuff, but also because you're less likely to

have serious relationships. This turns out to be important, because

a lot of startups involve someone moving.The founders of Kiko, for example, are now en route to the Bay Area

to start their next startup. It's a better place for what they

want to do. And it was easy for them to decide to go, because

neither as far as I know has a serious girlfriend, and everything

they own will fit in one car—or more precisely, will either fit

in one car or is crappy enough that they don't mind leaving it

behind.They at least were in Boston. What if they'd been in Nebraska,

like Evan Williams was at their age? Someone wrote recently that

the drawback of Y Combinator was that you had to move to participate.

It couldn't be any other way. The kind of conversations we have

with founders, we have to have in person. We fund a dozen startups

at a time, and we can't be in a dozen places at once. But even if

we could somehow magically save people from moving, we wouldn't.

We wouldn't be doing founders a favor by letting them stay in

Nebraska. Places that aren't

startup hubs are toxic to startups.

You can tell that from indirect evidence. You can tell how hard

it must be to start a startup in Houston or Chicago or Miami from

the microscopically small number, per capita, that succeed

there. I don't know exactly what's suppressing all the startups in these

towns—probably a hundred subtle little things—but something

must be.

[2]Maybe this will change. Maybe the increasing cheapness of startups

will mean they'll be able to survive anywhere, instead of only in

the most hospitable environments. Maybe 37signals is the pattern

for the future. But maybe not. Historically there have always

been certain towns that were centers for certain industries, and

if you weren't in one of them you were at a disadvantage. So my

guess is that 37signals is an anomaly. We're looking at a pattern

much older than "Web 2.0" here.Perhaps the reason more startups per capita happen in the Bay Area

than Miami is simply that there are more founder-type people there.

Successful startups are almost never started by one person. Usually

they begin with a conversation in which someone mentions that

something would be a good idea for a company, and his friend says,

"Yeah, that is a good idea, let's try it." If you're missing that

second person who says "let's try it," the startup never happens.

And that is another area where undergrads have an edge. They're

surrounded by people willing to say that. At a good college you're

concentrated together with a lot of other ambitious and technically

minded people—probably more concentrated than you'll ever be

again. If your nucleus spits out a neutron, there's a good chance

it will hit another nucleus.The number one question people ask us at Y Combinator is: Where can

I find a co-founder? That's the biggest problem for someone starting

a startup at 30. When they were in school they knew a lot of good

co-founders, but by 30 they've either lost touch with them or these

people are tied down by jobs they don't want to leave.Viaweb was an anomaly in this respect too. Though we were comparatively

old, we weren't tied down by impressive jobs. I was trying to be

an artist, which is not very constraining, and Robert, though 29,

was still in grad school due to a little interruption in his academic

career back in 1988. So arguably the Worm made Viaweb possible.

Otherwise Robert would have been a junior professor at that age,

and he wouldn't have had time to work on crazy speculative projects

with me.Most of the questions people ask Y Combinator we have some kind of

answer for, but not the co-founder question. There is no good

answer. Co-founders really should be people you already know. And

by far the best place to meet them is school. You have a large

sample of smart people; you get to compare how they all perform on

identical tasks; and everyone's life is pretty fluid. A lot of

startups grow out of schools for this reason. Google, Yahoo, and

Microsoft, among others, were all founded by people who met in

school. (In Microsoft's case, it was high school.)Many students feel they should wait and get a little more experience

before they start a company. All other things being equal, they

should. But all other things are not quite as equal as they look.

Most students don't realize how rich they are in the scarcest

ingredient in startups, co-founders. If you wait too long, you may

find that your friends are now involved in some project they don't

want to abandon. The better they are, the more likely this is to

happen.One way to mitigate this problem might be to actively plan your

startup while you're getting those n years of experience. Sure,

go off and get jobs or go to grad school or whatever, but get

together regularly to scheme, so the idea of starting a startup

stays alive in everyone's brain. I don't know if this works, but

it can't hurt to try.It would be helpful just to realize what an advantage you have as

students. Some of your classmates are probably going to be successful

startup founders; at a great technical university, that is a near

certainty. So which ones? If I were you I'd look for the people

who are not just smart, but incurable

builders.

Look

for the people who keep starting projects, and finish at least some

of them. That's what we look for. Above all else, above academic

credentials and even the idea you apply with, we look for people

who build things.The other place co-founders meet is at work. Fewer do than at

school, but there are things you can do to improve the odds. The

most important, obviously, is to work somewhere that has a lot of

smart, young people. Another is to work for a company located in

a startup hub. It will be easier to talk a co-worker into quitting

with you in a place where startups are happening all around you.You might also want to look at the employment agreement you sign

when you get hired. Most will say that any ideas you think of while

you're employed by the company belong to them. In practice it's

hard for anyone to prove what ideas you had when, so the line gets

drawn at code. If you're going to start a startup, don't write any

of the code while you're still employed. Or at least discard any

code you wrote while still employed and start over. It's not so

much that your employer will find out and sue you. It won't come

to that; investors or acquirers or (if you're so lucky) underwriters

will nail you first. Between t = 0 and when you buy that yacht,

someone is going to ask if any of your code legally belongs

to anyone else, and you need to be able to say no.

[3]The most overreaching employee agreement I've seen so far is Amazon's.

In addition to the usual clauses about owning your ideas, you also

can't be a founder of a startup that has another founder who worked

at Amazon—even if you didn't know them or even work there at the

same time. I suspect they'd have a hard time enforcing this, but

it's a bad sign they even try. There are plenty of other places

to work; you may as well choose one that keeps more of your options

open.Speaking of cool places to work, there is of course Google. But I

notice something slightly frightening about Google: zero startups

come out of there. In that respect it's a black hole. People seem

to like working at Google too much to leave. So if you hope to start

a startup one day, the evidence so far suggests you shouldn't work

there.I realize this seems odd advice. If they make your life so good

that you don't want to leave, why not work there? Because, in

effect, you're probably getting a local maximum. You need a certain

activation energy to start a startup. So an employer who's fairly

pleasant to work for can lull you into staying indefinitely, even

if it would be a net win for you to leave.

[4]The best place to work, if you want to start a startup, is probably

a startup. In addition to being the right sort of experience, one

way or another it will be over quickly. You'll either end up rich,

in which case problem solved, or the startup will get bought, in

which case it it will start to suck to work there and it will be

easy to leave, or most likely, the thing will blow up and you'll

be free again.Your final advantage, ignorance, may not sound very useful. I

deliberately used a controversial word for it; you might equally

call it innocence. But it seems to be a powerful force. My Y

Combinator co-founder Jessica Livingston is just about to publish

a book of interviews

with startup founders, and I noticed a remarkable pattern in them.

One after another said that if they'd known how hard it would be,

they would have been too intimidated to start.Ignorance can be useful when it's a counterweight to other forms

of stupidity. It's useful in starting startups because you're

capable of more than you realize. Starting startups is harder than

you expect, but you're also capable of more than you expect, so

they balance out.Most people look at a company like Apple and think, how could I

ever make such a thing? Apple is an institution, and I'm just a

person. But every institution was at one point just a handful of

people in a room deciding to start something. Institutions are

made up, and made up by people no different from you.I'm not saying everyone could start a startup. I'm sure most people

couldn't; I don't know much about the population at large. When

you get to groups I know well, like hackers, I can say more precisely.

At the top schools, I'd guess as many as a quarter of the CS majors

could make it as startup founders if they wanted.That "if they wanted" is an important qualification—so important

that it's almost cheating to append it like that—because once you

get over a certain threshold of intelligence, which most CS majors

at top schools are past, the deciding factor in whether you succeed

as a founder is how much you want to. You don't have to be that

smart. If you're not a genius, just start a startup in some unsexy

field where you'll have less competition, like software for human

resources departments. I picked that example at random, but I feel

safe in predicting that whatever they have now, it wouldn't take

genius to do better. There are a lot of people out there working

on boring stuff who are desperately in need of better software, so

however short you think you fall of Larry and Sergey, you can ratchet

down the coolness of the idea far enough to compensate.As well as preventing you from being intimidated, ignorance can

sometimes help you discover new ideas. Steve Wozniak

put this very strongly:

All the best things that I did at Apple came from (a) not having

money and (b) not having done it before, ever. Every single thing

that we came out with that was really great, I'd never once done

that thing in my life.

When you know nothing, you have to reinvent stuff for yourself, and

if you're smart your reinventions may be better than what preceded

them. This is especially true in fields where the rules change.

All our ideas about software were developed in a time when processors

were slow, and memories and disks were tiny. Who knows what obsolete

assumptions are embedded in the conventional wisdom? And the way

these assumptions are going to get fixed is not by explicitly

deallocating them, but by something more akin to garbage collection.

Someone ignorant but smart will come along and reinvent everything,

and in the process simply fail to reproduce certain existing ideas.MinusSo much for the advantages of young founders. What about the

disadvantages? I'm going to start with what goes wrong and try to

trace it back to the root causes.What goes wrong with young founders is that they build stuff that

looks like class projects. It was only recently that we figured

this out ourselves. We noticed a lot of similarities between the

startups that seemed to be falling behind, but we couldn't figure

out how to put it into words. Then finally we realized what it

was: they were building class projects.But what does that really mean? What's wrong with class projects?

What's the difference between a class project and a real startup?

If we could answer that question it would be useful not just to

would-be startup founders but to students in general, because we'd

be a long way toward explaining the mystery of the so-called real

world.There seem to be two big things missing in class projects: (1) an

iterative definition of a real problem and (2) intensity.The first is probably unavoidable. Class projects will inevitably

solve fake problems. For one thing, real problems are rare and

valuable. If a professor wanted to have students solve real problems,

he'd face the same paradox as someone trying to give an example of

whatever "paradigm" might succeed the Standard Model of physics.

There may well be something that does, but if you could think of

an example you'd be entitled to the Nobel Prize. Similarly, good

new problems are not to be had for the asking.In technology the difficulty is compounded by the fact that real

startups tend to discover the problem they're solving by a process

of evolution. Someone has an idea for something; they build it;

and in doing so (and probably only by doing so) they realize

the problem they should be solving is another one. Even if the

professor let you change your project description on the fly, there

isn't time enough to do that in a college class, or a market to

supply evolutionary pressures. So class

projects are mostly about implementation, which is the least

of your problems in a startup.It's not just that in a startup you work on the idea as well as

implementation. The very implementation is different. Its main

purpose is to refine the idea. Often the only value of most of the

stuff you build in the first six months is that it proves your

initial idea was mistaken. And that's extremely valuable. If

you're free of a misconception that everyone else still shares,

you're in a powerful position. But you're not thinking that way

about a class project. Proving your initial plan was mistaken would

just get you a bad grade. Instead of building stuff to throw away,

you tend to want every line of code to go toward that final goal

of showing you did a lot of work.That leads to our second difference: the way class projects are

measured. Professors will tend to judge you by the distance between

the starting point and where you are now. If someone has achieved

a lot, they should get a good grade. But customers will judge you

from the other direction: the distance remaining between where you

are now and the features they need. The market doesn't give a shit

how hard you worked. Users just want your software to do what they

need, and you get a zero otherwise. That is one of the most

distinctive differences between school and the real world: there

is no reward for putting in a good effort. In fact, the whole

concept of a "good effort" is a fake idea adults invented to encourage

kids. It is not found in nature.Such lies seem to be helpful to kids. But unfortunately when you

graduate they don't give you a list of all the lies they told you

during your education. You have to get them beaten out of you by

contact with the real world. And this is why so many jobs want

work experience. I couldn't understand that when I was in college.

I knew how to program. In fact, I could tell I knew how to program

better than most people doing it for a living. So what was this

mysterious "work experience" and why did I need it?Now I know what it is, and part of the confusion is grammatical.

Describing it as "work experience" implies it's like experience

operating a certain kind of machine, or using a certain programming

language. But really what work experience refers to is not some

specific expertise, but the elimination of certain habits left over

from childhood.One of the defining qualities of kids is that they flake. When

you're a kid and you face some hard test, you can cry and say "I

can't" and they won't make you do it. Of course, no one can make

you do anything in the grownup world either. What they do instead

is fire you. And when motivated by that

you find you can do a lot more than you realized. So one of the

things employers expect from someone with "work experience" is the

elimination of the flake reflex—the ability to get things done,

with no excuses.The other thing you get from work experience is an understanding

of what work is, and in particular, how intrinsically horrible it

is. Fundamentally the equation is a brutal one: you have to spend

most of your waking hours doing stuff someone else wants, or starve.

There are a few places where the work is so interesting that this

is concealed, because what other people want done happens to coincide

with what you want to work on. But you only have to imagine what

would happen if they diverged to see the underlying reality.It's not so much that adults lie to kids about this as never explain

it. They never explain what the deal is with money. You know from

an early age that you'll have some sort of job, because everyone

asks what you're going to "be" when you grow up. What they

don't tell you is that as a kid you're sitting on the shoulders of

someone else who's treading water, and that starting working means

you get thrown into the water on your own, and have to start treading

water yourself or sink. "Being" something is incidental; the

immediate problem is not to drown.The relationship between work and money tends to dawn on you only

gradually. At least it did for me. One's first thought tends to

be simply "This sucks. I'm in debt. Plus I have to get up on monday

and go to work." Gradually you realize that these two things are

as tightly connected as only a market can make them.So the most important advantage 24 year old founders have over 20

year old founders is that they know what they're trying to avoid.

To the average undergrad the idea of getting rich translates into

buying Ferraris, or being admired. To someone who has learned from

experience about the relationship between money and work, it

translates to something way more important: it means you get to opt

out of the brutal equation that governs the lives of 99.9% of people.

Getting rich means you can stop treading water.Someone who gets this will work much harder at making a startup

succeed—with the proverbial energy of a drowning man, in fact.

But understanding the relationship between money and work also

changes the way you work. You don't get money just for working,

but for doing things other people want. Someone who's figured that

out will automatically focus more on the user. And that cures the

other half of the class-project syndrome. After you've been working

for a while, you yourself tend to measure what you've done the same

way the market does.Of course, you don't have to spend years working to learn this

stuff. If you're sufficiently perceptive you can grasp these things

while you're still in school. Sam Altman did. He must have, because

Loopt is no class project. And as his example suggests, this can

be valuable knowledge. At a minimum, if you get this stuff, you

already have most of what you gain from the "work experience"

employers consider so desirable. But of course if you really get

it, you can use this information in a way that's more valuable to

you than that.NowSo suppose you think you might start a startup at some point, either

when you graduate or a few years after. What should you do now?

For both jobs and grad school, there are ways to prepare while

you're in college. If you want to get a job when you graduate, you

should get summer jobs at places you'd like to work. If you want

to go to grad school, it will help to work on research projects as

an undergrad. What's the equivalent for startups? How do you keep

your options maximally open?One thing you can do while you're still in school is to learn how

startups work. Unfortunately that's not easy. Few if any colleges

have classes about startups. There may be business school classes

on entrepreneurship, as they call it over there, but these are

likely to be a waste of time. Business schools like to talk about

startups, but philosophically they're at the opposite end of the

spectrum. Most books on startups also seem to be useless. I've

looked at a few and none get it right. Books in most fields are

written by people who know the subject from experience, but for

startups there's a unique problem: by definition the founders of

successful startups don't need to write books to make money. As a

result most books on the subject end up being written by people who

don't understand it.So I'd be skeptical of classes and books. The way to learn about

startups is by watching them in action, preferably by working at

one. How do you do that as an undergrad? Probably by sneaking in

through the back door. Just hang around a lot and gradually start

doing things for them. Most startups are (or should be) very

cautious about hiring. Every hire increases the burn rate, and bad

hires early on are hard to recover from. However, startups usually

have a fairly informal atmosphere, and there's always a lot that

needs to be done. If you just start doing stuff for them, many

will be too busy to shoo you away. You can thus gradually work

your way into their confidence, and maybe turn it into an official

job later, or not, whichever you prefer. This won't work for all

startups, but it would work for most I've known.Number two, make the most of the great advantage of school: the

wealth of co-founders. Look at the people around you and ask

yourself which you'd like to work with. When you apply that test,

you may find you get surprising results. You may find you'd prefer

the quiet guy you've mostly ignored to someone who seems impressive

but has an attitude to match. I'm not suggesting you suck up to

people you don't really like because you think one day they'll be

successful. Exactly the opposite, in fact: you should only start

a startup with someone you like, because a startup will put your

friendship through a stress test. I'm just saying you should think

about who you really admire and hang out with them, instead of

whoever circumstances throw you together with.Another thing you can do is learn skills that will be useful to you

in a startup. These may be different from the skills you'd learn

to get a job. For example, thinking about getting a job will make

you want to learn programming languages you think employers want,

like Java and C++. Whereas if you start a startup, you get to pick

the language, so you have to think about which will actually let

you get the most done. If you use that test you might end up

learning Ruby or Python instead.

But the most important skill for a startup founder isn't a programming

technique. It's a knack for understanding users and figuring out

how to give them what they want. I know I repeat this, but that's

because it's so important. And it's a skill you can learn, though

perhaps habit might be a better word. Get into the habit of thinking

of software as having users. What do those users want? What would

make them say wow?This is particularly valuable for undergrads, because the concept

of users is missing from most college programming classes. The way

you get taught programming in college would be like teaching writing

as grammar, without mentioning that its purpose is to communicate

something to an audience. Fortunately an audience for software is

now only an http request away. So in addition to the programming

you do for your classes, why not build some kind of website people

will find useful? At the very least it will teach you how to write

software with users. In the best case, it might not just be

preparation for a startup, but the startup itself, like it was for

Yahoo and Google.Notes[1]

Even the desire to protect one's children seems weaker, judging

from things people have historically done to their kids

rather than risk their community's disapproval. (I assume we still

do things that will be regarded in the future as barbaric, but

historical abuses are easier for us to see.)[2]

Worrying that Y Combinator makes founders move for 3 months

also suggests one underestimates how hard it is to start a startup.

You're going to have to put up with much greater inconveniences than

that.[3]

Most employee agreements

say that any idea relating to the company's present or potential

future business belongs to them. Often as not the second clause could

include any possible startup, and anyone doing due diligence for an

investor or acquirer will assume the worst.To be safe either (a) don't use code written while you

were still employed in your previous job, or (b) get your employer to

renounce, in writing, any claim to the code you write for your side

project. Many will consent to (b) rather than

lose a prized employee. The downside is that you'll have to tell them

exactly what your project does.[4]

Geshke and Warnock only founded Adobe because Xerox ignored

them. If Xerox had used what they built, they would probably

never have left PARC.Thanks to Jessica Livingston and Robert Morris for reading

drafts of this, and to Jeff Arnold and the SIPB for inviting me to

speak.

Comment on this essay.Chinese TranslationArabic Translation

How to Present to Investors

Want to start a startup? Get funded by

Y Combinator.

August 2006, rev. April 2007, September 2010In a few days it will be Demo Day, when the startups we funded

this summer present to investors. Y Combinator funds startups twice

a year, in January and June. Ten weeks later we invite all the

investors we know to hear them present what they've built so far.Ten weeks is not much time. The average startup probably doesn't

have much to show for itself after ten weeks. But the average

startup fails. When you look at the ones that went on to do great

things, you find a lot that began with someone pounding out a

prototype in a week or two of nonstop work. Startups are a

counterexample to the rule that haste makes waste.(Too much money seems to be as bad for startups as too much time,

so we don't give them much money either.)A week before Demo Day, we have a dress rehearsal called Rehearsal Day.

At other Y Combinator events we allow outside guests, but not at

Rehearsal Day. No one except the other founders gets to see the rehearsals.The presentations on Rehearsal Day are often pretty rough. But this is

to be expected. We try to pick founders who are good at building

things, not ones who are slick presenters. Some of the founders

are just out of college, or even still in it, and have never spoken

to a group of people they didn't already know.So we concentrate on the basics. On Demo Day each startup will

only get ten minutes, so we encourage them to focus on just two

goals: (a) explain what you're doing, and (b) explain why users

will want it.That might sound easy, but it's not when the speakers have no

experience presenting, and they're explaining technical matters to

an audience that's mostly non-technical.This situation is constantly repeated when startups present to

investors: people who are bad at explaining, talking to people who

are bad at understanding. Practically every successful startup,

including stars like Google, presented at some point to investors

who didn't get it and turned them down. Was it because the founders

were bad at presenting, or because the investors were obtuse? It's

probably always some of both.At the most recent Rehearsal Day, we four Y Combinator partners found

ourselves saying a lot of the same things we said at the last two.

So at dinner afterward we collected all our tips about presenting

to investors. Most startups face similar challenges, so we hope

these will be useful to a wider audience.

1. Explain what you're doing.Investors' main question when judging a very early startup is whether

you've made a compelling product. Before they can judge whether

you've built a good x, they have to understand what kind of x you've

built. They will get very frustrated if instead of telling them

what you do, you make them sit through some kind of preamble.Say what you're doing as soon as possible, preferably in the first

sentence. "We're Jeff and Bob and we've built an easy to use web-based

database. Now we'll show it to you and explain why people need

this."If you're a great public speaker you may be able to violate this

rule. Last year one founder spent the whole first half of his talk

on a fascinating analysis of the limits of the conventional desktop

metaphor. He got away with it, but unless you're a captivating

speaker, which most hackers aren't, it's better to play it safe.2. Get rapidly to demo.This section is now obsolete for YC founders presenting

at Demo Day, because Demo Day presentations are now so short

that they rarely include much if any demo. They seem to work

just as well without, however, which makes me think I was

wrong to emphasize demos so much before.A demo explains what you've made more effectively than any verbal

description. The only thing worth talking about first is the problem

you're trying to solve and why it's important. But don't spend

more than a tenth of your time on that. Then demo.When you demo, don't run through a catalog of features. Instead

start with the problem you're solving, and then show how your product

solves it. Show features in an order driven by some kind of purpose,

rather than the order in which they happen to appear on the screen.If you're demoing something web-based, assume that the network

connection will mysteriously die 30 seconds into your presentation,

and come prepared with a copy of the server software running on

your laptop.3. Better a narrow description than a vague one.One reason founders resist describing their projects concisely is

that, at this early stage, there are all kinds of possibilities.

The most concise descriptions seem misleadingly narrow. So for

example a group that has built an easy web-based database might

resist calling their applicaton that, because it could be so much

more. In fact, it could be anything...The problem is, as you approach (in the calculus sense) a description

of something that could be anything, the content of your description

approaches zero. If you describe your web-based database as "a

system to allow people to collaboratively leverage the value of

information," it will go in one investor ear and out the other.

They'll just discard that sentence as meaningless boilerplate, and

hope, with increasing impatience, that in the next sentence you'll

actually explain what you've made.Your primary goal is not to describe everything your system might

one day become, but simply to convince investors you're worth talking

to further. So approach this like an algorithm that gets the right

answer by successive approximations. Begin with a description

that's gripping but perhaps overly narrow, then flesh it out to the

extent you can. It's the same principle as incremental development:

start with a simple prototype, then add features, but at every point

have working code. In this case, "working code" means a working

description in the investor's head.4. Don't talk and drive.Have one person talk while another uses the computer. If the same

person does both, they'll inevitably mumble downwards at the computer

screen instead of talking clearly at the audience.As long as you're standing near the audience and looking at them,

politeness (and habit) compel them to pay attention to you. Once

you stop looking at them to fuss with something on your computer,

their minds drift off to the errands they have to run later.5. Don't talk about secondary matters at length.If you only have a few minutes, spend them explaining what your

product does and why it's great. Second order issues like competitors

or resumes should be single slides you go through quickly at the

end. If you have impressive resumes, just flash them on the screen

for 15 seconds and say a few words. For competitors, list the top

3 and explain in one sentence each what they lack

that you have. And put this kind of thing at the end, after you've

made it clear what you've built.6. Don't get too deeply into business models.It's good to talk about how you plan to make money, but mainly

because it shows you care about that and have thought about it.

Don't go into detail about your business model, because (a) that's

not what smart investors care about in a brief presentation, and

(b) any business model you have at this point is probably wrong

anyway.Recently a VC who came to speak at Y Combinator talked about a

company he just invested in. He said their business model was wrong

and would probably change three times before they got it right.

The founders were experienced guys who'd done startups before and

who'd just succeeded in getting millions from one of the top VC

firms, and even their business model was crap. (And yet he invested

anyway, because he expected it to be crap at this stage.)If you're solving an important problem, you're going to sound a lot

smarter talking about that than the business model. The business

model is just a bunch of guesses, and guesses about stuff that's

probably not your area of expertise. So don't spend your precious

few minutes talking about crap when you could be talking about

solid, interesting things you know a lot about: the problem you're

solving and what you've built so far.As well as being a bad use of time, if your business model seems

spectacularly wrong, that will push the stuff you want investors

to remember out of their heads. They'll just remember you as the

company with the boneheaded plan for making money, rather than the

company that solved that important problem.7. Talk slowly and clearly at the audience.Everyone at Rehearsal Day could see the difference between the people

who'd been out in the world for a while and had presented to groups,

and those who hadn't.You need to use a completely different voice and manner talking to

a roomful of people than you would in conversation. Everyday life

gives you no practice in this. If you can't already do it, the

best solution is to treat it as a consciously artificial trick,

like juggling.However, that doesn't mean you should talk like some kind of

announcer. Audiences tune that out. What you need to do is talk

in this artificial way, and yet make it seem conversational. (Writing

is the same. Good writing is an elaborate effort to seem spontaneous.)If you want to write out your whole presentation beforehand and

memorize it, that's ok. That has worked for some groups in the

past. But make sure to write something that sounds like spontaneous,

informal speech, and deliver it that way too.Err on the side of speaking slowly. At Rehearsal Day, one of the founders

mentioned a rule actors use: if you feel you're speaking too slowly,

you're speaking at about the right speed.8. Have one person talk.Startups often want to show that all the founders are equal partners.

This is a good instinct; investors dislike unbalanced teams. But

trying to show it by partitioning the presentation is going too

far. It's distracting. You can demonstrate your respect

for one another in more subtle ways. For example, when one of the

groups presented at Demo Day, the more extroverted of the two

founders did most of the talking, but he described his co-founder

as the best hacker he'd ever met, and you could tell he meant it.Pick the one or at most two best speakers, and have them do most

of the talking.Exception: If one of the founders is an expert in some specific

technical field, it can be good for them to talk about that for a

minute or so. This kind of "expert witness" can add credibility,

even if the audience doesn't understand all the details. If Jobs

and Wozniak had 10 minutes to present the Apple II, it might be a good plan

to have Jobs speak for 9 minutes and have Woz speak for a minute

in the middle about some of the technical feats he'd pulled off in

the design. (Though of course if it were actually those two, Jobs

would speak for the entire 10 minutes.)9. Seem confident.Between the brief time available and their lack of technical

background, many in the audience will have a hard time evaluating

what you're doing. Probably the single biggest piece of evidence,

initially, will be your own confidence in it. You have

to show you're impressed with what you've made.And I mean show, not tell. Never say "we're passionate" or "our

product is great." People just ignore that—or worse, write you

off as bullshitters. Such messages must be implicit.What you must not do is seem nervous and apologetic. If you've

truly made something good, you're doing investors a favor by

telling them about it. If you don't genuinely believe that, perhaps

you ought to change what your company is doing. If you don't believe

your startup has such promise that you'd be doing them a favor by

letting them invest, why are you investing your time in it?10. Don't try to seem more than you are.Don't worry if your company is just a few months old and doesn't

have an office yet, or your founders are technical people with no

business experience. Google was like that once, and they turned out

ok. Smart investors can see past such superficial flaws. They're

not looking for finished, smooth presentations. They're looking

for raw talent. All you need to convince them of is that you're

smart and that you're onto something good. If you try too hard to

conceal your rawness—by trying to seem corporate, or pretending

to know about stuff you don't—you may just conceal your talent.You can afford to be candid about what you haven't figured out yet.

Don't go out of your way to bring it up (e.g. by having a slide

about what might go wrong), but don't try to pretend either that

you're further along than you are. If you're a hacker and you're

presenting to experienced investors, they're probably better at

detecting bullshit than you are at producing it.11. Don't put too many words on slides.When there are a lot of words on a slide, people just skip reading

it. So look at your slides and ask of each word "could I cross

this out?" This includes gratuitous clip art. Try to get your

slides under 20 words if you can.Don't read your slides. They should be something in the background

as you face the audience and talk to them, not something you face

and read to an audience sitting behind you.Cluttered sites don't do well in demos, especially when they're

projected onto a screen. At the very least, crank up the font size

big enough to make all the text legible. But cluttered sites are

bad anyway, so perhaps you should use this opportunity to make your

design simpler.12. Specific numbers are good.If you have any kind of data, however preliminary, tell the audience.

Numbers stick in people's heads. If you can claim that the median

visitor generates 12 page views, that's great.But don't give them more than four or five numbers, and only give

them numbers specific to you. You don't need to tell them the size

of the market you're in. Who cares, really, if it's 500 million

or 5 billion a year? Talking about that is like an actor at the

beginning of his career telling his parents how much Tom Hanks

makes. Yeah, sure, but first you have to become Tom Hanks. The

important part is not whether he makes ten million a year or a

hundred, but how you get there.13. Tell stories about users.The biggest fear of investors looking at early stage startups is

that you've built something based on your own a priori theories of

what the world needs, but that no one will actually want. So it's

good if you can talk about problems specific users have and how you

solve them.Greg Mcadoo said one thing Sequoia looks for is the "proxy for

demand." What are people doing now, using inadequate tools, that

shows they need what you're making?Another sign of user need is when people pay a lot for something.

It's easy to convince investors there will be demand for

a cheaper alternative to something popular, if you preserve

the qualities that made it popular.The best stories about user needs are about your own. A remarkable

number of famous startups grew out of some need the founders had:

Apple, Microsoft, Yahoo, Google. Experienced investors know that,

so stories of this type will get their attention. The next best

thing is to talk about the needs of people you know personally,

like your friends or siblings.14. Make a soundbite stick in their heads.Professional investors hear a lot of pitches. After a while they

all blur together. The first cut is simply to be one of those

they remember. And the way to ensure that is to create a descriptive

phrase about yourself that sticks in their heads.In Hollywood, these phrases seem to be of the form "x meets y."

In the startup world, they're usually "the x of y" or "the x y."

Viaweb's was "the Microsoft Word of ecommerce."Find one and launch it clearly (but apparently casually) in your

talk, preferably near the beginning.It's a good exercise for you, too, to sit down and try to figure

out how to describe your startup in one compelling phrase. If you

can't, your plans may not be sufficiently focused.How to Fund a StartupHackers' Guide to InvestorsSpanish TranslationJapanese TranslationRussian Translation

Image: Casey Muller: Trevor Blackwell at Rehearsal Day, summer 2006

Copy What You Like

July 2006

When I was in high school I spent a lot of time imitating bad

writers. What we studied in English classes was mostly fiction,

so I assumed that was the highest form of writing. Mistake number

one. The stories that seemed to be most admired were ones in which

people suffered in complicated ways. Anything funny or

gripping was ipso facto suspect, unless it was old enough to be hard to

understand, like Shakespeare or Chaucer. Mistake number two. The

ideal medium seemed the short story, which I've since learned had

quite a brief life, roughly coincident with the peak of magazine

publishing. But since their size made them perfect for use in

high school classes, we read a lot of them, which gave us the

impression the short story was flourishing. Mistake number three.

And because they were so short, nothing really had to happen; you

could just show a randomly truncated slice of life, and that was

considered advanced. Mistake number four. The result was that I

wrote a lot of stories in which nothing happened except that someone

was unhappy in a way that seemed deep.For most of college I was a philosophy major. I was very impressed

by the papers published in philosophy journals. They were so

beautifully typeset, and their tone was just captivating—alternately

casual and buffer-overflowingly technical. A fellow would be walking

along a street and suddenly modality qua modality would spring upon

him. I didn't ever quite understand these papers, but I figured

I'd get around to that later, when I had time to reread them more

closely. In the meantime I tried my best to imitate them. This

was, I can now see, a doomed undertaking, because they weren't

really saying anything. No philosopher ever refuted another, for

example, because no one said anything definite enough to refute.

Needless to say, my imitations didn't say anything either.In grad school I was still wasting time imitating the wrong things.

There was then a fashionable type of program called an expert system,

at the core of which was something called an inference engine. I

looked at what these things did and thought "I could write that in

a thousand lines of code." And yet eminent professors were writing

books about them, and startups were selling them for a year's salary

a copy. What an opportunity, I thought; these impressive things

seem easy to me; I must be pretty sharp. Wrong. It was simply a

fad. The books the professors wrote about expert systems are now

ignored. They were not even on a path to anything interesting.

And the customers paying so much for them were largely the same

government agencies that paid thousands for screwdrivers and toilet

seats.How do you avoid copying the wrong things? Copy only what you

genuinely like. That would have saved me in all three cases. I

didn't enjoy the short stories we had to read in English classes;

I didn't learn anything from philosophy papers; I didn't use expert

systems myself. I believed these things were good because they

were admired.It can be hard to separate the things you like from the things

you're impressed with. One trick is to ignore presentation. Whenever

I see a painting impressively hung in a museum, I ask myself: how

much would I pay for this if I found it at a garage sale, dirty and

frameless, and with no idea who painted it? If you walk around a

museum trying this experiment, you'll find you get some truly

startling results. Don't ignore this data point just because it's

an outlier.Another way to figure out what you like is to look at what you enjoy

as guilty pleasures. Many things people like, especially if they're

young and ambitious, they like largely for the feeling of virtue

in liking them. 99% of people reading Ulysses are thinking

"I'm reading Ulysses" as they do it. A guilty pleasure is

at least a pure one. What do you read when you don't feel up to being

virtuous? What kind of book do you read and feel sad that there's

only half of it left, instead of being impressed that you're half

way through? That's what you really like.Even when you find genuinely good things to copy, there's another

pitfall to be avoided. Be careful to copy what makes them good,

rather than their flaws. It's easy to be drawn into imitating

flaws, because they're easier to see, and of course easier to copy

too. For example, most painters in the eighteenth and nineteenth

centuries used brownish colors. They were imitating the great

painters of the Renaissance, whose paintings by that time were brown

with dirt. Those paintings have since been cleaned, revealing

brilliant colors; their imitators are of course still brown.It was painting, incidentally, that cured me of copying the wrong

things. Halfway through grad school I decided I wanted to try being

a painter, and the art world was so manifestly corrupt that it

snapped the leash of credulity. These people made philosophy

professors seem as scrupulous as mathematicians. It was so clearly

a choice of doing good work xor being an insider that I was forced

to see the distinction. It's there to some degree in almost every

field, but I had till then managed to avoid facing it.That was one of the most valuable things I learned from painting:

you have to figure out for yourself what's

good. You can't trust

authorities. They'll lie to you on this one.

Comment on this essay.Chinese TranslationRomanian TranslationSpanish TranslationRussian Translation

The Island Test

July 2006I've discovered a handy test for figuring out what you're addicted

to. Imagine you were going to spend the weekend at a friend's house

on a little island off the coast of Maine. There are no shops on

the island and you won't be able to leave while you're there. Also,

you've never been to this house before, so you can't assume it will

have more than any house might.What, besides clothes and toiletries, do you make a point of packing?

That's what you're addicted to. For example, if you find yourself

packing a bottle of vodka (just in case), you may want to stop and

think about that.For me the list is four things: books, earplugs, a notebook, and a

pen.There are other things I might bring if I thought of it, like music,

or tea, but I can live without them. I'm not so addicted to caffeine

that I wouldn't risk the house not having any tea, just for a

weekend.Quiet is another matter. I realize it seems a bit eccentric to

take earplugs on a trip to an island off the coast of Maine. If

anywhere should be quiet, that should. But what if the person in

the next room snored? What if there was a kid playing basketball?

(Thump, thump, thump... thump.) Why risk it? Earplugs are small.Sometimes I can think with noise. If I already have momentum on

some project, I can work in noisy places. I can edit an essay or

debug code in an airport. But airports are not so bad: most of the

noise is whitish. I couldn't work with the sound of a sitcom coming

through the wall, or a car in the street playing thump-thump music.And of course there's another kind of thinking, when you're starting

something new, that requires complete quiet. You never

know when this will strike. It's just as well to carry plugs.The notebook and pen are professional equipment, as it were. Though

actually there is something druglike about them, in the sense that

their main purpose is to make me feel better. I hardly ever go

back and read stuff I write down in notebooks. It's just that if

I can't write things down, worrying about remembering one idea gets

in the way of having the next. Pen and paper wick ideas.The best notebooks I've found are made by a company called Miquelrius.

I use their smallest size, which is about 2.5 x 4 in.

The secret to writing on such

narrow pages is to break words only when you run out of space, like

a Latin inscription. I use the cheapest plastic Bic ballpoints,

partly because their gluey ink doesn't seep through pages, and

partly so I don't worry about losing them.I only started carrying a notebook about three years ago. Before

that I used whatever scraps of paper I could find. But the problem

with scraps of paper is that they're not ordered. In a notebook

you can guess what a scribble means by looking at the pages

around it. In the scrap era I was constantly finding notes I'd

written years before that might say something I needed to remember,

if I could only figure out what.As for books, I know the house would probably have something to

read. On the average trip I bring four books and only read one of

them, because I find new books to read en route. Really bringing

books is insurance.I realize this dependence on books is not entirely good—that what

I need them for is distraction. The books I bring on trips are

often quite virtuous, the sort of stuff that might be assigned

reading in a college class. But I know my motives aren't virtuous.

I bring books because if the world gets boring I need to be able

to slip into another distilled by some writer. It's like eating

jam when you know you should be eating fruit.There is a point where I'll do without books. I was walking in

some steep mountains once, and decided I'd rather just think, if I

was bored, rather than carry a single unnecessary ounce. It wasn't

so bad. I found I could entertain myself by having ideas instead

of reading other people's. If you stop eating jam, fruit starts

to taste better.So maybe I'll try not bringing books on some future trip. They're

going to have to pry the plugs out of my cold, dead ears, however.Spanish TranslationJapanese Translation

The Power of the Marginal

Want to start a startup? Get funded by

Y Combinator.

June 2006(This essay is derived from talks at Usenix 2006 and

Railsconf 2006.)A couple years ago my friend Trevor and I went to look at the Apple

garage. As we stood there, he said that as a kid growing up in

Saskatchewan he'd been amazed at the dedication Jobs and Wozniak

must have had to work in a garage."Those guys must have been

freezing!"That's one of California's hidden advantages: the mild climate means

there's lots of marginal space. In cold places that margin gets

trimmed off. There's a sharper line between outside and inside,

and only projects that are officially sanctioned — by organizations,

or parents, or wives, or at least by oneself — get proper indoor

space. That raises the activation energy for new ideas. You can't

just tinker. You have to justify.Some of Silicon Valley's most famous companies began in garages:

Hewlett-Packard in 1938, Apple in 1976, Google in 1998. In Apple's

case the garage story is a bit of an urban legend. Woz says all

they did there was assemble some computers, and that he did all the

actual design of the Apple I and Apple II in his apartment or his

cube at HP.

[1]

This was apparently too marginal even for Apple's PR

people.By conventional standards, Jobs and Wozniak were marginal people

too. Obviously they were smart, but they can't have looked good

on paper. They were at the time a pair of college dropouts with

about three years of school between them, and hippies to boot.

Their previous business experience consisted of making "blue boxes"

to hack into the phone system, a business with the rare distinction

of being both illegal and unprofitable.OutsidersNow a startup operating out of a garage in Silicon Valley would

feel part of an exalted tradition, like the poet in his garret, or

the painter who can't afford to heat his studio and thus has to

wear a beret indoors. But in 1976 it didn't seem so cool. The

world hadn't yet realized that starting a computer company was in

the same category as being a writer or a painter. It hadn't been

for long. Only in the preceding couple years had the dramatic fall

in the cost of hardware allowed outsiders to compete.In 1976, everyone looked down on a company operating out of a garage,

including the founders. One of the first things Jobs did when they

got some money was to rent office space. He wanted Apple to seem

like a real company.They already had something few real companies ever have: a fabulously well

designed product. You'd think they'd have had more confidence.

But I've talked to a lot of startup founders, and it's always this

way. They've built something that's going to change the world, and

they're worried about some nit like not having proper business

cards.That's the paradox I want to explore: great new things often come

from the margins, and yet the people who discover them are looked

down on by everyone, including themselves.It's an old idea that new things come from the margins. I want to

examine its internal structure. Why do great ideas come from the

margins? What kind of ideas? And is there anything we can do to

encourage the process?InsidersOne reason so many good ideas come from the margin is simply that

there's so much of it. There have to be more outsiders than insiders,

if insider means anything. If the number of outsiders is huge it

will always seem as if a lot of ideas come from them, even if few

do per capita. But I think there's more going on than this. There

are real disadvantages to being an insider, and in some kinds of

work they can outweigh the advantages.Imagine, for example, what would happen if the government decided

to commission someone to write an official Great American Novel.

First there'd be a huge ideological squabble over who to choose.

Most of the best writers would be excluded for having offended one

side or the other. Of the remainder, the smart ones would refuse

such a job, leaving only a few with the wrong sort of ambition.

The committee would choose one at the height of his career — that

is, someone whose best work was behind him — and hand over the

project with copious free advice about how the book should show in

positive terms the strength and diversity of the American people,

etc, etc.The unfortunate writer would then sit down to work with a huge

weight of expectation on his shoulders. Not wanting to blow such

a public commission, he'd play it safe. This book had better command

respect, and the way to ensure that would be to make it a tragedy.

Audiences have to be enticed to laugh, but if you kill people they

feel obliged to take you seriously. As everyone knows, America

plus tragedy equals the Civil War, so that's what it would have to

be about. Better stick to the standard cartoon version that the

Civil War was about slavery; people would be confused otherwise;

plus you can show a lot of strength and diversity. When finally

completed twelve years later, the book would be a 900-page pastiche

of existing popular novels — roughly Gone with the Wind plus

Roots. But its bulk and celebrity would make it a bestseller

for a few months, until blown out of the water by a talk-show host's

autobiography. The book would be made into a movie and thereupon

forgotten, except by the more waspish sort of reviewers, among whom

it would be a byword for bogusness like Milli Vanilli or Battlefield

Earth.Maybe I got a little carried away with this example. And yet is

this not at each point the way such a project would play out? The

government knows better than to get into the novel business, but

in other fields where they have a natural monopoly, like nuclear

waste dumps, aircraft carriers, and regime change, you'd find plenty

of projects isomorphic to this one — and indeed, plenty that were

less successful.This little thought experiment suggests a few of the disadvantages

of insider projects: the selection of the wrong kind of people, the

excessive scope, the inability to take risks, the need to seem

serious, the weight of expectations, the power of vested interests,

the undiscerning audience, and perhaps most dangerous, the tendency

of such work to become a duty rather than a pleasure.TestsA world with outsiders and insiders implies some kind of test for

distinguishing between them. And the trouble with most tests for

selecting elites is that there are two ways to pass them: to be

good at what they try to measure, and to be good at hacking the

test itself.So the first question to ask about a field is how honest its tests

are, because this tells you what it means to be an outsider. This

tells you how much to trust your instincts when you disagree with

authorities, whether it's worth going through the usual channels

to become one yourself, and perhaps whether you want to work in

this field at all.Tests are least hackable when there are consistent standards for

quality, and the people running the test really care about its

integrity. Admissions to PhD programs in the hard sciences are

fairly honest, for example. The professors will get whoever they

admit as their own grad students, so they try hard to choose well,

and they have a fair amount of data to go on. Whereas undergraduate

admissions seem to be much more hackable.One way to tell whether a field has consistent standards is the

overlap between the leading practitioners and the people who teach

the subject in universities. At one end of the scale you have

fields like math and physics, where nearly all the teachers are

among the best practitioners. In the middle are medicine, law,

history, architecture, and computer science, where many are. At

the bottom are business, literature, and the visual arts, where

there's almost no overlap between the teachers and the leading

practitioners. It's this end that gives rise to phrases like "those

who can't do, teach."Incidentally, this scale might be helpful in deciding what to study

in college. When I was in college the rule seemed to be that you

should study whatever you were most interested in. But in retrospect

you're probably better off studying something moderately interesting

with someone who's good at it than something very interesting with

someone who isn't. You often hear people say that you shouldn't

major in business in college, but this is actually an instance of

a more general rule: don't learn things from teachers who are bad

at them.How much you should worry about being an outsider depends on the

quality of the insiders. If you're an amateur mathematician and

think you've solved a famous open problem, better go back and check.

When I was in grad school, a friend in the math department had the

job of replying to people who sent in proofs of Fermat's last theorem

and so on, and it did not seem as if he saw it as a valuable source

of tips — more like manning a mental health hotline. Whereas if

the stuff you're writing seems different from what English professors

are interested in, that's not necessarily a problem.Anti-TestsWhere the method of selecting the elite is thoroughly corrupt, most

of the good people will be outsiders. In art, for example, the

image of the poor, misunderstood genius is not just one possible

image of a great artist: it's the standard image. I'm not

saying it's correct, incidentally, but it is telling how well this

image has stuck. You couldn't make a rap like that stick to math

or medicine.

[2]If it's corrupt enough, a test becomes an anti-test, filtering out

the people it should select by making them to do things only the

wrong people would do. Popularity in high school

seems to be such a test. There are plenty of similar ones in the grownup

world. For example, rising up through the hierarchy of the average

big company demands an attention to politics few thoughtful people

could spare.

[3]

Someone like Bill Gates can grow a company under

him, but it's hard to imagine him having the patience to climb the

corporate ladder at General Electric — or Microsoft, actually.It's kind of strange when you think about it, because lord-of-the-flies

schools and bureaucratic companies are both the default. There are

probably a lot of people who go from one to the other and never

realize the whole world doesn't work this way.I think that's one reason big companies are so often blindsided by

startups.

People at big companies don't realize the extent to which

they live in an environment that is one large, ongoing test for the

wrong qualities.If you're an outsider, your best chances for beating insiders are

obviously in fields where corrupt tests select a lame elite. But

there's a catch: if the tests are corrupt, your victory won't be

recognized, at least in your lifetime. You may feel you don't need

that, but history suggests it's dangerous to work in fields with

corrupt tests. You may beat the insiders, and yet not do as good

work, on an absolute scale, as you would in a field that was more

honest.Standards in art, for example, were almost as corrupt in the first

half of the eighteenth century as they are today. This was the era

of those fluffy idealized portraits of countesses with their lapdogs.

Chardin

decided to skip all that and paint ordinary things as he

saw them. He's now considered the best of that period — and yet

not the equal of Leonardo or Bellini or Memling, who all had the

additional encouragement of honest standards.It can be worth participating in a corrupt contest, however, if

it's followed by another that isn't corrupt. For example, it would

be worth competing with a company that can spend more than you on

marketing, as long as you can survive to the next round, when

customers compare your actual products. Similarly, you shouldn't

be discouraged by the comparatively corrupt test of college admissions,

because it's followed immediately by less hackable tests.

[4]RiskEven in a field with honest tests, there are still advantages to

being an outsider. The most obvious is that outsiders have nothing

to lose. They can do risky things, and if they fail, so what? Few

will even notice.The eminent, on the other hand, are weighed down by their eminence.

Eminence is like a suit: it impresses the wrong people, and it

constrains the wearer.Outsiders should realize the advantage they have here. Being able

to take risks is hugely valuable. Everyone values safety too much,

both the obscure and the eminent. No one wants to look like a fool.

But it's very useful to be able to. If most of your ideas aren't

stupid, you're probably being too conservative. You're not bracketing

the problem.Lord Acton said we should judge talent at its best and character

at its worst. For example, if you write one great book and ten bad

ones, you still count as a great writer — or at least, a better

writer than someone who wrote eleven that were merely good. Whereas

if you're a quiet, law-abiding citizen most of the time but

occasionally cut someone up and bury them in your backyard, you're

a bad guy.Almost everyone makes the mistake of treating ideas as if they were

indications of character rather than talent — as if having a stupid

idea made you stupid. There's a huge weight of tradition advising

us to play it safe. "Even a fool is thought wise if he keeps

silent," says the Old Testament (Proverbs 17:28).Well, that may be fine advice for a bunch of goatherds in Bronze

Age Palestine. There conservatism would be the order of the day.

But times have changed. It might still be reasonable to stick with

the Old Testament in political questions, but materially the world

now has a lot more state. Tradition is less of a guide, not just

because things change faster, but because the space of possibilities

is so large. The more complicated the world gets, the more valuable

it is to be willing to look like a fool.DelegationAnd yet the more successful people become, the more heat they get

if they screw up — or even seem to screw up. In this respect, as

in many others, the eminent are prisoners of their own success. So

the best way to understand the advantages of being an outsider may

be to look at the disadvantages of being an insider.If you ask eminent people what's wrong with their lives, the first

thing they'll complain about is the lack of time. A friend of mine

at Google is fairly high up in the company and went to work for

them long before they went public. In other words, he's now rich

enough not to have to work. I asked him if he could still endure

the annoyances of having a job, now that he didn't have to. And

he said that there weren't really any annoyances, except — and he

got a wistful look when he said this — that he got so much

email.The eminent feel like everyone wants to take a bite out of them.

The problem is so widespread that people pretending to be eminent

do it by pretending to be overstretched.The lives of the eminent become scheduled, and that's not good for

thinking. One of the great advantages of being an outsider is long,

uninterrupted blocks of time. That's what I remember about grad

school: apparently endless supplies of time, which I spent worrying

about, but not writing, my dissertation. Obscurity is like health

food — unpleasant, perhaps, but good for you. Whereas fame tends

to be like the alcohol produced by fermentation. When it reaches

a certain concentration, it kills off the yeast that produced it.The eminent generally respond to the shortage of time by turning

into managers. They don't have time to work. They're surrounded

by junior people they're supposed to help or supervise. The obvious

solution is to have the junior people do the work. Some good

stuff happens this way, but there are problems it doesn't work so

well for: the kind where it helps to have everything in one head.For example, it recently emerged that the famous glass artist Dale

Chihuly hasn't actually blown glass for 27 years. He has assistants

do the work for him. But one of the most valuable sources of ideas

in the visual arts is the resistance of the medium. That's why oil

paintings look so different from watercolors. In principle you

could make any mark in any medium; in practice the medium steers

you. And if you're no longer doing the work yourself, you stop

learning from this.So if you want to beat those eminent enough to delegate, one way

to do it is to take advantage of direct contact with the medium.

In the arts it's obvious how: blow your own glass, edit your own

films, stage your own plays. And in the process pay close attention

to accidents and to new ideas you have on the fly. This technique

can be generalized to any sort of work: if you're an outsider, don't

be ruled by plans. Planning is often just a weakness forced on

those who delegate.Is there a general rule for finding problems best solved in one

head? Well, you can manufacture them by taking any project usually

done by multiple people and trying to do it all yourself. Wozniak's

work was a classic example: he did everything himself, hardware and

software, and the result was miraculous. He claims not one bug was

ever found in the Apple II, in either hardware or software.Another way to find good problems to solve in one head is to focus

on the grooves in the chocolate bar — the places where tasks are

divided when they're split between several people. If you want to

beat delegation, focus on a vertical slice: for example, be both

writer and editor, or both design buildings and construct them.One especially good groove to span is the one between tools and

things made with them. For example, programming languages and

applications are usually written by different people, and this is

responsible for a lot of the worst flaws in

programming languages.

I think every language should be designed simultaneously with a

large application written in it, the way C was with Unix.Techniques for competing with delegation translate well into business,

because delegation is endemic there. Instead of avoiding it as a

drawback of senility, many companies embrace it as a sign of maturity.

In big companies software is often designed, implemented, and sold

by three separate types of people. In startups one person may have

to do all three. And though this feels stressful, it's one reason

startups win. The needs of customers and the means of satisfying

them are all in one head.FocusThe very skill of insiders can be a weakness. Once someone is good

at something, they tend to spend all their time doing that. This

kind of focus is very valuable, actually. Much of the skill of

experts is the ability to ignore false trails. But focus has

drawbacks: you don't learn from other fields, and when a new approach

arrives, you may be the last to notice.For outsiders this translates into two ways to win. One is to work

on a variety of things. Since you can't derive as much benefit

(yet) from a narrow focus, you may as well cast a wider net and

derive what benefit you can from similarities between fields. Just

as you can compete with delegation by working on larger vertical

slices, you can compete with specialization by working on larger

horizontal slices — by both writing and illustrating your book, for

example.The second way to compete with focus is to see what focus overlooks.

In particular, new things. So if you're not good at anything yet,

consider working on something so new that no one else is either.

It won't have any prestige yet, if no one is good at it, but you'll

have it all to yourself.The potential of a new medium is usually underestimated, precisely

because no one has yet explored its possibilities. Before

Durer

tried making engravings, no one took them very seriously. Engraving

was for making little devotional images — basically fifteenth century

baseball cards of saints. Trying to make masterpieces in this

medium must have seemed to Durer's contemporaries the way that,

say, making masterpieces in

comics

might seem to the average person

today.In the computer world we get not new mediums but new platforms: the

minicomputer, the microprocessor, the web-based application. At

first they're always dismissed as being unsuitable for real work.

And yet someone always decides to try anyway, and it turns out you

can do more than anyone expected. So in the future when you hear

people say of a new platform: yeah, it's popular and cheap, but not

ready yet for real work, jump on it.As well as being more comfortable working on established lines,

insiders generally have a vested interest in perpetuating them.

The professor who made his reputation by discovering some new idea

is not likely to be the one to discover its replacement. This is

particularly true with companies, who have not only skill and pride

anchoring them to the status quo, but money as well. The Achilles

heel of successful companies is their inability to cannibalize

themselves. Many innovations consist of replacing something with

a cheaper alternative, and companies just don't want to see a path

whose immediate effect is to cut an existing source of revenue.So if you're an outsider you should actively seek out contrarian

projects. Instead of working on things the eminent have made

prestigious, work on things that could steal that prestige.The really juicy new approaches are not the ones insiders reject

as impossible, but those they ignore as undignified. For example,

after Wozniak designed the Apple II he offered it first to his

employer, HP. They passed. One of the reasons was that, to save

money, he'd designed the Apple II to use a TV as a monitor, and HP

felt they couldn't produce anything so declasse.LessWozniak used a TV as a monitor for the simple reason that he couldn't

afford a monitor. Outsiders are not merely free but compelled to

make things that are cheap and lightweight. And both are good bets

for growth: cheap things spread faster, and lightweight things

evolve faster.The eminent, on the other hand, are almost forced to work on a large

scale. Instead of garden sheds they must design huge art museums.

One reason they work on big things is that they can: like our

hypothetical novelist, they're flattered by such opportunities.

They also know that big projects will by their sheer bulk impress

the audience. A garden shed, however lovely, would be easy to

ignore; a few might even snicker at it. You can't snicker at a

giant museum, no matter how much you dislike it. And finally, there

are all those people the eminent have working for them; they have

to choose projects that can keep them all busy.Outsiders are free of all this. They can work on small things, and

there's something very pleasing about small things. Small things

can be perfect; big ones always have something wrong with them.

But there's a

magic

in small things that goes beyond such rational

explanations. All kids know it. Small things have more personality.Plus making them is more fun. You can do what you want; you don't

have to satisfy committees. And perhaps most important, small

things can be done fast. The prospect of seeing the finished project

hangs in the air like the smell of dinner cooking. If you work

fast, maybe you could have it done tonight.Working on small things is also a good way to learn. The most

important kinds of learning happen one project at a time. ("Next

time, I won't...") The faster you cycle through projects, the

faster you'll evolve.Plain materials have a charm like small scale. And in addition

there's the challenge of making do with less. Every designer's

ears perk up at the mention of that game, because it's a game you

can't lose. Like the JV playing the varsity, if you even tie, you

win. So paradoxically there are cases where fewer resources yield

better results, because the designers' pleasure at their own ingenuity

more than compensates.

[5]So if you're an outsider, take advantage of your ability to make

small and inexpensive things. Cultivate the pleasure and simplicity

of that kind of work; one day you'll miss it.ResponsibilityWhen you're old and eminent, what will you miss about being young

and obscure? What people seem to miss most is the lack of

responsibilities.Responsibility is an occupational disease of eminence. In principle

you could avoid it, just as in principle you could avoid getting

fat as you get old, but few do. I sometimes suspect that responsibility

is a trap and that the most virtuous route would be to shirk it,

but regardless it's certainly constraining.When you're an outsider you're constrained too, of course. You're

short of money, for example. But that constrains you in different

ways. How does responsibility constrain you? The worst thing is

that it allows you not to focus on real work. Just as the most

dangerous forms of

procrastination

are those that seem like work,

the danger of responsibilities is not just that they can consume a

whole day, but that they can do it without setting off the

kind of alarms you'd set off if you spent a whole day sitting on a

park bench.A lot of the pain of being an outsider is being aware of one's own

procrastination. But this is actually a good thing. You're at

least close enough to work that the smell of it makes you hungry.As an outsider, you're just one step away from getting things done.

A huge step, admittedly, and one that most people never seem to

make, but only one step. If you can summon up the energy to get

started, you can work on projects with an intensity (in both senses)

that few insiders can match. For insiders work turns into a duty,

laden with responsibilities and expectations. It's never so pure

as it was when they were young.Work like a dog being taken for a walk, instead of an ox being yoked

to the plow. That's what they miss.AudienceA lot of outsiders make the mistake of doing the opposite; they

admire the eminent so much that they copy even their flaws. Copying

is a good way to learn, but copy the right things. When I was in

college I imitated the pompous diction of famous professors. But

this wasn't what made them eminent — it was more a flaw their

eminence had allowed them to sink into. Imitating it was like

pretending to have gout in order to seem rich.Half the distinguishing qualities of the eminent are actually

disadvantages. Imitating these is not only a waste of time, but

will make you seem a fool to your models, who are often well aware

of it.What are the genuine advantages of being an insider? The greatest

is an audience. It often seems to outsiders that the great advantage

of insiders is money — that they have the resources to do what they

want. But so do people who inherit money, and that doesn't seem

to help, not as much as an audience. It's good for morale to know

people want to see what you're making; it draws work out of you.If I'm right that the defining advantage of insiders is an audience,

then we live in exciting times, because just in the last ten years

the Internet has made audiences a lot more liquid. Outsiders don't

have to content themselves anymore with a proxy audience of a few

smart friends. Now, thanks to the Internet, they can start to grow

themselves actual audiences. This is great news for the marginal,

who retain the advantages of outsiders while increasingly being

able to siphon off what had till recently been the prerogative of

the elite.Though the Web has been around for more than ten years, I think

we're just beginning to see its democratizing effects. Outsiders

are still learning how to steal audiences. But more importantly,

audiences are still learning how to be stolen — they're still just

beginning to realize how much

deeper bloggers can dig than

journalists, how much

more interesting

a democratic news site can be than a

front page controlled by editors, and how much

funnier

a bunch of kids

with webcams can be than mass-produced sitcoms.The big media companies shouldn't worry that people will post their

copyrighted material on YouTube. They should worry that people

will post their own stuff on YouTube, and audiences will watch that

instead.HackingIf I had to condense the power of the marginal into one sentence

it would be: just try hacking something together. That phrase draws

in most threads I've mentioned here. Hacking something together

means deciding what to do as you're doing it, not a subordinate

executing the vision of his boss. It implies the result won't

be pretty, because it will be made quickly out of inadequate

materials. It may work, but it won't be the sort of thing the

eminent would want to put their name on. Something hacked together

means something that barely solves the problem, or maybe doesn't

solve the problem at all, but another you discovered en route. But

that's ok, because the main value of that initial version is not the

thing itself, but what it leads to. Insiders who daren't walk

through the mud in their nice clothes will never make it to the

solid ground on the other side.The word "try" is an especially valuable component. I disagree

here with Yoda, who said there is no try. There is try. It implies

there's no punishment if you fail. You're driven by curiosity

instead of duty. That means the wind of procrastination will be

in your favor: instead of avoiding this work, this will be what you

do as a way of avoiding other work. And when you do it, you'll be

in a better mood. The more the work depends on imagination, the

more that matters, because most people have more ideas when they're

happy.If I could go back and redo my twenties, that would be one thing

I'd do more of: just try hacking things together. Like many people

that age, I spent a lot of time worrying about what I should do.

I also spent some time trying to build stuff. I should have spent

less time worrying and more time building. If you're not sure what

to do, make something.Raymond Chandler's advice to thriller writers was "When in doubt,

have a man come through a door with a gun in his hand." He followed

that advice. Judging from his books, he was often in doubt. But

though the result is occasionally cheesy, it's never boring. In

life, as in books, action is underrated.Fortunately the number of things you can just hack together keeps

increasing. People fifty years ago would be astonished that one

could just hack together a movie, for example. Now you can even

hack together distribution. Just make stuff and put it online.InappropriateIf you really want to score big, the place to focus is the margin

of the margin: the territories only recently captured from the

insiders. That's where you'll find the juiciest projects still

undone, either because they seemed too risky, or simply because

there were too few insiders to explore everything.This is why I spend most of my time writing

essays lately. The

writing of essays used to be limited to those who could get them

published. In principle you could have written them and just shown

them to your friends; in practice that didn't work.

[6]

An

essayist needs the resistance of an audience, just as an engraver

needs the resistance of the plate.Up till a few years ago, writing essays was the ultimate insider's

game. Domain experts were allowed to publish essays about their

field, but the pool allowed to write on general topics was about

eight people who went to the right parties in New York. Now the

reconquista has overrun this territory, and, not surprisingly, found

it sparsely cultivated. There are so many essays yet unwritten.

They tend to be the naughtier ones; the insiders have pretty much

exhausted the motherhood and apple pie topics.This leads to my final suggestion: a technique for determining when

you're on the right track. You're on the right track when people

complain that you're unqualified, or that you've done something

inappropriate. If people are complaining, that means you're doing

something rather than sitting around, which is the first step. And

if they're driven to such empty forms of complaint, that means

you've probably done something good.If you make something and people complain that it doesn't work,

that's a problem. But if the worst thing they can hit you with is

your own status as an outsider, that implies that in every other

respect you've succeeded. Pointing out that someone is unqualified

is as desperate as resorting to racial slurs. It's just a legitimate

sounding way of saying: we don't like your type around here.But the best thing of all is when people call what you're doing

inappropriate. I've been hearing this word all my life and I only

recently realized that it is, in fact, the sound of the homing

beacon. "Inappropriate" is the null criticism. It's merely the

adjective form of "I don't like it."So that, I think, should be the highest goal for the marginal. Be

inappropriate. When you hear people saying that, you're golden.

And they, incidentally, are busted.Notes[1]

The facts about Apple's early history are from an interview

with Steve

Wozniak in Jessica Livingston's

Founders at Work.[2]

As usual the popular image is several decades behind reality.

Now the misunderstood artist is not a chain-smoking drunk who pours

his soul into big, messy canvases that philistines see and say

"that's not art" because it isn't a picture of anything. The

philistines have now been trained that anything hung on a wall

is art. Now the misunderstood artist is a coffee-drinking vegan

cartoonist whose work they see and say "that's not art" because it

looks like stuff they've seen in the Sunday paper.[3]

In fact this would do fairly well as a definition of politics:

what determines rank in the absence of objective tests.[4]

In high school you're led to believe your whole future depends

on where you go to college, but it turns out only to buy you a couple

years. By your mid-twenties the people worth impressing

already judge you more by what

you've done than where you went to school.[5]

Managers are presumably wondering, how can I make this miracle

happen? How can I make the people working for me do more with less?

Unfortunately the constraint probably has to be self-imposed. If

you're expected to do more with less, then you're being

starved, not eating virtuously.[6]

Without the prospect of publication, the closest most people

come to writing essays is to write in a journal. I find I never

get as deeply into subjects as I do in proper essays. As the name

implies, you don't go back and rewrite journal entries over

and over for two weeks.Thanks to Sam Altman, Trevor Blackwell, Paul Buchheit, Sarah

Harlin, Jessica Livingston, Jackie McDonough, Robert Morris, Olin

Shivers, and Chris Small for reading drafts of this, and to Chris

Small and Chad Fowler for inviting me to speak.

Japanese TranslationChinese Translation

Why Startups Condense in America

May 2006(This essay is derived from a keynote at Xtech.)Startups happen in clusters. There are a lot of them in Silicon

Valley and Boston, and few in Chicago or Miami. A country that

wants startups will probably also have to reproduce whatever makes

these clusters form.I've claimed that the recipe is a

great university near a town smart

people like. If you set up those conditions within the US, startups

will form as inevitably as water droplets condense on a cold piece

of metal. But when I consider what it would take to reproduce

Silicon Valley in another country, it's clear the US is a particularly

humid environment. Startups condense more easily here.It is by no means a lost cause to try to create a silicon valley

in another country. There's room not merely to equal Silicon Valley,

but to surpass it. But if you want to do that, you have to

understand the advantages startups get from being in America.1. The US Allows Immigration.For example, I doubt it would be possible to reproduce Silicon

Valley in Japan, because one of Silicon Valley's most distinctive

features is immigration. Half the people there speak with accents.

And the Japanese don't like immigration. When they think about how

to make a Japanese silicon valley, I suspect they unconsciously

frame it as how to make one consisting only of Japanese people.

This way of framing the question probably guarantees failure.A silicon valley has to be a mecca for the smart and the ambitious,

and you can't have a mecca if you don't let people into it.Of course, it's not saying much that America is more open to

immigration than Japan. Immigration policy is one area where a

competitor could do better.2. The US Is a Rich Country.I could see India one day producing a rival to Silicon Valley.

Obviously they have the right people: you can tell that by the

number of Indians in the current Silicon Valley. The problem with

India itself is that it's still so poor.In poor countries, things we take for granted are missing. A friend

of mine visiting India sprained her ankle falling down the steps

in a railway station. When she turned to see what had happened,

she found the steps were all different heights. In industrialized

countries we walk down steps our whole lives and never think about

this, because there's an infrastructure that prevents such a staircase

from being built.The US has never been so poor as some countries are now. There

have never been swarms of beggars in the streets of American cities.

So we have no data about what it takes to get from the swarms-of-beggars

stage to the silicon-valley stage. Could you have both at once,

or does there have to be some baseline prosperity before you get a

silicon valley?I suspect there is some speed limit to the evolution

of an economy. Economies are made out of people, and attitudes can

only change a certain amount per generation.

[1]3. The US Is Not (Yet) a Police State.Another country I could see wanting to have a silicon valley is

China. But I doubt they could do it yet either. China still seems

to be a police state, and although present rulers seem enlightened

compared to the last, even enlightened despotism can probably only

get you part way toward being a great economic power.It can get you factories for building things designed elsewhere.

Can it get you the designers, though? Can imagination flourish

where people can't criticize the government? Imagination means

having odd ideas, and it's hard to have odd ideas about technology

without also having odd ideas about politics. And in any case,

many technical ideas do have political implications. So if you

squash dissent, the back pressure will propagate into technical

fields.

[2]Singapore would face a similar problem. Singapore seems very aware

of the importance of encouraging startups. But while energetic

government intervention may be able to make a port run efficiently,

it can't coax startups into existence. A state that bans chewing

gum has a long way to go before it could create a San Francisco.Do you need a San Francisco? Might there not be an alternate route

to innovation that goes through obedience and cooperation instead

of individualism? Possibly, but I'd bet not. Most imaginative

people seem to share a certain prickly independence,

whenever and wherever they lived. You see it in Diogenes telling

Alexander to get out of his light and two thousand years later in

Feynman breaking into safes at Los Alamos.

[3]

Imaginative people

don't want to follow or lead. They're most productive when everyone

gets to do what they want.Ironically, of all rich countries the US has lost the most civil

liberties recently. But I'm not too worried yet. I'm hoping once

the present administration is out, the natural openness of American

culture will reassert itself.4. American Universities Are Better.You need a great university to seed a silicon valley, and so far

there are few outside the US. I asked a handful of American computer

science professors which universities in Europe were most admired,

and they all basically said "Cambridge" followed by a long pause

while they tried to think of others. There don't seem to be many

universities elsewhere that compare with the best in America, at

least in technology.In some countries this is the result of a deliberate policy. The

German and Dutch governments, perhaps from fear of elitism, try to

ensure that all universities are roughly equal in quality. The

downside is that none are especially good. The best professors

are spread out, instead of being concentrated as they are in the

US. This probably makes them less productive, because they don't

have good colleagues to inspire them. It also means no one university

will be good enough to act as a mecca, attracting talent from abroad

and causing startups to form around it.The case of Germany is a strange one. The Germans invented the

modern university, and up till the 1930s theirs were the best in

the world. Now they have none that stand out. As I was mulling

this over, I found myself thinking: "I can understand why German

universities declined in the 1930s, after they excluded Jews. But

surely they should have bounced back by now." Then I realized:

maybe not. There are few Jews left in Germany and most Jews I know

would not want to move there. And if you took any great American

university and removed the Jews, you'd have some pretty big gaps.

So maybe it would be a lost cause trying to create a silicon valley

in Germany, because you couldn't establish the level of university

you'd need as a seed.

[4]It's natural for US universities to compete with one another because

so many are private. To reproduce the quality of American universities

you probably also have to reproduce this. If universities are

controlled by the central government, log-rolling will pull them

all toward the mean: the new Institute of X will end up at the

university in the district of a powerful politician, instead of

where it should be.5. You Can Fire People in America.I think one of the biggest obstacles to creating startups in Europe

is the attitude toward employment. The famously rigid labor laws

hurt every company, but startups especially, because startups have

the least time to spare for bureaucratic hassles.The difficulty of firing people is a particular problem for startups

because they have no redundancy. Every person has to do their

job well.But the problem is more than just that some startup might have a

problem firing someone they needed to. Across industries and

countries, there's a strong inverse correlation between performance

and job security. Actors and directors are fired at the end of

each film, so they have to deliver every time. Junior professors

are fired by default after a few years unless the university chooses

to grant them tenure. Professional athletes know they'll be pulled

if they play badly for just a couple games. At the other end of

the scale (at least in the US) are auto workers, New York City

schoolteachers, and civil servants, who are all nearly impossible

to fire. The trend is so clear that you'd have to be willfully

blind not to see it.Performance isn't everything, you say? Well, are auto workers,

schoolteachers, and civil servants happier than actors,

professors, and professional athletes?European public opinion will apparently tolerate people being fired

in industries where they really care about performance. Unfortunately

the only industry they care enough about so far is soccer. But

that is at least a precedent.6. In America Work Is Less Identified with Employment.The problem in more traditional places like Europe and Japan goes

deeper than the employment laws. More dangerous is the attitude

they reflect: that an employee is a kind of servant, whom the

employer has a duty to protect. It used to be that way in America

too. In 1970 you were still supposed to get a job with a big

company, for whom ideally you'd work your whole career. In return

the company would take care of you: they'd try not to fire you,

cover your medical expenses, and support you in old age.Gradually employment has been shedding such paternalistic overtones

and becoming simply an economic exchange. But the importance of

the new model is not just that it makes it easier for startups to

grow. More important, I think, is that it it makes it easier for

people to start startups.Even in the US most kids graduating from college still think they're

supposed to get jobs, as if you couldn't be productive without being

someone's employee. But the less you identify work with employment,

the easier it becomes to start a startup. When you see your career

as a series of different types of work, instead of a lifetime's

service to a single employer, there's less risk in starting your

own company, because you're only replacing one segment instead of

discarding the whole thing.The old ideas are so powerful that even the most successful startup

founders have had to struggle against them. A year after the

founding of Apple, Steve Wozniak still hadn't quit HP. He still

planned to work there for life. And when Jobs found someone to

give Apple serious venture funding, on the condition that Woz quit,

he initially refused, arguing that he'd designed both the Apple I

and the Apple II while working at HP, and there was no reason he

couldn't continue.7. America Is Not Too Fussy.If there are any laws regulating businesses, you can assume larval

startups will break most of them, because they don't know what the

laws are and don't have time to find out.For example, many startups in America begin in places where it's

not really legal to run a business. Hewlett-Packard, Apple, and

Google were all run out of garages. Many more startups, including

ours, were initially run out of apartments. If the laws against

such things were actually enforced, most startups wouldn't happen.That could be a problem in fussier countries. If Hewlett and Packard

tried running an electronics company out of their garage in

Switzerland, the old lady next door would report them to the municipal

authorities.But the worst problem in other countries is probably the effort

required just to start a company. A friend of mine started a company

in Germany in the early 90s, and was shocked to discover, among

many other regulations, that you needed $20,000 in capital to

incorporate. That's one reason I'm not typing this on an Apfel

laptop. Jobs and Wozniak couldn't have come up with that kind of

money in a company financed by selling a VW bus and an HP calculator.

We couldn't have started Viaweb either.

[5]Here's a tip for governments that want to encourage startups: read

the stories of existing startups, and then try to simulate what

would have happened in your country. When you hit something that

would have killed Apple, prune it off.Startups are marginal.

They're started by the poor and the

timid; they begin in marginal space and spare time; they're started

by people who are supposed to be doing something else; and though

businesses, their founders often know nothing about business. Young

startups are fragile. A society that trims its margins sharply

will kill them all.8. America Has a Large Domestic Market.What sustains a startup in the beginning is the prospect of getting

their initial product out. The successful ones therefore make the

first version as simple as possible. In the US they usually begin

by making something just for the local market.This works in America, because the local market is 300 million

people. It wouldn't work so well in Sweden. In a small country,

a startup has a harder task: they have to sell internationally from

the start.The EU was designed partly to simulate a single, large domestic

market. The problem is that the inhabitants still speak many

different languages. So a software startup in Sweden is still at

a disadvantage relative to one in the US, because they have to deal

with internationalization from the beginning. It's significant

that the most famous recent startup in Europe, Skype, worked on a

problem that was intrinsically international.However, for better or worse it looks as if Europe will in a few

decades speak a single language. When I was a student in Italy in

1990, few Italians spoke English. Now all educated people seem to

be expected to-- and Europeans do not like to seem uneducated. This

is presumably a taboo subject, but if present trends continue,

French and German will eventually go the way of Irish and Luxembourgish:

they'll be spoken in homes and by eccentric nationalists.9. America Has Venture Funding.Startups are easier to start in America because funding is easier

to get. There are now a few VC firms outside the US, but startup

funding doesn't only come from VC firms. A more important source,

because it's more personal and comes earlier in the process, is

money from individual angel investors. Google might never have got

to the point where they could raise millions from VC funds if they

hadn't first raised a hundred thousand from Andy Bechtolsheim. And

he could help them because he was one of the founders of Sun. This

pattern is repeated constantly in startup hubs. It's this pattern

that makes them startup hubs.The good news is, all you have to do to get the process rolling is

get those first few startups successfully launched. If they stick

around after they get rich, startup founders will almost automatically

fund and encourage new startups.The bad news is that the cycle is slow. It probably takes five

years, on average, before a startup founder can make angel investments.

And while governments might be able to set up local VC funds

by supplying the money themselves and recruiting people from existing

firms to run them, only organic growth can produce angel investors.Incidentally, America's private universities are one reason there's

so much venture capital. A lot of the money in VC funds comes from

their endowments. So another advantage of private universities is

that a good chunk of the country's wealth is managed by enlightened

investors.10. America Has Dynamic Typing for Careers.Compared to other industrialized countries the US is disorganized

about routing people into careers. For example, in America people

often don't decide to go to medical school till they've finished

college. In Europe they generally decide in high school.The European approach reflects the old idea that each person has a

single, definite occupation-- which is not far from the idea that

each person has a natural "station" in life. If this were true,

the most efficient plan would be to discover each person's station

as early as possible, so they could receive the training appropriate

to it.In the US things are more haphazard. But that turns out to be an

advantage as an economy gets more liquid, just as dynamic typing

turns out to work better than static for ill-defined problems. This

is particularly true with startups. "Startup founder" is not the

sort of career a high school student would choose. If you ask at

that age, people will choose conservatively. They'll choose

well-understood occupations like engineer, or doctor, or lawyer.Startups are the kind of thing people don't plan, so you're more

likely to get them in a society where it's ok to make career decisions

on the fly.For example, in theory the purpose of a PhD program is to train you

to do research. But fortunately in the US this is another rule

that isn't very strictly enforced. In the US most people in CS PhD

programs are there simply because they wanted to learn more. They

haven't decided what they'll do afterward. So American grad schools

spawn a lot of startups, because students don't feel they're failing

if they don't go into research.Those worried about America's "competitiveness" often suggest

spending more on public schools. But perhaps America's lousy public

schools have a hidden advantage. Because they're so bad, the kids

adopt an attitude of waiting for college. I did; I knew I was

learning so little that I wasn't even learning what the choices

were, let alone which to choose. This is demoralizing, but it does

at least make you keep an open mind.Certainly if I had to choose between bad high schools and good

universities, like the US, and good high schools and bad universities,

like most other industrialized countries, I'd take the US system.

Better to make everyone feel like a late bloomer than a failed child

prodigy.AttitudesThere's one item conspicuously missing from this list: American

attitudes. Americans are said to be more entrepreneurial, and less

afraid of risk. But America has no monopoly on this. Indians and

Chinese seem plenty entrepreneurial, perhaps more than Americans.Some say Europeans are less energetic, but I don't believe it. I

think the problem with Europe is not that they lack balls, but that

they lack examples.Even in the US, the most successful startup founders are often

technical people who are quite timid, initially, about the idea of

starting their own company. Few are the sort of backslapping

extroverts one thinks of as typically American. They can usually

only summon up the activation energy to start a startup when they

meet people who've done it and realize they could too.I think what holds back European hackers is simply that they don't

meet so many people who've done it. You see that variation even

within the US. Stanford students are more entrepreneurial than

Yale students, but not because of some difference in their characters;

the Yale students just have fewer examples.I admit there seem to be different attitudes toward ambition in

Europe and the US. In the US it's ok to be overtly ambitious, and

in most of Europe it's not. But this can't be an intrinsically

European quality; previous generations of Europeans were as ambitious

as Americans. What happened? My hypothesis is that ambition was

discredited by the terrible things ambitious people did in the first

half of the twentieth century. Now swagger is out. (Even now the

image of a very ambitious German presses a button or two, doesn't

it?)It would be surprising if European attitudes weren't affected by

the disasters of the twentieth century. It takes a while to be

optimistic after events like that. But ambition is human nature.

Gradually it will re-emerge.

[6]How To Do BetterI don't mean to suggest by this list that America is the perfect

place for startups. It's the best place so far, but the sample

size is small, and "so far" is not very long. On historical time

scales, what we have now is just a

prototype.So let's look at Silicon Valley the way you'd look at a product

made by a competitor. What weaknesses could you exploit? How could

you make something users would like better? The users in this case

are those critical few thousand people you'd like to move to your

silicon valley.To start with, Silicon Valley is too far from San Francisco. Palo

Alto, the original ground zero, is about thirty miles away, and the

present center more like forty. So people who come to work in

Silicon Valley face an unpleasant choice: either live in the boring

sprawl of the valley proper, or live in San Francisco and endure

an hour commute each way.The best thing would be if the silicon valley were not merely closer

to the interesting city, but interesting itself. And there is a

lot of room for improvement here. Palo Alto is not so bad, but

everything built since is the worst sort of strip development. You

can measure how demoralizing it is by the number of people who will

sacrifice two hours a day commuting rather than live there.Another area in which you could easily surpass Silicon Valley is

public transportation. There is a train running the length of it,

and by American standards it's not bad. Which is to say that to

Japanese or Europeans it would seem like something out of the third

world.The kind of people you want to attract to your silicon valley like

to get around by train, bicycle, and on foot. So if you want to

beat America, design a town that puts cars last. It will be a while

before any American city can bring itself to do that.Capital GainsThere are also a couple things you could do to beat America at the

national level. One would be to have lower capital gains taxes.

It doesn't seem critical to have the lowest income taxes,

because to take advantage of those, people have to move.

[7]

But

if capital gains rates vary, you move assets, not yourself, so

changes are reflected at market speeds. The lower the rate, the

cheaper it is to buy stock in growing companies as opposed to real

estate, or bonds, or stocks bought for the dividends they pay.So if you want to encourage startups you should have a low rate on

capital gains. Politicians are caught between a rock and a hard

place here, however: make the capital gains rate low and be accused

of creating "tax breaks for the rich," or make it high and starve

growing companies of investment capital. As Galbraith said,

politics is a matter of choosing between the unpalatable and the

disastrous. A lot of governments experimented with the disastrous

in the twentieth century; now the trend seems to be toward the

merely unpalatable.Oddly enough, the leaders now are European countries like Belgium,

which has a capital gains tax rate of zero.ImmigrationThe other place you could beat the US would be with smarter immigration

policy. There are huge gains to be made here. Silicon valleys are

made of people, remember.Like a company whose software runs on Windows, those in the current

Silicon Valley are all too aware of the shortcomings of the INS,

but there's little they can do about it. They're hostages of the

platform.America's immigration system has never been well run, and since

2001 there has been an additional admixture of paranoia. What

fraction of the smart people who want to come to America can even

get in? I doubt even half. Which means if you made a competing

technology hub that let in all smart people, you'd immediately get

more than half the world's top talent, for free.US immigration policy is particularly ill-suited to startups, because

it reflects a model of work from the 1970s. It assumes good technical

people have college degrees, and that work means working for a big

company.If you don't have a college degree you can't get an H1B visa, the

type usually issued to programmers. But a test that excludes Steve

Jobs, Bill Gates, and Michael Dell can't be a good one. Plus you

can't get a visa for working on your own company, only for working

as an employee of someone else's. And if you want to apply for

citizenship you daren't work for a startup at all, because if your

sponsor goes out of business, you have to start over.American immigration policy keeps out most smart people, and channels

the rest into unproductive jobs. It would be easy to do better.

Imagine if, instead, you treated immigration like recruiting-- if

you made a conscious effort to seek out the smartest people and get

them to come to your country.A country that got immigration right would have a huge advantage.

At this point you could become a mecca for smart people simply by

having an immigration system that let them in.A Good VectorIf you look at the kinds of things you have to do to create an

environment where startups condense, none are great sacrifices.

Great universities? Livable towns? Civil liberties? Flexible

employment laws? Immigration policies that let in smart people?

Tax laws that encourage growth? It's not as if you have to risk

destroying your country to get a silicon valley; these are all good

things in their own right.And then of course there's the question, can you afford not to? I

can imagine a future in which the default choice of ambitious young

people is to start their own company

rather than work for someone else's. I'm not sure that will happen,

but it's where the trend points now. And if that is the future,

places that don't have startups will be a whole step behind,

like those that missed the Industrial Revolution.Notes[1]

On the verge of the Industrial Revolution, England was already

the richest country in the world. As far as such things can be

compared, per capita income in England in 1750 was higher than

India's in 1960.Deane, Phyllis, The First Industrial Revolution, Cambridge

University Press, 1965.[2]

This has already happened once in China, during the Ming

Dynasty, when the country turned its back on industrialization at

the command of the court. One of Europe's advantages was that it

had no government powerful enough to do that.[3]

Of course, Feynman and Diogenes were from adjacent traditions,

but Confucius, though more polite, was no more willing to be told

what to think.[4]

For similar reasons it might be a lost cause to try to establish

a silicon valley in Israel. Instead of no Jews moving there, only

Jews would move there, and I don't think you could build a silicon

valley out of just Jews any more than you could out of just Japanese.(This is not a remark about the qualities of these groups, just their

sizes. Japanese are only about 2% of the world population, and

Jews about .2%.)[5]

According to the World Bank, the initial capital requirement

for German companies is 47.6% of the per capita income. Doh.World Bank, Doing Business in 2006, http://doingbusiness.org[6]

For most of the twentieth century, Europeans looked back on

the summer of 1914 as if they'd been living in a dream world. It

seems more accurate (or at least, as accurate) to call the years

after 1914 a nightmare than to call those before a dream. A lot

of the optimism Europeans consider distinctly American is simply

what they too were feeling in 1914.[7]

The point where things start to go wrong seems to be about

50%. Above that people get serious about tax avoidance. The reason

is that the payoff for avoiding tax grows hyperexponentially (x/1-x

for 0 < x < 1). If your income tax rate is 10%, moving to Monaco

would only give you 11% more income, which wouldn't even cover the

extra cost. If it's 90%, you'd get ten times as much income. And

at 98%, as it was briefly in Britain in the 70s, moving to Monaco

would give you fifty times as much income. It seems quite likely

that European governments of the 70s never drew this curve.Thanks to Trevor Blackwell, Matthias Felleisen, Jessica

Livingston, Robert Morris, Neil Rimer, Hugues Steinier, Brad

Templeton, Fred Wilson, and Stephen Wolfram for reading

drafts of this, and to Ed Dumbill for inviting me to speak.French TranslationRussian TranslationJapanese TranslationArabic Translation

How to Be Silicon Valley

May 2006(This essay is derived from a keynote at Xtech.)Could you reproduce Silicon Valley elsewhere, or is there something

unique about it?It wouldn't be surprising if it were hard to reproduce in other

countries, because you couldn't reproduce it in most of the US

either. What does it take to make a silicon valley even here?What it takes is the right people. If you could get the right ten

thousand people to move from Silicon Valley to Buffalo, Buffalo

would become Silicon Valley.

[1]That's a striking departure from the past. Up till a couple decades

ago, geography was destiny for cities. All great cities were located

on waterways, because cities made money by trade, and water was the

only economical way to ship.Now you could make a great city anywhere, if you could get the right

people to move there. So the question of how to make a silicon

valley becomes: who are the right people, and how do you get them

to move?Two TypesI think you only need two kinds of people to create a technology

hub: rich people and nerds. They're the limiting reagents in the

reaction that produces startups, because they're the only ones

present when startups get started. Everyone else will move.Observation bears this out: within the US, towns have become startup

hubs if and only if they have both rich people and nerds. Few

startups happen in Miami, for example, because although it's full

of rich people, it has few nerds. It's not the kind of place nerds

like.Whereas Pittsburgh has the opposite problem: plenty of nerds, but

no rich people. The top US Computer Science departments are said

to be MIT, Stanford, Berkeley, and Carnegie-Mellon. MIT yielded

Route 128. Stanford and Berkeley yielded Silicon Valley. But

Carnegie-Mellon? The record skips at that point. Lower down the

list, the University of Washington yielded a high-tech community

in Seattle, and the University of Texas at Austin yielded one in

Austin. But what happened in Pittsburgh? And in Ithaca, home of

Cornell, which is also high on the list?I grew up in Pittsburgh and went to college at Cornell, so I can

answer for both. The weather is terrible, particularly in winter,

and there's no interesting old city to make up for it, as there is

in Boston. Rich people don't want to live in Pittsburgh or Ithaca.

So while there are plenty of hackers who could start startups,

there's no one to invest in them.Not BureaucratsDo you really need the rich people? Wouldn't it work to have the

government invest in the nerds? No, it would not. Startup investors

are a distinct type of rich people. They tend to have a lot of

experience themselves in the technology business. This (a) helps

them pick the right startups, and (b) means they can supply advice

and connections as well as money. And the fact that they have a

personal stake in the outcome makes them really pay attention.Bureaucrats by their nature are the exact opposite sort of people

from startup investors. The idea of them making startup investments

is comic. It would be like mathematicians running Vogue-- or

perhaps more accurately, Vogue editors running a math journal.

[2]Though indeed, most things bureaucrats do, they do badly. We just

don't notice usually, because they only have to compete against

other bureaucrats. But as startup investors they'd have to compete

against pros with a great deal more experience and motivation.Even corporations that have in-house VC groups generally forbid

them to make their own investment decisions. Most are only allowed

to invest in deals where some reputable private VC firm is willing

to act as lead investor.Not BuildingsIf you go to see Silicon Valley, what you'll see are buildings.

But it's the people that make it Silicon Valley, not the buildings.

I read occasionally about attempts to set up "technology

parks" in other places, as if the active ingredient of Silicon

Valley were the office space. An article about Sophia Antipolis

bragged that companies there included Cisco, Compaq, IBM, NCR, and

Nortel. Don't the French realize these aren't startups?Building office buildings for technology companies won't get you a

silicon valley, because the key stage in the life of a startup

happens before they want that kind of space. The key stage is when

they're three guys operating out of an apartment. Wherever the

startup is when it gets funded, it will stay. The defining quality

of Silicon Valley is not that Intel or Apple or Google have offices

there, but that they were started there.So if you want to reproduce Silicon Valley, what you need to reproduce

is those two or three founders sitting around a kitchen table

deciding to start a company. And to reproduce that you need those

people.UniversitiesThe exciting thing is, all you need are the people. If you could

attract a critical mass of nerds and investors to live somewhere,

you could reproduce Silicon Valley. And both groups are highly

mobile. They'll go where life is good. So what makes a place good

to them?What nerds like is other nerds. Smart people will go wherever other

smart people are. And in particular, to great universities. In

theory there could be other ways to attract them, but so far

universities seem to be indispensable. Within the US, there are

no technology hubs without first-rate universities-- or at least,

first-rate computer science departments.So if you want to make a silicon valley, you not only need a

university, but one of the top handful in the world. It has to be

good enough to act as a magnet, drawing the best people from thousands

of miles away. And that means it has to stand up to existing magnets

like MIT and Stanford.This sounds hard. Actually it might be easy. My professor friends,

when they're deciding where they'd like to work, consider one thing

above all: the quality of the other faculty. What attracts professors

is good colleagues. So if you managed to recruit, en masse, a

significant number of the best young researchers, you could create

a first-rate university from nothing overnight. And you could do

that for surprisingly little. If you paid 200 people hiring bonuses

of $3 million apiece, you could put together a faculty that would

bear comparison with any in the world. And from that point the

chain reaction would be self-sustaining. So whatever it costs to

establish a mediocre university, for an additional half billion or

so you could have a great one.

[3]PersonalityHowever, merely creating a new university would not be enough to

start a silicon valley. The university is just the seed. It has

to be planted in the right soil, or it won't germinate. Plant it

in the wrong place, and you just create Carnegie-Mellon.To spawn startups, your university has to be in a town that has

attractions other than the university. It has to be a place where

investors want to live, and students want to stay after they graduate.The two like much the same things, because most startup investors

are nerds themselves. So what do nerds look for in a town? Their

tastes aren't completely different from other people's, because a

lot of the towns they like most in the US are also big tourist

destinations: San Francisco, Boston, Seattle. But their tastes

can't be quite mainstream either, because they dislike other big

tourist destinations, like New York, Los Angeles, and Las Vegas.There has been a lot written lately about the "creative class." The

thesis seems to be that as wealth derives increasingly from ideas,

cities will prosper only if they attract those who have them. That

is certainly true; in fact it was the basis of Amsterdam's prosperity

400 years ago.A lot of nerd tastes they share with the creative class in general.

For example, they like well-preserved old neighborhoods instead of

cookie-cutter suburbs, and locally-owned shops and restaurants

instead of national chains. Like the rest of the creative class,

they want to live somewhere with personality.What exactly is personality? I think it's the feeling that each

building is the work of a distinct group of people. A town with

personality is one that doesn't feel mass-produced. So if you want

to make a startup hub-- or any town to attract the "creative class"--

you probably have to ban large development projects.

When a large tract has been developed by a single organization, you

can always tell.

[4]Most towns with personality are old, but they don't have to be.

Old towns have two advantages: they're denser, because they were

laid out before cars, and they're more varied, because they were

built one building at a time. You could have both now. Just have

building codes that ensure density, and ban large scale developments.A corollary is that you have to keep out the biggest developer of

all: the government. A government that asks "How can we build a

silicon valley?" has probably ensured failure by the way they framed

the question. You don't build a silicon valley; you let one grow.NerdsIf you want to attract nerds, you need more than a town with

personality. You need a town with the right personality. Nerds

are a distinct subset of the creative class, with different tastes

from the rest. You can see this most clearly in New York, which

attracts a lot of creative people, but few nerds.

[5]What nerds like is the kind of town where people walk around smiling.

This excludes LA, where no one walks at all, and also New York,

where people walk, but not smiling. When I was in grad school in

Boston, a friend came to visit from New York. On the subway back

from the airport she asked "Why is everyone smiling?" I looked and

they weren't smiling. They just looked like they were compared to

the facial expressions she was used to.If you've lived in New York, you know where these facial expressions

come from. It's the kind of place where your mind may be excited,

but your body knows it's having a bad time. People don't so much

enjoy living there as endure it for the sake of the excitement.

And if you like certain kinds of excitement, New York is incomparable.

It's a hub of glamour, a magnet for all the shorter half-life

isotopes of style and fame.Nerds don't care about glamour, so to them the appeal of New York

is a mystery. People who like New York will pay a fortune for a

small, dark, noisy apartment in order to live in a town where the

cool people are really cool. A nerd looks at that deal and sees

only: pay a fortune for a small, dark, noisy apartment.Nerds will pay a premium to live in a town where the smart people

are really smart, but you don't have to pay as much for that. It's

supply and demand: glamour is popular, so you have to pay a lot for

it.Most nerds like quieter pleasures. They like cafes instead of

clubs; used bookshops instead of fashionable clothing shops; hiking

instead of dancing; sunlight instead of tall buildings. A nerd's

idea of paradise is Berkeley or Boulder.YouthIt's the young nerds who start startups, so it's those specifically

the city has to appeal to. The startup hubs in the US are all

young-feeling towns. This doesn't mean they have to be new.

Cambridge has the oldest town plan in America, but it feels young

because it's full of students.What you can't have, if you want to create a silicon valley, is a

large, existing population of stodgy people. It would be a waste

of time to try to reverse the fortunes of a declining industrial town

like Detroit or Philadelphia by trying to encourage startups. Those

places have too much momentum in the wrong direction. You're better

off starting with a blank slate in the form of a small town. Or

better still, if there's a town young people already flock to, that

one.The Bay Area was a magnet for the young and optimistic for decades

before it was associated with technology. It was a place people

went in search of something new. And so it became synonymous with

California nuttiness. There's still a lot of that there. If you

wanted to start a new fad-- a new way to focus one's "energy," for

example, or a new category of things not to eat-- the Bay Area would

be the place to do it. But a place that tolerates oddness in the

search for the new is exactly what you want in a startup hub, because

economically that's what startups are. Most good startup ideas

seem a little crazy; if they were obviously good ideas, someone

would have done them already.(How many people are going to want computers in their houses?

What, another search engine?)That's the connection between technology and liberalism. Without

exception the high-tech cities in the US are also the most liberal.

But it's not because liberals are smarter that this is so. It's

because liberal cities tolerate odd ideas, and smart people by

definition have odd ideas.Conversely, a town that gets praised for being "solid" or representing

"traditional values" may be a fine place to live, but it's never

going to succeed as a startup hub. The 2004 presidential election,

though a disaster in other respects, conveniently supplied us with

a county-by-county

map of such places.

[6]To attract the young, a town must have an intact center. In most

American cities the center has been abandoned, and the growth, if

any, is in the suburbs. Most American cities have been turned

inside out. But none of the startup hubs has: not San Francisco,

or Boston, or Seattle. They all have intact centers.

[7]

My guess is that no city with a dead center could be turned into a

startup hub. Young people don't want to live in the suburbs.Within the US, the two cities I think could most easily be turned

into new silicon valleys are Boulder and Portland. Both have the

kind of effervescent feel that attracts the young. They're each

only a great university short of becoming a silicon valley, if they

wanted to.TimeA great university near an attractive town. Is that all it takes?

That was all it took to make the original Silicon Valley. Silicon

Valley traces its origins to William Shockley, one of the inventors

of the transistor. He did the research that won him the Nobel Prize

at Bell Labs, but when he started his own company in 1956 he moved

to Palo Alto to do it. At the time that was an odd thing to do.

Why did he? Because he had grown up there and remembered how nice

it was. Now Palo Alto is suburbia, but then it was a charming

college town-- a charming college town with perfect weather and San

Francisco only an hour away.The companies that rule Silicon Valley now are all descended in

various ways from Shockley Semiconductor. Shockley was a difficult

man, and in 1957 his top people-- "the traitorous eight"-- left to

start a new company, Fairchild Semiconductor. Among them were

Gordon Moore and Robert Noyce, who went on to found Intel, and

Eugene Kleiner, who founded the VC firm Kleiner Perkins. Forty-two

years later, Kleiner Perkins funded Google, and the partner responsible

for the deal was John Doerr, who came to Silicon Valley in 1974 to

work for Intel.So although a lot of the newest companies in Silicon Valley don't

make anything out of silicon, there always seem to be multiple links

back to Shockley. There's a lesson here: startups beget startups.

People who work for startups start their own. People who get rich

from startups fund new ones. I suspect this kind of organic growth

is the only way to produce a startup hub, because it's the only way

to grow the expertise you need.That has two important implications. The first is that you need

time to grow a silicon valley. The university you could create in

a couple years, but the startup community around it has to grow

organically. The cycle time is limited by the time it takes a

company to succeed, which probably averages about five years.The other implication of the organic growth hypothesis is that you

can't be somewhat of a startup hub. You either have a self-sustaining

chain reaction, or not. Observation confirms this too: cities

either have a startup scene, or they don't. There is no middle

ground. Chicago has the third largest metropolitan area in America.

As source of startups it's negligible compared to Seattle, number 15.The good news is that the initial seed can be quite small. Shockley

Semiconductor, though itself not very successful, was big enough.

It brought a critical mass of experts in an important new technology

together in a place they liked enough to stay.CompetingOf course, a would-be silicon valley faces an obstacle the original

one didn't: it has to compete with Silicon Valley. Can that be

done? Probably.One of Silicon Valley's biggest advantages is its venture capital

firms. This was not a factor in Shockley's day, because VC funds

didn't exist. In fact, Shockley Semiconductor and Fairchild

Semiconductor were not startups at all in our sense. They were

subsidiaries-- of Beckman Instruments and Fairchild Camera and

Instrument respectively. Those companies were apparently willing

to establish subsidiaries wherever the experts wanted to live.Venture investors, however, prefer to fund startups within an hour's

drive. For one, they're more likely to notice startups nearby.

But when they do notice startups in other towns they prefer them

to move. They don't want to have to travel to attend board meetings,

and in any case the odds of succeeding are higher in a startup hub.The centralizing effect of venture firms is a double one: they cause

startups to form around them, and those draw in more startups through

acquisitions. And although the first may be weakening because it's

now so cheap to start some startups, the second seems as strong as ever.

Three of the most admired

"Web 2.0" companies were started outside the usual startup hubs,

but two of them have already been reeled in through acquisitions.Such centralizing forces make it harder for new silicon valleys to

get started. But by no means impossible. Ultimately power rests

with the founders. A startup with the best people will beat one

with funding from famous VCs, and a startup that was sufficiently

successful would never have to move. So a town that

could exert enough pull over the right people could resist and

perhaps even surpass Silicon Valley.For all its power, Silicon Valley has a great weakness: the paradise

Shockley found in 1956 is now one giant parking lot. San Francisco

and Berkeley are great, but they're forty miles away. Silicon

Valley proper is soul-crushing suburban sprawl. It

has fabulous weather, which makes it significantly better than the

soul-crushing sprawl of most other American cities. But a competitor

that managed to avoid sprawl would have real leverage. All a city

needs is to be the kind of place the next traitorous eight look at

and say "I want to stay here," and that would be enough to get the

chain reaction started.Notes[1]

It's interesting to consider how low this number could be

made. I suspect five hundred would be enough, even if they could

bring no assets with them. Probably just thirty, if I could pick them,

would be enough to turn Buffalo into a significant startup hub.[2]

Bureaucrats manage to allocate research funding moderately

well, but only because (like an in-house VC fund) they outsource

most of the work of selection. A professor at a famous university

who is highly regarded by his peers will get funding, pretty much

regardless of the proposal. That wouldn't work for startups, whose

founders aren't sponsored by organizations, and are often unknowns.[3]

You'd have to do it all at once, or at least a whole department

at a time, because people would be more likely to come if they

knew their friends were. And you should probably start from scratch,

rather than trying to upgrade an existing university, or much energy

would be lost in friction.[4]

Hypothesis: Any plan in which multiple independent buildings

are gutted or demolished to be "redeveloped" as a single project

is a net loss of personality for the city, with the exception of

the conversion of buildings not previously public, like warehouses.[5]

A few startups get started in New York, but less

than a tenth as many per capita as in Boston, and mostly

in less nerdy fields like finance and media.[6]

Some blue counties are false positives (reflecting the

remaining power of Democractic party machines), but there are no

false negatives. You can safely write off all the red counties.[7]

Some "urban renewal" experts took a shot at destroying Boston's

in the 1960s, leaving the area around city hall a bleak wasteland,

but most neighborhoods successfully resisted them.Thanks to Chris Anderson, Trevor Blackwell, Marc Hedlund,

Jessica Livingston, Robert Morris, Greg Mcadoo, Fred Wilson,

and Stephen Wolfram for

reading drafts of this, and to Ed Dumbill for inviting me to speak.(The second part of this talk became Why Startups

Condense in America.)VC Deals by RegionStartup Jobs by RegionThey Would Be GodsInterview: Richard HodgsonSanta Clara Valley, 1971Scattered AbroadRussian TranslationSpanish TranslationJapanese TranslationPortuguese TranslationArabic Translation

If you liked this, you may also like

Hackers & Painters.

The Hardest Lessons for Startups to Learn

April 2006(This essay is derived from a talk at the 2006

Startup School.)The startups we've funded so far are pretty quick, but they seem

quicker to learn some lessons than others. I think it's because

some things about startups are kind of counterintuitive.We've now

invested

in enough companies that I've learned a trick

for determining which points are the counterintuitive ones:

they're the ones I have to keep repeating.So I'm going to number these points, and maybe with future startups

I'll be able to pull off a form of Huffman coding. I'll make them

all read this, and then instead of nagging them in detail, I'll

just be able to say: number four!

1. Release Early.The thing I probably repeat most is this recipe for a startup: get

a version 1 out fast, then improve it based on users' reactions.By "release early" I don't mean you should release something full

of bugs, but that you should release something minimal. Users hate

bugs, but they don't seem to mind a minimal version 1, if there's

more coming soon.There are several reasons it pays to get version 1 done fast. One

is that this is simply the right way to write software, whether for

a startup or not. I've been repeating that since 1993, and I haven't seen much since to

contradict it. I've seen a lot of startups die because they were

too slow to release stuff, and none because they were too quick.

[1]One of the things that will surprise you if you build something

popular is that you won't know your users. Reddit now has almost half a million

unique visitors a month. Who are all those people? They have no

idea. No web startup does. And since you don't know your users,

it's dangerous to guess what they'll like. Better to release

something and let them tell you.Wufoo took this to heart and released

their form-builder before the underlying database. You can't even

drive the thing yet, but 83,000 people came to sit in the driver's

seat and hold the steering wheel. And Wufoo got valuable feedback

from it: Linux users complained they used too much Flash, so they

rewrote their software not to. If they'd waited to release everything

at once, they wouldn't have discovered this problem till it was

more deeply wired in.Even if you had no users, it would still be important to release

quickly, because for a startup the initial release acts as a shakedown

cruise. If anything major is broken-- if the idea's no good,

for example, or the founders hate one another-- the stress of getting

that first version out will expose it. And if you have such problems

you want to find them early.Perhaps the most important reason to release early, though, is that

it makes you work harder. When you're working on something that

isn't released, problems are intriguing. In something that's out

there, problems are alarming. There is a lot more urgency once you

release. And I think that's precisely why people put it off. They

know they'll have to work a lot harder once they do.

[2]

2. Keep Pumping Out Features.Of course, "release early" has a second component, without which

it would be bad advice. If you're going to start with something

that doesn't do much, you better improve it fast.What I find myself repeating is "pump out features." And this rule

isn't just for the initial stages. This is something all startups

should do for as long as they want to be considered startups.I don't mean, of course, that you should make your application ever

more complex. By "feature" I mean one unit of hacking-- one quantum

of making users' lives better.As with exercise, improvements beget improvements. If you run every

day, you'll probably feel like running tomorrow. But if you skip

running for a couple weeks, it will be an effort to drag yourself

out. So it is with hacking: the more ideas you implement, the more

ideas you'll have. You should make your system better at least in

some small way every day or two.This is not just a good way to get development done; it is also a

form of marketing. Users love a site that's constantly improving.

In fact, users expect a site to improve. Imagine if you visited a

site that seemed very good, and then returned two months later and

not one thing had changed. Wouldn't it start to seem lame?

[3]They'll like you even better when you improve in response to their

comments, because customers are used to companies ignoring them.

If you're the rare exception-- a company that actually listens--

you'll generate fanatical loyalty. You won't need to advertise,

because your users will do it for you.This seems obvious too, so why do I have to keep repeating it? I

think the problem here is that people get used to how things are.

Once a product gets past the stage where it has glaring flaws, you

start to get used to it, and gradually whatever features it happens

to have become its identity. For example, I doubt many people at

Yahoo (or Google for that matter) realized how much better web mail

could be till Paul Buchheit showed them.I think the solution is to assume that anything you've made is far

short of what it could be. Force yourself, as a sort of intellectual

exercise, to keep thinking of improvements. Ok, sure, what you

have is perfect. But if you had to change something, what would

it be?If your product seems finished, there are two possible explanations:

(a) it is finished, or (b) you lack imagination. Experience suggests

(b) is a thousand times more likely.

3. Make Users Happy.Improving constantly is an instance of a more general rule: make

users happy. One thing all startups have in common is that they

can't force anyone to do anything. They can't force anyone to use

their software, and they can't force anyone to do deals with them.

A startup has to sing for its supper. That's why the successful

ones make great things. They have to, or die.When you're running a startup you feel like a little bit of debris

blown about by powerful winds. The most powerful wind is users.

They can either catch you and loft you up into the sky, as they did

with Google, or leave you flat on the pavement, as they do with

most startups. Users are a fickle wind, but more powerful than any

other. If they take you up, no competitor can keep you down.As a little piece of debris, the rational thing for you to do is

not to lie flat, but to curl yourself into a shape the wind will

catch.I like the wind metaphor because it reminds you how impersonal the

stream of traffic is. The vast majority of people who visit your

site will be casual visitors. It's them you have to design your

site for. The people who really care will find what they want by

themselves.The median visitor will arrive with their finger poised on the Back

button. Think about your own experience: most links you

follow lead to something lame. Anyone who has used the web for

more than a couple weeks has been trained to click on Back after

following a link. So your site has to say "Wait! Don't click on

Back. This site isn't lame. Look at this, for example."There are two things you have to do to make people pause. The most

important is to explain, as concisely as possible, what the hell

your site is about. How often have you visited a site that seemed

to assume you already knew what they did? For example, the corporate

site that says the

company makes

enterprise content management solutions for business that enable

organizations to unify people, content and processes to minimize

business risk, accelerate time-to-value and sustain lower total

cost of ownership.

An established company may get away with such an opaque description,

but no startup can. A startup

should be able to explain in one or two sentences exactly what it

does.

[4]

And not just to users. You need this for everyone:

investors, acquirers, partners, reporters, potential employees, and

even current employees. You probably shouldn't even start a company

to do something that can't be described compellingly in one or two

sentences.The other thing I repeat is to give people everything you've got,

right away. If you have something impressive, try to put it on the

front page, because that's the only one most visitors will see.

Though indeed there's a paradox here: the more you push the good

stuff toward the front, the more likely visitors are to explore

further.

[5]In the best case these two suggestions get combined: you tell

visitors what your site is about by showing them. One of the

standard pieces of advice in fiction writing is "show, don't tell."

Don't say that a character's angry; have him grind his teeth, or

break his pencil in half. Nothing will explain what your site does

so well as using it.The industry term here is "conversion." The job of your site is

to convert casual visitors into users-- whatever your definition

of a user is. You can measure this in your growth rate. Either

your site is catching on, or it isn't, and you must know which. If

you have decent growth, you'll win in the end, no matter how obscure

you are now. And if you don't, you need to fix something.

4. Fear the Right Things.Another thing I find myself saying a lot is "don't worry." Actually,

it's more often "don't worry about this; worry about that instead."

Startups are right to be paranoid, but they sometimes fear the wrong

things.Most visible disasters are not so alarming as they seem. Disasters

are normal in a startup: a founder quits, you discover a patent

that covers what you're doing, your servers keep crashing, you run

into an insoluble technical problem, you have to change your name,

a deal falls through-- these are all par for the course. They won't

kill you unless you let them.Nor will most competitors. A lot of startups worry "what if Google

builds something like us?" Actually big companies are not the ones

you have to worry about-- not even Google. The people at Google

are smart, but no smarter than you; they're not as motivated, because

Google is not going to go out of business if this one product fails;

and even at Google they have a lot of bureaucracy to slow them down.What you should fear, as a startup, is not the established players,

but other startups you don't know exist yet. They're way more

dangerous than Google because, like you, they're cornered animals.Looking just at existing competitors can give you a false sense of

security. You should compete against what someone else could be

doing, not just what you can see people doing. A corollary is that

you shouldn't relax just because you have no visible competitors

yet. No matter what your idea, there's someone else out there

working on the same thing.That's the downside of it being easier to start a startup: more people

are doing it. But I disagree with Caterina Fake when she says that

makes this a bad time to start a startup. More people are starting

startups, but not as many more as could. Most college graduates

still think they have to get a job. The average person can't ignore

something that's been beaten into their head since they were three

just because serving web pages recently got a lot cheaper.And in any case, competitors are not the biggest threat. Way more

startups hose themselves than get crushed by competitors. There

are a lot of ways to do it, but the three main ones are internal

disputes, inertia, and ignoring users. Each is, by itself, enough

to kill you. But if I had to pick the worst, it would be ignoring

users. If you want a recipe for a startup that's going to die,

here it is: a couple of founders who have some great idea they know

everyone is going to love, and that's what they're going to build,

no matter what.Almost everyone's initial plan is broken. If companies stuck to

their initial plans, Microsoft would be selling programming languages,

and Apple would be selling printed circuit boards. In both cases

their customers told them what their business should be-- and they

were smart enough to listen.As Richard Feynman said, the imagination of nature is greater than

the imagination of man. You'll find more interesting things by

looking at the world than you could ever produce just by thinking.

This principle is very powerful. It's why the best abstract painting

still falls short of Leonardo, for example. And it applies to

startups too. No idea for a product could ever be so clever as the

ones you can discover by smashing a beam of prototypes into a beam

of users.

5. Commitment Is a Self-Fulfilling Prophecy.I now have enough experience with startups to be able to say what

the most important quality is in a startup founder, and it's not

what you might think. The most important quality in a startup

founder is determination. Not intelligence-- determination.This is a little depressing. I'd like to believe Viaweb succeeded

because we were smart, not merely determined. A lot of people in

the startup world want to believe that. Not just founders, but

investors too. They like the idea of inhabiting a world ruled by

intelligence. And you can tell they really believe this, because

it affects their investment decisions.Time after time VCs invest in startups founded by eminent professors.

This may work in biotech, where a lot of startups simply commercialize

existing research, but in software you want to invest in students,

not professors. Microsoft, Yahoo, and Google were all founded by

people who dropped out of school to do it. What students lack in

experience they more than make up in dedication.Of course, if you want to get rich, it's not enough merely to be

determined. You have to be smart too, right? I'd like to think

so, but I've had an experience that convinced me otherwise: I spent

several years living in New York.You can lose quite a lot in the brains department and it won't kill

you. But lose even a little bit in the commitment department, and

that will kill you very rapidly.Running a startup is like walking on your hands: it's possible, but

it requires extraordinary effort. If an ordinary employee were

asked to do the things a startup founder has to, he'd be very

indignant. Imagine if you were hired at some big company, and in

addition to writing software ten times faster than you'd ever had

to before, they expected you to answer support calls, administer

the servers, design the web site, cold-call customers, find the

company office space, and go out and get everyone lunch.And to do all this not in the calm, womb-like atmosphere of a big

company, but against a backdrop of constant disasters. That's the

part that really demands determination. In a startup, there's

always some disaster happening. So if you're the least bit inclined

to find an excuse to quit, there's always one right there.But if you lack commitment, chances are it will have been hurting

you long before you actually quit. Everyone who deals with startups

knows how important commitment is, so if they sense you're ambivalent,

they won't give you much attention. If you lack commitment, you'll

just find that for some mysterious reason good things happen to

your competitors but not to you. If you lack commitment, it will

seem to you that you're unlucky.Whereas if you're determined to stick around, people will pay

attention to you, because odds are they'll have to deal with you

later. You're a local, not just a tourist, so everyone has to come

to terms with you.At Y Combinator we sometimes mistakenly fund teams who have the

attitude that they're going to give this startup thing a shot for

three months, and if something great happens, they'll stick with

it-- "something great" meaning either that someone wants to buy

them or invest millions of dollars in them. But if this is your

attitude, "something great" is very unlikely to happen to you,

because both acquirers and investors judge you by your level of

commitment.If an acquirer thinks you're going to stick around no matter what,

they'll be more likely to buy you, because if they don't and you

stick around, you'll probably grow, your price will go up, and

they'll be left wishing they'd bought you earlier. Ditto for

investors. What really motivates investors, even big VCs, is not

the hope of good returns, but the fear of missing out.

[6]

So if

you make it clear you're going to succeed no matter what, and the only

reason you need them is to make it happen a little faster, you're

much more likely to get money.You can't fake this. The only way to convince everyone that you're

ready to fight to the death is actually to be ready to.You have to be the right kind of determined, though. I carefully

chose the word determined rather than stubborn, because stubbornness

is a disastrous quality in a startup. You have to be determined,

but flexible, like a running back. A successful running back doesn't

just put his head down and try to run through people. He improvises:

if someone appears in front of him, he runs around them; if someone

tries to grab him, he spins out of their grip; he'll even run in

the wrong direction briefly if that will help. The one thing he'll

never do is stand still.

[7]

6. There Is Always Room.I was talking recently to a startup founder about whether it might

be good to add a social component to their software. He said he

didn't think so, because the whole social thing was tapped out.

Really? So in a hundred years the only social networking sites

will be the Facebook, MySpace, Flickr, and Del.icio.us? Not likely.There is always room for new stuff. At every point in history,

even the darkest bits of the dark ages, people were discovering

things that made everyone say "why didn't anyone think of that

before?" We know this continued to be true up till 2004, when the

Facebook was founded-- though strictly speaking someone else did

think of that.The reason we don't see the opportunities all around us is that we

adjust to however things are, and assume that's how things have to

be. For example, it would seem crazy to most people to try to make

a better search engine than Google. Surely that field, at least,

is tapped out. Really? In a hundred years-- or even twenty-- are

people still going to search for information using something like

the current Google? Even Google probably doesn't think that.In particular, I don't think there's any limit to the number of

startups. Sometimes you hear people saying "All these guys starting

startups now are going to be disappointed. How many little startups

are Google and Yahoo going to buy, after all?" That sounds cleverly

skeptical, but I can prove it's mistaken. No one proposes that

there's some limit to the number of people who can be employed in

an economy consisting of big, slow-moving companies with a couple

thousand people each. Why should there be any limit to the number

who could be employed by small, fast-moving companies with ten each?

It seems to me the only limit would be the number of people who

want to work that hard.The limit on the number of startups is not the number that can get

acquired by Google and Yahoo-- though it seems even that should

be unlimited, if the startups were actually worth buying-- but the

amount of wealth that can be created. And I don't think there's

any limit on that, except cosmological ones.So for all practical purposes, there is no limit to the number of

startups. Startups make wealth, which means they make things people

want, and if there's a limit on the number of things people want,

we are nowhere near it. I still don't even have a flying car.

7. Don't Get Your Hopes Up.This is another one I've been repeating since long before Y Combinator.

It was practically the corporate motto at Viaweb.Startup founders are naturally optimistic. They wouldn't do it

otherwise. But you should treat your optimism the way you'd treat

the core of a nuclear reactor: as a source of power that's also

very dangerous. You have to build a shield around it, or it will

fry you.The shielding of a reactor is not uniform; the reactor would be

useless if it were. It's pierced in a few places to let pipes in.

An optimism shield has to be pierced too. I think the place to

draw the line is between what you expect of yourself, and what you

expect of other people. It's ok to be optimistic about what you

can do, but assume the worst about machines and other people.This is particularly necessary in a startup, because you tend to

be pushing the limits of whatever you're doing. So things don't

happen in the smooth, predictable way they do in the rest of the

world. Things change suddenly, and usually for the worse.Shielding your optimism is nowhere more important than with deals.

If your startup is doing a deal, just assume it's not going to

happen. The VCs who say they're going to invest in you aren't.

The company that says they're going to buy you isn't. The big

customer who wants to use your system in their whole company won't.

Then if things work out you can be pleasantly surprised.The reason I warn startups not to get their hopes up is not to save

them from being disappointed when things fall through. It's

for a more practical reason: to prevent them from leaning their

company against something that's going to fall over, taking them

with it.For example, if someone says they want to invest in you, there's a

natural tendency to stop looking for other investors. That's why

people proposing deals seem so positive: they want you to

stop looking. And you want to stop too, because doing deals is a

pain. Raising money, in particular, is a huge time sink. So you

have to consciously force yourself to keep looking.Even if you ultimately do the first deal, it will be to your advantage

to have kept looking, because you'll get better terms. Deals are

dynamic; unless you're negotiating with someone unusually honest,

there's not a single point where you shake hands and the deal's

done. There are usually a lot of subsidiary questions to be cleared

up after the handshake, and if the other side senses weakness-- if

they sense you need this deal-- they will be very tempted to screw

you in the details.VCs and corp dev guys are professional negotiators. They're trained

to take advantage of weakness.

[8]

So while they're often nice

guys, they just can't help it. And as pros they do this more than

you. So don't even try to bluff them. The only way a startup can

have any leverage in a deal is genuinely not to need it. And if

you don't believe in a deal, you'll be less likely to depend on it.So I want to plant a hypnotic suggestion in your heads: when you

hear someone say the words "we want to invest in you" or "we want

to acquire you," I want the following phrase to appear automatically

in your head: don't get your hopes up. Just continue running

your company as if this deal didn't exist. Nothing is more likely

to make it close.The way to succeed in a startup is to focus on the goal of getting

lots of users, and keep walking swiftly toward it while investors

and acquirers scurry alongside trying to wave money in your face.

Speed, not MoneyThe way I've described it, starting a startup sounds pretty stressful.

It is. When I talk to the founders of the companies we've funded,

they all say the same thing: I knew it would be hard, but I didn't

realize it would be this hard.So why do it? It would be worth enduring a lot of pain and stress

to do something grand or heroic, but just to make money? Is making

money really that important?No, not really. It seems ridiculous to me when people take business

too seriously. I regard making money as a boring errand to be got

out of the way as soon as possible. There is nothing grand or

heroic about starting a startup per se.So why do I spend so much time thinking about startups? I'll tell

you why. Economically, a startup is best seen not as a way to get

rich, but as a way to work faster. You have to make a living, and

a startup is a way to get that done quickly, instead of letting it

drag on through your whole life.

[9]We take it for granted most of the time, but human life is fairly

miraculous. It is also palpably short. You're given this marvellous

thing, and then poof, it's taken away. You can see why people

invent gods to explain it. But even to people who don't believe

in gods, life commands respect. There are times in most of our

lives when the days go by in a blur, and almost everyone has a

sense, when this happens, of wasting something precious. As Ben

Franklin said, if you love life, don't waste time, because time is

what life is made of.So no, there's nothing particularly grand about making money. That's

not what makes startups worth the trouble. What's important about

startups is the speed. By compressing the dull but necessary task

of making a living into the smallest possible time, you show respect

for life, and there is something grand about that.Notes[1]

Startups can die from releasing something full of bugs, and not

fixing them fast enough, but I don't know of any that died from

releasing something stable but minimal very early, then promptly

improving it.[2]

I know this is why I haven't released Arc. The moment I do,

I'll have people nagging me for features.[3]

A web site is different from a book or movie or desktop application

in this respect. Users judge a site not as a single snapshot, but

as an animation with multiple frames. Of the two, I'd say the rate of

improvement is more important to users than where you currently

are.[4]

It should not always tell this to users, however. For example,

MySpace is basically a replacement mall for mallrats. But it was

wiser for them, initially, to pretend that the site was about bands.[5]

Similarly, don't make users register to try your site. Maybe

what you have is so valuable that visitors should gladly register

to get at it. But they've been trained to expect the opposite.

Most of the things they've tried on the web have sucked-- and

probably especially those that made them register.[6]

VCs have rational reasons for behaving this way. They don't

make their money (if they make money) off their median investments.

In a typical fund, half the companies fail, most of the rest generate

mediocre returns, and one or two "make the fund" by succeeding

spectacularly. So if they miss just a few of the most promising

opportunities, it could hose the whole fund.[7]

The attitude of a running back doesn't translate to soccer.

Though it looks great when a forward dribbles past multiple defenders,

a player who persists in trying such things will do worse in the

long term than one who passes.[8]

The reason Y Combinator never negotiates valuations

is that we're not professional negotiators, and don't want to turn

into them.[9]

There are two ways to do

work you love: (a) to make money, then work

on what you love, or (b) to get a job where you get paid to work on

stuff you love. In practice the first phases of both

consist mostly of unedifying schleps, and in (b) the second phase is less

secure.Thanks to Sam Altman, Trevor Blackwell, Beau Hartshorne, Jessica

Livingston, and Robert Morris for reading drafts of this.Romanian TranslationRussian TranslationFrench TranslationJapanese Translation

See Randomness

April 2006, rev August 2009Plato quotes Socrates as saying "the unexamined life is not worth

living." Part of what he meant was that the proper role of humans is to

think, just as the proper role of anteaters is to poke their noses

into anthills.A lot of ancient philosophy had the quality — and I

don't mean this in an insulting way — of the kind of conversations

freshmen have late at night in common rooms:

What is our purpose? Well, we humans are

as conspicuously different from other animals as the anteater.

In our case the distinguishing feature is the ability to reason.

So obviously that is what we should be doing, and a human who

doesn't is doing a bad job of being human — is no better than an

animal.

Now we'd give a different answer. At least, someone Socrates's age

would. We'd ask why we even suppose we have a "purpose" in life.

We may be better adapted for some things than others; we

may be happier doing things we're adapted for; but why assume

purpose?The history of ideas

is a history of gradually discarding the assumption that it's all

about us. No, it turns out, the earth is not the center of the

universe — not even the center of the solar system. No, it turns

out, humans are not created by God in his own image; they're just

one species among many, descended not merely from apes, but from

microorganisms. Even the concept of "me" turns out to be fuzzy

around the edges if you examine it closely.The idea that we're the center of things is difficult to discard.

So difficult that there's probably room to discard more. Richard

Dawkins made another step in that direction only in the last several

decades, with the idea of the

selfish gene.

No, it turns

out, we're not even the protagonists: we're just the latest model

vehicle our genes have constructed to travel around in. And having

kids is our genes heading for the lifeboats. Reading

that book snapped my brain out of its previous way of thinking the

way Darwin's must have when it first appeared.(Few people can experience now what Darwin's contemporaries did

when The Origin of Species was first published, because everyone

now is raised either to take evolution for granted, or to regard

it as a heresy. No one encounters the idea of natural selection for

the first time as an adult.)So if you want to discover things that have been overlooked till

now, one really good place to look is in our blind spot: in our

natural, naive belief that it's all about us. And expect to encounter

ferocious opposition if you do.Conversely, if you have to choose between two theories, prefer the

one that doesn't center on you.This principle isn't only for big ideas. It works in everyday life,

too. For example, suppose you're saving a piece of cake in the fridge, and you

come home one day to find your housemate has eaten

it. Two possible theories:

a) Your housemate did it deliberately to upset you. He knew

you were saving that piece of cake.b) Your housemate was hungry.

I say pick b. No one knows who said "never attribute to malice what

can be explained by incompetence," but it is a powerful idea.

Its more general version is our answer to the Greeks:

Don't see purpose where there isn't.

Or better still, the positive version:

See randomness.Korean Translation

Are Software Patents Evil?

March 2006(This essay is derived from a talk at Google.)A few weeks ago I found to my surprise that I'd been granted four patents.

This was all the more surprising

because I'd only applied for three. The patents aren't mine, of

course. They were assigned to Viaweb, and became Yahoo's when they

bought us. But the news set me thinking about the question of

software patents generally.Patents are a hard problem. I've had to advise most of the startups

we've funded about them, and despite years of experience I'm still

not always sure I'm giving the right advice.One thing I do feel pretty certain of is that if you're against

software patents, you're against patents in general. Gradually our

machines consist more and more of software. Things that used to

be done with levers and cams and gears are now done with loops and

trees and closures. There's nothing special about physical embodiments

of control systems that should make them patentable, and the software

equivalent not.Unfortunately, patent law is inconsistent on this point. Patent

law in most countries says that algorithms aren't patentable. This

rule is left over from a time when "algorithm" meant something like

the Sieve of Eratosthenes. In 1800, people could not see as readily

as we can that a great many patents on mechanical objects were

really patents on the algorithms they embodied.Patent lawyers still have to pretend that's what they're doing when

they patent algorithms. You must not use the word "algorithm" in

the title of a patent application, just as you must not use the

word "essays" in the title of a book. If you want to patent an

algorithm, you have to frame it as a computer system executing that algorithm.

Then it's mechanical; phew. The default euphemism for algorithm

is "system and method." Try a patent search for that phrase and

see how many results you get.Since software patents are no different from hardware patents,

people who say "software patents are evil" are saying simply "patents

are evil." So why do so many people complain about software patents

specifically?I think the problem is more with the patent office than the concept

of software patents. Whenever software meets government, bad things

happen, because software changes fast and government changes slow.

The patent office has been overwhelmed by both the volume and the

novelty of applications for software patents, and as a result they've

made a lot of mistakes.The most common is to grant patents that shouldn't be granted. To

be patentable, an invention has to be more than new. It also has

to be non-obvious. And this, especially, is where the USPTO has

been dropping the ball. Slashdot has an icon that expresses the

problem vividly: a knife and fork with the words "patent pending"

superimposed.The scary thing is, this is the only icon they have for

patent stories. Slashdot readers now take it for granted that a

story about a patent will be about a bogus patent.

That's how bad the problem has become.The problem with Amazon's notorious one-click patent, for example,

is not that it's a software patent, but that it's obvious. Any

online store that kept people's shipping addresses would have

implemented this. The reason Amazon did it first was not that they

were especially smart, but because they were one of the earliest

sites with enough clout to force customers to log in before they

could buy something.

[1]We, as hackers, know the USPTO is letting people patent the knives

and forks of our world. The problem is, the USPTO are not hackers.

They're probably good at judging new inventions for casting steel

or grinding lenses, but they don't understand software yet.At this point an optimist would be tempted to add "but they will

eventually." Unfortunately that might not be true. The problem

with software patents is an instance of a more general one: the

patent office takes a while to understand new technology. If so,

this problem will only get worse, because the rate of technological

change seems to be increasing. In thirty years, the patent office

may understand the sort of things we now patent as software, but

there will be other new types of inventions they understand even

less.Applying for a patent is a negotiation. You generally apply for a

broader patent than you think you'll be granted, and the examiners

reply by throwing out some of your claims and granting others. So

I don't really blame Amazon for applying for the one-click patent.

The big mistake was the patent office's, for not insisting on

something narrower, with real technical content. By granting such

an over-broad patent, the USPTO in effect slept with Amazon on the

first date. Was Amazon supposed to say no?Where Amazon went over to the dark side was not in applying for the

patent, but in enforcing it. A lot of companies (Microsoft, for

example) have been granted large numbers of preposterously over-broad

patents, but they keep them mainly for defensive purposes. Like

nuclear weapons, the main role of big companies' patent portfolios

is to threaten anyone who attacks them with a counter-suit. Amazon's

suit against Barnes & Noble was thus the equivalent of a nuclear

first strike.That suit probably hurt Amazon more than it helped them. Barnes &

Noble was a lame site; Amazon would have crushed them anyway. To

attack a rival they could have ignored, Amazon put a lasting black

mark on their own reputation. Even now I think if you asked hackers

to free-associate about Amazon, the one-click patent would turn up

in the first ten topics.Google clearly doesn't feel that merely holding patents is evil.

They've applied for a lot of them. Are they hypocrites? Are patents

evil?There are really two variants of that question, and people answering

it often aren't clear in their own minds which they're answering.

There's a narrow variant: is it bad, given the current legal system,

to apply for patents? and also a broader one: is it bad that the

current legal system allows patents?These are separate questions. For example, in preindustrial societies

like medieval Europe, when someone attacked you, you didn't call

the police. There were no police. When attacked, you were supposed

to fight back, and there were conventions about how to do it. Was

this wrong? That's two questions: was it wrong to take justice

into your own hands, and was it wrong that you had to? We tend to

say yes to the second, but no to the first. If no one else will

defend you, you have to defend yourself.

[2]The situation with patents is similar. Business is a kind of

ritualized warfare. Indeed, it evolved from actual warfare: most

early traders switched on the fly from merchants to pirates depending

on how strong you seemed. In business there are certain rules

describing how companies may and may not compete with one another,

and someone deciding that they're going to play by their own rules

is missing the point. Saying "I'm not going to apply for patents

just because everyone else does" is not like saying "I'm not going

to lie just because everyone else does." It's more like saying

"I'm not going to use TCP/IP just because everyone else does." Oh

yes you are.A closer comparison might be someone seeing a hockey game for the

first time, realizing with shock that the players were deliberately

bumping into one another, and deciding that one would on no account

be so rude when playing hockey oneself.Hockey allows checking. It's part of the game. If your team refuses

to do it, you simply lose. So it is in business. Under the present

rules, patents are part of the game.What does that mean in practice? We tell the startups we fund not

to worry about infringing patents, because startups rarely get sued

for patent infringement. There are only two reasons someone might

sue you: for money, or to prevent you from competing with them.

Startups are too poor to be worth suing for money. And in practice

they don't seem to get sued much by competitors, either. They don't

get sued by other startups because (a) patent suits are an expensive

distraction, and (b) since the other startups are as young as they

are, their patents probably haven't issued yet.

[3]

Nor do startups,

at least in the software business, seem to get sued much by established

competitors. Despite all the patents Microsoft holds, I don't know

of an instance where they sued a startup for patent infringement.

Companies like Microsoft and Oracle don't win by winning lawsuits.

That's too uncertain. They win by locking competitors out of their

sales channels. If you do manage to threaten them, they're more

likely to buy you than sue you.When you read of big companies filing patent suits against smaller

ones, it's usually a big company on the way down, grasping at

straws. For example, Unisys's attempts to enforce their patent on

LZW compression. When you see a big company threatening patent

suits, sell. When a company starts fighting over IP, it's a sign

they've lost the real battle, for users.A company that sues competitors for patent infringement is like

a defender who has been beaten so thoroughly that he turns to plead

with the referee. You don't do that if you can still reach the

ball, even if you genuinely believe you've been fouled. So a company

threatening patent suits is a company in trouble.When we were working on Viaweb, a bigger company in the e-commerce

business was granted a patent on online ordering, or something like

that. I got a call from a VP there asking if we'd like to license

it. I replied that I thought the patent was completely bogus, and

would never hold up in court. "Ok," he replied. "So, are you guys

hiring?"If your startup grows big enough, however, you'll start to get sued,

no matter what you do. If you go public, for example, you'll be

sued by multiple patent trolls who hope you'll pay them off to go

away. More on them later.In other words, no one will sue you for patent infringement till

you have money, and once you have money, people will sue you whether

they have grounds to or not. So I advise fatalism. Don't waste

your time worrying about patent infringement. You're probably

violating a patent every time you tie your shoelaces. At the start,

at least, just worry about making something great and getting lots

of users. If you grow to the point where anyone considers you worth

attacking, you're doing well.We do advise the companies we fund to apply for patents, but not

so they can sue competitors. Successful startups either get bought

or grow into big companies. If a startup wants to grow into a big

company, they should apply for patents to build up the patent

portfolio they'll need to maintain an armed truce with other big

companies. If they want to get bought, they should apply for patents

because patents are part of the mating dance with acquirers.Most startups that succeed do it by getting bought, and most acquirers

care about patents. Startup acquisitions are usually a build-vs-buy

decision for the acquirer. Should we buy this little startup or

build our own? And two things, especially, make them decide not

to build their own: if you already have a large and rapidly growing

user base, and if you have a fairly solid patent application on

critical parts of your software.There's a third reason big companies should prefer buying to building:

that if they built their own, they'd screw it up. But few big

companies are smart enough yet to admit this to themselves. It's

usually the acquirer's engineers who are asked how hard it would

be for the company to build their own, and they overestimate their

abilities.

[4]

A patent seems to change the balance. It gives the

acquirer an excuse to admit they couldn't copy what you're doing.

It may also help them to grasp what's special about your technology.Frankly, it surprises me how small a role patents play in the

software business. It's kind of ironic, considering all the dire

things experts say about software patents stifling innovation, but

when one looks closely at the software business, the most striking

thing is how little patents seem to matter.In other fields, companies regularly sue competitors for patent

infringement. For example, the airport baggage scanning business

was for many years a cozy duopoly shared between two companies,

InVision and L-3. In 2002 a startup called Reveal appeared, with

new technology that let them build scanners a third the size. They

were sued for patent infringement before they'd even released a

product.You rarely hear that kind of story in our world. The one example

I've found is, embarrassingly enough, Yahoo, which filed a patent

suit against a gaming startup called Xfire in 2005. Xfire doesn't

seem to be a very big deal, and it's hard to say why Yahoo felt

threatened. Xfire's VP of engineering had worked at Yahoo on similar

stuff-- in fact, he was listed as an inventor on the patent Yahoo

sued over-- so perhaps there was something personal about it. My

guess is that someone at Yahoo goofed. At any rate they didn't

pursue the suit very vigorously.Why do patents play so small a role in software? I can think of

three possible reasons.One is that software is so complicated that patents by themselves

are not worth very much. I may be maligning other fields here, but

it seems that in most types of engineering you can hand the details

of some new technique to a group of medium-high quality people and

get the desired result. For example, if someone develops a new

process for smelting ore that gets a better yield, and you assemble

a team of qualified experts and tell them about it, they'll be able

to get the same yield. This doesn't seem to work in software.

Software is so subtle and unpredictable that "qualified experts"

don't get you very far.That's why we rarely hear phrases like "qualified expert" in the

software business. What that level of ability can get you is, say,

to make your software compatible with some other piece of software--

in eight months, at enormous cost. To do anything harder you need

individual brilliance. If you assemble a team of qualified experts

and tell them to make a new web-based email program, they'll get

their asses kicked by a team of inspired nineteen year olds.Experts can implement, but they can't design.

Or rather, expertise in implementation is the only kind most people,

including the experts themselves, can measure.

[5]But design is a definite skill. It's not just an airy intangible.

Things always seem intangible when you don't understand them.

Electricity seemed an airy intangible to most people in 1800. Who

knew there was so much to know about it? So it is with design.

Some people are good at it and some people are bad at it, and there's

something very tangible they're good or bad at.The reason design counts so much in software is probably that there

are fewer constraints than on physical things. Building physical

things is expensive and dangerous. The space of possible choices

is smaller; you tend to have to work as part of a larger group; and

you're subject to a lot of regulations. You don't have any of that

if you and a couple friends decide to create a new web-based

application.Because there's so much scope for design in software, a successful

application tends to be way more than the sum of its patents. What

protects little companies from being copied by bigger competitors

is not just their patents, but the thousand little things the big

company will get wrong if they try.The second reason patents don't count for much in our world is that

startups rarely attack big companies head-on, the way Reveal did.

In the software business, startups beat established companies by

transcending them. Startups don't build desktop word processing

programs to compete with Microsoft Word.

[6]

They build Writely.

If this paradigm is crowded, just wait for the next one; they run

pretty frequently on this route.Fortunately for startups, big companies are extremely good at denial.

If you take the trouble to attack them from an oblique angle, they'll

meet you half-way and maneuver to keep you in their blind spot. To

sue a startup would mean admitting it was dangerous, and that often

means seeing something the big company doesn't want to see. IBM

used to sue its mainframe competitors regularly, but they didn't

bother much about the microcomputer industry because they didn't

want to see the threat it posed. Companies building web based apps

are similarly protected from Microsoft, which even now doesn't want

to imagine a world in which Windows is irrelevant.The third reason patents don't seem to matter very much in software

is public opinion-- or rather, hacker opinion. In a recent interview,

Steve Ballmer coyly left open the possibility of attacking Linux

on patent grounds. But I doubt Microsoft would ever be so stupid.

They'd face the mother of all boycotts. And not just from the

technical community in general; a lot of their own people would

rebel.Good hackers care a lot about matters of principle, and they are

highly mobile. If a company starts misbehaving, smart people won't

work there. For some reason this seems to be more true in software

than other businesses. I don't think it's because hackers have

intrinsically higher principles so much as that their skills are

easily transferrable. Perhaps we can split the difference and say

that mobility gives hackers the luxury of being principled.Google's "don't be evil" policy may for this reason be the most

valuable thing they've discovered. It's very constraining in some

ways. If Google does do something evil, they get doubly whacked

for it: once for whatever they did, and again for hypocrisy. But

I think it's worth it. It helps them to hire the best people, and

it's better, even from a purely selfish point of view, to be

constrained by principles than by stupidity.(I wish someone would get this point across to the present

administration.)I'm not sure what the proportions are of the preceding three

ingredients, but the custom among the big companies seems to be not

to sue the small ones, and the startups are mostly too busy and too

poor to sue one another. So despite the huge number of software

patents there's not a lot of suing going on. With one exception:

patent trolls.Patent trolls are companies consisting mainly of lawyers whose whole

business is to accumulate patents and threaten to sue companies who

actually make things. Patent trolls, it seems safe to say, are

evil. I feel a bit stupid saying that, because when you're saying

something that Richard Stallman and Bill Gates would both agree

with, you must be perilously close to tautologies.The CEO of Forgent, one of the most notorious patent trolls, says

that what his company does is "the American way." Actually that's

not true. The American way is to make money by creating wealth, not by suing people.

[7]

What companies like Forgent do is actually the proto-industrial

way. In the period just before the industrial revolution, some of

the greatest fortunes in countries like England and France were

made by courtiers who extracted some lucrative right from the crown--

like the right to collect taxes on the import of silk-- and then

used this to squeeze money from the merchants in that business. So

when people compare patent trolls to the mafia, they're more right

than they know, because the mafia too are not merely bad, but bad

specifically in the sense of being an obsolete business model.Patent trolls seem to have caught big companies by surprise. In

the last couple years they've extracted hundreds of millions of

dollars from them. Patent trolls are hard to fight precisely because

they create nothing. Big companies are safe from being sued by

other big companies because they can threaten a counter-suit. But

because patent trolls don't make anything, there's nothing they can

be sued for. I predict this loophole will get closed fairly quickly,

at least by legal standards. It's clearly an abuse of the system,

and the victims are powerful.

[8]But evil as patent trolls are, I don't think they hamper innovation

much. They don't sue till a startup has made money, and by that

point the innovation that generated it has already happened. I

can't think of a startup that avoided working on some problem because

of patent trolls.So much for hockey as the game is played now. What about the more

theoretical question of whether hockey would be a better game without

checking? Do patents encourage or discourage innovation?This is a very hard question to answer in the general case. People

write whole books on the topic. One of my main hobbies is the

history of technology, and even though I've studied the subject for

years, it would take me several weeks of research to be able to say

whether patents have in general been a net win.One thing I can say is that 99.9% of the people who express opinions

on the subject do it not based on such research, but out of a kind

of religious conviction. At least, that's the polite way of putting

it; the colloquial version involves speech coming out of organs not

designed for that purpose.Whether they encourage innovation or not, patents were at least

intended to. You don't get a patent for nothing. In return for

the exclusive right to use an idea, you have to publish it,

and it was largely to encourage such openness that patents were

established.Before patents, people protected ideas by keeping them secret. With

patents, central governments said, in effect, if you tell everyone

your idea, we'll protect it for you. There is a parallel here to

the rise of civil order, which happened at roughly the same time.

Before central governments were powerful enough to enforce order,

rich people had private armies. As governments got more powerful,

they gradually compelled magnates to cede most responsibility for

protecting them. (Magnates still have bodyguards, but no longer

to protect them from other magnates.)Patents, like police, are involved in many abuses. But in both

cases the default is something worse. The choice is not "patents

or freedom?" any more than it is "police or freedom?" The actual

questions are respectively "patents or secrecy?" and "police or

gangs?"As with gangs, we have some idea what secrecy would be like, because

that's how things used to be. The economy of medieval Europe was

divided up into little tribes, each jealously guarding their

privileges and secrets. In Shakespeare's time, "mystery" was

synonymous with "craft." Even today we can see an echo of the

secrecy of medieval guilds, in the now pointless secrecy of the

Masons.The most memorable example of medieval industrial secrecy is probably

Venice, which forbade glassblowers to leave the city, and sent

assassins after those who tried. We might like to think we wouldn't

go so far, but the movie industry has already tried to pass laws

prescribing three year prison terms just for putting movies on

public networks. Want to try a frightening thought experiment? If

the movie industry could have any law they wanted, where would they

stop? Short of the death penalty, one assumes, but how close would

they get?Even worse than the spectacular abuses might be the overall decrease

in efficiency that would accompany increased secrecy. As anyone

who has dealt with organizations that operate on a "need to know"

basis can attest, dividing information up into little cells is

terribly inefficient. The flaw in the "need to know" principle is

that you don't know who needs to know something. An idea

from one area might spark a great discovery in another. But the

discoverer doesn't know he needs to know it.If secrecy were the only protection for ideas, companies wouldn't

just have to be secretive with other companies; they'd have to be

secretive internally. This would encourage what is already the

worst trait of big companies.I'm not saying secrecy would be worse than patents, just that we

couldn't discard patents for free. Businesses would become more

secretive to compensate, and in some fields this might get ugly.

Nor am I defending the current patent system. There is clearly a

lot that's broken about it. But the breakage seems to affect

software less than most other fields.In the software business I know from experience whether patents

encourage or discourage innovation, and the answer is the type that

people who like to argue about public policy least like to hear:

they don't affect innovation much, one way or the other. Most

innovation in the software business happens in startups, and startups

should simply ignore other companies' patents. At least, that's

what we advise, and we bet money on that advice.The only real role of patents, for most startups, is as an element

of the mating dance with acquirers. There patents do help a little.

And so they do encourage innovation indirectly, in that they give

more power to startups, which is where, pound for pound, the most

innovation happens. But even in the mating dance, patents are of

secondary importance. It matters more to make something great and

get a lot of users.Notes[1]

You have to be careful here, because a great discovery often

seems obvious in retrospect. One-click ordering, however, is not

such a discovery.[2]

"Turn the other cheek" skirts the issue; the critical question

is not how to deal with slaps, but sword thrusts.[3]

Applying for a patent is now very slow, but it might actually

be bad if that got fixed. At the moment the time it takes to get

a patent is conveniently just longer than the time it takes a startup

to succeed or fail.[4]

Instead of the canonical "could you build this?" maybe the corp

dev guys should be asking "will you build this?" or even "why haven't

you already built this?"[5]

Design ability is so hard to measure that you can't even trust

the design world's internal standards. You can't assume that someone

with a degree in design is any good at design, or that an eminent

designer is any better than his peers. If that worked, any company

could build products as good as Apple's just by hiring

sufficiently qualified designers.[6]

If anyone wanted to try, we'd be interested to hear from them.

I suspect it's one of those things that's not as hard as everyone

assumes.[7]

Patent trolls can't even claim, like speculators, that they

"create" liquidity.[8]

If big companies don't want to wait for the government to take

action, there is a way to fight back themselves. For a long time

I thought there wasn't, because there was nothing to grab onto.

But there is one resource patent trolls need: lawyers. Big technology

companies between them generate a lot of legal business. If they

agreed among themselves never to do business with any firm employing

anyone who had worked for a patent troll, either as an employee or

as outside counsel, they could probably starve the trolls of the

lawyers they need.Thanks to Dan Bloomberg, Paul Buchheit, Sarah Harlin,

Jessica Livingston, and Peter Norvig

for reading drafts of this, to Joel Lehrer and Peter Eng for answering

my questions about patents, and to Ankur Pansari for inviting me

to speak.Japanese Translation

6,631,372

March 2006, rev August 2009A couple days ago I found to my surprise that I'd been granted a

patent.

It issued in 2003, but no one told me. I wouldn't know about it

now except that a few months ago, while visiting Yahoo, I happened

to run into a Big Cheese I knew from working there in the late

nineties. He brought up something called Revenue Loop, which Viaweb

had been working on when they bought us.The idea is basically that you sort search results not in order of

textual "relevance" (as search engines did then) nor in order of

how much advertisers bid (as Overture did) but in order of the bid

times the number of transactions. Ordinarily you'd do this for

shopping searches, though in fact one of the features of our scheme

is that it automatically detects which searches are shopping searches.If you just order the results in order of bids, you can make the

search results useless, because the first results could be dominated

by lame sites that had bid the most. But if you order results by

bid multiplied by transactions, far from selling out, you're getting

a better measure of relevance. What could be a better sign that

someone was satisfied with a search result than going to the site

and buying something?And, of course, this algorithm automatically maximizes the revenue

of the search engine.Everyone is focused on this type of approach now, but few were in

1998. In 1998 it was all about selling banner ads. We didn't know

that, so we were pretty excited when we figured out what seemed to

us the optimal way of doing shopping searches.When Yahoo was thinking of buying us, we had a meeting with Jerry

Yang in New York. For him, I now realize, this was supposed to be

one of those meetings when you check out a company you've pretty

much decided to buy, just to make sure they're ok guys. We weren't

expected to do more than chat and seem smart and reasonable. He

must have been dismayed when I jumped up to the whiteboard and

launched into a presentation of our exciting new technology.I was just as dismayed when he didn't seem to care at all about it.

At the time I thought, "boy, is this guy poker-faced. We present

to him what has to be the optimal way of sorting product search

results, and he's not even curious." I didn't realize till much later

why he didn't care. In 1998, advertisers were overpaying enormously

for ads on web sites.

In 1998, if advertisers paid the maximum that traffic was worth to

them, Yahoo's revenues would have decreased.Things are different now, of course. Now this sort of thing is all

the rage. So when I ran into the Yahoo exec I knew from the old

days in the Yahoo cafeteria a few months ago, the first thing he

remembered was not (fortunately) all the fights I had with him, but

Revenue Loop."Well," I said, "I think we actually applied for a patent on it.

I'm not sure what happened to the application after I left.""Really? That would be an important patent."So someone investigated, and sure enough, that patent application

had continued in the pipeline for several years after, and finally

issued in 2003.The main thing that struck me on reading it, actually, is that

lawyers at some point messed up my nice clear writing. Some clever

person with a spell checker reduced one section to Zen-like incomprehensibility:

Also, common spelling errors will tend to get fixed. For example,

if users searching for "compact disc player" end up spending

considerable money at sites offering compact disc players, then

those pages will have a higher relevance for that search phrase,

even though the phrase "compact disc player" is not present on

those pages.

(That "compat disc player" wasn't a typo, guys.)For the fine prose of the original, see the provisional application

of February 1998, back when we were still Viaweb and couldn't afford

to pay lawyers to turn every "a lot of" into "considerable."

Why YC

March 2006, rev August 2009Yesterday one of the founders we funded asked me why we started

Y

Combinator. Or more precisely, he asked if we'd started YC mainly

for fun.Kind of, but not quite. It is enormously fun to be able to work

with Rtm and Trevor again. I missed that after we sold Viaweb, and

for all the years after I always had a background process running,

looking for something we could do together. There is definitely

an aspect of a band reunion to Y Combinator. Every couple days I

slip and call it "Viaweb."Viaweb we started very explicitly to make money. I was sick of

living from one freelance project to the next, and decided to just

work as hard as I could till I'd made enough to solve the problem

once and for all. Viaweb was sometimes fun, but it wasn't designed

for fun, and mostly it wasn't. I'd be surprised if any startup is.

All startups are mostly schleps.The real reason we started Y Combinator is neither selfish nor

virtuous. We didn't start it mainly to make money; we have no idea

what our average returns might be, and won't know for years. Nor

did we start YC mainly to help out young would-be founders, though

we do like the idea, and comfort ourselves occasionally with the

thought that if all our investments tank, we will thus have been

doing something unselfish. (It's oddly nondeterministic.)The real reason we started Y Combinator is one probably only a

hacker would understand. We did it because it seems such a great

hack. There are thousands of smart people who could start companies

and don't, and with a relatively small amount of force applied at

just the right place, we can spring on the world a stream of new

startups that might otherwise not have existed.In a way this is virtuous, because I think startups are a good

thing. But really what motivates us is the completely amoral desire

that would motivate any hacker who looked at some complex device

and realized that with a tiny tweak he could make it run more

efficiently. In this case, the device is the world's economy, which

fortunately happens to be open source.

How to Do What You Love

Want to start a startup? Get funded by

Y Combinator.

January 2006To do something well you have to like it. That idea is not exactly

novel. We've got it down to four words: "Do what you love." But

it's not enough just to tell people that. Doing what you love is

complicated.The very idea is foreign to what most of us learn as kids. When I

was a kid, it seemed as if work and fun were opposites by definition.

Life had two states: some of the time adults were making you do

things, and that was called work; the rest of the time you could

do what you wanted, and that was called playing. Occasionally the

things adults made you do were fun, just as, occasionally, playing

wasn't—for example, if you fell and hurt yourself. But except

for these few anomalous cases, work was pretty much defined as

not-fun.And it did not seem to be an accident. School, it was implied, was

tedious because it was preparation for grownup work.The world then was divided into two groups, grownups and kids.

Grownups, like some kind of cursed race, had to work. Kids didn't,

but they did have to go to school, which was a dilute version of

work meant to prepare us for the real thing. Much as we disliked

school, the grownups all agreed that grownup work was worse, and

that we had it easy.Teachers in particular all seemed to believe implicitly that work

was not fun. Which is not surprising: work wasn't fun for most of

them. Why did we have to memorize state capitals instead of playing

dodgeball? For the same reason they had to watch over a bunch of

kids instead of lying on a beach. You couldn't just do what you

wanted.I'm not saying we should let little kids do whatever they want.

They may have to be made to work on certain things. But if we make

kids work on dull stuff, it might be wise to tell them that tediousness

is not the defining quality of work, and indeed that the reason

they have to work on dull stuff now is so they can work on more

interesting stuff later.

[1]Once, when I was about 9 or 10, my father told me I could be whatever

I wanted when I grew up, so long as I enjoyed it. I remember that

precisely because it seemed so anomalous. It was like being told

to use dry water. Whatever I thought he meant, I didn't think he

meant work could literally be fun—fun like playing. It

took me years to grasp that.JobsBy high school, the prospect of an actual job was on the horizon.

Adults would sometimes come to speak to us about their work, or we

would go to see them at work. It was always understood that they

enjoyed what they did. In retrospect I think one may have: the

private jet pilot. But I don't think the bank manager really did.The main reason they all acted as if they enjoyed their work was

presumably the upper-middle class convention that you're supposed

to. It would not merely be bad for your career to say that you

despised your job, but a social faux-pas.Why is it conventional to pretend to like what you do? The first

sentence of this essay explains that. If you have to like something

to do it well, then the most successful people will all like what

they do. That's where the upper-middle class tradition comes from.

Just as houses all over America are full of

chairs

that are, without

the owners even knowing it, nth-degree imitations of chairs designed

250 years ago for French kings, conventional attitudes about work

are, without the owners even knowing it, nth-degree imitations of

the attitudes of people who've done great things.What a recipe for alienation. By the time they reach an age to

think about what they'd like to do, most kids have been thoroughly

misled about the idea of loving one's work. School has trained

them to regard work as an unpleasant duty. Having a job is said

to be even more onerous than schoolwork. And yet all the adults

claim to like what they do. You can't blame kids for thinking "I

am not like these people; I am not suited to this world."Actually they've been told three lies: the stuff they've been taught

to regard as work in school is not real work; grownup work is not

(necessarily) worse than schoolwork; and many of the adults around

them are lying when they say they like what they do.The most dangerous liars can be the kids' own parents. If you take

a boring job to give your family a high standard of living, as so

many people do, you risk infecting your kids with the idea that

work is boring.

[2]

Maybe it would be better for kids in this one

case if parents were not so unselfish. A parent who set an example

of loving their work might help their kids more than an expensive

house.

[3]It was not till I was in college that the idea of work finally broke

free from the idea of making a living. Then the important question

became not how to make money, but what to work on. Ideally these

coincided, but some spectacular boundary cases (like Einstein in

the patent office) proved they weren't identical.The definition of work was now to make some original contribution

to the world, and in the process not to starve. But after the habit

of so many years my idea of work still included a large component

of pain. Work still seemed to require discipline, because only

hard problems yielded grand results, and hard problems couldn't

literally be fun. Surely one had to force oneself to work on them.If you think something's supposed to hurt, you're less likely to

notice if you're doing it wrong. That about sums up my experience

of graduate school.BoundsHow much are you supposed to like what you do? Unless you

know that, you don't know when to stop searching. And if, like most

people, you underestimate it, you'll tend to stop searching too

early. You'll end up doing something chosen for you by your parents,

or the desire to make money, or prestige—or sheer inertia.Here's an upper bound: Do what you love doesn't mean, do what you

would like to do most this second. Even Einstein probably

had moments when he wanted to have a cup of coffee, but told himself

he ought to finish what he was working on first.It used to perplex me when I read about people who liked what they

did so much that there was nothing they'd rather do. There didn't

seem to be any sort of work I liked that much. If I had a

choice of (a) spending the next hour working on something or (b)

be teleported to Rome and spend the next hour wandering about, was

there any sort of work I'd prefer? Honestly, no.But the fact is, almost anyone would rather, at any given moment,

float about in the Carribbean, or have sex, or eat some delicious

food, than work on hard problems. The rule about doing what you

love assumes a certain length of time. It doesn't mean, do what

will make you happiest this second, but what will make you happiest

over some longer period, like a week or a month.Unproductive pleasures pall eventually. After a while you get tired

of lying on the beach. If you want to stay happy, you have to do

something.As a lower bound, you have to like your work more than any unproductive

pleasure. You have to like what you do enough that the concept of

"spare time" seems mistaken. Which is not to say you have to spend

all your time working. You can only work so much before you get

tired and start to screw up. Then you want to do something else—even something mindless. But you don't regard this time as the

prize and the time you spend working as the pain you endure to earn

it.I put the lower bound there for practical reasons. If your work

is not your favorite thing to do, you'll have terrible problems

with procrastination. You'll have to force yourself to work, and

when you resort to that the results are distinctly inferior.To be happy I think you have to be doing something you not only

enjoy, but admire. You have to be able to say, at the end, wow,

that's pretty cool. This doesn't mean you have to make something.

If you learn how to hang glide, or to speak a foreign language

fluently, that will be enough to make you say, for a while at least,

wow, that's pretty cool. What there has to be is a test.So one thing that falls just short of the standard, I think, is

reading books. Except for some books in math and the hard sciences,

there's no test of how well you've read a book, and that's why

merely reading books doesn't quite feel like work. You have to do

something with what you've read to feel productive.I think the best test is one Gino Lee taught me: to try to do things

that would make your friends say wow. But it probably wouldn't

start to work properly till about age 22, because most people haven't

had a big enough sample to pick friends from before then.SirensWhat you should not do, I think, is worry about the opinion of

anyone beyond your friends. You shouldn't worry about prestige.

Prestige is the opinion of the rest of the world. When you can ask

the opinions of people whose judgement you respect, what does it

add to consider the opinions of people you don't even know?

[4]This is easy advice to give. It's hard to follow, especially when

you're young.

[5]

Prestige is like a powerful magnet that warps

even your beliefs about what you enjoy. It causes you to work not

on what you like, but what you'd like to like.That's what leads people to try to write novels, for example. They

like reading novels. They notice that people who write them win

Nobel prizes. What could be more wonderful, they think, than to

be a novelist? But liking the idea of being a novelist is not

enough; you have to like the actual work of novel-writing if you're

going to be good at it; you have to like making up elaborate lies.Prestige is just fossilized inspiration. If you do anything well

enough, you'll make it prestigious. Plenty of things we now

consider prestigious were anything but at first. Jazz comes to

mind—though almost any established art form would do. So just

do what you like, and let prestige take care of itself.Prestige is especially dangerous to the ambitious. If you want to

make ambitious people waste their time on errands, the way to do

it is to bait the hook with prestige. That's the recipe for getting

people to give talks, write forewords, serve on committees, be

department heads, and so on. It might be a good rule simply to

avoid any prestigious task. If it didn't suck, they wouldn't have

had to make it prestigious.Similarly, if you admire two kinds of work equally, but one is more

prestigious, you should probably choose the other. Your opinions

about what's admirable are always going to be slightly influenced

by prestige, so if the two seem equal to you, you probably have

more genuine admiration for the less prestigious one.The other big force leading people astray is money. Money by itself

is not that dangerous. When something pays well but is regarded

with contempt, like telemarketing, or prostitution, or personal

injury litigation, ambitious people aren't tempted by it. That

kind of work ends up being done by people who are "just trying to

make a living." (Tip: avoid any field whose practitioners say

this.) The danger is when money is combined with prestige, as in,

say, corporate law, or medicine. A comparatively safe and prosperous

career with some automatic baseline prestige is dangerously tempting

to someone young, who hasn't thought much about what they really

like.The test of whether people love what they do is whether they'd do

it even if they weren't paid for it—even if they had to work at

another job to make a living. How many corporate lawyers would do

their current work if they had to do it for free, in their spare

time, and take day jobs as waiters to support themselves?This test is especially helpful in deciding between different kinds

of academic work, because fields vary greatly in this respect. Most

good mathematicians would work on math even if there were no jobs

as math professors, whereas in the departments at the other end of

the spectrum, the availability of teaching jobs is the driver:

people would rather be English professors than work in ad agencies,

and publishing papers is the way you compete for such jobs. Math

would happen without math departments, but it is the existence of

English majors, and therefore jobs teaching them, that calls into

being all those thousands of dreary papers about gender and identity

in the novels of Conrad. No one does

that

kind of thing for fun.The advice of parents will tend to err on the side of money. It

seems safe to say there are more undergrads who want to be novelists

and whose parents want them to be doctors than who want to be doctors

and whose parents want them to be novelists. The kids think their

parents are "materialistic." Not necessarily. All parents tend to

be more conservative for their kids than they would for themselves,

simply because, as parents, they share risks more than rewards. If

your eight year old son decides to climb a tall tree, or your teenage

daughter decides to date the local bad boy, you won't get a share

in the excitement, but if your son falls, or your daughter gets

pregnant, you'll have to deal with the consequences.DisciplineWith such powerful forces leading us astray, it's not surprising

we find it so hard to discover what we like to work on. Most people

are doomed in childhood by accepting the axiom that work = pain.

Those who escape this are nearly all lured onto the rocks by prestige

or money. How many even discover something they love to work on?

A few hundred thousand, perhaps, out of billions.It's hard to find work you love; it must be, if so few do. So don't

underestimate this task. And don't feel bad if you haven't succeeded

yet. In fact, if you admit to yourself that you're discontented,

you're a step ahead of most people, who are still in denial. If

you're surrounded by colleagues who claim to enjoy work that you

find contemptible, odds are they're lying to themselves. Not

necessarily, but probably.Although doing great work takes less discipline than people think—because the way to do great work is to find something you like so

much that you don't have to force yourself to do it—finding

work you love does usually require discipline. Some people are

lucky enough to know what they want to do when they're 12, and just

glide along as if they were on railroad tracks. But this seems the

exception. More often people who do great things have careers with

the trajectory of a ping-pong ball. They go to school to study A,

drop out and get a job doing B, and then become famous for C after

taking it up on the side.Sometimes jumping from one sort of work to another is a sign of

energy, and sometimes it's a sign of laziness. Are you dropping

out, or boldly carving a new path? You often can't tell yourself.

Plenty of people who will later do great things seem to be disappointments

early on, when they're trying to find their niche.Is there some test you can use to keep yourself honest? One is to

try to do a good job at whatever you're doing, even if you don't

like it. Then at least you'll know you're not using dissatisfaction

as an excuse for being lazy. Perhaps more importantly, you'll get

into the habit of doing things well.Another test you can use is: always produce. For example, if you

have a day job you don't take seriously because you plan to be a

novelist, are you producing? Are you writing pages of fiction,

however bad? As long as you're producing, you'll know you're not

merely using the hazy vision of the grand novel you plan to write

one day as an opiate. The view of it will be obstructed by the all

too palpably flawed one you're actually writing."Always produce" is also a heuristic for finding the work you love.

If you subject yourself to that constraint, it will automatically

push you away from things you think you're supposed to work on,

toward things you actually like. "Always produce" will discover

your life's work the way water, with the aid of gravity, finds the

hole in your roof.Of course, figuring out what you like to work on doesn't mean you

get to work on it. That's a separate question. And if you're

ambitious you have to keep them separate: you have to make a conscious

effort to keep your ideas about what you want from being contaminated

by what seems possible.

[6]It's painful to keep them apart, because it's painful to observe

the gap between them. So most people pre-emptively lower their

expectations. For example, if you asked random people on the street

if they'd like to be able to draw like Leonardo, you'd find most

would say something like "Oh, I can't draw." This is more a statement

of intention than fact; it means, I'm not going to try. Because

the fact is, if you took a random person off the street and somehow

got them to work as hard as they possibly could at drawing for the

next twenty years, they'd get surprisingly far. But it would require

a great moral effort; it would mean staring failure in the eye every

day for years. And so to protect themselves people say "I can't."Another related line you often hear is that not everyone can do

work they love—that someone has to do the unpleasant jobs. Really?

How do you make them? In the US the only mechanism for forcing

people to do unpleasant jobs is the draft, and that hasn't been

invoked for over 30 years. All we can do is encourage people to

do unpleasant work, with money and prestige.If there's something people still won't do, it seems as if society

just has to make do without. That's what happened with domestic

servants. For millennia that was the canonical example of a job

"someone had to do." And yet in the mid twentieth century servants

practically disappeared in rich countries, and the rich have just

had to do without.So while there may be some things someone has to do, there's a good

chance anyone saying that about any particular job is mistaken.

Most unpleasant jobs would either get automated or go undone if no

one were willing to do them.Two RoutesThere's another sense of "not everyone can do work they love"

that's all too true, however. One has to make a living, and it's

hard to get paid for doing work you love. There are two routes to

that destination:

The organic route: as you become more eminent, gradually to

increase the parts of your job that you like at the expense of

those you don't.The two-job route: to work at things you don't like to get money

to work on things you do.

The organic route is more common. It happens naturally to anyone

who does good work. A young architect has to take whatever work

he can get, but if he does well he'll gradually be in a position

to pick and choose among projects. The disadvantage of this route

is that it's slow and uncertain. Even tenure is not real freedom.The two-job route has several variants depending on how long you

work for money at a time. At one extreme is the "day job," where

you work regular hours at one job to make money, and work on what

you love in your spare time. At the other extreme you work at

something till you make enough not to

have to work for money again.The two-job route is less common than the organic route, because

it requires a deliberate choice. It's also more dangerous. Life

tends to get more expensive as you get older, so it's easy to get

sucked into working longer than you expected at the money job.

Worse still, anything you work on changes you. If you work too

long on tedious stuff, it will rot your brain. And the best paying

jobs are most dangerous, because they require your full attention.The advantage of the two-job route is that it lets you jump over

obstacles. The landscape of possible jobs isn't flat; there are

walls of varying heights between different kinds of work.

[7]

The trick of maximizing the parts of your job that you like can get you

from architecture to product design, but not, probably, to music.

If you make money doing one thing and then work on another, you

have more freedom of choice.Which route should you take? That depends on how sure you are of

what you want to do, how good you are at taking orders, how much

risk you can stand, and the odds that anyone will pay (in your

lifetime) for what you want to do. If you're sure of the general

area you want to work in and it's something people are likely to

pay you for, then you should probably take the organic route. But

if you don't know what you want to work on, or don't like to take

orders, you may want to take the two-job route, if you can stand

the risk.Don't decide too soon. Kids who know early what they want to do

seem impressive, as if they got the answer to some math question

before the other kids. They have an answer, certainly, but odds

are it's wrong.A friend of mine who is a quite successful doctor complains constantly

about her job. When people applying to medical school ask her for

advice, she wants to shake them and yell "Don't do it!" (But she

never does.) How did she get into this fix? In high school she

already wanted to be a doctor. And she is so ambitious and determined

that she overcame every obstacle along the way—including,

unfortunately, not liking it.Now she has a life chosen for her by a high-school kid.When you're young, you're given the impression that you'll get

enough information to make each choice before you need to make it.

But this is certainly not so with work. When you're deciding what

to do, you have to operate on ridiculously incomplete information.

Even in college you get little idea what various types of work are

like. At best you may have a couple internships, but not all jobs

offer internships, and those that do don't teach you much more about

the work than being a batboy teaches you about playing baseball.In the design of lives, as in the design of most other things, you

get better results if you use flexible media. So unless you're

fairly sure what you want to do, your best bet may be to choose a

type of work that could turn into either an organic or two-job

career. That was probably part of the reason I chose computers.

You can be a professor, or make a lot of money, or morph it into

any number of other kinds of work.It's also wise, early on, to seek jobs that let you do many different

things, so you can learn faster what various kinds of work are like.

Conversely, the extreme version of the two-job route is dangerous

because it teaches you so little about what you like. If you work

hard at being a bond trader for ten years, thinking that you'll

quit and write novels when you have enough money, what happens when

you quit and then discover that you don't actually like writing

novels?Most people would say, I'd take that problem. Give me a million

dollars and I'll figure out what to do. But it's harder than it

looks. Constraints give your life shape. Remove them and most

people have no idea what to do: look at what happens to those who

win lotteries or inherit money. Much as everyone thinks they want

financial security, the happiest people are not those who have it,

but those who like what they do. So a plan that promises freedom

at the expense of knowing what to do with it may not be as good as

it seems.Whichever route you take, expect a struggle. Finding work you love

is very difficult. Most people fail. Even if you succeed, it's

rare to be free to work on what you want till your thirties or

forties. But if you have the destination in sight you'll be more

likely to arrive at it. If you know you can love work, you're in

the home stretch, and if you know what work you love, you're

practically there.Notes[1]

Currently we do the opposite: when we make kids do boring work,

like arithmetic drills, instead of admitting frankly that it's

boring, we try to disguise it with superficial decorations.[2]

One father told me about a related phenomenon: he found himself

concealing from his family how much he liked his work. When he

wanted to go to work on a saturday, he found it easier to say that

it was because he "had to" for some reason, rather than admitting

he preferred to work than stay home with them.[3]

Something similar happens with suburbs. Parents move to suburbs

to raise their kids in a safe environment, but suburbs are so dull

and artificial that by the time they're fifteen the kids are convinced

the whole world is boring.[4]

I'm not saying friends should be the only audience for your

work. The more people you can help, the better. But friends should

be your compass.[5]

Donald Hall said young would-be poets were mistaken to be so

obsessed with being published. But you can imagine what it would

do for a 24 year old to get a poem published in The New Yorker.

Now to people he meets at parties he's a real poet. Actually he's

no better or worse than he was before, but to a clueless audience

like that, the approval of an official authority makes all the

difference. So it's a harder problem than Hall realizes. The

reason the young care so much about prestige is that the people

they want to impress are not very discerning.[6]

This is isomorphic to the principle that you should prevent

your beliefs about how things are from being contaminated by how

you wish they were. Most people let them mix pretty promiscuously.

The continuing popularity of religion is the most visible index of

that.[7]

A more accurate metaphor would be to say that the graph of jobs

is not very well connected.Thanks to Trevor Blackwell, Dan Friedman, Sarah Harlin,

Jessica Livingston, Jackie McDonough, Robert Morris, Peter Norvig,

David Sloo, and Aaron Swartz

for reading drafts of this.Hebrew TranslationJapanese TranslationChinese TranslationRussian TranslationSlovak TranslationItalian TranslationGerman TranslationSpanish TranslationFrench TranslationHungarian TranslationPortuguese TranslationSerbian TranslationGreek TranslationVietnamese Translation

Good and Bad Procrastination

December 2005The most impressive people I know are all terrible procrastinators.

So could it be that procrastination isn't always bad?Most people who write about procrastination write about how to cure

it. But this is, strictly speaking, impossible. There are an

infinite number of things you could be doing. No matter what you

work on, you're not working on everything else. So the question

is not how to avoid procrastination, but how to procrastinate well.There are three variants of procrastination, depending on what you

do instead of working on something: you could work on (a) nothing,

(b) something less important, or (c) something more important. That

last type, I'd argue, is good procrastination.That's the "absent-minded professor," who forgets to shave, or eat,

or even perhaps look where he's going while he's thinking about

some interesting question. His mind is absent from the everyday

world because it's hard at work in another.That's the sense in which the most impressive people I know are all

procrastinators. They're type-C procrastinators: they put off

working on small stuff to work on big stuff.What's "small stuff?" Roughly, work that has zero chance of being

mentioned in your obituary. It's hard to say at the time what will

turn out to be your best work (will it be your magnum opus on

Sumerian temple architecture, or the detective thriller you wrote

under a pseudonym?), but there's a whole class of tasks you can

safely rule out: shaving, doing your laundry, cleaning the house,

writing thank-you notes—anything that might be called an errand.Good procrastination is avoiding errands to do real work.Good in a sense, at least. The people who want you to do the errands

won't think it's good. But you probably have to annoy them if you

want to get anything done. The mildest seeming people, if they

want to do real work, all have a certain degree of ruthlessness

when it comes to avoiding errands.Some errands, like replying to letters, go away if you

ignore them (perhaps taking friends with them). Others, like mowing

the lawn, or filing tax returns, only get worse if you put them

off. In principle it shouldn't work to put off the second kind of

errand. You're going to have to do whatever it is eventually. Why

not (as past-due notices are always saying) do it now?The reason it pays to put off even those errands is that real work

needs two things errands don't: big chunks of time, and the

right mood. If you get inspired by some project, it can be a net

win to blow off everything you were supposed to do for the next few

days to work on it. Yes, those errands may cost you more time when

you finally get around to them. But if you get a lot done during

those few days, you will be net more productive.In fact, it may not be a difference in degree, but a difference in

kind. There may be types of work that can only be done in long,

uninterrupted stretches, when inspiration hits, rather than dutifully

in scheduled little slices. Empirically it seems to be so. When

I think of the people I know who've done great things, I don't

imagine them dutifully crossing items off to-do lists. I imagine

them sneaking off to work on some new idea.Conversely, forcing someone to perform errands synchronously is

bound to limit their productivity. The cost of an interruption is

not just the time it takes, but that it breaks the time on either

side in half. You probably only have to interrupt someone a couple

times a day before they're unable to work on hard problems at all.I've wondered a lot about why

startups are most productive at the

very beginning, when they're just a couple guys in an apartment.

The main reason may be that there's no one to interrupt them yet.

In theory it's good when the founders finally get enough money to

hire people to do some of the work for them. But it may be better

to be overworked than interrupted. Once you dilute a startup with

ordinary office workers—with type-B procrastinators—the whole

company starts to resonate at their frequency. They're interrupt-driven,

and soon you are too.Errands are so effective at killing great projects that a lot of

people use them for that purpose. Someone who has decided to write

a novel, for example, will suddenly find that the house needs

cleaning. People who fail to write novels don't do it by sitting

in front of a blank page for days without writing anything. They

do it by feeding the cat, going out to buy something they need for

their apartment, meeting a friend for coffee, checking email. "I

don't have time to work," they say. And they don't; they've made

sure of that.(There's also a variant where one has no place to work. The cure

is to visit the places where famous people worked, and see how

unsuitable they were.)I've used both these excuses at one time or another. I've learned

a lot of tricks for making myself work over the last 20 years, but

even now I don't win consistently. Some days I get real work done.

Other days are eaten up by errands. And I know it's usually my

fault: I let errands eat up the day, to avoid

facing some hard problem.The most dangerous form of procrastination is unacknowledged type-B

procrastination, because it doesn't feel like procrastination.

You're "getting things done." Just the wrong things.Any advice about procrastination that concentrates on crossing

things off your to-do list is not only incomplete, but positively

misleading, if it doesn't consider the possibility that the to-do

list is itself a form of type-B procrastination. In fact, possibility

is too weak a word. Nearly everyone's is. Unless you're working

on the biggest things you could be working on, you're type-B

procrastinating, no matter how much you're getting done.In his famous essay You and Your Research

(which I recommend to

anyone ambitious, no matter what they're working on), Richard Hamming

suggests that you ask yourself three questions:

What are the most important problems in your field? Are you working on one of them? Why not?

Hamming was at Bell Labs when he started asking such questions. In

principle anyone there ought to have been able to work on the most

important problems in their field. Perhaps not everyone can make

an equally dramatic mark on the world; I don't know; but whatever

your capacities, there are projects that stretch them. So Hamming's

exercise can be generalized to:

What's the best thing you could be working on, and why aren't

you?

Most people will shy away from this question. I shy away from it

myself; I see it there on the page and quickly move on to the next

sentence. Hamming used to go around actually asking people this,

and it didn't make him popular. But it's a question anyone ambitious

should face.The trouble is, you may end up hooking a very big fish with this

bait. To do good work, you need to do more than find good projects.

Once you've found them, you have to get yourself to work on them,

and that can be hard. The bigger the problem, the harder it is to

get yourself to work on it.Of course, the main reason people find it difficult to work on a

particular problem is that they don't

enjoy it. When you're young,

especially, you often find yourself working on stuff you don't

really like-- because it seems impressive, for example, or because

you've been assigned to work on it. Most grad students are stuck

working on big problems they don't really like, and grad school is

thus synonymous with procrastination.But even when you like what you're working on, it's easier to get

yourself to work on small problems than big ones. Why? Why is it

so hard to work on big problems? One reason is that you may not

get any reward in the forseeable future. If you work on something

you can finish in a day or two, you can expect to have a nice feeling

of accomplishment fairly soon. If the reward is indefinitely far

in the future, it seems less real.Another reason people don't work on big projects is, ironically,

fear of wasting time. What if they fail? Then all the time they

spent on it will be wasted. (In fact it probably won't be, because

work on hard projects almost always leads somewhere.)But the trouble with big problems can't be just that they promise

no immediate reward and might cause you to waste a lot of time. If

that were all, they'd be no worse than going to visit your in-laws.

There's more to it than that. Big problems are terrifying.

There's an almost physical pain in facing them. It's like having

a vacuum cleaner hooked up to your imagination. All your initial

ideas get sucked out immediately, and you don't have any more, and

yet the vacuum cleaner is still sucking.You can't look a big problem too directly in the eye. You have to

approach it somewhat obliquely. But you have to adjust the angle

just right: you have to be facing the big problem directly enough

that you catch some of the excitement radiating from it, but not

so much that it paralyzes you. You can tighten the angle once you

get going, just as a sailboat can sail closer to the wind once it

gets underway.If you want to work on big things, you seem to have to trick yourself

into doing it. You have to work on small things that could grow

into big things, or work on successively larger things, or split

the moral load with collaborators. It's not a sign of weakness to

depend on such tricks. The very best work has been done this way.When I talk to people who've managed to make themselves work on big

things, I find that all blow off errands, and all feel guilty about

it. I don't think they should feel guilty. There's more to do

than anyone could. So someone doing the best work they can is

inevitably going to leave a lot of errands undone. It seems a

mistake to feel bad about that.I think the way to "solve" the problem of procrastination is to let

delight pull you instead of making a to-do list push you. Work on

an ambitious project you really enjoy, and sail as close to the

wind as you can, and you'll leave the right things undone.Thanks to Trevor Blackwell, Jessica Livingston, and Robert

Morris for reading drafts of this.Romanian TranslationRussian TranslationHebrew TranslationGerman TranslationPortuguese TranslationItalian TranslationJapanese TranslationSpanish Translation

Web 2.0

Want to start a startup? Get funded by

Y Combinator.

November 2005Does "Web 2.0" mean anything? Till recently I thought it didn't,

but the truth turns out to be more complicated. Originally, yes,

it was meaningless. Now it seems to have acquired a meaning. And

yet those who dislike the term are probably right, because if it

means what I think it does, we don't need it.I first heard the phrase "Web 2.0" in the name of the Web 2.0

conference in 2004. At the time it was supposed to mean using "the

web as a platform," which I took to refer to web-based applications.

[1]So I was surprised at a conference this summer when Tim O'Reilly

led a session intended to figure out a definition of "Web 2.0."

Didn't it already mean using the web as a platform? And if it

didn't already mean something, why did we need the phrase at all?OriginsTim says the phrase "Web 2.0" first

arose in "a brainstorming session between

O'Reilly and Medialive International." What is Medialive International?

"Producers of technology tradeshows and conferences," according to

their site. So presumably that's what this brainstorming session

was about. O'Reilly wanted to organize a conference about the web,

and they were wondering what to call it.I don't think there was any deliberate plan to suggest there was a

new version of the web. They just wanted to make the point

that the web mattered again. It was a kind of semantic deficit

spending: they knew new things were coming, and the "2.0" referred

to whatever those might turn out to be.And they were right. New things were coming. But the new version

number led to some awkwardness in the short term. In the process

of developing the pitch for the first conference, someone must have

decided they'd better take a stab at explaining what that "2.0"

referred to. Whatever it meant, "the web as a platform" was at

least not too constricting.The story about "Web 2.0" meaning the web as a platform didn't live

much past the first conference. By the second conference, what

"Web 2.0" seemed to mean was something about democracy. At least,

it did when people wrote about it online. The conference itself

didn't seem very grassroots. It cost $2800, so the only people who

could afford to go were VCs and people from big companies.And yet, oddly enough, Ryan Singel's article

about the conference in Wired News spoke of "throngs of

geeks." When a friend of mine asked Ryan about this, it was news

to him. He said he'd originally written something like "throngs

of VCs and biz dev guys" but had later shortened it just to "throngs,"

and that this must have in turn been expanded by the editors into

"throngs of geeks." After all, a Web 2.0 conference would presumably

be full of geeks, right?Well, no. There were about 7. Even Tim O'Reilly was wearing a

suit, a sight so alien I couldn't parse it at first. I saw

him walk by and said to one of the O'Reilly people "that guy looks

just like Tim.""Oh, that's Tim. He bought a suit."

I ran after him, and sure enough, it was. He explained that he'd

just bought it in Thailand.The 2005 Web 2.0 conference reminded me of Internet trade shows

during the Bubble, full of prowling VCs looking for the next hot

startup. There was that same odd atmosphere created by a large

number of people determined not to miss out. Miss out on what?

They didn't know. Whatever was going to happen—whatever Web 2.0

turned out to be.I wouldn't quite call it "Bubble 2.0" just because VCs are eager

to invest again. The Internet is a genuinely big deal. The bust

was as much an overreaction as

the boom. It's to be expected that once we started to pull out of

the bust, there would be a lot of growth in this area, just as there

was in the industries that spiked the sharpest before the Depression.The reason this won't turn into a second Bubble is that the IPO

market is gone. Venture investors

are driven by exit strategies. The reason they were funding all

those laughable startups during the late 90s was that they hoped

to sell them to gullible retail investors; they hoped to be laughing

all the way to the bank. Now that route is closed. Now the default

exit strategy is to get bought, and acquirers are less prone to

irrational exuberance than IPO investors. The closest you'll get

to Bubble valuations is Rupert Murdoch paying $580 million for

Myspace. That's only off by a factor of 10 or so.1. AjaxDoes "Web 2.0" mean anything more than the name of a conference

yet? I don't like to admit it, but it's starting to. When people

say "Web 2.0" now, I have some idea what they mean. And the fact

that I both despise the phrase and understand it is the surest proof

that it has started to mean something.One ingredient of its meaning is certainly Ajax, which I can still

only just bear to use without scare quotes. Basically, what "Ajax"

means is "Javascript now works." And that in turn means that

web-based applications can now be made to work much more like desktop

ones.As you read this, a whole new generation

of software is being written to take advantage of Ajax. There

hasn't been such a wave of new applications since microcomputers

first appeared. Even Microsoft sees it, but it's too late for them

to do anything more than leak "internal"

documents designed to give the impression they're on top of this

new trend.In fact the new generation of software is being written way too

fast for Microsoft even to channel it, let alone write their own

in house. Their only hope now is to buy all the best Ajax startups

before Google does. And even that's going to be hard, because

Google has as big a head start in buying microstartups as it did

in search a few years ago. After all, Google Maps, the canonical

Ajax application, was the result of a startup they bought.So ironically the original description of the Web 2.0 conference

turned out to be partially right: web-based applications are a big

component of Web 2.0. But I'm convinced they got this right by

accident. The Ajax boom didn't start till early 2005, when Google

Maps appeared and the term "Ajax" was coined.2. DemocracyThe second big element of Web 2.0 is democracy. We now have several

examples to prove that amateurs can

surpass professionals, when they have the right kind of system to

channel their efforts. Wikipedia

may be the most famous. Experts have given Wikipedia middling

reviews, but they miss the critical point: it's good enough. And

it's free, which means people actually read it. On the web, articles

you have to pay for might as well not exist. Even if you were

willing to pay to read them yourself, you can't link to them.

They're not part of the conversation.Another place democracy seems to win is in deciding what counts as

news. I never look at any news site now except Reddit.

[2]

I know if something major

happens, or someone writes a particularly interesting article, it

will show up there. Why bother checking the front page of any

specific paper or magazine? Reddit's like an RSS feed for the whole

web, with a filter for quality. Similar sites include Digg, a technology news site that's

rapidly approaching Slashdot in popularity, and del.icio.us, the collaborative

bookmarking network that set off the "tagging" movement. And whereas

Wikipedia's main appeal is that it's good enough and free, these

sites suggest that voters do a significantly better job than human

editors.The most dramatic example of Web 2.0 democracy is not in the selection

of ideas, but their production.

I've noticed for a while that the stuff I read on individual people's

sites is as good as or better than the stuff I read in newspapers

and magazines. And now I have independent evidence: the top links

on Reddit are generally links to individual people's sites rather

than to magazine articles or news stories.My experience of writing

for magazines suggests an explanation. Editors. They control the

topics you can write about, and they can generally rewrite whatever

you produce. The result is to damp extremes. Editing yields 95th

percentile writing—95% of articles are improved by it, but 5% are

dragged down. 5% of the time you get "throngs of geeks."On the web, people can publish whatever they want. Nearly all of

it falls short of the editor-damped writing in print publications.

But the pool of writers is very, very large. If it's large enough,

the lack of damping means the best writing online should surpass

the best in print.

[3]

And now that the web has evolved mechanisms

for selecting good stuff, the web wins net. Selection beats damping,

for the same reason market economies beat centrally planned ones.Even the startups are different this time around. They are to the

startups of the Bubble what bloggers are to the print media. During

the Bubble, a startup meant a company headed by an MBA that was

blowing through several million dollars of VC money to "get big

fast" in the most literal sense. Now it means a smaller, younger, more technical group that just

decided to make something great. They'll decide later if they want

to raise VC-scale funding, and if they take it, they'll take it on

their terms.3. Don't Maltreat UsersI think everyone would agree that democracy and Ajax are elements

of "Web 2.0." I also see a third: not to maltreat users. During

the Bubble a lot of popular sites were quite high-handed with users.

And not just in obvious ways, like making them register, or subjecting

them to annoying ads. The very design of the average site in the

late 90s was an abuse. Many of the most popular sites were loaded

with obtrusive branding that made them slow to load and sent the

user the message: this is our site, not yours. (There's a physical

analog in the Intel and Microsoft stickers that come on some

laptops.)I think the root of the problem was that sites felt they were giving

something away for free, and till recently a company giving anything

away for free could be pretty high-handed about it. Sometimes it

reached the point of economic sadism: site owners assumed that the

more pain they caused the user, the more benefit it must be to them.

The most dramatic remnant of this model may be at salon.com, where

you can read the beginning of a story, but to get the rest you have

sit through a movie.At Y Combinator we advise all the startups we fund never to lord

it over users. Never make users register, unless you need to in

order to store something for them. If you do make users register,

never make them wait for a confirmation link in an email; in fact,

don't even ask for their email address unless you need it for some

reason. Don't ask them any unnecessary questions. Never send them

email unless they explicitly ask for it. Never frame pages you

link to, or open them in new windows. If you have a free version

and a pay version, don't make the free version too restricted. And

if you find yourself asking "should we allow users to do x?" just

answer "yes" whenever you're unsure. Err on the side of generosity.In How to Start a Startup I advised startups

never to let anyone fly under them, meaning never to let any other

company offer a cheaper, easier solution. Another way to fly low

is to give users more power. Let users do what they want. If you

don't and a competitor does, you're in trouble.iTunes is Web 2.0ish in this sense. Finally you can buy individual

songs instead of having to buy whole albums. The recording industry

hated the idea and resisted it as long as possible. But it was

obvious what users wanted, so Apple flew under the labels.

[4]

Though really it might be better to describe iTunes as Web 1.5.

Web 2.0 applied to music would probably mean individual bands giving

away DRMless songs for free.The ultimate way to be nice to users is to give them something for

free that competitors charge for. During the 90s a lot of people

probably thought we'd have some working system for micropayments

by now. In fact things have gone in the other direction. The most

successful sites are the ones that figure out new ways to give stuff

away for free. Craigslist has largely destroyed the classified ad

sites of the 90s, and OkCupid looks likely to do the same to the

previous generation of dating sites.Serving web pages is very, very cheap. If you can make even a

fraction of a cent per page view, you can make a profit. And

technology for targeting ads continues to improve. I wouldn't be

surprised if ten years from now eBay had been supplanted by an

ad-supported freeBay (or, more likely, gBay).Odd as it might sound, we tell startups that they should try to

make as little money as possible. If you can figure out a way to

turn a billion dollar industry into a fifty million dollar industry,

so much the better, if all fifty million go to you. Though indeed,

making things cheaper often turns out to generate more money in the

end, just as automating things often turns out to generate more

jobs.The ultimate target is Microsoft. What a bang that balloon is going

to make when someone pops it by offering a free web-based alternative

to MS Office.

[5]

Who will? Google? They seem to be taking their

time. I suspect the pin will be wielded by a couple of 20 year old

hackers who are too naive to be intimidated by the idea. (How hard

can it be?)The Common ThreadAjax, democracy, and not dissing users. What do they all have in

common? I didn't realize they had anything in common till recently,

which is one of the reasons I disliked the term "Web 2.0" so much.

It seemed that it was being used as a label for whatever happened

to be new—that it didn't predict anything.But there is a common thread. Web 2.0 means using the web the way

it's meant to be used. The "trends" we're seeing now are simply

the inherent nature of the web emerging from under the broken models

that got imposed on it during the Bubble.I realized this when I read an interview with

Joe Kraus, the co-founder of Excite.

[6]

Excite really never got the business model right at all. We fell

into the classic problem of how when a new medium comes out it

adopts the practices, the content, the business models of the old

medium—which fails, and then the more appropriate models get

figured out.

It may have seemed as if not much was happening during the years

after the Bubble burst. But in retrospect, something was happening:

the web was finding its natural angle of repose. The democracy

component, for example—that's not an innovation, in the sense of

something someone made happen. That's what the web naturally tends

to produce.Ditto for the idea of delivering desktop-like applications over the

web. That idea is almost as old as the web. But the first time

around it was co-opted by Sun, and we got Java applets. Java has

since been remade into a generic replacement for C++, but in 1996

the story about Java was that it represented a new model of software.

Instead of desktop applications, you'd run Java "applets" delivered

from a server.This plan collapsed under its own weight. Microsoft helped kill it,

but it would have died anyway. There was no uptake among hackers.

When you find PR firms promoting

something as the next development platform, you can be sure it's

not. If it were, you wouldn't need PR firms to tell you, because

hackers would already be writing stuff on top of it, the way sites

like Busmonster used Google Maps as a

platform before Google even meant it to be one.The proof that Ajax is the next hot platform is that thousands of

hackers have spontaneously started building things on top

of it. Mikey likes it.There's another thing all three components of Web 2.0 have in common.

Here's a clue. Suppose you approached investors with the following

idea for a Web 2.0 startup:

Sites like del.icio.us and flickr allow users to "tag" content

with descriptive tokens. But there is also huge source of

implicit tags that they ignore: the text within web links.

Moreover, these links represent a social network connecting the

individuals and organizations who created the pages, and by using

graph theory we can compute from this network an estimate of the

reputation of each member. We plan to mine the web for these

implicit tags, and use them together with the reputation hierarchy

they embody to enhance web searches.

How long do you think it would take them on average to realize that

it was a description of Google?Google was a pioneer in all three components of Web 2.0: their core

business sounds crushingly hip when described in Web 2.0 terms,

"Don't maltreat users" is a subset of "Don't be evil," and of course

Google set off the whole Ajax boom with Google Maps.Web 2.0 means using the web as it was meant to be used, and Google

does. That's their secret. They're sailing with the wind, instead of sitting

becalmed praying for a business model, like the print media, or

trying to tack upwind by suing their customers, like Microsoft and

the record labels.

[7]Google doesn't try to force things to happen their way. They try

to figure out what's going to happen, and arrange to be standing

there when it does. That's the way to approach technology—and

as business includes an ever larger technological component, the

right way to do business.The fact that Google is a "Web 2.0" company shows that, while

meaningful, the term is also rather bogus. It's like the word

"allopathic." It just means doing things right, and it's a bad

sign when you have a special word for that.

Notes[1]

From the conference

site, June 2004: "While the first wave of the Web was closely

tied to the browser, the second wave extends applications across

the web and enables a new generation of services and business

opportunities." To the extent this means anything, it seems to be

about

web-based applications.[2]

Disclosure: Reddit was funded by

Y Combinator. But although

I started using it out of loyalty to the home team, I've become a

genuine addict. While we're at it, I'm also an investor in

!MSFT, having sold all my shares earlier this year.[3]

I'm not against editing. I spend more time editing than

writing, and I have a group of picky friends who proofread almost

everything I write. What I dislike is editing done after the fact

by someone else.[4]

Obvious is an understatement. Users had been climbing in through

the window for years before Apple finally moved the door.[5]

Hint: the way to create a web-based alternative to Office may

not be to write every component yourself, but to establish a protocol

for web-based apps to share a virtual home directory spread across

multiple servers. Or it may be to write it all yourself.[6]

In Jessica Livingston's

Founders at

Work.[7]

Microsoft didn't sue their customers directly, but they seem

to have done all they could to help SCO sue them.Thanks to Trevor Blackwell, Sarah Harlin, Jessica Livingston, Peter

Norvig, Aaron Swartz, and Jeff Weiner for reading drafts of this, and to the

guys at O'Reilly and Adaptive Path for answering my questions.Interview About Web 2.0Spanish TranslationGerman TranslationRussian TranslationJapanese Translation

If you liked this, you may also like

Hackers & Painters.

How to Fund a Startup

Want to start a startup? Get funded by

Y Combinator.

November 2005

Venture funding works like gears. A typical startup goes through

several rounds of funding, and at each round you want to take just

enough money to reach the speed where you can shift into the next

gear.Few startups get it quite right. Many are underfunded. A few are

overfunded, which is like trying to start driving in third gear.I think it would help founders to understand funding better—not

just the mechanics of it, but what investors are thinking. I was

surprised recently when I realized that all the worst problems we

faced in our startup were due not to competitors, but investors.

Dealing with competitors was easy by comparison.I don't mean to suggest that our investors were nothing but a drag

on us. They were helpful in negotiating deals, for example. I

mean more that conflicts with investors are particularly nasty.

Competitors punch you in the jaw, but investors have you by the

balls.Apparently our situation was not unusual. And if trouble with

investors is one of the biggest threats to a startup, managing them

is one of the most important skills founders need to learn.Let's start by talking about the five sources of startup funding.

Then we'll trace the life of a hypothetical (very fortunate) startup

as it shifts gears through successive rounds.Friends and FamilyA lot of startups get their first funding from friends and family.

Excite did, for example: after the founders graduated from college,

they borrowed $15,000 from their parents to start a company. With

the help of some part-time jobs they made it last 18 months.If your friends or family happen to be rich, the line blurs between

them and angel investors. At Viaweb we got our first $10,000 of

seed money from our friend Julian, but he was sufficiently rich

that it's hard to say whether he should be classified as a friend

or angel. He was also a lawyer, which was great, because it meant

we didn't have to pay legal bills out of that initial small sum.The advantage of raising money from friends and family is that

they're easy to find. You already know them. There are three main

disadvantages: you mix together your business and personal life;

they will probably not be as well connected as angels or venture

firms; and they may not be accredited investors, which could

complicate your life later.The SEC defines an "accredited investor" as someone with over a

million dollars in liquid assets or an income of over $200,000 a

year. The regulatory burden is much lower if a company's shareholders

are all accredited investors. Once you take money from the general

public you're more restricted in what you can do.

[1]A startup's life will be more complicated, legally, if any of the

investors aren't accredited. In an IPO, it might not merely add

expense, but change the outcome. A lawyer I asked about it said:

When the company goes public, the SEC will carefully study all

prior issuances of stock by the company and demand that it take

immediate action to cure any past violations of securities laws.

Those remedial actions can delay, stall or even kill the IPO.

Of course the odds of any given startup doing an IPO are small.

But not as small as they might seem. A lot of startups that end up

going public didn't seem likely to at first. (Who could have guessed

that the company Wozniak and Jobs started in their spare time selling

plans for microcomputers would yield one of the biggest IPOs of the

decade?) Much of the value of a startup consists of that tiny

probability multiplied by the huge outcome.It wasn't because they weren't accredited investors that I didn't

ask my parents for seed money, though. When we were starting Viaweb,

I didn't know about the concept of an accredited investor, and

didn't stop to think about the value of investors' connections.

The reason I didn't take money from my parents was that I didn't

want them to lose it.ConsultingAnother way to fund a startup is to get a job. The best sort of

job is a consulting project in which you can build whatever software

you wanted to sell as a startup. Then you can gradually transform

yourself from a consulting company into a product company, and have

your clients pay your development expenses.This is a good plan for someone with kids, because it takes most

of the risk out of starting a startup. There never has to be a

time when you have no revenues. Risk and reward are usually

proportionate, however: you should expect a plan that cuts the risk

of starting a startup also to cut the average return. In this case,

you trade decreased financial risk for increased risk that your

company won't succeed as a startup.But isn't the consulting company itself a startup? No, not generally.

A company has to be more than small and newly founded to be a

startup. There are millions of small businesses in America, but

only a few thousand are startups. To be a startup, a company has

to be a product business, not a service business. By which I mean

not that it has to make something physical, but that it has to have

one thing it sells to many people, rather than doing custom work

for individual clients. Custom work doesn't scale. To be a startup

you need to be the band that sells a million copies of a song, not

the band that makes money by playing at individual weddings and bar

mitzvahs.The trouble with consulting is that clients have an awkward habit

of calling you on the phone. Most startups operate close to the

margin of failure, and the distraction of having to deal with clients

could be enough to put you over the edge. Especially if you have

competitors who get to work full time on just being a startup.So you have to be very disciplined if you take the consulting route.

You have to work actively to prevent your company growing into a

"weed tree," dependent on this source of easy but low-margin money.

[2]Indeed, the biggest danger of consulting may be that it gives you

an excuse for failure. In a startup, as in grad school, a lot of

what ends up driving you are the expectations of your family and

friends. Once you start a startup and tell everyone that's what

you're doing, you're now on a path labelled "get rich or bust." You

now have to get rich, or you've failed.Fear of failure is an extraordinarily powerful force. Usually it

prevents people from starting things, but once you publish some

definite ambition, it switches directions and starts working in

your favor. I think it's a pretty clever piece of jiujitsu to set

this irresistible force against the slightly less immovable object

of becoming rich. You won't have it driving you if your stated

ambition is merely to start a consulting company that you will one

day morph into a startup.An advantage of consulting, as a way to develop a product, is that

you know you're making something at least one customer wants. But

if you have what it takes to start a startup you should have

sufficient vision not to need this crutch.Angel InvestorsAngels are individual rich people. The word was first used

for backers of Broadway plays, but now applies to individual investors

generally. Angels who've made money in technology are preferable,

for two reasons: they understand your situation, and they're a

source of contacts and advice.The contacts and advice can be more important than the money. When

del.icio.us took money from investors, they took money from, among

others, Tim O'Reilly. The amount he put in was small compared to

the VCs who led the round, but Tim is a smart and influential guy

and it's good to have him on your side.You can do whatever you want with money from consulting or friends

and family. With angels we're now talking about venture funding

proper, so it's time to introduce the concept of exit strategy.

Younger would-be founders are often surprised that investors expect

them either to sell the company or go public. The reason is that

investors need to get their capital back. They'll only consider

companies that have an exit strategy—meaning companies that could

get bought or go public.This is not as selfish as it sounds. There are few large, private

technology companies. Those that don't fail all seem to get bought

or go public. The reason is that employees are investors too—of

their time—and they want just as much to be able to cash out. If

your competitors offer employees stock options that might make them

rich, while you make it clear you plan to stay private, your

competitors will get the best people. So the principle of an "exit"

is not just something forced on startups by investors, but part of

what it means to be a startup.Another concept we need to introduce now is valuation. When someone

buys shares in a company, that implicitly establishes a value for

it. If someone pays $20,000 for 10% of a company, the company is

in theory worth $200,000. I say "in theory" because in early stage

investing, valuations are voodoo. As a company gets more established,

its valuation gets closer to an actual market value. But in a newly

founded startup, the valuation number is just an artifact of the

respective contributions of everyone involved.Startups often "pay" investors who will help the company in some

way by letting them invest at low valuations. If I had a startup

and Steve Jobs wanted to invest in it, I'd give him the stock for

$10, just to be able to brag that he was an investor. Unfortunately,

it's impractical (if not illegal) to adjust the valuation of the

company up and down for each investor. Startups' valuations are

supposed to rise over time. So if you're going to sell cheap stock

to eminent angels, do it early, when it's natural for the company

to have a low valuation.Some angel investors join together in syndicates. Any city where

people start startups will have one or more of them. In Boston the

biggest is the Common

Angels. In the Bay Area it's the Band

of Angels. You can find groups near you through the Angel Capital Association.

[3]

However, most angel investors don't belong to these groups. In

fact, the more prominent the angel, the less likely they are to

belong to a group.Some angel groups charge you money to pitch your idea to them.

Needless to say, you should never do this.One of the dangers of taking investment from individual angels,

rather than through an angel group or investment firm, is that they

have less reputation to protect. A big-name VC firm will not screw

you too outrageously, because other founders would avoid them if

word got out. With individual angels you don't have this protection,

as we found to our dismay in our own startup. In many startups'

lives there comes a point when you're at the investors'

mercy—when you're out of money and the only place to get more is your

existing investors. When we got into such a scrape, our investors

took advantage of it in a way that a name-brand VC probably wouldn't

have.Angels have a corresponding advantage, however: they're also not

bound by all the rules that VC firms are. And so they can, for

example, allow founders to cash out partially in a funding round,

by selling some of their stock directly to the investors. I think

this will become more common; the average founder is eager to do

it, and selling, say, half a million dollars worth of stock will

not, as VCs fear, cause most founders to be any less committed to

the business.The same angels who tried to screw us also let us do this, and so

on balance I'm grateful rather than angry. (As in families, relations

between founders and investors can be complicated.)The best way to find angel investors is through personal introductions.

You could try to cold-call angel groups near you, but angels, like

VCs, will pay more attention to deals recommended by someone they

respect.Deal terms with angels vary a lot. There are no generally accepted

standards. Sometimes angels' deal terms are as fearsome as VCs'.

Other angels, particularly in the earliest stages, will invest based

on a two-page agreement.Angels who only invest occasionally may not themselves know what

terms they want. They just want to invest in this startup. What

kind of anti-dilution protection do they want? Hell if they know.

In these situations, the deal terms tend to be random: the angel

asks his lawyer to create a vanilla agreement, and the terms end

up being whatever the lawyer considers vanilla. Which in practice

usually means, whatever existing agreement he finds lying around

his firm. (Few legal documents are created from scratch.)These heaps o' boilerplate are a problem for small startups, because

they tend to grow into the union of all preceding documents. I

know of one startup that got from an angel investor what amounted

to a five hundred pound handshake: after deciding to invest, the

angel presented them with a 70-page agreement. The startup didn't

have enough money to pay a lawyer even to read it, let alone negotiate

the terms, so the deal fell through.One solution to this problem would be to have the startup's lawyer

produce the agreement, instead of the angel's. Some angels might

balk at this, but others would probably welcome it.Inexperienced angels often get cold feet when the time comes to

write that big check. In our startup, one of the two angels in the

initial round took months to pay us, and only did after repeated

nagging from our lawyer, who was also, fortunately, his lawyer.It's obvious why investors delay. Investing in startups is risky!

When a company is only two months old, every day you wait

gives you 1.7% more data about their trajectory. But the investor

is already being compensated for that risk in the low price of the

stock, so it is unfair to delay.Fair or not, investors do it if you let them. Even VCs do it. And

funding delays are a big distraction for founders, who ought to be

working on their company, not worrying about investors. What's a

startup to do? With both investors and acquirers, the only leverage

you have is competition. If an investor knows you have other

investors lined up, he'll be a lot more eager to close-- and not

just because he'll worry about losing the deal, but because if other

investors are interested, you must be worth investing in. It's the

same with acquisitions. No one wants to buy you till someone else

wants to buy you, and then everyone wants to buy you.The key to closing deals is never to stop pursuing alternatives.

When an investor says he wants to invest in you, or an acquirer

says they want to buy you, don't believe it till you get the

check. Your natural tendency when an investor says yes will

be to relax and go back to writing code. Alas, you can't; you have

to keep looking for more investors, if only to get this one to act.

[4]Seed Funding FirmsSeed firms are like angels in that they invest relatively small

amounts at early stages, but like VCs in that they're companies

that do it as a business, rather than individuals making occasional

investments on the side.Till now, nearly all seed firms have been so-called "incubators,"

so Y Combinator gets called

one too, though the only thing we have in common is that we invest

in the earliest phase.According to the National Association of Business Incubators, there

are about 800 incubators in the US. This is an astounding number,

because I know the founders of a lot of startups, and I can't think

of one that began in an incubator.What is an incubator? I'm not sure myself. The defining quality

seems to be that you work in their space. That's where the name

"incubator" comes from. They seem to vary a great deal in other

respects. At one extreme is the sort of pork-barrel project where

a town gets money from the state government to renovate a vacant

building as a "high-tech incubator," as if it were merely lack of

the right sort of office space that had till now prevented the town

from becoming a

startup hub.

At the other extreme are places like

Idealab, which generates ideas for new startups internally and hires

people to work for them.The classic Bubble incubators, most of which now seem to be dead,

were like VC firms except that they took a much bigger role in the

startups they funded. In addition to working in their space, you

were supposed to use their office staff, lawyers, accountants, and

so on.Whereas incubators tend (or tended) to exert more control than VCs,

Y Combinator exerts less.

And we think it's better if startups operate out of their own

premises, however crappy, than the offices of their investors. So

it's annoying that we keep getting called an "incubator," but perhaps

inevitable, because there's only one of us so far and no word yet

for what we are. If we have to be called something, the obvious

name would be "excubator." (The name is more excusable if one

considers it as meaning that we enable people to escape cubicles.)Because seed firms are companies rather than individual people,

reaching them is easier than reaching angels. Just go to their web

site and send them an email. The importance of personal introductions

varies, but is less than with angels or VCs.The fact that seed firms are companies also means the investment

process is more standardized. (This is generally true with angel

groups too.) Seed firms will probably have set deal terms they use

for every startup they fund. The fact that the deal terms are

standard doesn't mean they're favorable to you, but if other startups

have signed the same agreements and things went well for them, it's

a sign the terms are reasonable.Seed firms differ from angels and VCs in that they invest exclusively

in the earliest phases—often when the company is still just an

idea. Angels and even VC firms occasionally do this, but they also

invest at later stages.The problems are different in the early stages. For example, in

the first couple months a startup may completely redefine their idea. So seed investors usually care less

about the idea than the people. This is true of all venture funding,

but especially so in the seed stage.Like VCs, one of the advantages of seed firms is the advice they

offer. But because seed firms operate in an earlier phase, they

need to offer different kinds of advice. For example, a seed firm

should be able to give advice about how to approach VCs, which VCs

obviously don't need to do; whereas VCs should be able to give

advice about how to hire an "executive team," which is not an issue

in the seed stage.In the earliest phases, a lot of the problems are technical, so

seed firms should be able to help with technical as well as business

problems.Seed firms and angel investors generally want to invest in the

initial phases of a startup, then hand them off to VC firms for the

next round. Occasionally startups go from seed funding direct to

acquisition, however, and I expect this to become increasingly

common.Google has been aggressively pursuing this route, and now Yahoo is too. Both

now compete directly with VCs. And this is a smart move. Why wait

for further funding rounds to jack up a startup's price? When a

startup reaches the point where VCs have enough information to

invest in it, the acquirer should have enough information to buy

it. More information, in fact; with their technical depth, the

acquirers should be better at picking winners than VCs.Venture Capital FundsVC firms are like seed firms in that they're actual companies, but

they invest other people's money, and much larger amounts of it.

VC investments average several million dollars. So they tend to

come later in the life of a startup, are harder to get, and come

with tougher terms.The word "venture capitalist" is sometimes used loosely for any

venture investor, but there is a sharp difference between VCs and

other investors: VC firms are organized as funds, much like

hedge funds or mutual funds. The fund managers, who are called

"general partners," get about 2% of the fund annually as a management

fee, plus about 20% of the fund's gains.There is a very sharp dropoff in performance among VC firms, because

in the VC business both success and failure are self-perpetuating.

When an investment scores spectacularly, as Google did for Kleiner

and Sequoia, it generates a lot of good publicity for the VCs. And

many founders prefer to take money from successful VC firms, because

of the legitimacy it confers. Hence a vicious (for the losers)

cycle: VC firms that have been doing badly will only get the deals

the bigger fish have rejected, causing them to continue to do badly.As a result, of the thousand or so VC funds in the US now, only

about 50 are likely to make money, and it is very hard for a new

fund to break into this group.In a sense, the lower-tier VC firms are a bargain for founders.

They may not be quite as smart or as well connected as the big-name

firms, but they are much hungrier for deals. This means you should

be able to get better terms from them.Better how? The most obvious is valuation: they'll take less of

your company. But as well as money, there's power. I think founders

will increasingly be able to stay on as CEO, and on terms that will

make it fairly hard to fire them later.The most dramatic change, I predict,

is that VCs will allow founders to

cash out partially by selling

some of their stock direct to the VC firm. VCs have traditionally

resisted letting founders get anything before the ultimate "liquidity

event." But they're also desperate for deals. And since I know

from my own experience that the rule against buying stock from

founders is a stupid one, this is a natural place for things to

give as venture funding becomes more and more a seller's market.The disadvantage of taking money from less known firms is that

people will assume, correctly or not, that you were turned down by

the more exalted ones. But, like where you went to college, the

name of your VC stops mattering once you have some performance to

measure. So the more confident you are, the less you need a

brand-name VC. We funded Viaweb entirely with angel money; it never

occurred to us that the backing of a well known VC firm would make

us seem more impressive.

[5]Another danger of less known firms is that, like angels, they have

less reputation to protect. I suspect it's the lower-tier firms

that are responsible for most of the tricks that have given VCs

such a bad reputation among hackers. They are doubly hosed: the

general partners themselves are less able, and yet they have harder

problems to solve, because the top VCs skim off all the best deals,

leaving the lower-tier firms exactly the startups that are likely

to blow up.For example, lower-tier firms are much more likely to pretend to

want to do a deal with you just to lock you up while they decide

if they really want to. One experienced CFO said:

The better ones usually will not give a term sheet unless they

really want to do a deal. The second or third tier firms have a

much higher break rate—it could be as high as 50%.

It's obvious why: the lower-tier firms' biggest fear, when chance

throws them a bone, is that one of the big dogs will notice and

take it away. The big dogs don't have to worry about that.Falling victim to this trick could really hurt you. As one

VC told me:

If you were talking to four VCs, told three of them that you

accepted a term sheet, and then have to call them back to tell

them you were just kidding, you are absolutely damaged goods.

Here's a partial solution: when a VC offers you a term sheet, ask

how many of their last 10 term sheets turned into deals. This will

at least force them to lie outright if they want to mislead you.Not all the people who work at VC firms are partners. Most firms

also have a handful of junior employees called something like

associates or analysts. If you get a call from a VC

firm, go to their web site and check whether the person you talked

to is a partner. Odds are it will be a junior person; they scour

the web looking for startups their bosses could invest in. The

junior people will tend to seem very positive about your company.

They're not pretending; they want to believe you're a hot

prospect, because it would be a huge coup for them if their firm

invested in a company they discovered. Don't be misled by this

optimism. It's the partners who decide, and they view things with

a colder eye.Because VCs invest large amounts, the money comes with more

restrictions. Most only come into effect if the company gets into

trouble. For example, VCs generally write it into the deal that

in any sale, they get their investment back first. So if the company

gets sold at a low price, the founders could get nothing. Some VCs

now require that in any sale they get 4x their investment back

before the common stock holders (that is, you) get anything, but

this is an abuse that should be resisted.Another difference with large investments is that the founders are

usually required to accept "vesting"—to surrender their stock and

earn it back over the next 4-5 years. VCs don't want to invest

millions in a company the founders could just walk away from.

Financially, vesting has little effect, but in some situations it

could mean founders will have less power. If VCs got de facto

control of the company and fired one of the founders, he'd lose any

unvested stock unless there was specific protection against this.

So vesting would in that situation force founders to toe the line.The most noticeable change when a startup takes serious funding is

that the founders will no longer have complete control. Ten years

ago VCs used to insist that founders step down as CEO and hand the

job over to a business guy they supplied. This is less the rule

now, partly because the disasters of the Bubble showed that generic

business guys don't make such great CEOs.But while founders will increasingly be able to stay on as CEO,

they'll have to cede some power, because the board of directors

will become more powerful. In the seed stage, the board is generally

a formality; if you want to talk to the other board members, you

just yell into the next room. This stops with VC-scale money. In

a typical VC funding deal, the board of directors might be composed

of two VCs, two founders, and one outside person acceptable to both.

The board will have ultimate power, which means the founders now

have to convince instead of commanding.This is not as bad as it sounds, however. Bill Gates is in the

same position; he doesn't have majority control of Microsoft; in

principle he also has to convince instead of commanding. And yet

he seems pretty commanding, doesn't he? As long as things are going

smoothly, boards don't interfere much. The danger comes when there's

a bump in the road, as happened to Steve Jobs at Apple.Like angels, VCs prefer to invest in deals that come to them through

people they know. So while nearly all VC funds have some address

you can send your business plan to, VCs privately admit the chance

of getting funding by this route is near zero. One recently told

me that he did not know a single startup that got funded this way.I suspect VCs accept business plans "over the transom" more as a

way to keep tabs on industry trends than as a source of deals. In

fact, I would strongly advise against mailing your business plan

randomly to VCs, because they treat this as evidence of laziness.

Do the extra work of getting personal introductions. As one VC put

it:

I'm not hard to find. I know a lot of people. If you can't find

some way to reach me, how are you going to create a successful

company?

One of the most difficult problems for startup founders is deciding

when to approach VCs. You really only get one chance, because they

rely heavily on first impressions. And you can't approach some and

save others for later, because (a) they ask who else you've talked

to and when and (b) they talk among themselves. If you're talking

to one VC and he finds out that you were rejected by another several

months ago, you'll definitely seem shopworn.So when do you approach VCs? When you can convince them. If the

founders have impressive resumes and the idea isn't hard to understand,

you could approach VCs quite early. Whereas if the founders are

unknown and the idea is very novel, you might have to launch the

thing and show that users loved it before VCs would be convinced.If several VCs are interested in you, they will sometimes be willing

to split the deal between them. They're more likely to do this if

they're close in the VC pecking order. Such deals may be a net win

for founders, because you get multiple VCs interested in your

success, and you can ask each for advice about the other. One

founder I know wrote:

Two-firm deals are great. It costs you a little more equity, but

being able to play the two firms off each other (as well as ask

one if the other is being out of line) is invaluable.

When you do negotiate with VCs, remember that they've done this a

lot more than you have. They've invested in dozens of startups,

whereas this is probably the first you've founded. But don't let

them or the situation intimidate you. The average founder is smarter

than the average VC. So just do what you'd do in any complex,

unfamiliar situation: proceed deliberately, and question anything

that seems odd.It is, unfortunately, common for VCs to put terms in an agreement

whose consequences surprise founders later, and also common for VCs

to defend things they do by saying that they're standard in the

industry. Standard, schmandard; the whole industry is only a few

decades old, and rapidly evolving. The concept of "standard" is a

useful one when you're operating on a small scale (Y Combinator

uses identical terms for every deal because for tiny seed-stage

investments it's not worth the overhead of negotiating individual

deals), but it doesn't apply at the VC level. On that scale, every

negotiation is unique.Most successful startups get money from more than one of the preceding

five sources.

[6]

And, confusingly, the names of funding sources

also tend to be used as the names of different rounds. The best

way to explain how it all works is to follow the case of a hypothetical

startup.Stage 1: Seed RoundOur startup begins when a group of three friends have an idea--

either an idea for something they might build, or simply the idea

"let's start a company." Presumably they already have some source

of food and shelter. But if you have food and shelter, you probably

also have something you're supposed to be working on: either

classwork, or a job. So if you want to work full-time on a startup,

your money situation will probably change too.A lot of startup founders say they started the company without any

idea of what they planned to do. This is actually less common than

it seems: many have to claim they thought of the idea after quitting

because otherwise their former employer would own it.The three friends decide to take the leap. Since most startups are

in competitive businesses, you not only want to work full-time on

them, but more than full-time. So some or all of the friends quit

their jobs or leave school. (Some of the founders in a startup can

stay in grad school, but at least one has to make the company his

full-time job.)They're going to run the company out of one of their apartments at

first, and since they don't have any users they don't have to pay

much for infrastructure. Their main expenses are setting up the

company, which costs a couple thousand dollars in legal work and

registration fees, and the living expenses of the founders.The phrase "seed investment" covers a broad range. To some VC firms

it means $500,000, but to most startups it means several months'

living expenses. We'll suppose our group of friends start with

$15,000 from their friend's rich uncle, who they give 5% of the

company in return. There's only common stock at this stage. They

leave 20% as an options pool for later employees (but they set

things up so that they can issue this stock to themselves if they

get bought early and most is still unissued), and the three founders

each get 25%.By living really cheaply they think they can make the remaining

money last five months. When you have five months' runway left,

how soon do you need to start looking for your next round? Answer:

immediately. It takes time to find investors, and time (always

more than you expect) for the deal to close even after they say

yes. So if our group of founders know what they're doing they'll

start sniffing around for angel investors right away. But of course

their main job is to build version 1 of their software.The friends might have liked to have more money in this first phase,

but being slightly underfunded teaches them an important lesson.

For a startup, cheapness is power. The lower your costs, the more

options you have—not just at this stage, but at every point till

you're profitable. When you have a high "burn rate," you're always

under time pressure, which means (a) you don't have time for your

ideas to evolve, and (b) you're often forced to take deals you don't

like.Every startup's rule should be: spend little, and work fast.After ten weeks' work the three friends have built a prototype that

gives one a taste of what their product will do. It's not what

they originally set out to do—in the process of writing it, they

had some new ideas. And it only does a fraction of what the finished

product will do, but that fraction includes stuff that no one else

has done before.They've also written at least a skeleton business plan, addressing

the five fundamental questions: what they're going to do, why users

need it, how large the market is, how they'll make money, and who

the competitors are and why this company is going to beat them.

(That last has to be more specific than "they suck" or "we'll work

really hard.")If you have to choose between spending time on the demo or the

business plan, spend most on the demo. Software is not only more

convincing, but a better way to explore ideas.Stage 2: Angel RoundWhile writing the prototype, the group has been traversing their

network of friends in search of angel investors. They find some

just as the prototype is demoable. When they demo it, one of the

angels is willing to invest. Now the group is looking for more

money: they want enough to last for a year, and maybe to hire a

couple friends. So they're going to raise $200,000.The angel agrees to invest at a pre-money valuation of $1 million.

The company issues $200,000 worth of new shares to the angel; if

there were 1000 shares before the deal, this means 200 additional

shares. The angel now owns 200/1200 shares, or a sixth of the

company, and all the previous shareholders' percentage ownership

is diluted by a sixth. After the deal, the capitalization table

looks like this:

shareholder shares percent

-------------------------------

angel 200 16.7

uncle 50 4.2

each founder 250 20.8

option pool 200 16.7

---- -----

total 1200 100

To keep things simple, I had the angel do a straight cash for stock

deal. In reality the angel might be more likely to make the

investment in the form of a convertible loan. A convertible loan

is a loan that can be converted into stock later; it works out the

same as a stock purchase in the end, but gives the angel more

protection against being squashed by VCs in future rounds.Who pays the legal bills for this deal? The startup, remember,

only has a couple thousand left. In practice this turns out to be

a sticky problem that usually gets solved in some improvised way.

Maybe the startup can find lawyers who will do it cheaply in the

hope of future work if the startup succeeds. Maybe someone has a

lawyer friend. Maybe the angel pays for his lawyer to represent

both sides. (Make sure if you take the latter route that the lawyer

is representing you rather than merely advising you, or his

only duty is to the investor.)An angel investing $200k would probably expect a seat on the board

of directors. He might also want preferred stock, meaning a special

class of stock that has some additional rights over the common stock

everyone else has. Typically these rights include vetoes over major

strategic decisions, protection against being diluted in future

rounds, and the right to get one's investment back first if the

company is sold.Some investors might expect the founders to accept vesting for a

sum this size, and others wouldn't. VCs are more likely to require

vesting than angels. At Viaweb we managed to raise $2.5 million

from angels without ever accepting vesting, largely because we were

so inexperienced that we were appalled at the idea. In practice

this turned out to be good, because it made us harder to push around.Our experience was unusual; vesting is the norm for amounts that

size. Y Combinator doesn't require vesting, because (a) we invest

such small amounts, and (b) we think it's unnecessary, and that the

hope of getting rich is enough motivation to keep founders at work.

But maybe if we were investing millions we would think differently.I should add that vesting is also a way for founders to protect

themselves against one another. It solves the problem of what to

do if one of the founders quits. So some founders impose it on

themselves when they start the company.The angel deal takes two weeks to close, so we are now three months

into the life of the company.The point after you get the first big chunk of angel money will

usually be the happiest phase in a startup's life. It's a lot like

being a postdoc: you have no immediate financial worries, and few

responsibilities. You get to work on juicy kinds of work, like

designing software. You don't have to spend time on bureaucratic

stuff, because you haven't hired any bureaucrats yet. Enjoy it

while it lasts, and get as much done as you can, because you will

never again be so productive.With an apparently inexhaustible sum of money sitting safely in the

bank, the founders happily set to work turning their prototype into

something they can release. They hire one of their friends—at

first just as a consultant, so they can try him out—and then a

month later as employee #1. They pay him the smallest salary he can

live on, plus 3% of the company in restricted stock, vesting over

four years. (So after this the option pool is down to 13.7%).

[7]

They also spend a little money on a freelance graphic designer.How much stock do you give early employees? That varies so much

that there's no conventional number. If you get someone really

good, really early, it might be wise to give him as much stock as

the founders. The one universal rule is that the amount of stock

an employee gets decreases polynomially with the age of the company.

In other words, you get rich as a power of how early you were. So

if some friends want you to come work for their startup, don't wait

several months before deciding.A month later, at the end of month four, our group of founders have

something they can launch. Gradually through word of mouth they

start to get users. Seeing the system in use by real users—people

they don't know—gives them lots of new ideas. Also they find

they now worry obsessively about the status of their server. (How

relaxing founders' lives must have been when startups wrote VisiCalc.)By the end of month six, the system is starting to have a solid

core of features, and a small but devoted following. People start

to write about it, and the founders are starting to feel like experts

in their field.We'll assume that their startup is one that could put millions more

to use. Perhaps they need to spend a lot on marketing, or build

some kind of expensive infrastructure, or hire highly paid salesmen.

So they decide to start talking to VCs. They get introductions to

VCs from various sources: their angel investor connects them with

a couple; they meet a few at conferences; a couple VCs call them

after reading about them.Step 3: Series A RoundArmed with their now somewhat fleshed-out business plan and able

to demo a real, working system, the founders visit the VCs they

have introductions to. They find the VCs intimidating and inscrutable.

They all ask the same question: who else have you pitched to? (VCs

are like high school girls: they're acutely aware of their position

in the VC pecking order, and their interest in a company is a

function of the interest other VCs show in it.)One of the VC firms says they want to invest and offers the founders

a term sheet. A term sheet is a summary of what the deal terms

will be when and if they do a deal; lawyers will fill in the details

later. By accepting the term sheet, the startup agrees to turn

away other VCs for some set amount of time while this firm does the

"due diligence" required for the deal. Due diligence is the corporate

equivalent of a background check: the purpose is to uncover any

hidden bombs that might sink the company later, like serious design

flaws in the product, pending lawsuits against the company,

intellectual property issues, and so on. VCs' legal and financial

due diligence is pretty thorough, but the technical due diligence

is generally a joke.

[8]The due diligence discloses no ticking bombs, and six weeks later

they go ahead with the deal. Here are the terms: a $2 million

investment at a pre-money valuation of $4 million, meaning that

after the deal closes the VCs will own a third of the company (2 /

(4 + 2)). The VCs also insist that prior to the deal the option

pool be enlarged by an additional hundred shares. So the total

number of new shares issued is 750, and the cap table becomes:

shareholder shares percent

-------------------------------

VCs 650 33.3

angel 200 10.3

uncle 50 2.6

each founder 250 12.8

employee 36\* 1.8 \*unvested

option pool 264 13.5

---- -----

total 1950 100

This picture is unrealistic in several respects. For example, while

the percentages might end up looking like this, it's unlikely that

the VCs would keep the existing numbers of shares. In fact, every

bit of the startup's paperwork would probably be replaced, as if

the company were being founded anew. Also, the money might come

in several tranches, the later ones subject to various

conditions—though this is apparently more common in deals with lower-tier VCs

(whose lot in life is to fund more dubious startups) than with the

top firms.And of course any VCs reading this are probably rolling on the floor

laughing at how my hypothetical VCs let the angel keep his 10.3 of

the company. I admit, this is the Bambi version; in simplifying

the picture, I've also made everyone nicer. In the real world, VCs

regard angels the way a jealous husband feels about his wife's

previous boyfriends. To them the company didn't exist before they

invested in it.

[9]I don't want to give the impression you have to do an angel round

before going to VCs. In this example I stretched things out to

show multiple sources of funding in action. Some startups could go

directly from seed funding to a VC round; several of the companies

we've funded have.The founders are required to vest their shares over four years, and

the board is now reconstituted to consist of two VCs, two founders,

and a fifth person acceptable to both. The angel investor cheerfully

surrenders his board seat.At this point there is nothing new our startup can teach us about

funding—or at least, nothing good.

[10]

The startup will almost

certainly hire more people at this point; those millions must be

put to work, after all. The company may do additional funding

rounds, presumably at higher valuations. They may if they are

extraordinarily fortunate do an IPO, which we should remember is

also in principle a round of funding, regardless of its de facto

purpose. But that, if not beyond the bounds of possibility, is

beyond the scope of this article.Deals Fall ThroughAnyone who's been through a startup will find the preceding portrait

to be missing something: disasters. If there's one thing all

startups have in common, it's that something is always going wrong.

And nowhere more than in matters of funding.For example, our hypothetical startup never spent more than half

of one round before securing the next. That's more ideal than

typical. Many startups—even successful ones—come close to

running out of money at some point. Terrible things happen to

startups when they run out of money, because they're designed for

growth, not adversity.But the most unrealistic thing about the series of deals I've

described is that they all closed. In the startup world, closing

is not what deals do. What deals do is fall through. If you're

starting a startup you would do well to remember that. Birds fly;

fish swim; deals fall through.Why? Partly the reason deals seem to fall through so often is that

you lie to yourself. You want the deal to close, so you start to

believe it will. But even correcting for this, startup deals fall

through alarmingly often—far more often than, say, deals to buy

real estate. The reason is that it's such a risky environment.

People about to fund or acquire a startup are prone to wicked cases

of buyer's remorse. They don't really grasp the risk they're taking

till the deal's about to close. And then they panic. And not just

inexperienced angel investors, but big companies too.So if you're a startup founder wondering why some angel investor

isn't returning your phone calls, you can at least take comfort in

the thought that the same thing is happening to other deals a hundred

times the size.The example of a startup's history that I've presented is like a

skeleton—accurate so far as it goes, but needing to be fleshed

out to be a complete picture. To get a complete picture, just add

in every possible disaster.A frightening prospect? In a way. And yet also in a way encouraging.

The very uncertainty of startups frightens away almost everyone.

People overvalue stability—especially young

people, who ironically need it least. And so in starting a startup,

as in any really bold undertaking, merely deciding to do it gets

you halfway there. On the day of the race, most of the other runners

won't show up.

Notes[1]

The aim of such regulations is to protect widows and orphans

from crooked investment schemes; people with a million dollars in

liquid assets are assumed to be able to protect themselves.

The unintended consequence is that the investments that generate

the highest returns, like hedge funds, are available only to the

rich.[2]

Consulting is where product companies go to die. IBM is the

most famous example. So starting as a consulting company is like

starting out in the grave and trying to work your way up into the

world of the living.[3]

If "near you" doesn't mean the Bay Area, Boston, or Seattle,

consider moving. It's not a coincidence you haven't heard of many

startups from Philadelphia.[4]

Investors are often compared to sheep. And they are like sheep,

but that's a rational response to their situation. Sheep act the

way they do for a reason. If all the other sheep head for a certain

field, it's probably good grazing. And when a wolf appears, is he

going to eat a sheep in the middle of the flock, or one near the

edge?[5]

This was partly confidence, and partly simple ignorance. We

didn't know ourselves which VC firms were the impressive ones. We

thought software was all that mattered. But that turned out to be

the right direction to be naive in: it's much better to overestimate

than underestimate the importance of making a good product.[6]

I've omitted one source: government grants. I don't think

these are even worth thinking about for the average startup.

Governments may mean well when they set up grant programs to encourage

startups, but what they give with one hand they take away with the

other: the process of applying is inevitably so arduous, and the

restrictions on what you can do with the money so burdensome, that

it would be easier to take a job to get the money.

You should be especially suspicious of grants whose purpose is some

kind of social engineering-- e.g. to encourage more startups to be

started in Mississippi. Free money to start a startup in a place

where few succeed is hardly free.Some government agencies run venture funding groups, which make

investments rather than giving grants. For example, the CIA runs

a venture fund called In-Q-Tel that is modelled on private sector

funds and apparently generates good returns. They would probably

be worth approaching—if you don't mind taking money from the CIA.[7]

Options have largely been replaced with restricted stock, which

amounts to the same thing. Instead of earning the right to buy

stock, the employee gets the stock up front, and earns the right

not to have to give it back. The shares set aside for this purpose

are still called the "option pool."[8]

First-rate technical people do not generally hire themselves

out to do due diligence for VCs. So the most difficult

part for startup founders is often responding politely to the inane

questions of the "expert" they send to look you over.[9]

VCs regularly wipe out angels by issuing arbitrary amounts of

new stock. They seem to have a standard piece of casuistry for

this situation: that the angels are no longer working to help the

company, and so don't deserve to keep their stock. This of course

reflects a willful misunderstanding of what investment means; like

any investor, the angel is being compensated for risks he took

earlier. By a similar logic, one could argue that the VCs should

be deprived of their shares when the company goes public.[10]

One new thing the company might encounter is a down

round, or a funding round at valuation lower than the previous

round. Down rounds are bad news; it is generally the common stock

holders who take the hit. Some of the most fearsome provisions in

VC deal terms have to do with down rounds—like "full ratchet

anti-dilution," which is as frightening as it sounds.Founders are tempted to ignore these clauses, because they think

the company will either be a big success or a complete bust. VCs

know otherwise: it's not uncommon for startups to have moments of

adversity before they ultimately succeed. So it's worth negotiating

anti-dilution provisions, even though you don't think you need to,

and VCs will try to make you feel that you're being gratuitously

troublesome.Thanks to Sam Altman, Hutch Fishman, Steve Huffman, Jessica

Livingston, Sesha Pratap, Stan Reiss, Andy Singleton, Zak Stone,

and Aaron Swartz for reading drafts of this.Arabic Translation

The Venture Capital Squeeze

November 2005In the next few years, venture capital funds will find themselves

squeezed from four directions. They're already stuck with a seller's

market, because of the huge amounts they raised at the end of the

Bubble and still haven't invested. This by itself is not the end

of the world. In fact, it's just a more extreme version of the

norm

in the VC business: too much money chasing too few deals.Unfortunately, those few deals now want less and less money, because

it's getting so cheap to start a startup. The four causes: open

source, which makes software free; Moore's law, which makes hardware

geometrically closer to free; the Web, which makes promotion free

if you're good; and better languages, which make development a lot

cheaper.When we started our startup in 1995, the first three were our biggest

expenses. We had to pay $5000 for the Netscape Commerce Server,

the only software that then supported secure http connections. We

paid $3000 for a server with a 90 MHz processor and 32 meg of

memory. And we paid a PR firm about $30,000 to promote our launch.Now you could get all three for nothing. You can get the software

for free; people throw away computers more powerful than our first

server; and if you make something good you can generate ten times

as much traffic by word of mouth online than our first PR firm got

through the print media.And of course another big change for the average startup is that

programming languages have improved-- or rather, the median language has. At most startups ten years

ago, software development meant ten programmers writing code in

C++. Now the same work might be done by one or two using Python

or Ruby.During the Bubble, a lot of people predicted that startups would

outsource their development to India. I think a better model for

the future is David Heinemeier Hansson, who outsourced his development

to a more powerful language instead. A lot of well-known applications

are now, like BaseCamp, written by just one programmer. And one

guy is more than 10x cheaper than ten, because (a) he won't waste

any time in meetings, and (b) since he's probably a founder, he can

pay himself nothing.Because starting a startup is so cheap, venture capitalists now

often want to give startups more money than the startups want to

take. VCs like to invest several million at a time. But as one

VC told me after a startup he funded would only take about half a

million, "I don't know what we're going to do. Maybe we'll just

have to give some of it back." Meaning give some of the fund back

to the institutional investors who supplied it, because it wasn't

going to be possible to invest it all.Into this already bad situation comes the third problem: Sarbanes-Oxley.

Sarbanes-Oxley is a law, passed after the Bubble, that drastically

increases the regulatory burden on public companies. And in addition

to the cost of compliance, which is at least two million dollars a

year, the law introduces frightening legal exposure for corporate

officers. An experienced CFO I know said flatly: "I would not

want to be CFO of a public company now."You might think that responsible corporate governance is an area

where you can't go too far. But you can go too far in any law, and

this remark convinced me that Sarbanes-Oxley must have. This CFO

is both the smartest and the most upstanding money guy I know. If

Sarbanes-Oxley deters people like him from being CFOs of public

companies, that's proof enough that it's broken.Largely because of Sarbanes-Oxley, few startups go public now. For

all practical purposes, succeeding now equals getting bought. Which

means VCs are now in the business of finding promising little 2-3

man startups and pumping them up into companies that cost $100

million to acquire. They didn't mean to be in this business; it's

just what their business has evolved into.Hence the fourth problem: the acquirers have begun to realize they

can buy wholesale. Why should they wait for VCs to make the startups

they want more expensive? Most of what the VCs add, acquirers don't

want anyway. The acquirers already have brand recognition and HR

departments. What they really want is the software and the developers,

and that's what the startup is in the early phase: concentrated

software and developers.Google, typically, seems to have been the first to figure this out.

"Bring us your startups early," said Google's speaker at the Startup School. They're quite

explicit about it: they like to acquire startups at just the point

where they would do a Series A round. (The Series A round is the

first round of real VC funding; it usually happens in the first

year.) It is a brilliant strategy, and one that other big technology

companies will no doubt try to duplicate. Unless they want to have

still more of their lunch eaten by Google.Of course, Google has an advantage in buying startups: a lot of the

people there are rich, or expect to be when their options vest.

Ordinary employees find it very hard to recommend an acquisition;

it's just too annoying to see a bunch of twenty year olds get rich

when you're still working for salary. Even if it's the right thing

for your company to do.The Solution(s)Bad as things look now, there is a way for VCs to save themselves.

They need to do two things, one of which won't surprise them, and

another that will seem an anathema.Let's start with the obvious one: lobby to get Sarbanes-Oxley

loosened. This law was created to prevent future Enrons, not to

destroy the IPO market. Since the IPO market was practically dead

when it passed, few saw what bad effects it would have. But now

that technology has recovered from the last bust, we can see clearly

what a bottleneck Sarbanes-Oxley has become.Startups are fragile plants—seedlings, in fact. These seedlings

are worth protecting, because they grow into the trees of the

economy. Much of the economy's growth is their growth. I think

most politicians realize that. But they don't realize just how

fragile startups are, and how easily they can become collateral

damage of laws meant to fix some other problem.Still more dangerously, when you destroy startups, they make very

little noise. If you step on the toes of the coal industry, you'll

hear about it. But if you inadvertantly squash the startup industry,

all that happens is that the founders of the next Google stay in

grad school instead of starting a company.My second suggestion will seem shocking to VCs: let founders cash

out partially in the Series A round. At the moment, when VCs invest

in a startup, all the stock they get is newly issued and all the

money goes to the company. They could buy some stock directly from

the founders as well.Most VCs have an almost religious rule against doing this. They

don't want founders to get a penny till the company is sold or goes

public. VCs are obsessed with control, and they worry that they'll

have less leverage over the founders if the founders have any money.This is a dumb plan. In fact, letting the founders sell a little stock

early would generally be better for the company, because it would

cause the founders' attitudes toward risk to be aligned with the

VCs'. As things currently work, their attitudes toward risk tend

to be diametrically opposed: the founders, who have nothing, would

prefer a 100% chance of $1 million to a 20% chance of $10 million,

while the VCs can afford to be "rational" and prefer the latter.Whatever they say, the reason founders are selling their companies

early instead of doing Series A rounds is that they get paid up

front. That first million is just worth so much more than the

subsequent ones. If founders could sell a little stock early,

they'd be happy to take VC money and bet the rest on a bigger

outcome.So why not let the founders have that first million, or at least

half million? The VCs would get same number of shares for the

money. So what if some of the money would go to the

founders instead of the company?Some VCs will say this is

unthinkable—that they want all their money to be put to work

growing the company. But the fact is, the huge size of current VC

investments is dictated by the structure

of VC funds, not the needs of startups. Often as not these large

investments go to work destroying the company rather than growing

it.The angel investors who funded our startup let the founders sell

some stock directly to them, and it was a good deal for everyone.

The angels made a huge return on that investment, so they're happy.

And for us founders it blunted the terrifying all-or-nothingness

of a startup, which in its raw form is more a distraction than a

motivator.If VCs are frightened at the idea of letting founders partially

cash out, let me tell them something still more frightening: you

are now competing directly with Google.

Thanks to Trevor Blackwell, Sarah Harlin, Jessica

Livingston, and Robert Morris for reading drafts of this.Romanian TranslationHebrew TranslationJapanese Translation

If you liked this, you may also like

Hackers & Painters.

Ideas for Startups

Want to start a startup? Get funded by

Y Combinator.

October 2005(This essay is derived from a talk at the 2005

Startup School.)How do you get good ideas for

startups? That's probably the number

one question people ask me.I'd like to reply with another question: why do people think it's

hard to come up with ideas for startups?That might seem a stupid thing to ask. Why do they think

it's hard? If people can't do it, then it is hard, at least

for them. Right?Well, maybe not. What people usually say is not that they can't

think of ideas, but that they don't have any. That's not quite the

same thing. It could be the reason they don't have any is that

they haven't tried to generate them.I think this is often the case. I think people believe that coming

up with ideas for startups is very hard-- that it must be

very hard-- and so they don't try do to it. They assume ideas are

like miracles: they either pop into your head or they don't.I also have a theory about why people think this. They overvalue

ideas. They think creating a startup is just a matter of implementing

some fabulous initial idea. And since a successful startup is worth

millions of dollars, a good idea is therefore a million dollar idea.If coming up with an idea for a startup equals coming up with a

million dollar idea, then of course it's going to seem hard. Too

hard to bother trying. Our instincts tell us something so valuable

would not be just lying around for anyone to discover.Actually, startup ideas are not million dollar ideas, and here's

an experiment you can try to prove it: just try to sell one. Nothing

evolves faster than markets. The fact that there's no market for

startup ideas suggests there's no demand. Which means, in the

narrow sense of the word, that startup ideas are worthless.QuestionsThe fact is, most startups end up nothing like the initial idea.

It would be closer to the truth to say the main value of your initial

idea is that, in the process of discovering it's broken, you'll

come up with your real idea.The initial idea is just a starting point-- not a blueprint, but a

question. It might help if they were expressed that way. Instead

of saying that your idea is to make a collaborative, web-based

spreadsheet, say: could one make a collaborative, web-based

spreadsheet? A few grammatical tweaks, and a woefully incomplete

idea becomes a promising question to explore.There's a real difference, because an assertion provokes objections

in a way a question doesn't. If you say: I'm going to build a

web-based spreadsheet, then critics-- the most dangerous of which

are in your own head-- will immediately reply that you'd be competing

with Microsoft, that you couldn't give people the kind of UI they

expect, that users wouldn't want to have their data on your servers,

and so on.A question doesn't seem so challenging. It becomes: let's try

making a web-based spreadsheet and see how far we get. And everyone

knows that if you tried this you'd be able to make something

useful. Maybe what you'd end up with wouldn't even be a spreadsheet.

Maybe it would be some kind of new spreasheet-like collaboration

tool that doesn't even have a name yet. You wouldn't have thought

of something like that except by implementing your way toward it.Treating a startup idea as a question changes what you're looking

for. If an idea is a blueprint, it has to be right. But if it's

a question, it can be wrong, so long as it's wrong in a way that

leads to more ideas.One valuable way for an idea to be wrong is to be only a partial

solution. When someone's working on a problem that seems too

big, I always ask: is there some way to bite off some subset of the

problem, then gradually expand from there? That will generally

work unless you get trapped on a local maximum, like 1980s-style

AI, or C.UpwindSo far, we've reduced the problem from thinking of a million dollar

idea to thinking of a mistaken question. That doesn't seem so hard,

does it?To generate such questions you need two things: to be familiar with

promising new technologies, and to have the right kind of friends.

New technologies are the ingredients startup ideas are made of, and

conversations with friends are the kitchen they're cooked in.Universities have both, and that's why so many startups grow out

of them. They're filled with new technologies, because they're

trying to produce research, and only things that are new count as

research. And they're full of exactly the right kind of people to

have ideas with: the other students, who will be not only smart but

elastic-minded to a fault.The opposite extreme would be a well-paying but boring job at a big

company. Big companies are biased against new technologies, and

the people you'd meet there would be wrong too.In an essay I wrote for high school students,

I said a good rule of thumb was to stay upwind-- to

work on things that maximize your future options. The principle

applies for adults too, though perhaps it has to be modified to:

stay upwind for as long as you can, then cash in the potential

energy you've accumulated when you need to pay for kids.I don't think people consciously realize this, but one reason

downwind jobs like churning out Java for a bank pay so well is

precisely that they are downwind. The market price for that kind

of work is higher because it gives you fewer options for the future.

A job that lets you work on exciting new stuff will tend to pay

less, because part of the compensation is in the form of the new

skills you'll learn.Grad school is the other end of the spectrum from a coding job at

a big company: the pay's low but you spend most of your time working

on new stuff. And of course, it's called "school," which makes

that clear to everyone, though in fact all jobs are some percentage

school.The right environment for having startup ideas need not be a

university per se. It just has to be a situation with a large

percentage of school.It's obvious why you want exposure to new technology, but why do

you need other people? Can't you just think of new ideas yourself?

The empirical answer is: no. Even Einstein needed people to bounce

ideas off. Ideas get developed in the process of explaining them

to the right kind of person. You need that resistance, just

as a carver needs the resistance of the wood.This is one reason Y Combinator has a rule against investing in

startups with only one founder. Practically every successful company

has at least two. And because startup founders work under great

pressure, it's critical they be friends.I didn't realize it till I was writing this, but that may help

explain why there are so few female startup founders. I read on

the Internet (so it must be true) that only 1.7% of VC-backed

startups are founded by women. The percentage of female hackers

is small, but not that small. So why the discrepancy?When you realize that successful startups tend to have multiple

founders who were already friends, a

possible explanation emerges. People's best friends are likely to

be of the same sex, and if one group is a minority in some population,

pairs of them will be a minority squared.

[1]DoodlingWhat these groups of co-founders do together is more complicated

than just sitting down and trying to think of ideas. I suspect the

most productive setup is a kind of together-alone-together sandwich.

Together you talk about some hard problem, probably getting nowhere.

Then, the next morning, one of you has an idea in the shower about

how to solve it. He runs eagerly to to tell the others, and together

they work out the kinks.What happens in that shower? It seems to me that ideas just pop

into my head. But can we say more than that?Taking a shower is like a form of meditation. You're alert, but

there's nothing to distract you. It's in a situation like this,

where your mind is free to roam, that it bumps into new ideas.What happens when your mind wanders? It may be like doodling. Most

people have characteristic ways of doodling. This habit is

unconscious, but not random: I found my doodles changed after I

started studying painting. I started to make the kind of gestures

I'd make if I were drawing from life. They were atoms of drawing,

but arranged randomly.

[2]Perhaps letting your mind wander is like doodling with ideas. You

have certain mental gestures you've learned in your work, and when

you're not paying attention, you keep making these same gestures,

but somewhat randomly. In effect, you call the same functions on

random arguments. That's what a metaphor is: a function applied

to an argument of the wrong type.Conveniently, as I was writing this, my mind wandered: would it be

useful to have metaphors in a programming language? I don't know;

I don't have time to think about this. But it's convenient because

this is an example of what I mean by habits of mind. I spend a lot

of time thinking about language design, and my habit of always

asking "would x be useful in a programming language" just got

invoked.If new ideas arise like doodles, this would explain why you have

to work at something for a while before you have any. It's not

just that you can't judge ideas till you're an expert in a field.

You won't even generate ideas, because you won't have any habits

of mind to invoke.Of course the habits of mind you invoke on some field don't have

to be derived from working in that field. In fact, it's often

better if they're not. You're not just looking for good ideas, but

for good new ideas, and you have a better chance of generating

those if you combine stuff from distant fields. As hackers, one

of our habits of mind is to ask, could one open-source x? For

example, what if you made an open-source operating system? A fine

idea, but not very novel. Whereas if you ask, could you make an

open-source play? you might be onto something.Are some kinds of work better sources of habits of mind than others?

I suspect harder fields may be better sources, because to attack

hard problems you need powerful solvents. I find math is a good

source of metaphors-- good enough that it's worth studying just for

that. Related fields are also good sources, especially when they're

related in unexpected ways. Everyone knows computer science and

electrical engineering are related, but precisely because everyone

knows it, importing ideas from one to the other doesn't yield great

profits. It's like importing something from Wisconsin to Michigan.

Whereas (I claim) hacking and painting are

also related, in the sense that hackers and painters are both

makers,

and this source of new ideas is practically virgin territory.ProblemsIn theory you could stick together ideas at random and see what you

came up with. What if you built a peer-to-peer dating site? Would

it be useful to have an automatic book? Could you turn theorems

into a commodity? When you assemble ideas at random like this,

they may not be just stupid, but semantically ill-formed. What

would it even mean to make theorems a commodity? You got me. I

didn't think of that idea, just its name.You might come up with something useful this way, but I never have.

It's like knowing a fabulous sculpture is hidden inside a block of

marble, and all you have to do is remove the marble that isn't part

of it. It's an encouraging thought, because it reminds you there

is an answer, but it's not much use in practice because the search

space is too big.I find that to have good ideas I need to be working on some problem.

You can't start with randomness. You have to start with a problem,

then let your mind wander just far enough for new ideas to form.In a way, it's harder to see problems than their solutions. Most

people prefer to remain in denial about problems. It's obvious

why: problems are irritating. They're problems! Imagine if people

in 1700 saw their lives the way we'd see them. It would have been

unbearable. This denial is such a powerful force that, even when

presented with possible solutions, people often prefer to believe

they wouldn't work.I saw this phenomenon when I worked on spam filters. In 2002, most

people preferred to ignore spam, and most of those who didn't

preferred to believe the heuristic filters then available were the

best you could do.I found spam intolerable, and I felt it had to be possible to

recognize it statistically. And it turns out that was all you

needed to solve the problem. The algorithm I used was ridiculously

simple. Anyone who'd really tried to solve the problem would have

found it. It was just that no one had really tried to solve the

problem.

[3]Let me repeat that recipe: finding the problem intolerable and

feeling it must be possible to solve it. Simple as it seems, that's

the recipe for a lot of startup ideas.WealthSo far most of what I've said applies to ideas in general. What's

special about startup ideas? Startup ideas are ideas for companies,

and companies have to make money. And the way to make money is to

make something people want.Wealth is what people want. I don't mean that as some kind of

philosophical statement; I mean it as a tautology.So an idea for a startup is an idea for something people want.

Wouldn't any good idea be something people want? Unfortunately

not. I think new theorems are a fine thing to create, but there

is no great demand for them. Whereas there appears to be great

demand for celebrity gossip magazines. Wealth is defined democratically.

Good ideas and valuable ideas are not quite the same thing; the

difference is individual tastes.But valuable ideas are very close to good ideas, especially in

technology. I think they're so close that you can get away with

working as if the goal were to discover good ideas, so long as, in

the final stage, you stop and ask: will people actually pay for

this? Only a few ideas are likely to make it that far and then get

shot down; RPN calculators might be one example.One way to make something people want is to look at stuff people

use now that's broken. Dating sites are a prime example. They

have millions of users, so they must be promising something people

want. And yet they work horribly. Just ask anyone who uses them.

It's as if they used the worse-is-better approach but stopped after

the first stage and handed the thing over to marketers.Of course, the most obvious breakage in the average computer user's

life is Windows itself. But this is a special case: you can't

defeat a monopoly by a frontal attack. Windows can and will be

overthrown, but not by giving people a better desktop OS. The way

to kill it is to redefine the problem as a superset of the current

one. The problem is not, what operating system should people use

on desktop computers? but how should people use applications?

There are answers to that question that don't even involve desktop

computers.Everyone thinks Google is going to solve this problem, but it is a

very subtle one, so subtle that a company as big as Google might

well get it wrong. I think the odds are better than 50-50 that the

Windows killer-- or more accurately, Windows transcender-- will

come from some little startup.Another classic way to make something people want is to take a

luxury and make it into a commmodity. People must want something

if they pay a lot for it. And it is a very rare product that can't

be made dramatically cheaper if you try.This was Henry Ford's plan. He made cars, which had been a luxury

item, into a commodity. But the idea is much older than Henry Ford.

Water mills transformed mechanical power from a luxury into a

commodity, and they were used in the Roman empire. Arguably

pastoralism transformed a luxury into a commodity.When you make something cheaper you can sell more of them. But if

you make something dramatically cheaper you often get qualitative

changes, because people start to use it in different ways. For

example, once computers get so cheap that most people can have one

of their own, you can use them as communication devices.Often to make something dramatically cheaper you have to redefine

the problem. The Model T didn't have all the features previous

cars did. It only came in black, for example. But it solved the

problem people cared most about, which was getting from place to

place.One of the most useful mental habits I know I learned from Michael

Rabin: that the best way to solve a problem is often to redefine

it. A lot of people use this technique without being consciously

aware of it, but Rabin was spectacularly explicit. You need a big

prime number? Those are pretty expensive. How about if I give you

a big number that only has a 10 to the minus 100 chance of not being

prime? Would that do? Well, probably; I mean, that's probably

smaller than the chance that I'm imagining all this anyway.Redefining the problem is a particularly juicy heuristic when you

have competitors, because it's so hard for rigid-minded people to

follow. You can work in plain sight and they don't realize the

danger. Don't worry about us. We're just working on search. Do

one thing and do it well, that's our motto.Making things cheaper is actually a subset of a more general

technique: making things easier. For a long time it was most of

making things easier, but now that the things we build are so

complicated, there's another rapidly growing subset: making things

easier to use.This is an area where there's great room for improvement. What you

want to be able to say about technology is: it just works. How

often do you say that now?Simplicity takes effort-- genius, even. The average programmer

seems to produce UI designs that are almost willfully bad. I was

trying to use the stove at my mother's house a couple weeks ago.

It was a new one, and instead of physical knobs it had buttons and

an LED display. I tried pressing some buttons I thought would cause

it to get hot, and you know what it said? "Err." Not even "Error."

"Err." You can't just say "Err" to the user of a stove.

You should design the UI so that errors are impossible. And the

boneheads who designed this stove even had an example of such a UI

to work from: the old one. You turn one knob to set the temperature

and another to set the timer. What was wrong with that? It just

worked.It seems that, for the average engineer, more options just means

more rope to hang yourself. So if you want to start a startup, you

can take almost any existing technology produced by a big company,

and assume you could build something way easier to use.Design for ExitSuccess for a startup approximately equals getting bought. You

need some kind of exit strategy, because you can't get the smartest

people to work for you without giving them options likely to be

worth something. Which means you either have to get bought or go

public, and the number of startups that go public is very small.If success probably means getting bought, should you make that a

conscious goal? The old answer was no: you were supposed to pretend

that you wanted to create a giant, public company, and act surprised

when someone made you an offer. Really, you want to buy us? Well,

I suppose we'd consider it, for the right price.I think things are changing. If 98% of the time success means

getting bought, why not be open about it? If 98% of the time you're

doing product development on spec for some big company, why not

think of that as your task? One advantage of this approach is that

it gives you another source of ideas: look at big companies, think

what they should

be doing, and do it yourself. Even if

they already know it, you'll probably be done faster.Just be sure to make something multiple acquirers will want. Don't

fix Windows, because the only potential acquirer is Microsoft, and

when there's only one acquirer, they don't have to hurry. They can

take their time and copy you instead of buying you. If you want

to get market price, work on something where there's competition.If an increasing number of startups are created to do product

development on spec, it will be a natural counterweight to monopolies.

Once some type of technology is captured by a monopoly, it will

only evolve at big company rates instead of startup rates, whereas

alternatives will evolve with especial speed. A free market

interprets monopoly as damage and routes around it.The Woz RouteThe most productive way to generate startup ideas is also the

most unlikely-sounding: by accident. If you look at how famous

startups got started, a lot of them weren't initially supposed to

be startups. Lotus began with a program Mitch Kapor wrote for a

friend. Apple got started because Steve Wozniak wanted to build

microcomputers, and his employer, Hewlett-Packard, wouldn't let him

do it at work. Yahoo began as David Filo's personal collection of

links.This is not the only way to start startups. You can sit down and

consciously come up with an idea for a company; we did. But measured

in total market cap, the build-stuff-for-yourself model might be

more fruitful. It certainly has to be the most fun way to come up

with startup ideas. And since a startup ought to have multiple

founders who were already friends before they decided to start a

company, the rather surprising conclusion is that the best way to

generate startup ideas is to do what hackers do for fun: cook up

amusing hacks with your friends.It seems like it violates some kind of conservation law, but there

it is: the best way to get a "million dollar idea" is just to do

what hackers enjoy doing anyway.

Notes[1]

This phenomenon may account for a number of discrepancies

currently blamed on various forbidden isms. Never attribute to

malice what can be explained by math.[2]

A lot of classic abstract expressionism is doodling of this type:

artists trained to paint from life using the same gestures but

without using them to represent anything. This explains why such

paintings are (slightly) more interesting than random marks would be.[3]

Bill Yerazunis had solved the problem, but he got there by

another path. He made a general-purpose file classifier so good

that it also worked for spam.One Specific IdeaRomanian TranslationJapanese TranslationTraditional Chinese TranslationRussian TranslationArabic Translation

What I Did this Summer

October 2005The first Summer Founders Program has just finished. We were

surprised how well it went. Overall only about 10% of startups

succeed, but if I had to guess now, I'd predict three or four of

the eight startups we funded will make it.Of the startups that needed further funding, I believe all have

either closed a round or are likely to soon. Two have already

turned down (lowball) acquisition offers.We would have been happy if just one of the eight seemed promising

by the end of the summer. What's going on? Did some kind of anomaly

make this summer's applicants especially good? We worry about that,

but we can't think of one. We'll find out this winter.The whole summer was full of surprises. The best was that the hypothesis we were testing seems to be

correct. Young hackers can start viable companies. This is good

news for two reasons: (a) it's an encouraging thought, and (b) it

means that Y Combinator, which is predicated on the idea, is not

hosed.AgeMore precisely, the hypothesis was that success in a startup depends

mainly on how smart and energetic you are, and much less on how old

you are or how much business experience you have. The results so

far bear this out. The 2005 summer founders ranged in age from 18

to 28 (average 23), and there is no correlation between their ages

and how well they're doing.This should not really be surprising. Bill Gates and Michael Dell

were both 19 when they started the companies that made them famous.

Young founders are not a new phenomenon: the trend began as soon

as computers got cheap enough for college kids to afford them.Another of our hypotheses was that you can start a startup on less

money than most people think. Other investors were surprised to

hear the most we gave any group was $20,000. But we knew it was

possible to start on that little because we started Viaweb on

$10,000.And so it proved this summer. Three months' funding is enough to

get into second gear. We had a demo day for potential investors

ten weeks in, and seven of the eight groups had a prototype ready

by that time. One, Reddit, had

already launched, and were able to give a demo of their live site.A researcher who studied the SFP startups said the one thing they

had in common was that they all worked ridiculously hard. People

this age are commonly seen as lazy. I think in some cases it's not

so much that they lack the appetite for work, but that the work

they're offered is unappetizing.The experience of the SFP suggests that if you let motivated people

do real work, they work hard, whatever their age. As one of the

founders said "I'd read that starting a startup consumed your life,

but I had no idea what that meant until I did it."I'd feel guilty if I were a boss making people work this hard. But

we're not these people's bosses. They're working on their own

projects. And what makes them work is not us but their competitors.

Like good athletes, they don't work hard because the coach yells

at them, but because they want to win.We have less power than bosses, and yet the founders work harder

than employees. It seems like a win for everyone. The only catch

is that we get on average only about 5-7% of the upside, while an

employer gets nearly all of it. (We're counting on it being 5-7%

of a much larger number.)As well as working hard, the groups all turned out to be extraordinarily

responsible. I can't think of a time when one failed to do something

they'd promised to, even by being late for an appointment. This

is another lesson the world has yet to learn. One of the founders

discovered that the hardest part of arranging a meeting with

executives at a big cell phone carrier was getting a rental company

to rent him a car, because he was too young.I think the problem here is much the same as with the apparent

laziness of people this age. They seem lazy because the work they're

given is pointless, and they act irresponsible because they're not

given any power. Some of them, anyway. We only have a sample size

of about twenty, but it seems so far that if you let people in their

early twenties be their own bosses, they rise to the occasion.MoraleThe summer founders were as a rule very idealistic. They also

wanted very much to get rich. These qualities might seem incompatible,

but they're not. These guys want to get rich, but they want to do

it by changing the world. They wouldn't (well, seven of the eight

groups wouldn't) be interested in making money by speculating in

stocks. They want to make something people use.I think this makes them more effective as founders. As hard as

people will work for money, they'll work harder for a cause. And

since success in a startup depends so much on motivation, the

paradoxical result is that the people likely to make the most money

are those who aren't in it just for the money.The founders of Kiko, for example,

are working on an Ajax calendar. They want to get rich, but they

pay more attention to design than they would if that were their

only motivation. You can tell just by looking at it.I never considered it till this summer, but this might be another

reason startups run by hackers tend to do better than those run by

MBAs. Perhaps it's not just that hackers understand technology

better, but that they're driven by more powerful motivations.

Microsoft, as I've said before, is a dangerously misleading example.

Their mean corporate culture only works for monopolies.

Google is a better model.Considering that the summer founders are the sharks in this ocean,

we were surprised how frightened most of them were of competitors.

But now that I think of it, we were just as frightened when we

started Viaweb. For the first year, our initial reaction to news

of a competitor was always: we're doomed. Just as a hypochondriac

magnifies his symptoms till he's convinced he has some terrible

disease, when you're not used to competitors you magnify them into

monsters.Here's a handy rule for startups: competitors are rarely as dangerous

as they seem. Most will self-destruct before you can destroy them.

And it certainly doesn't matter how many of them there are, any

more than it matters to the winner of a marathon how many runners

are behind him."It's a crowded market," I remember one founder saying worriedly."Are you the current leader?" I asked."Yes.""Is anyone able to develop software faster than you?""Probably not.""Well, if you're ahead now, and you're the fastest, then you'll

stay ahead. What difference does it make how many others there

are?"Another group was worried when they realized they had to rewrite

their software from scratch. I told them it would be a bad sign

if they didn't. The main function of your initial version is to

be rewritten.That's why we advise groups to ignore issues like scalability,

internationalization, and heavy-duty security at first. [1] I can

imagine an advocate of "best practices" saying these ought to be

considered from the start. And he'd be right, except that they

interfere with the primary function of software in a startup: to

be a vehicle for experimenting with its own design. Having to

retrofit internationalization or scalability is a pain, certainly.

The only bigger pain is not needing to, because your initial version

was too big and rigid to evolve into something users wanted.I suspect this is another reason startups beat big companies.

Startups can be irresponsible and release version 1s that are light

enough to evolve. In big companies, all the pressure is in the

direction of over-engineering.What Got LearnedOne thing we were curious about this summer was where these groups

would need help. That turned out to vary a lot. Some we helped

with technical advice-- for example, about how to set up an application

to run on multiple servers. Most we helped with strategy questions,

like what to patent, and what to charge for and what to give away.

Nearly all wanted advice about dealing with future investors: how

much money should they take and what kind of terms should they

expect?However, all the groups quickly learned how to deal with stuff like

patents and investors. These problems aren't intrinsically difficult,

just unfamiliar.It was surprising-- slightly frightening even-- how fast they

learned. The weekend before the demo day for investors, we had a

practice session where all the groups gave their presentations.

They were all terrible. We tried to explain how to make them better,

but we didn't have much hope. So on demo day I told the assembled

angels and VCs that these guys were hackers, not MBAs, and so while

their software was good, we should not expect slick presentations

from them.The groups then proceeded to give fabulously slick presentations.

Gone were the mumbling recitations of lists of features. It was

as if they'd spent the past week at acting school. I still don't

know how they did it.Perhaps watching each others' presentations helped them see what

they'd been doing wrong. Just as happens in college, the summer

founders learned a lot from one another-- maybe more than they

learned from us. A lot of the problems they face are the same,

from dealing with investors to hacking Javascript.I don't want to give the impression there were no problems this

summer. A lot went wrong, as usually happens with startups. One

group got an "exploding

term-sheet" from some VCs. Pretty much all the groups who had

dealings with big companies found that big companies do everything

infinitely slowly. (This is to be expected. If big companies

weren't incapable, there would be no room for startups to exist.)

And of course there were the usual nightmares associated with

servers. In short, the disasters this summer were just the usual childhood

diseases. Some of this summer's eight startups will

probably die eventually; it would be extraordinary if all eight

succeeded. But what kills them will not be dramatic, external

threats, but a mundane, internal one: not getting enough done.So far, though, the news is all good. In fact, we were surprised

how much fun the summer was for us. The main reason was how much

we liked the founders. They're so earnest and hard-working. They

seem to like us too. And this illustrates another advantage of

investing over hiring: our relationship with them is way better

than it would be between a boss and an employee. Y Combinator ends

up being more like an older brother than a parent.I was surprised how much time I spent making introductions.

Fortunately I discovered that when a startup needed to talk to

someone, I could usually get to the right person by at most one

hop. I remember wondering, how did my friends get to be so eminent?

and a second later realizing: shit, I'm forty.Another surprise was that the three-month batch format,

which we were forced into by the constraints of the summer, turned

out to be an advantage. When we started Y Combinator, we planned

to invest the way other venture firms do: as proposals came in,

we'd evaluate them and decide yes or no. The SFP

was just an experiment to get things started. But it worked so

well that we plan to do

all

our investing this way, one cycle in

the summer and one in winter. It's more efficient for us, and

better for the startups too.Several groups said our weekly dinners saved them from a common

problem afflicting startups: working so hard that one has no social

life. (I remember that part all too well.) This way, they were

guaranteed a social event at least once a week.IndependenceI've heard Y Combinator described as an "incubator." Actually we're

the opposite: incubators exert more control than ordinary VCs, and

we make a point of exerting less. Among other things, incubators

usually make you work in their office-- that's where the

word "incubator" comes from. That seems the wrong model. If

investors get too involved, they smother one of the most powerful

forces in a startup: the feeling that it's your own company.Incubators were conspicuous failures during the Bubble. There's

still debate about whether this was because of the Bubble, or because

they're a bad idea. My vote is they're a bad idea. I think they

fail because they select for the wrong people. When we were starting

a startup, we would never have taken funding from an "incubator."

We can find office space, thanks; just give us the money. And

people with that attitude are the ones likely to succeed in startups.Indeed, one quality all the founders shared this summer was a spirit

of independence. I've been wondering about that. Are some people

just a lot more independent than others, or would everyone be this

way if they were allowed to?As with most nature/nurture questions, the answer is probably: some

of each. But my main conclusion from the summer is that there's

more environment in the mix than most people realize. I could see

that from how the founders' attitudes changed during the

summer. Most were emerging from twenty or so years of being told

what to do. They seemed a little surprised at having total freedom.

But they grew into it really quickly; some of these guys now seem

about four inches taller (metaphorically) than they did at the

beginning of the summer.When we asked the summer founders what surprised them most about

starting a company, one said "the most shocking thing is that it

worked."It will take more experience to know for sure, but my guess is that

a lot of hackers could do this-- that if you put people in a position

of independence, they develop the qualities they need. Throw them

off a cliff, and most will find on the way down that they have

wings.The reason this is news to anyone is that the same forces work in

the other direction too. Most hackers are

employees, and this molds

you into someone to whom starting a startup seems impossible as

surely as starting a startup molds you into someone who can handle

it.If I'm right, "hacker" will mean something different in twenty years

than it does now. Increasingly it will mean the people who run the

company. Y Combinator is just accelerating a process that would

have happened anyway. Power is shifting from the people who deal

with money to the people who create technology, and if our experience

this summer is any guide, this will be a good thing.Notes[1] By heavy-duty security I mean efforts to protect against truly

determined attackers.The image

shows us, the 2005 summer founders, and Smartleaf

co-founders Mark Nitzberg and Olin Shivers at the 30-foot table

Kate Courteau designed for us. Photo by Alex Lewin.Thanks to Sarah Harlin, Steve Huffman, Jessica Livingston,

Zak Stone, and Aaron Swartz for reading drafts of this.

Romanian TranslationJapanese Translation

Inequality and Risk

August 2005(This essay is derived from a talk at Defcon 2005.)Suppose you wanted to get rid of economic inequality. There are

two ways to do it: give money to the poor, or take it away from the

rich. But they amount to the same thing, because if you want to

give money to the poor, you have to get it from somewhere. You

can't get it from the poor, or they just end up where they started.

You have to get it from the rich.There is of course a way to make the poor richer without simply

shifting money from the rich. You could help the poor become more

productive — for example, by improving access to education. Instead

of taking money from engineers and giving it to checkout clerks,

you could enable people who would have become checkout clerks to

become engineers.This is an excellent strategy for making the poor richer. But the

evidence of the last 200 years shows that it doesn't reduce economic

inequality, because it makes the rich richer too. If there

are more engineers, then there are more opportunities to hire them

and to sell them things. Henry Ford couldn't have made a fortune

building cars in a society in which most people were still subsistence

farmers; he would have had neither workers nor customers.If you want to reduce economic inequality instead of just improving

the overall standard of living, it's not enough just to raise up

the poor. What if one of your newly minted engineers gets ambitious

and goes on to become another Bill Gates? Economic inequality will

be as bad as ever. If you actually want to compress the gap between

rich and poor, you have to push down on the top as well as pushing

up on the bottom.How do you push down on the top? You could try to decrease the

productivity of the people who make the most money: make the best

surgeons operate with their left hands, force popular actors to

overeat, and so on. But this approach is hard to implement. The

only practical solution is to let people do the best work they can,

and then (either by taxation or by limiting what they can charge)

to confiscate whatever you deem to be surplus.So let's be clear what reducing economic inequality means. It is

identical with taking money from the rich.When you transform a mathematical expression into another form, you

often notice new things. So it is in this case. Taking money from

the rich turns out to have consequences one might not foresee when

one phrases the same idea in terms of "reducing inequality."The problem is, risk and reward have to be proportionate. A bet

with only a 10% chance of winning has to pay more than one with a

50% chance of winning, or no one will take it. So if you lop off

the top of the possible rewards, you thereby decrease people's

willingness to take risks.Transposing into our original expression, we get: decreasing economic

inequality means decreasing the risk people are willing to take.There are whole classes of risks that are no longer worth taking

if the maximum return is decreased. One reason high tax rates are

disastrous is that this class of risks includes starting new

companies.InvestorsStartups are intrinsically risky. A startup

is like a small boat

in the open sea. One big wave and you're sunk. A competing product,

a downturn in the economy, a delay in getting funding or regulatory

approval, a patent suit, changing technical standards, the departure

of a key employee, the loss of a big account — any one of these can

destroy you overnight. It seems only about 1 in 10 startups succeeds.

[1]Our startup paid its first round of outside investors 36x. Which

meant, with current US tax rates, that it made sense to invest in

us if we had better than a 1 in 24 chance of succeeding. That

sounds about right. That's probably roughly how we looked when we

were a couple of nerds with no business experience operating out

of an apartment.If that kind of risk doesn't pay, venture investing, as we know it,

doesn't happen.That might be ok if there were other sources of capital for new

companies. Why not just have the government, or some large

almost-government organization like Fannie Mae, do the venture

investing instead of private funds?I'll tell you why that wouldn't work. Because then you're asking

government or almost-government employees to do the one thing they

are least able to do: take risks.As anyone who has worked for the government knows, the important

thing is not to make the right choices, but to make choices that

can be justified later if they fail. If there is a safe option,

that's the one a bureaucrat will choose. But that is exactly the

wrong way to do venture investing. The nature of the business means

that you want to make terribly risky choices, if the upside looks

good enough.VCs are currently

paid in a way that makes them

focus on the upside:

they get a percentage of the fund's gains. And that helps overcome

their understandable fear of investing in a company run by nerds

who look like (and perhaps are) college students.If VCs weren't allowed to get rich, they'd behave like bureaucrats.

Without hope of gain, they'd have only fear of loss. And so they'd

make the wrong choices. They'd turn down the nerds in favor of the

smooth-talking MBA in a suit, because that investment would be

easier to justify later if it failed.FoundersBut even if you could somehow redesign venture funding to work

without allowing VCs to become rich, there's another kind of investor

you simply cannot replace: the startups' founders and early employees.What they invest is their time and ideas. But these are equivalent

to money; the proof is that investors are willing (if forced) to

treat them as interchangeable, granting the same status to "sweat

equity" and the equity they've purchased with cash.The fact that you're investing time doesn't change the relationship

between risk and reward. If you're going to invest your time in

something with a small chance of succeeding, you'll only do it if

there is a proportionately large payoff.

[2]

If large payoffs aren't allowed, you may as well play it safe.Like many startup founders, I did it to get rich. But not because

I wanted to buy expensive things. What I wanted was security. I

wanted to make enough money that I didn't have to worry about money.

If I'd been forbidden to make enough from a startup to do this, I

would have sought security by some other means: for example, by

going to work for a big, stable organization from which it would

be hard to get fired. Instead of busting my ass in a startup, I

would have tried to get a nice, low-stress job at a big research

lab, or tenure at a university.That's what everyone does in societies where risk isn't rewarded.

If you can't ensure your own security, the next best thing is to

make a nest for yourself in some large organization where your

status depends mostly on seniority.

[3]Even if we could somehow replace investors, I don't see how we could

replace founders. Investors mainly contribute money, which in

principle is the same no matter what the source. But the founders

contribute ideas. You can't replace those.Let's rehearse the chain of argument so far. I'm heading for a

conclusion to which many readers will have to be dragged kicking

and screaming, so I've tried to make each link unbreakable. Decreasing

economic inequality means taking money from the rich. Since risk

and reward are equivalent, decreasing potential rewards automatically

decreases people's appetite for risk. Startups are intrinsically

risky. Without the prospect of rewards proportionate to the risk,

founders will not invest their time in a startup. Founders are

irreplaceable. So eliminating economic inequality means eliminating

startups.Economic inequality is not just a consequence of startups.

It's the engine that drives them, in the same way a fall of water

drives a water mill. People start startups in the hope of becoming

much richer than they were before. And if your society tries to

prevent anyone from being much richer than anyone else, it will

also prevent one person from being much richer at t2 than t1.GrowthThis argument applies proportionately. It's not just that if you

eliminate economic inequality, you get no startups. To the extent

you reduce economic inequality, you decrease the number of startups.

[4]

Increase taxes, and willingness to take risks decreases in

proportion.And that seems bad for everyone. New technology and new jobs both

come disproportionately from new companies. Indeed, if you don't

have startups, pretty soon you won't have established companies

either, just as, if you stop having kids, pretty soon you won't

have any adults.It sounds benevolent to say we ought to reduce economic inequality.

When you phrase it that way, who can argue with you? Inequality

has to be bad, right? It sounds a good deal less benevolent to say

we ought to reduce the rate at which new companies are founded.

And yet the one implies the other.Indeed, it may be that reducing investors' appetite for risk doesn't

merely kill off larval startups, but kills off the most promising

ones especially. Startups yield faster growth at greater risk than

established companies. Does this trend also hold among startups?

That is, are the riskiest startups the ones that generate most

growth if they succeed? I suspect the answer is yes. And that's

a chilling thought, because it means that if you cut investors'

appetite for risk, the most beneficial startups are the first to

go.Not all rich people got that way from startups, of course. What

if we let people get rich by starting startups, but taxed away all

other surplus wealth? Wouldn't that at least decrease inequality?Less than you might think. If you made it so that people could

only get rich by starting startups, people who wanted to get rich

would all start startups. And that might be a great thing. But I

don't think it would have much effect on the distribution of wealth.

People who want to get rich will do whatever they have to. If

startups are the only way to do it, you'll just get far more people

starting startups. (If you write the laws very carefully, that is.

More likely, you'll just get a lot of people doing things that can

be made to look on paper like startups.)If we're determined to eliminate economic inequality, there is still

one way out: we could say that we're willing to go ahead and do

without startups. What would happen if we did?At a minimum, we'd have to accept lower rates of technological

growth. If you believe that large, established companies could

somehow be made to develop new technology as fast as startups, the

ball is in your court to explain how. (If you can come up with a

remotely plausible story, you can make a fortune writing business

books and consulting for large companies.)

[5]Ok, so we get slower growth. Is that so bad? Well, one reason

it's bad in practice is that other countries might not agree to

slow down with us. If you're content to develop new technologies

at a slower rate than the rest of the world, what happens is that

you don't invent anything at all. Anything you might discover has

already been invented elsewhere. And the only thing you can offer

in return is raw materials and cheap labor. Once you sink that

low, other countries can do whatever they like with you: install

puppet governments, siphon off your best workers, use your women

as prostitutes, dump their toxic waste on your territory — all the

things we do to poor countries now. The only defense is to isolate

yourself, as communist countries did in the twentieth century. But

the problem then is, you have to become a police state to enforce

it.

Wealth and PowerI realize startups are not the main target of those who want to

eliminate economic inequality. What they really dislike is the

sort of wealth that becomes self-perpetuating through an alliance

with power. For example, construction firms that fund politicians'

campaigns in return for government contracts, or rich parents who

get their children into good colleges by sending them to expensive

schools designed for that purpose. But if you try to attack this type of wealth

through economic policy, it's hard to hit without destroying

startups as collateral damage.The problem here is not wealth, but corruption. So why not go after

corruption?We don't need to prevent people from being rich if we can prevent

wealth from translating into power. And there has been progress

on that front. Before he died of drink in 1925, Commodore Vanderbilt's

wastrel grandson Reggie ran down pedestrians on five separate

occasions, killing two of them. By 1969, when Ted Kennedy drove

off the bridge at Chappaquiddick, the limit seemed to be down to

one. Today it may well be zero. But what's changed is not variation

in wealth. What's changed is the ability to translate wealth into

power.How do you break the connection between wealth and power? Demand

transparency. Watch closely how power is exercised, and demand an

account of how decisions are made. Why aren't all police interrogations

videotaped? Why did 36% of Princeton's class of 2007 come from

prep schools, when only 1.7% of American kids attend them? Why did

the US really invade Iraq? Why don't government officials disclose

more about their finances, and why only during their term of office?A friend of mine who knows a lot about computer security says the

single most important step is to log everything. Back when he was

a kid trying to break into computers, what worried him most was the

idea of leaving a trail. He was more inconvenienced by the need

to avoid that than by any obstacle deliberately put in his path.Like all illicit connections, the connection between wealth and

power flourishes in secret. Expose all transactions, and you will

greatly reduce it. Log everything. That's a strategy that already

seems to be working, and it doesn't have the side effect of making

your whole country poor.I don't think many people realize there is a connection between

economic inequality and risk. I didn't fully grasp it till recently.

I'd known for years of course that if one didn't score in a startup,

the other alternative was to get a cozy, tenured research job. But

I didn't understand the equation governing my behavior. Likewise,

it's obvious empirically that a country that doesn't let people get

rich is headed for disaster, whether it's Diocletian's Rome or

Harold Wilson's Britain. But I did not till recently understand

the role risk played.If you try to attack wealth, you end up nailing risk as well, and

with it growth. If we want a fairer world, I think we're better

off attacking one step downstream, where wealth turns into power.Notes

[1]

Success here is defined from the initial investors' point of

view: either an IPO, or an acquisition for more than the valuation

at the last round of funding. The conventional 1 in 10 success rate

is suspiciously neat, but conversations with VCs suggest it's roughly

correct for startups overall. Top VC firms expect to do better.[2]

I'm not claiming founders sit down and calculate the expected

after-tax return from a startup. They're motivated by examples of

other people who did it. And those examples do reflect after-tax returns.[3]

Conjecture: The variation in wealth in a (non-corrupt)

country or organization

will be inversely proportional to the prevalence of systems of

seniority. So if you suppress variation in wealth, seniority will

become correspondingly more important. So far, I know of no

counterexamples, though in very corrupt countries you may get

both simultaneously. (Thanks to Daniel Sobral for pointing

this out.)[4]

In a country with a truly feudal economy, you might be able to

redistribute wealth successfully, because there are no startups to

kill.[5]

The speed at which startups develop new techology is the other

reason they pay so well. As I explained in "How to Make Wealth", what you do in a startup is compress a

lifetime's worth of work into a few years. It seems as

dumb to discourage that as to discourage risk-taking.

Thanks to Chris Anderson, Trevor Blackwell, Dan Giffin,

Jessica Livingston, and Evan Williams for reading drafts of this

essay, and to Langley Steinert, Sangam Pant, and Mike Moritz for

information about venture investing.Romanian TranslationDutch TranslationTraditional Chinese TranslationJapanese TranslationHebrew Translation

If you liked this, you may also like

Hackers & Painters.

After the Ladder

August 2005Thirty years ago, one was supposed to work one's way up the corporate

ladder. That's less the rule now. Our generation wants to get

paid up front. Instead of developing a product for some big company

in the expectation of getting job security in return, we develop

the product ourselves, in a startup, and sell it to the big company.

At the very least we want options.Among other things, this shift has created the appearance of a rapid

increase in economic inequality. But really the two cases are not

as different as they look in economic statistics.Economic statistics are misleading because they ignore the value

of safe jobs. An easy job from which one can't be fired is worth

money; exchanging the two is one of the commonest forms of

corruption. A sinecure is, in effect, an annuity. Except sinecures

don't appear in economic statistics. If they did, it would be clear

that in practice socialist countries have nontrivial disparities

of wealth, because they usually have a class of powerful bureaucrats

who are paid mostly by seniority and can never be fired.While not a sinecure, a position on the corporate ladder was genuinely

valuable, because big companies tried not to fire people, and

promoted from within based largely on seniority. A position on the

corporate ladder had a value analogous to the "goodwill" that is a

very real element in the valuation of companies. It meant one could

expect future high paying jobs.One of main causes of the decay of the corporate ladder is the trend

for takeovers that began in the 1980s. Why waste your time climbing

a ladder that might disappear before you reach the top?And, by no coincidence, the corporate ladder was one of the reasons

the early corporate raiders were so successful. It's not only

economic statistics that ignore the value of safe jobs. Corporate

balance sheets do too. One reason it was profitable to carve up 1980s

companies and sell them for parts was that they hadn't formally

acknowledged their implicit debt to employees who had done good

work and expected to be rewarded with high-paying executive jobs

when their time came.In the movie Wall Street, Gordon Gekko

ridicules a company overloaded with vice presidents. But the company

may not be as corrupt as it seems; those VPs' cushy jobs were

probably payment for work done earlier.I like the new model better. For one thing, it seems a bad plan

to treat jobs as rewards. Plenty of good engineers got made into

bad managers that way. And the old system meant people had to deal

with a lot more corporate politics, in order to protect the work

they'd invested in a position on the ladder.The big disadvantage of the new system is that it involves more risk. If you develop ideas in a startup instead

of within a big company, any number of random factors could sink

you before you can finish. But maybe the older generation would

laugh at me for saying that the way we do things is riskier. After

all, projects within big companies were always getting cancelled

as a result of arbitrary decisions from higher up. My father's

entire industry (breeder reactors) disappeared that way.For better or worse, the idea of the corporate ladder is probably

gone for good. The new model seems more liquid, and more efficient.

But it is less of a change, financially, than one might think. Our

fathers weren't that stupid.Romanian TranslationJapanese Translation

What Business Can Learn from Open Source

August 2005(This essay is derived from a talk at Oscon 2005.)Lately companies have been paying more attention to open source.

Ten years ago there seemed a real danger Microsoft would extend its

monopoly to servers. It seems safe to say now that open source has

prevented that. A recent survey found 52% of companies are replacing

Windows servers with Linux servers.

[1]More significant, I think, is which 52% they are. At this point,

anyone proposing to run Windows on servers should be prepared to

explain what they know about servers that Google, Yahoo, and Amazon

don't.But the biggest thing business has to learn from open source is not

about Linux or Firefox, but about the forces that produced them.

Ultimately these will affect a lot more than what software you use.We may be able to get a fix on these underlying forces by triangulating

from open source and blogging. As you've probably noticed, they

have a lot in common.Like open source, blogging is something people do themselves, for

free, because they enjoy it. Like open source hackers, bloggers

compete with people working for money, and often win. The method

of ensuring quality is also the same: Darwinian. Companies ensure

quality through rules to prevent employees from screwing up. But

you don't need that when the audience can communicate with one

another. People just produce whatever they want; the good stuff

spreads, and the bad gets ignored. And in both cases, feedback

from the audience improves the best work.Another thing blogging and open source have in common is the Web.

People have always been willing to do great work

for free, but before the Web it was harder to reach an audience

or collaborate on projects.AmateursI think the most important of the new principles business has to learn is

that people work a lot harder on stuff they like. Well, that's

news to no one. So how can I claim business has to learn it? When

I say business doesn't know this, I mean the structure of business

doesn't reflect it.Business still reflects an older model, exemplified by the French

word for working: travailler. It has an English cousin, travail,

and what it means is torture.

[2]This turns out not to be the last word on work, however.

As societies get richer, they learn something about

work that's a lot like what they learn about diet. We know now that the

healthiest diet is the one our peasant ancestors were forced to

eat because they were poor. Like rich food, idleness

only seems desirable when you don't get enough of it. I think we were

designed to work, just as we were designed to eat a certain amount

of fiber, and we feel bad if we don't.There's a name for people who work for the love of it: amateurs.

The word now has such bad connotations that we forget its etymology,

though it's staring us in the face. "Amateur" was originally rather

a complimentary word. But the thing to be in the twentieth century

was professional, which amateurs, by definition, are not.That's why the business world was so surprised by one lesson from

open source: that people working for love often surpass those working

for money. Users don't switch from Explorer to Firefox because

they want to hack the source. They switch because it's a better

browser.It's not that Microsoft isn't trying. They know controlling the

browser is one of the keys to retaining their monopoly. The problem

is the same they face in operating systems: they can't pay people

enough to build something better than a group of inspired hackers

will build for free.I suspect professionalism was always overrated-- not just in the

literal sense of working for money, but also connotations like

formality and detachment. Inconceivable as it would have seemed

in, say, 1970, I think professionalism was largely a fashion,

driven by conditions that happened to exist in the twentieth century.One of the most powerful of those was the existence of "channels." Revealingly,

the same term was used for both products and information: there

were distribution channels, and TV and radio channels.It was the narrowness of such channels that made professionals

seem so superior to amateurs. There were only a few jobs as

professional journalists, for example, so competition ensured the

average journalist was fairly good. Whereas anyone can express

opinions about current events in a bar. And so the average person

expressing his opinions in a bar sounds like an idiot compared to

a journalist writing about the subject.On the Web, the barrier for publishing your ideas is even lower.

You don't have to buy a drink, and they even let kids in.

Millions of people are publishing online, and the average

level of what they're writing, as you might expect, is not very

good. This has led some in the media to conclude that blogs don't

present much of a threat-- that blogs are just a fad.Actually, the fad is the word "blog," at least the way the print

media now use it. What they mean by "blogger" is not someone who

publishes in a weblog format, but anyone who publishes online.

That's going to become a problem as the Web becomes the default

medium for publication. So I'd

like to suggest an alternative word for someone who publishes online.

How about "writer?"Those in the print media who dismiss the writing online because of

its low average quality are missing an important point: no one reads

the average blog. In the old world of channels, it meant something

to talk about average quality, because that's what you were getting

whether you liked it or not.

But now you can read any writer you want. So the average

quality of writing online isn't what the print media are competing

against. They're competing against the best writing online. And,

like Microsoft, they're losing.I know that from my own experience as a reader. Though most print

publications are online, I probably

read two or three articles on individual people's sites for every

one I read on the site of a newspaper or magazine.And when I read, say, New York Times stories, I never reach

them through the Times front page. Most I find through aggregators

like Google News or Slashdot or Delicious. Aggregators show how

much better

you can do than the channel. The New York Times front page is

a list of articles written by people who work for the New York Times. Delicious

is a list of articles that are interesting. And it's only now that

you can see the two side by side that you notice how little overlap there is.Most articles in the print media are boring. For example, the

president notices that a majority of voters now think invading Iraq

was a mistake, so he makes an address to the nation to drum up

support. Where is the man bites dog in that? I didn't hear the

speech, but I could probably tell you exactly what he said. A

speech like that is, in the most literal sense, not news: there is

nothing new in it.

[3]Nor is there anything new, except the names and places, in most

"news" about things going wrong. A child is abducted; there's a

tornado; a ferry sinks; someone gets bitten by a shark; a small

plane crashes. And what do you learn about the world from these

stories? Absolutely nothing. They're outlying data points; what

makes them gripping also makes them irrelevant.As in software, when professionals produce such crap, it's not

surprising if amateurs can do better. Live by the channel, die by

the channel: if you depend on an oligopoly, you sink into bad habits

that are hard to overcome when you suddenly get competition.

[4]WorkplacesAnother thing blogs and open source software have in common is that

they're often made by people working at home. That may not seem

surprising. But it should be. It's the architectural equivalent

of a home-made aircraft shooting down an F-18. Companies spend

millions to build office buildings for a single purpose: to be a

place to work. And yet people working in their own homes,

which aren't even designed to be workplaces, end up

being more productive.This proves something a lot of us have suspected. The average

office is a miserable place to get work done. And a lot of what

makes offices bad are the very qualities we associate with

professionalism. The sterility

of offices is supposed to suggest efficiency. But suggesting

efficiency is a different thing from actually being efficient.The atmosphere of the average workplace is to productivity what

flames painted on the side of a car are to speed. And it's not

just the way offices look that's bleak. The way people act is just

as bad.Things are different in a startup. Often as not a startup begins

in an apartment. Instead of matching beige cubicles

they have an assortment of furniture they bought used. They work

odd hours, wearing the most casual of clothing. They look at

whatever they want online without worrying whether it's "work safe."

The cheery, bland language of the office is replaced by wicked humor. And

you know what? The company at this stage is probably the most

productive it's ever going to be.Maybe it's not a coincidence. Maybe some aspects of professionalism

are actually a net lose.To me the most demoralizing aspect of the traditional office is

that you're supposed to be there at certain times. There are usually

a few people in a company who really have to, but the reason most

employees work fixed hours is that the company can't measure their

productivity.The basic idea behind office hours is that if you can't make people

work, you can at least prevent them from having fun. If employees

have to be in the building a certain number of hours a day, and are

forbidden to do non-work things while there, then they must be

working. In theory. In practice they spend a lot of their time

in a no-man's land, where they're neither working nor having fun.If you could measure how much work people did, many companies

wouldn't need any fixed workday. You could just say: this is what

you have to do. Do it whenever you like, wherever you like. If

your work requires you to talk to other people in the company, then

you may need to be here a certain amount. Otherwise we don't care.That may seem utopian, but it's what we told people who came to

work for our company. There were no fixed office hours. I never

showed up before 11 in the morning. But we weren't saying this to

be benevolent. We were saying: if you work here we expect you to

get a lot done. Don't try to fool us just by being here a lot.The problem with the facetime model is not just that it's demoralizing, but

that the people pretending to work interrupt

the ones actually working. I'm convinced the facetime model

is the main reason large organizations have so many meetings.

Per capita, large organizations accomplish very little.

And yet all those people have to be on site at least eight hours a

day. When so much time goes in one end and so little achievement

comes out the other, something has to give. And meetings are the

main mechanism for taking up the slack.For one year I worked at a regular nine to five job, and I remember

well the strange, cozy feeling that comes over one during meetings.

I was very aware, because of the novelty, that I was being paid for

programming. It seemed just amazing, as if there was a machine on

my desk that spat out a dollar bill every two minutes no matter

what I did. Even while I was in the bathroom! But because the

imaginary machine was always running, I felt I always ought to be

working. And so meetings felt wonderfully relaxing. They

counted as work, just like programming, but they were so much easier.

All you had to do was sit and look attentive.Meetings are like an opiate with a network effect. So is email,

on a smaller scale. And in addition to the direct cost in time,

there's the cost in fragmentation-- breaking people's day up into

bits too small to be useful.You can see how dependent you've become on something by removing

it suddenly. So for big companies I propose the following experiment.

Set aside one day where meetings are forbidden-- where everyone has to

sit at their desk all day and work without interruption on

things they can do without talking to anyone else.

Some amount of communication is necessary in most jobs, but I'm

sure many employees could find eight hours worth of stuff they could

do by themselves. You could call it "Work Day."The other problem with pretend work

is that it often looks better than real work. When I'm

writing or hacking I spend as much time just thinking as I do

actually typing. Half the time I'm sitting drinking a cup of tea,

or walking around the neighborhood. This is a critical phase--

this is where ideas come from-- and yet I'd feel guilty doing this

in most offices, with everyone else looking busy.It's hard to see how bad some practice is till you have something

to compare it to. And that's one reason open source, and even blogging

in some cases, are so important. They show us what real work looks like.We're funding eight new startups at the moment. A friend asked

what they were doing for office space, and seemed surprised when I

said we expected them to work out of whatever apartments they found

to live in. But we didn't propose that to save money. We did it

because we want their software to be good. Working in crappy

informal spaces is one of the things startups do right without

realizing it. As soon as you get into an office, work and life

start to drift apart.That is one of the key tenets of professionalism. Work and life

are supposed to be separate. But that part, I'm convinced, is a

mistake.Bottom-UpThe third big lesson we can learn from open source and

blogging is that ideas can bubble up from the bottom, instead of

flowing down from the top. Open source and blogging both work

bottom-up: people make what they want, and the best stuff

prevails.Does this sound familiar? It's the principle of a market economy.

Ironically, though open source and blogs are done for free, those

worlds resemble market economies, while most companies, for all

their talk about the value of free markets, are run internally like

communist states.There are two forces that together steer design: ideas about

what to do next, and the enforcement of quality. In the channel

era, both flowed down from the top. For example, newspaper editors

assigned stories to reporters, then edited what they wrote.Open source and blogging show us things don't have to work that

way. Ideas and even the enforcement of quality can flow bottom-up.

And in both cases the results are not merely acceptable, but better.

For example, open source software is more reliable precisely because

it's open source; anyone can find mistakes.The same happens with writing. As we got close to publication, I

found I was very worried about the essays in

Hackers

& Painters

that hadn't been online. Once an essay has had a couple thousand

page views I feel reasonably confident about it. But these had had

literally orders of magnitude less scrutiny. It felt like

releasing software without testing it.That's what all publishing used to be like. If

you got ten people to read a manuscript, you were lucky. But I'd

become so used to publishing online that the old method now seemed

alarmingly unreliable, like navigating by dead reckoning once you'd

gotten used to a GPS.The other thing I like about publishing online is that you can write

what you want and publish when you want. Earlier this year I wrote

something that seemed suitable for a magazine, so

I sent it to an editor I know.

As I was waiting to hear back, I found to my surprise that I was

hoping they'd reject it. Then I could put it online right away.

If they accepted it, it wouldn't be read by anyone for months, and

in the meantime I'd have to fight word-by-word to save it from being

mangled by some twenty five year old copy editor.

[5]Many employees would like to build great things for the companies

they work for, but more often than not management won't let them.

How many of us have heard stories of employees going to management

and saying, please let us build this thing to make money for you--

and the company saying no? The most famous example is probably Steve Wozniak,

who originally wanted to build microcomputers for his then-employer, HP.

And they turned him down. On the blunderometer, this episode ranks

with IBM accepting a non-exclusive license for DOS. But I think this

happens all the time. We just don't hear about it usually,

because to prove yourself right you have to quit

and start your own company, like Wozniak did.StartupsSo these, I think, are the three big lessons open source and blogging

have to teach business: (1) that people work harder on stuff they

like, (2) that the standard office environment is very unproductive,

and (3) that bottom-up often works better than top-down.I can imagine managers at this point saying: what is this guy talking

about? What good does it do me to know that my programmers

would be more productive

working at home on their own projects? I need their asses in here

working on version 3.2 of our software, or we're never going to

make the release date.And it's true, the benefit that specific manager could derive from

the forces I've described is near zero. When I say business can

learn from open source, I don't mean any specific business can. I

mean business can learn about new conditions the same way a gene

pool does. I'm not claiming companies can get smarter, just that

dumb ones will die.So what will business look like when it has assimilated the lessons

of open source and blogging? I think the big obstacle preventing

us from seeing the future of business is the assumption that people

working for you have to be employees. But think about what's going

on underneath: the company has some money, and they pay it to the

employee in the hope that he'll make something worth more than they

paid him. Well, there are other ways to arrange that relationship.

Instead of paying the guy money as a salary, why not give it to him

as investment? Then instead of coming to your office to work on

your projects, he can work wherever he wants on projects of his own.Because few of us know any alternative, we have no idea how much

better we could do than the traditional employer-employee relationship.

Such customs evolve with glacial slowness. Our

employer-employee relationship still retains a big chunk of

master-servant DNA.

[6]I dislike being on either end of it.

I'll work my ass off for a customer, but I resent being told what

to do by a boss. And being a boss is also horribly frustrating;

half the time it's easier just to do stuff yourself than to get

someone else to do it for you.

I'd rather do almost anything than give or receive a

performance review.On top of its unpromising origins, employment

has accumulated a lot of cruft over the years. The list of what

you can't ask in job interviews is now so long that for convenience

I assume it's infinite. Within the

office you now have to walk on eggshells lest anyone

say or do

something that makes the company prey to a lawsuit. And God help

you if you fire anyone.Nothing shows more clearly that employment is not an ordinary economic

relationship than companies being sued for firing people. In any

purely economic relationship you're free to do what you want. If

you want to stop buying steel pipe from one supplier and start

buying it from another, you don't have to explain why. No one can

accuse you of unjustly switching pipe suppliers. Justice implies

some kind of paternal obligation that isn't there in

transactions between equals.Most of the legal restrictions on employers are intended to protect

employees. But you can't have action without an equal and opposite

reaction. You can't expect employers to have some kind of paternal

responsibility toward employees without putting employees in the

position of children. And that seems a bad road to go down.Next time you're in a moderately large city, drop by the main post

office and watch the body language of the people working there.

They have the same sullen resentment as children made to do

something they don't want to. Their union has exacted pay

increases and work restrictions that would have been the envy of

previous generations of postal workers, and yet they don't seem any

happier for it. It's demoralizing

to be on the receiving end of a paternalistic relationship, no

matter how cozy the terms. Just ask any teenager.I see the disadvantages of the employer-employee relationship because

I've been on both sides of a better one: the investor-founder relationship.

I wouldn't claim it's painless. When I was running a

startup, the thought of our investors used to keep me up at night.

And now that I'm an investor,

the thought of our startups keeps me

up at night. All the pain of whatever problem you're trying to

solve is still there.

But the pain hurts less when it isn't

mixed with resentment.I had the misfortune to participate in what amounted to a controlled

experiment to prove that. After Yahoo bought our startup I went

to work for them. I was doing exactly the same work, except with

bosses. And to my horror I started acting like a child. The

situation pushed buttons I'd forgotten

I had.The big advantage of investment over employment, as the examples of open

source and blogging suggest, is that people working on projects of

their own are enormously more productive. And a

startup is a project

of one's own in two senses, both of them important: it's creatively

one's own, and also economically ones's own.Google is a rare example of a big company in tune with the forces

I've described. They've tried hard to make their offices less sterile

than the usual cube farm. They give employees who do great work

large grants of stock to simulate the rewards of a startup. They

even let hackers spend 20% of their time on their own projects.Why not let people spend 100% of their time on their own projects,

and instead of trying to approximate the value of what they create,

give them the actual market value? Impossible? That is in fact

what venture capitalists do.So am I claiming that no one is going to be an employee anymore--

that everyone should go and start a startup? Of course not.

But more people could do it than do it now.

At the moment, even the smartest students leave school thinking

they have to get a job.

Actually what they need to do is make

something valuable. A job is one way to do that, but the more

ambitious ones will ordinarily be better off taking money from an

investor than an employer.Hackers tend to think business is for MBAs. But business

administration is not what you're doing in a startup. What you're

doing is business creation. And the first phase of that

is mostly product creation-- that is, hacking. That's the

hard part. It's a lot harder to create something people love than

to take something people love and figure out how to make money from

it.Another thing that keeps people away from starting startups is the

risk. Someone with kids and a mortgage should think twice before

doing it. But most young hackers have neither.And as the example of open source and blogging suggests, you'll

enjoy it more, even if you fail. You'll be working on your own

thing, instead of going to some office and doing what you're told.

There may be more pain in your own company, but it won't hurt as

much.That may be the greatest effect, in the long run, of the forces

underlying open source and blogging: finally ditching the old

paternalistic employer-employee relationship, and replacing it with

a purely economic one, between equals.

Notes[1]

Survey by Forrester Research reported in the cover story of

Business Week, 31 Jan 2005. Apparently someone believed you have to

replace the actual server in order to switch the operating system.[2]

It derives from the late Latin tripalium,

a torture device so called because it consisted of three stakes.

I don't know how the stakes were used. "Travel" has the same root.[3]

It would be much bigger news, in that sense, if the president

faced unscripted questions by giving a press conference.[4]

One measure of the incompetence of newspapers is that so many

still make you register to read stories. I have yet to find a blog

that tried that.[5]

They accepted the article, but I took so long to

send them the final version that by the time I did the section of

the magazine they'd accepted it for had disappeared in a reorganization.[6]

The word "boss" is derived from the Dutch baas, meaning

"master."Thanks to Sarah Harlin, Jessica Livingston, and Robert Morris for reading drafts of this.French TranslationRussian TranslationJapanese TranslationSpanish TranslationArabic Translation

Hiring is Obsolete

Want to start a startup? Get funded by

Y Combinator.

May 2005(This essay is derived from a talk at the Berkeley CSUA.)The three big powers on the Internet now are Yahoo, Google, and

Microsoft. Average age of their founders: 24. So it is pretty

well established now that grad students can start successful

companies. And if grad students can do it, why not undergrads?Like everything else in technology, the cost of starting a startup

has decreased dramatically. Now it's so low that it has disappeared

into the noise. The main cost of starting a Web-based

startup is food and rent. Which means it doesn't cost much more

to start a company than to be a total slacker. You can probably

start a startup on ten thousand dollars of seed funding, if you're

prepared to live on ramen.The less it costs to start a company, the less you need the permission

of investors to do it. So a lot of people will be able to start

companies now who never could have before.The most interesting subset may be those in their early twenties.

I'm not so excited about founders who have everything investors

want except intelligence, or everything except energy. The most

promising group to be liberated by the new, lower threshold are

those who have everything investors want except experience.Market RateI once claimed that nerds were unpopular

in secondary school mainly because they had better things to do

than work full-time at being popular. Some said I was just telling

people what they wanted to hear. Well, I'm now about to do that

in a spectacular way: I think undergraduates are undervalued.Or more precisely, I think few realize the huge

spread in the value of 20 year olds. Some, it's true, are not very

capable. But others are more capable than all but a handful of 30

year olds. [1]Till now the problem has always been that it's difficult to pick

them out. Every VC in the world, if they could go back in time,

would try to invest in Microsoft. But which would have then? How

many would have understood that this particular 19 year old was

Bill Gates?It's hard to judge the young because (a) they change rapidly, (b)

there is great variation between them, and (c) they're individually

inconsistent. That last one is a big problem. When you're young,

you occasionally say and do stupid things even when you're smart.

So if the algorithm is to filter out people who say stupid things,

as many investors and employers unconsciously do, you're going to

get a lot of false positives.Most organizations who hire people right out of college are only

aware of the average value of 22 year olds, which is not that high.

And so the idea for most of the twentieth century was that everyone

had to begin as a trainee in some

entry-level job. Organizations

realized there was a lot of variation in the incoming stream, but

instead of pursuing this thought they tended to suppress it, in the

belief that it was good for even the most promising kids to start

at the bottom, so they didn't get swelled heads.The most productive young people will always be undervalued

by large organizations, because the young have no performance to

measure yet, and any error in guessing their ability will tend

toward the mean.What's an especially productive 22 year old to do? One thing you

can do is go over the heads of organizations, directly to the users.

Any company that hires you is, economically, acting as a proxy for

the customer. The rate at which they value you (though they may

not consciously realize it) is an attempt to guess your value to

the user. But there's a way to appeal their judgement. If you

want, you can opt to be valued directly by users, by starting your

own company.The market is a lot more discerning than any employer. And it is

completely non-discriminatory. On the Internet, nobody knows you're

a dog. And more to the point, nobody knows you're 22. All users

care about is whether your site or software gives them what they

want. They don't care if the person behind it is a high school

kid.If you're really productive, why not make employers pay market rate

for you? Why go work as an ordinary employee for a big

company, when you could start a startup and make them buy it to get

you?When most people hear the word "startup," they think of the famous

ones that have gone public. But most startups that succeed do it

by getting bought. And usually the acquirer doesn't just want the

technology, but the people who created it as well.Often big companies buy startups before they're profitable. Obviously

in such cases they're not after revenues. What they want is the

development team and the software they've built so far. When a

startup gets bought for 2 or 3 million six months in, it's really

more of a hiring bonus than an acquisition.I think this sort of thing will happen more and more, and that it

will be better for everyone. It's obviously better for the people

who start the startup, because they get a big chunk of money up

front. But I think it will be better for the acquirers too. The

central problem in big companies, and the main reason they're so

much less productive than small companies, is the difficulty of

valuing each person's work. Buying larval startups solves that

problem for them: the acquirer doesn't pay till the developers have

proven themselves. Acquirers are protected on the downside, but

still get most of the upside.Product DevelopmentBuying startups also solves another problem afflicting big companies:

they can't do product development. Big companies are good at

extracting the value from existing products, but bad at creating

new ones.Why? It's worth studying this phenomenon in detail, because this

is the raison d'etre of startups.To start with, most big companies have some kind of turf to protect,

and this tends to warp their development decisions. For example,

Web-based applications are hot now, but

within Microsoft there must

be a lot of ambivalence about them, because the very idea of Web-based

software threatens the desktop. So any Web-based application that

Microsoft ends up with, will probably, like Hotmail, be something

developed outside the company.Another reason big companies are bad at developing new products is

that the kind of people who do that tend not to have much power in

big companies (unless they happen to be the CEO). Disruptive

technologies are developed by disruptive people. And they either

don't work for the big company, or have been outmaneuvered by yes-men

and have comparatively little influence.Big companies also lose because they usually only build one of each

thing. When you only have one Web browser, you can't do anything

really risky with it. If ten different startups design ten different

Web browsers and you take the best, you'll probably get something

better.The more general version of this problem is that there are too many

new ideas for companies to explore them all. There might be 500

startups right now who think they're making something Microsoft

might buy. Even Microsoft probably couldn't manage 500 development

projects in-house.Big companies also don't pay people the right way. People developing

a new product at a big company get paid roughly the same whether

it succeeds or fails. People at a startup expect to get rich if

the product succeeds, and get nothing if it fails. [2] So naturally

the people at the startup work a lot harder.The mere bigness of big companies is an obstacle. In startups,

developers are often forced to talk directly to users, whether they

want to or not, because there is no one else to do sales and support.

It's painful doing sales, but you learn much more from

trying to sell people something than reading what

they said in focus groups.And then of course, big companies are bad at product development

because they're bad at everything. Everything happens slower in

big companies than small ones, and product development is something

that has to happen fast, because you have to go through a lot of

iterations to get something good.TrendI think the trend of big companies buying startups will only

accelerate. One of the biggest remaining obstacles is pride. Most

companies, at least unconsciously, feel they ought to be able to

develop stuff in house, and that buying startups is to some degree

an admission of failure. And so, as people generally do with

admissions of failure, they put it off for as long as possible.

That makes the acquisition very expensive when it finally happens.What companies should do is go out and discover startups when they're

young, before VCs have puffed them up into something that costs

hundreds of millions to acquire. Much of what VCs add, the acquirer

doesn't need anyway.Why don't acquirers try to predict the companies they're going to

have to buy for hundreds of millions, and grab them early for a

tenth or a twentieth of that? Because they can't predict the winners

in advance? If they're only paying a twentieth as much, they only

have to predict a twentieth as well. Surely they can manage that.I think companies that acquire technology will gradually learn to

go after earlier stage startups. They won't necessarily buy them

outright. The solution may be some hybrid of investment and

acquisition: for example, to buy a chunk of the company and get an

option to buy the rest later.When companies buy startups, they're effectively fusing recruiting

and product development. And I think that's more efficient than

doing the two separately, because you always get people who are

really committed to what they're working on.Plus this method yields teams of developers who already work well

together. Any conflicts between them have been ironed out under

the very hot iron of running a startup. By the time the acquirer

gets them, they're finishing one another's sentences. That's

valuable in software, because so many bugs occur at the boundaries

between different people's code.InvestorsThe increasing cheapness of starting a company doesn't just give

hackers more power relative to employers. It also gives them more

power relative to investors.The conventional wisdom among VCs is that hackers shouldn't be

allowed to run their own companies. The founders are supposed to

accept MBAs as their bosses, and themselves take on some title like

Chief Technical Officer. There may be cases where this is a good

idea. But I think founders will increasingly be able to push back

in the matter of control, because they just don't need the investors'

money as much as they used to.Startups are a comparatively new phenomenon. Fairchild Semiconductor

is considered the first VC-backed startup, and they were founded

in 1959, less than fifty years ago. Measured on the time scale of

social change, what we have now is pre-beta. So we shouldn't assume

the way startups work now is the way they have to work.Fairchild needed a lot of money to get started. They had to build

actual factories. What does the first round of venture funding for

a Web-based startup get spent on today? More money can't get

software written faster; it isn't needed for facilities, because

those can now be quite cheap; all money can really buy you is sales

and marketing. A sales force is worth something, I'll admit. But

marketing is increasingly irrelevant. On the Internet, anything

genuinely good will spread by word of mouth.Investors' power comes from money. When startups need less money,

investors have less power over them. So future founders may not

have to accept new CEOs if they don't want them. The VCs will have

to be dragged kicking and screaming down this road, but like many

things people have to be dragged kicking and screaming toward, it

may actually be good for them.Google is a sign of the way things are going. As a condition of

funding, their investors insisted they hire someone old and experienced

as CEO. But from what I've heard the founders didn't just give in

and take whoever the VCs wanted. They delayed for an entire year,

and when they did finally take a CEO, they chose a guy with a PhD

in computer science.It sounds to me as if the founders are still the most powerful

people in the company, and judging by Google's performance, their

youth and inexperience doesn't seem to have hurt them. Indeed, I

suspect Google has done better than they would have if the founders

had given the VCs what they wanted, when they wanted it, and let

some MBA take over as soon as they got their first round of funding.I'm not claiming the business guys installed by VCs have no value.

Certainly they have. But they don't need to become the founders'

bosses, which is what that title CEO means. I predict that in the

future the executives installed by VCs will increasingly be COOs

rather than CEOs. The founders will run engineering directly, and

the rest of the company through the COO.The Open CageWith both employers and investors, the balance of power is slowly

shifting towards the young. And yet they seem the last to realize

it. Only the most ambitious undergrads even consider starting their

own company when they graduate. Most just want to get a job.Maybe this is as it should be. Maybe if the idea of starting a

startup is intimidating, you filter out the uncommitted. But I

suspect the filter is set a little too high. I think there are

people who could, if they tried, start successful startups, and who

instead let themselves be swept into the intake ducts of big

companies.Have you ever noticed that when animals are let out of cages, they

don't always realize at first that the door's open? Often they

have to be poked with a stick to get them out. Something similar

happened with blogs. People could have been publishing online in

1995, and yet blogging has only really taken off in the last couple

years. In 1995 we thought only professional writers were entitled

to publish their ideas, and that anyone else who did was a crank.

Now publishing online is becoming so popular that everyone wants

to do it, even print journalists. But blogging has not taken off

recently because of any technical innovation; it just took eight

years for everyone to realize the cage was open.I think most undergrads don't realize yet that the economic cage

is open. A lot have been told by their parents that the route to

success is to get a good job. This was true when their parents

were in college, but it's less true now. The route to success is

to build something valuable, and you don't have to be working for

an existing company to do that. Indeed, you can often do it better

if you're not.When I talk to undergrads, what surprises me most about them is how

conservative they are. Not politically, of course. I mean they

don't seem to want to take risks. This is a mistake, because the

younger you are, the more risk you can take.RiskRisk and reward are always proportionate. For example, stocks are

riskier than bonds, and over time always have greater returns. So

why does anyone invest in bonds? The catch is that phrase "over

time." Stocks will generate greater returns over thirty years, but

they might lose value from year to year. So what you should invest

in depends on how soon you need the money. If you're young, you

should take the riskiest investments you can find.All this talk about investing may seem very theoretical. Most

undergrads probably have more debts than assets. They may feel

they have nothing to invest. But that's not true: they have their

time to invest, and the same rule about risk applies there. Your

early twenties are exactly the time to take insane career risks.The reason risk is always proportionate to reward is that market

forces make it so. People will pay extra for stability. So if you

choose stability-- by buying bonds, or by going to work for a big

company-- it's going to cost you.Riskier career moves pay better on average, because there is less

demand for them. Extreme choices like starting a startup are so

frightening that most people won't even try. So you don't end up

having as much competition as you might expect, considering the

prizes at stake.The math is brutal. While perhaps 9 out of 10 startups fail, the

one that succeeds will pay the founders more than 10 times what

they would have made in an ordinary job. [3]

That's the sense in

which startups pay better "on average."Remember that. If you start a startup, you'll probably fail. Most

startups fail. It's the nature of the business. But it's not

necessarily a mistake to try something that has a 90% chance of

failing, if you can afford the risk. Failing at 40, when you have

a family to support, could be serious. But if you fail at 22, so

what? If you try to start a startup right out of college and it

tanks, you'll end up at 23 broke and a lot smarter. Which, if you

think about it, is roughly what you hope to get from a graduate

program.Even if your startup does tank, you won't harm your prospects with

employers. To make sure I asked some friends who work for big

companies. I asked managers at Yahoo, Google, Amazon, Cisco and

Microsoft how they'd feel about two candidates, both 24, with equal

ability, one who'd tried to start a startup that tanked, and another

who'd spent the two years since college working as a developer at

a big company. Every one responded that they'd prefer the guy who'd

tried to start his own company. Zod Nazem, who's in charge of

engineering at Yahoo, said:

I actually put more value on the guy with the failed

startup. And you can quote me!

So there you have it. Want to get hired by Yahoo? Start your own

company.The Man is the CustomerIf even big employers think highly of young hackers who start

companies, why don't more do it? Why are undergrads so conservative?

I think it's because they've spent so much time in institutions.The first twenty years of everyone's life consists of being piped

from one institution to another. You probably didn't have much

choice about the secondary schools you went to. And after high

school it was probably understood that you were supposed to go to

college. You may have had a few different colleges to choose

between, but they were probably pretty similar. So by this point

you've been riding on a subway line for twenty years, and the next

stop seems to be a job.Actually college is where the line ends. Superficially, going to

work for a company may feel like just the next in a series of

institutions, but underneath, everything is different. The end of

school is the fulcrum of your life, the point where you go from

net consumer to net producer.The other big change is that now, you're steering. You can go

anywhere you want. So it may be worth standing back and understanding

what's going on, instead of just doing the default thing.All through college, and probably long before that, most undergrads

have been thinking about what employers want. But what really

matters is what customers want, because they're the ones who give

employers the money to pay you.So instead of thinking about what employers want, you're probably

better off thinking directly about what users want. To the extent

there's any difference between the two, you can even use that to

your advantage if you start a company of your own. For example,

big companies like docile conformists. But this is merely an

artifact of their bigness, not something customers need.Grad SchoolI didn't consciously realize all this when I was graduating from

college-- partly because I went straight to grad school. Grad

school can be a pretty good deal, even if you think of one day

starting a startup. You can start one when you're done, or even

pull the ripcord part way through, like the founders of Yahoo and

Google.Grad school makes a good launch pad for startups, because you're

collected together with a lot of smart people, and you have bigger

chunks of time to work on your own projects than an undergrad or

corporate employee would. As long as you have a fairly tolerant

advisor, you can take your time developing an idea before turning

it into a company. David Filo and Jerry Yang started the Yahoo

directory in February 1994 and were getting a million hits a day

by the fall, but they didn't actually drop out of grad school and

start a company till March 1995.You could also try the startup first, and if it doesn't work, then

go to grad school. When startups tank they usually do it fairly

quickly. Within a year you'll know if you're wasting your time.If it fails, that is. If it succeeds, you may have to delay grad

school a little longer. But you'll have a much more enjoyable life

once there than you would on a regular grad student stipend.ExperienceAnother reason people in their early twenties don't start startups

is that they feel they don't have enough experience. Most investors

feel the same.I remember hearing a lot of that word "experience" when I was in

college. What do people really mean by it? Obviously it's not the

experience itself that's valuable, but something it changes in your

brain. What's different about your brain after you have "experience,"

and can you make that change happen faster?I now have some data on this, and I can tell you what tends to be

missing when people lack experience. I've said that every

startup needs three things: to start with good people,

to make something users want, and not to spend too much money. It's

the middle one you get wrong when you're inexperienced. There are

plenty of undergrads with enough technical skill to write good

software, and undergrads are not especially prone to waste money.

If they get something wrong, it's usually not realizing they have

to make something people want.This is not exclusively a failing of the young. It's common for

startup founders of all ages to build things no one wants.Fortunately, this flaw should be easy to fix. If undergrads were

all bad programmers, the problem would be a lot harder. It can

take years to learn how to program. But I don't think it takes

years to learn how to make things people want. My hypothesis is

that all you have to do is smack hackers on the side of the head

and tell them: Wake up. Don't sit here making up a priori theories

about what users need. Go find some users and see what they need.Most successful startups not only do something very specific, but

solve a problem people already know they have.The big change that "experience" causes in your brain is learning

that you need to solve people's problems. Once you grasp that, you

advance quickly to the next step, which is figuring out what those

problems are. And that takes some effort, because the way software

actually gets used, especially by the people who pay the most for

it, is not at all what you might expect. For example, the stated

purpose of Powerpoint is to present ideas. Its real role is to

overcome people's fear of public speaking. It allows you to give

an impressive-looking talk about nothing, and it causes the audience

to sit in a dark room looking at slides, instead of a bright one

looking at you.This kind of thing is out there for anyone to see. The key is to

know to look for it-- to realize that having an idea for a startup

is not like having an idea for a class project. The goal in a

startup is not to write a cool piece of software. It's to make

something people want. And to do that you have to look at users--

forget about hacking, and just look at users. This can be quite a

mental adjustment, because little if any of the software you write

in school even has users. A few steps before a Rubik's Cube is solved, it still looks like a

mess. I think there are a lot of undergrads whose brains are in a

similar position: they're only a few steps away from being able to

start successful startups, if they wanted to, but they don't realize

it. They have more than enough technical skill. They just haven't

realized yet that the way to create wealth is to make what users

want, and that employers are just proxies for users in which risk

is pooled.If you're young and smart, you don't need either of those. You

don't need someone else to tell you what users want, because you

can figure it out yourself. And you don't want to pool risk, because

the younger you are, the more risk you should take.A Public Service MessageI'd like to conclude with a joint message from me and your parents.

Don't drop out of college to start a startup. There's no rush.

There will be plenty of time to start companies after you graduate.

In fact, it may be just as well to go work for an existing company

for a couple years after you graduate, to learn how companies work.And yet, when I think about it, I can't imagine telling Bill Gates

at 19 that he should wait till he graduated to start a company.

He'd have told me to get lost. And could I have honestly claimed

that he was harming his future-- that he was learning less by working

at ground zero of the microcomputer revolution than he would have

if he'd been taking classes back at Harvard? No, probably not.And yes, while it is probably true that you'll learn some valuable

things by going to work for an existing company for a couple years

before starting your own, you'd learn a thing or two running your

own company during that time too.The advice about going to work for someone else would get an even

colder reception from the 19 year old Bill Gates. So I'm supposed

to finish college, then go work for another company for two years,

and then I can start my own? I have to wait till I'm 23? That's

four years. That's more than twenty percent of my life so

far. Plus in four years it will be way too late to make money

writing a Basic interpreter for the Altair.And he'd be right. The Apple II was launched just two years later.

In fact, if Bill had finished college and gone to work for another

company as we're suggesting, he might well have gone to work for

Apple. And while that would probably have been better for all of

us, it wouldn't have been better for him.So while I stand by our responsible advice to finish college and

then go work for a while before starting a startup, I have to admit

it's one of those things the old tell the young, but don't expect

them to listen to. We say this sort of thing mainly so we can claim

we warned you. So don't say I didn't warn you.

Notes[1]

The average B-17 pilot in World War II was in his early twenties.

(Thanks to Tad Marko for pointing this out.)[2] If a company tried to pay employees this way, they'd be called

unfair. And yet when they buy some startups and not others, no one

thinks of calling that unfair.

[3] The 1/10 success rate for startups is a bit of an urban legend.

It's suspiciously neat. My guess is the odds are slightly worse.Thanks to Jessica Livingston for reading drafts of this, to

the friends I promised anonymity to for their opinions about hiring,

and to Karen Nguyen and the Berkeley CSUA for organizing this talk.Russian TranslationRomanian TranslationJapanese Translation

If you liked this, you may also like

Hackers & Painters.

The Submarine

Breaking News:

The Suit is Back!

April 2005"Suits make a corporate comeback," says the New

York Times. Why does this sound familiar? Maybe because

the suit was also back in February,

September

2004, June

2004, March

2004, September

2003,

November

2002,

April 2002,

and February

2002.

Why do the media keep running stories saying suits are back? Because

PR firms tell

them to. One of the most surprising things I discovered

during my brief business career was the existence of the PR industry,

lurking like a huge, quiet submarine beneath the news. Of the

stories you read in traditional media that aren't about politics,

crimes, or disasters, more than half probably come from PR firms.I know because I spent years hunting such "press hits." Our startup spent

its entire marketing budget on PR: at a time when we were assembling

our own computers to save money, we were paying a PR firm $16,000

a month. And they were worth it. PR is the news equivalent of

search engine optimization; instead of buying ads, which readers

ignore, you get yourself inserted directly into the stories. [1]Our PR firm

was one of the best in the business. In 18 months, they got press

hits in over 60 different publications.

And we weren't the only ones they did great things for.

In 1997 I got a call from another

startup founder considering hiring them to promote his company. I

told him they were PR gods, worth every penny of their outrageous

fees. But I remember thinking his company's name was odd.

Why call an auction site "eBay"?

SymbiosisPR is not dishonest. Not quite. In fact, the reason the best PR

firms are so effective is precisely that they aren't dishonest.

They give reporters genuinely valuable information. A good PR firm

won't bug reporters just because the client tells them to; they've

worked hard to build their credibility with reporters, and they

don't want to destroy it by feeding them mere propaganda.If anyone is dishonest, it's the reporters. The main reason PR

firms exist is that reporters are lazy. Or, to put it more nicely,

overworked. Really they ought to be out there digging up stories

for themselves. But it's so tempting to sit in their offices and

let PR firms bring the stories to them. After all, they know good

PR firms won't lie to them.A good flatterer doesn't lie, but tells his victim selective truths

(what a nice color your eyes are). Good PR firms use the same

strategy: they give reporters stories that are true, but whose truth

favors their clients.For example, our PR firm often pitched stories about how the Web

let small merchants compete with big ones. This was perfectly true.

But the reason reporters ended up writing stories about this

particular truth, rather than some other one, was that small merchants

were our target market, and we were paying the piper.Different publications vary greatly in their reliance on PR firms.

At the bottom of the heap are the trade press, who make most of

their money from advertising and would give the magazines away for

free if advertisers would let them. [2] The average

trade publication is a bunch of ads, glued together by just enough

articles to make it look like a magazine. They're so desperate for

"content" that some will print your press releases almost verbatim,

if you take the trouble to write them to read like articles.At the other extreme are publications like the New York Times

and the Wall Street Journal. Their reporters do go out and

find their own stories, at least some of the time. They'll listen

to PR firms, but briefly and skeptically. We managed to get press

hits in almost every publication we wanted, but we never managed

to crack the print edition of the Times. [3]The weak point of the top reporters is not laziness, but vanity.

You don't pitch stories to them. You have to approach them as if

you were a specimen under their all-seeing microscope, and make it

seem as if the story you want them to run is something they thought

of themselves.Our greatest PR coup was a two-part one. We estimated, based on

some fairly informal math, that there were about 5000 stores on the

Web. We got one paper to print this number, which seemed neutral

enough. But once this "fact" was out there in print, we could quote

it to other publications, and claim that with 1000 users we had 20%

of the online store market.This was roughly true. We really did have the biggest share of the

online store market, and 5000 was our best guess at its size. But

the way the story appeared in the press sounded a lot more definite.Reporters like definitive statements. For example, many of the

stories about Jeremy Jaynes's conviction say that he was one of the

10 worst spammers. This "fact" originated in Spamhaus's ROKSO list,

which I think even Spamhaus would admit is a rough guess at the top

spammers. The first stories about Jaynes cited this source, but

now it's simply repeated as if it were part of the indictment.

[4]All you can say with certainty about Jaynes is that he was a fairly

big spammer. But reporters don't want to print vague stuff like

"fairly big." They want statements with punch, like "top ten." And

PR firms give them what they want.

Wearing suits, we're told, will make us

3.6

percent more productive.BuzzWhere the work of PR firms really does get deliberately misleading is in

the generation of "buzz." They usually feed the same story to

several different publications at once. And when readers see similar

stories in multiple places, they think there is some important trend

afoot. Which is exactly what they're supposed to think.When Windows 95 was launched, people waited outside stores

at midnight to buy the first copies. None of them would have been

there without PR firms, who generated such a buzz in

the news media that it became self-reinforcing, like a nuclear chain

reaction.I doubt PR firms realize it yet, but the Web makes it possible to

track them at work. If you search for the obvious phrases, you

turn up several efforts over the years to place stories about the

return of the suit. For example, the Reuters article

that got picked up by USA

Today in September 2004. "The suit is back," it begins.Trend articles like this are almost always the work of

PR firms. Once you know how to read them, it's straightforward to

figure out who the client is. With trend stories, PR firms usually

line up one or more "experts" to talk about the industry generally.

In this case we get three: the NPD Group, the creative director of

GQ, and a research director at Smith Barney. [5] When

you get to the end of the experts, look for the client. And bingo,

there it is: The Men's Wearhouse.Not surprising, considering The Men's Wearhouse was at that moment

running ads saying "The Suit is Back." Talk about a successful

press hit-- a wire service article whose first sentence is your own

ad copy.The secret to finding other press hits from a given pitch

is to realize that they all started from the same document back at

the PR firm. Search for a few key phrases and the names of the

clients and the experts, and you'll turn up other variants of this

story.Casual

fridays are out and dress codes are in writes Diane E. Lewis

in The Boston Globe. In a remarkable coincidence, Ms. Lewis's

industry contacts also include the creative director of GQ.Ripped jeans and T-shirts are out, writes Mary Kathleen Flynn in

US News & World Report. And she too knows the

creative director of GQ.Men's suits

are back writes Nicole Ford in Sexbuzz.Com ("the ultimate men's

entertainment magazine").Dressing

down loses appeal as men suit up at the office writes Tenisha

Mercer of The Detroit News.

Now that so many news articles are online, I suspect you could find

a similar pattern for most trend stories placed by PR firms. I

propose we call this new sport "PR diving," and I'm sure there are

far more striking examples out there than this clump of five stories.OnlineAfter spending years chasing them, it's now second nature

to me to recognize press hits for what they are. But before we

hired a PR firm I had no idea where articles in the mainstream media

came from. I could tell a lot of them were crap, but I didn't

realize why.Remember the exercises in critical reading you did in school, where

you had to look at a piece of writing and step back and ask whether

the author was telling the whole truth? If you really want to be

a critical reader, it turns out you have to step back one step

further, and ask not just whether the author is telling the truth,

but why he's writing about this subject at all.Online, the answer tends to be a lot simpler. Most people who

publish online write what they write for the simple reason that

they want to. You

can't see the fingerprints of PR firms all over the articles, as

you can in so many print publications-- which is one of the reasons,

though they may not consciously realize it, that readers trust

bloggers more than Business Week.I was talking recently to a friend who works for a

big newspaper. He thought the print media were in serious trouble,

and that they were still mostly in denial about it. "They think

the decline is cyclic," he said. "Actually it's structural."In other words, the readers are leaving, and they're not coming

back.

Why? I think the main reason is that the writing online is more honest.

Imagine how incongruous the New York Times article about

suits would sound if you read it in a blog:

The urge to look corporate-- sleek, commanding,

prudent, yet with just a touch of hubris on your well-cut sleeve--

is an unexpected development in a time of business disgrace.

The problem

with this article is not just that it originated in a PR firm.

The whole tone is bogus. This is the tone of someone writing down

to their audience.Whatever its flaws, the writing you find online

is authentic. It's not mystery meat cooked up

out of scraps of pitch letters and press releases, and pressed into

molds of zippy

journalese. It's people writing what they think.I didn't realize, till there was an alternative, just how artificial

most of the writing in the mainstream media was. I'm not saying

I used to believe what I read in Time and Newsweek. Since high

school, at least, I've thought of magazines like that more as

guides to what ordinary people were being

told to think than as

sources of information. But I didn't realize till the last

few years that writing for publication didn't have to mean writing

that way. I didn't realize you could write as candidly and

informally as you would if you were writing to a friend.Readers aren't the only ones who've noticed the

change. The PR industry has too.

A hilarious article

on the site of the PR Society of America gets to the heart of the

matter:

Bloggers are sensitive about becoming mouthpieces

for other organizations and companies, which is the reason they

began blogging in the first place.

PR people fear bloggers for the same reason readers

like them. And that means there may be a struggle ahead. As

this new kind of writing draws readers away from traditional media, we

should be prepared for whatever PR mutates into to compensate.

When I think

how hard PR firms work to score press hits in the traditional

media, I can't imagine they'll work any less hard to feed stories

to bloggers, if they can figure out how.

Notes[1] PR has at least

one beneficial feature: it favors small companies. If PR didn't

work, the only alternative would be to advertise, and only big

companies can afford that.[2] Advertisers pay

less for ads in free publications, because they assume readers

ignore something they get for free. This is why so many trade

publications nominally have a cover price and yet give away free

subscriptions with such abandon.[3] Different sections

of the Times vary so much in their standards that they're

practically different papers. Whoever fed the style section reporter

this story about suits coming back would have been sent packing by

the regular news reporters.[4] The most striking

example I know of this type is the "fact" that the Internet worm

of 1988 infected 6000 computers. I was there when it was cooked up,

and this was the recipe: someone guessed that there were about

60,000 computers attached to the Internet, and that the worm might

have infected ten percent of them.Actually no one knows how many computers the worm infected, because

the remedy was to reboot them, and this destroyed all traces. But

people like numbers. And so this one is now replicated

all over the Internet, like a little worm of its own.[5] Not all were

necessarily supplied by the PR firm. Reporters sometimes call a few

additional sources on their own, like someone adding a few fresh

vegetables to a can of soup.

Thanks to Ingrid Basset, Trevor Blackwell, Sarah Harlin, Jessica

Livingston, Jackie McDonough, Robert Morris, and Aaron Swartz (who

also found the PRSA article) for reading drafts of this.Correction: Earlier versions used a recent

Business Week article mentioning del.icio.us as an example

of a press hit, but Joshua Schachter tells me

it was spontaneous.The Web is a Writing EnvironmentA Sell-Out's TaleHow to Pitch BloggersBlogging for Milk7 Habits of Highly Effective Blog PRPR People Need To Learn To Deal With New GatekeepersMarqui Blogosphere ProgramPR WatchReal Men ExfoliateHow the News is MadeJanuary 2006: The suit is back yet againThe Decline of the TieJapanese Translation

If you liked this, you may also like

Hackers & Painters.

Why Smart People Have Bad Ideas

Want to start a startup? Get funded by

Y Combinator.

April 2005This summer, as an

experiment, some

friends and I are giving seed

funding to a bunch of new startups. It's an experiment because

we're prepared to fund younger founders than most investors would.

That's why we're doing it during the summer—so even college

students can participate.We know from Google and Yahoo that grad students can start successful

startups. And we know from experience that some undergrads are as

capable as most grad students. The accepted age for startup founders

has been creeping downward. We're trying to find the lower bound.

The deadline has now passed, and we're sifting through 227 applications.

We expected to divide them into two categories, promising

and unpromising. But we soon saw we needed a third: promising

people with unpromising ideas.

[1]The Artix PhaseWe should have expected this. It's very common for a group of

founders to go through one lame idea before realizing that a startup

has to make something people will pay for. In fact, we ourselves

did.Viaweb wasn't the first startup Robert Morris and I started. In

January 1995, we and a couple friends started a company called

Artix. The plan was to put art galleries on the Web. In retrospect,

I wonder how we could have wasted our time on anything so stupid.

Galleries are not especially excited about being on

the Web even now, ten years later. They don't want to have their

stock visible to any random visitor, like an antique store.

[2]Besides which, art dealers are the most technophobic people on

earth. They didn't become art dealers after a difficult choice

between that and a career in the hard sciences. Most of them had

never seen the Web before we came to tell them why they should be

on it. Some didn't even have computers. It doesn't do justice to

the situation to describe it as a hard sell; we soon sank

to building sites for free, and it was hard to convince galleries

even to do that.Gradually it dawned on us that

instead of trying to make Web sites for

people who didn't want them, we could make sites for

people who did. In fact, software that would let people who wanted

sites make their own. So we ditched Artix and

started a new company, Viaweb, to make software for building online stores.

That one succeeded.We're in good company here. Microsoft was not the first company

Paul Allen and Bill Gates started either. The first was called

Traf-o-data. It does not seem to have done as well as Micro-soft.

In Robert's defense, he was skeptical about Artix. I dragged him

into it.

[3]

But there were moments when he was optimistic. And

if we, who were 29 and 30 at the time, could get excited about such

a thoroughly boneheaded idea, we should not be surprised that hackers

aged 21 or 22 are pitching us ideas with little hope of making money.The Still Life EffectWhy does this happen? Why do good hackers have bad business ideas?Let's look at our case. One reason we had such a lame idea was

that it was the first thing we thought of. I was in New York trying

to be a starving artist at the time (the starving part is actually

quite easy), so I was haunting galleries anyway. When I learned

about the Web, it seemed natural to mix the two. Make Web sites

for galleries—that's the ticket!If you're going to spend years working on something, you'd think

it might be wise to spend at least a couple days considering different

ideas, instead of going with the first that comes into your head.

You'd think. But people don't. In fact, this is a constant problem

when you're painting still lifes. You plonk down a bunch of stuff

on a table, and maybe spend five or ten minutes rearranging it to

look interesting. But you're so impatient to get started painting

that ten minutes of rearranging feels very long. So you start

painting. Three days later, having spent twenty hours staring at

it, you're kicking yourself for having set up such an awkward and

boring composition, but by then it's too late.Part of the problem is that big projects tend to grow out of small

ones. You set up a still life to make a quick sketch when you have

a spare hour, and days later you're still working on it. I once

spent a month painting three versions of a still life I set up in

about four minutes. At each point (a day, a week, a month) I thought

I'd already put in so much time that it was too late to change.So the biggest cause of bad ideas is the still life effect: you

come up with a random idea, plunge into it, and then at each point

(a day, a week, a month) feel you've put so much time into it that

this must be the idea.How do we fix that? I don't think we should discard plunging.

Plunging into an idea is a good thing. The solution is at the other

end: to realize that having invested time in something doesn't make

it good.This is clearest in the case of names. Viaweb was originally

called Webgen, but we discovered someone else had a product called

that. We were so attached to our name that we offered him 5%

of the company if he'd let us have it. But he wouldn't, so

we had to think of another.

[4]

The best we could do was Viaweb,

which we disliked at first. It was like having a new mother. But

within three days we loved it, and Webgen sounded lame and

old-fashioned.If it's hard to change something so simple as a name, imagine

how hard it is to garbage-collect an idea. A name only has one

point of attachment into your head. An idea for a company gets

woven into your thoughts. So you must consciously discount for

that. Plunge in, by all means, but remember later to look at your

idea in the harsh light of morning and ask: is this something people

will pay for? Is this, of all the things we could make, the thing

people will pay most for?MuckThe second mistake we made with Artix is also very common. Putting

galleries on the Web seemed cool.One of the most valuable things my father taught me is an old

Yorkshire saying: where there's muck, there's brass. Meaning that

unpleasant work pays. And more to the point here, vice versa. Work

people like doesn't pay well, for reasons of supply and demand.

The most extreme case is developing programming languages, which

doesn't pay at all, because people like it so much they do it for

free.When we started Artix, I was still ambivalent about business. I

wanted to keep one foot in the art world. Big, big, mistake. Going

into business is like a hang-glider launch: you'd better do it

wholeheartedly, or not at all. The purpose of a company, and a

startup especially, is to make money. You can't have divided

loyalties.Which is not to say that you have to do the most disgusting sort

of work, like spamming, or starting a company whose only purpose

is patent litigation. What I mean is, if you're starting a company

that will do something cool, the aim had better be to make money

and maybe be cool, not to be cool and maybe make money.It's hard enough to make money that you can't do it by accident.

Unless it's your first priority, it's unlikely to happen at all.HyenasWhen I probe our motives with Artix, I see a third mistake: timidity.

If you'd proposed at the time that we go into the e-commerce business,

we'd have found the idea terrifying. Surely a field like that would

be dominated by fearsome startups with five million dollars of VC

money each. Whereas we felt pretty sure that we could hold our own

in the slightly less competitive business of generating Web sites

for art galleries.We erred ridiculously far on the side of safety. As it turns out,

VC-backed startups are not that fearsome. They're too busy trying

to spend all that

money to get software written. In 1995, the

e-commerce business was very competitive as measured in press

releases, but not as measured in software. And really it never

was. The big fish like Open Market (rest their souls) were just

consulting companies pretending to be product companies

[5], and

the offerings at our end of the market were a couple hundred lines

of Perl scripts. Or could have been implemented as a couple hundred

lines of Perl; in fact they were probably tens of thousands of lines

of C++ or Java. Once we actually took the plunge into e-commerce,

it turned out to be surprisingly easy to compete.So why were we afraid? We felt we were good at programming, but

we lacked confidence in our ability to do a mysterious, undifferentiated

thing we called "business." In fact there is no such thing as

"business." There's selling, promotion, figuring out what people

want, deciding how much to charge, customer support, paying your

bills, getting customers to pay you, getting incorporated, raising

money, and so on. And the combination is not as hard as it seems,

because some tasks (like raising money and getting incorporated)

are an O(1) pain in the ass, whether you're big or small, and others

(like selling and promotion) depend more on energy and imagination

than any kind of special training.Artix was like a hyena, content to survive on carrion because we

were afraid of the lions. Except the lions turned out not to have

any teeth, and the business of putting galleries online barely

qualified as carrion.A Familiar ProblemSum up all these sources of error, and it's no wonder we had such

a bad idea for a company. We did the first thing we thought of;

we were ambivalent about being in business at all; and we deliberately

chose an impoverished market to avoid competition.Looking at the applications for the Summer Founders Program, I see

signs of all three. But the first is by far the biggest problem.

Most of the groups applying have not stopped to ask: of all the

things we could do, is this the one with the best chance of

making money?If they'd already been through their Artix phase, they'd have learned

to ask that. After the reception we got from art dealers, we were

ready to. This time, we thought, let's make something people want.Reading the Wall Street Journal for a week should give anyone

ideas for two or three new startups. The articles are full of

descriptions of problems that need to be solved. But most of the

applicants don't seem to have looked far for ideas.We expected the most common proposal to be for multiplayer games.

We were not far off: this was the second most common.

The most common was some combination of a blog, a calendar,

a dating site, and Friendster. Maybe there is some new killer app

to be discovered here, but it seems perverse to go poking around

in this fog when there are valuable, unsolved problems lying about

in the open for anyone to see. Why did no one propose a new scheme

for micropayments? An ambitious project, perhaps, but I can't

believe we've considered every alternative. And newspapers and

magazines are (literally) dying for a solution.Why did so few applicants really think about what customers want?

I think the problem with many, as with people in their early twenties

generally, is that they've been trained their whole lives to jump

through predefined hoops. They've spent 15-20 years solving problems

other people have set for them. And how much time deciding what

problems would be good to solve? Two or three course projects?

They're good at solving problems, but bad at choosing them.But that, I'm convinced, is just the effect of training. Or more

precisely, the effect of grading. To make grading efficient,

everyone has to solve the same problem, and that means it has to

be decided in advance. It would be great if schools taught students

how to choose problems as well as how to solve them, but I don't

know how you'd run such a class in practice.Copper and TinThe good news is, choosing problems is something that can be learned.

I know that from experience. Hackers can learn to make things

customers want.

[6]This is a controversial view. One expert on "entrepreneurship"

told me that any startup had to include business people, because

only they could focus on what customers wanted. I'll probably

alienate this guy forever by quoting him, but I have to risk it,

because his email was such a perfect example of this view:

80% of MIT spinoffs succeed provided they

have at least one management person in the team at the start. The

business person represents the "voice of the customer" and that's

what keeps the engineers and product development on track.

This is, in my opinion, a crock. Hackers are perfectly capable of

hearing the voice of the customer without a business person to

amplify the signal for them. Larry Page and Sergey Brin were grad

students in computer science, which presumably makes them "engineers."

Do you suppose Google is only good because they had some business

guy whispering in their ears what customers wanted? It seems to

me the business guys who did the most for Google were the ones who

obligingly flew Altavista into a hillside just as Google was getting

started.The hard part about figuring out what customers want is figuring

out that you need to figure it out. But that's something you can

learn quickly. It's like seeing the other interpretation of an

ambiguous picture. As soon as someone tells you there's a rabbit

as well as a duck, it's hard not to see it.And compared to the sort of problems hackers are used to solving,

giving customers what they want is easy. Anyone who can write an

optimizing compiler can design a UI that doesn't confuse users,

once they choose to focus on that problem. And once you

apply that kind of brain power to petty but profitable questions,

you can create wealth very rapidly.That's the essence of a startup: having brilliant people do work

that's beneath them. Big companies try to hire the right person

for the job. Startups win because they don't—because they take

people so smart that they would in a big company be doing "research,"

and set them to work instead on problems of the most immediate and

mundane sort. Think Einstein designing refrigerators.

[7]If you want to learn what people want, read

Dale Carnegie's How to Win Friends and Influence People.

[8]

When a friend recommended this book, I couldn't believe he was

serious. But he insisted it was good, so I read it, and he was

right. It deals with the most difficult problem in human experience:

how to see things from other people's point of view, instead of

thinking only of yourself.Most smart people don't do that very well. But adding this ability

to raw brainpower is like adding tin to copper. The result is

bronze, which is so much harder that it seems a different metal.A hacker who has learned what to make, and not just how to make,

is extraordinarily powerful. And not just at making money: look

what a small group of volunteers has achieved with Firefox.Doing an Artix teaches you to make something people want in the

same way that not drinking anything would teach you how much you

depend on water. But it would be more convenient for all involved

if the Summer Founders didn't learn this on our dime—if they could

skip the Artix phase and go right on to make something customers

wanted. That, I think, is going to be the real experiment this

summer. How long will it take them to grasp this? We decided

we ought to have T-Shirts for the SFP, and we'd been thinking about

what to print on the back. Till now we'd been planning to use

If you can read this, I should be working.

but now we've decided it's going to be

Make something people want.

Notes[1]

SFP applicants: please don't assume that not being accepted

means we think your idea is bad. Because we want to keep the

number of startups small this first summer, we're going to have

to turn down some good proposals too.[2]

Dealers try to give each customer the impression that the stuff

they're showing him is something special that only a few people

have seen, when in fact it may have been sitting in their racks for

years while they tried to unload it on buyer after buyer.[3]

On the other hand, he was skeptical about Viaweb too. I have

a precise measure of that, because at one point in the first couple

months we made a bet: if he ever made a million dollars out of

Viaweb, he'd get his ear pierced. We didn't let him

off, either.[4]

I wrote a program to generate all the combinations of "Web"

plus a three letter word. I learned from this that most three

letter words are bad: Webpig, Webdog, Webfat, Webzit, Webfug. But

one of them was Webvia; I swapped them to make Viaweb.[5]

It's much easier to sell services than a product, just as it's

easier to make a living playing at weddings than by selling recordings.

But the margins are greater on products. So during the

Bubble a lot of companies used consulting to generate revenues

they could attribute to the sale of products, because it made a

better story for an IPO.[6]

Trevor Blackwell presents the following recipe for a startup:

"Watch people who have money to spend, see what they're wasting

their time on, cook up a solution, and try selling it to them. It's

surprising how small a problem can be and still provide a profitable

market for a solution."[7]

You need to offer especially large rewards to get great people

to do tedious work. That's why startups always pay equity rather

than just salary.[8]

Buy an old

copy from the 1940s or 50s instead of the current edition, which has been

rewritten to suit present fashions. The original edition contained

a few unPC ideas, but it's always better to read an original book,

bearing in mind that it's a book from a past era, than to read a

new version sanitized for your protection.Thanks to Bill Birch, Trevor Blackwell, Jessica Livingston,

and Robert Morris for reading drafts of this.Russian TranslationItalian TranslationJapanese Translation

If you liked this, you may also like

Hackers & Painters.

Return of the Mac

March 2005All the best hackers

I know are gradually switching to Macs. My

friend Robert said his whole research group at MIT recently bought

themselves Powerbooks. These guys are not the graphic designers

and grandmas who were buying Macs at Apple's low point in the

mid 1990s. They're about as hardcore OS hackers as you can get.The reason, of course, is OS X. Powerbooks are beautifully designed

and run FreeBSD. What more do you need to know?I got a Powerbook at the end of last year. When my IBM Thinkpad's

hard disk died soon after, it became my only laptop. And when my

friend Trevor showed up at my house recently, he was carrying a

Powerbook identical to mine.For most of us, it's not a switch to Apple, but a return. Hard as

this was to believe in the mid 90s, the Mac was in its time the

canonical hacker's computer.In the fall of 1983,

the professor in one of my college CS classes got up and announced,

like a prophet, that there would soon be a computer with half a MIPS

of processing power that would fit under an airline seat and cost

so little that we could save enough to buy one from a summer job.

The whole room gasped.

And when the Mac appeared, it was even better than we'd hoped. It

was small and powerful and cheap, as promised. But it was also

something we'd never considered a computer could be: fabulously

well designed.I had to have one. And I wasn't alone. In the mid to late 1980s,

all the hackers I knew were either writing software for the Mac,

or wanted to. Every futon sofa in Cambridge seemed to have the

same fat white book lying open on it. If you turned it over, it

said "Inside Macintosh." Then came Linux and FreeBSD, and hackers, who follow the most

powerful OS wherever it leads, found themselves switching to Intel

boxes. If you cared about design, you could buy a Thinkpad, which

was at least not actively repellent, if you could get the Intel and

Microsoft stickers

off the front. [1]With OS X, the hackers are back. When I walked into the Apple store

in Cambridge, it was like coming home. Much

was changed, but there was still that Apple coolness in the air,

that feeling that the show was being run by someone who really

cared, instead of random corporate deal-makers.So what, the business world may say. Who cares if hackers like Apple

again? How big is the hacker market, after all?Quite small, but important out of proportion to its size. When it

comes to computers, what hackers are doing now, everyone will be

doing in ten years. Almost all technology, from Unix to bitmapped

displays to the Web, became popular first within CS departments and

research labs, and gradually spread to the rest of the world.I remember telling my father back in 1986 that there was a new kind

of computer called a Sun that was a serious Unix machine, but

so small and cheap that you could

have one of your own to sit in front of, instead of sitting in front

of a VT100 connected to a single central Vax. Maybe, I suggested,

he should buy some stock in this company. I think he really wishes

he'd listened.In 1994 my friend Koling wanted to talk to his girlfriend in Taiwan,

and to save long-distance bills he wrote some software that would

convert sound to data packets that could be sent over the Internet.

We weren't sure at the time whether this was a proper use of the

Internet, which was still then a quasi-government entity. What he

was doing is now called VoIP, and it is a huge and rapidly growing

business.If you want to know what ordinary people will be doing with computers

in ten years, just walk around the CS department at a good university.

Whatever they're doing, you'll be doing.In the matter of "platforms" this tendency is even more pronounced,

because novel software originates with

great hackers, and they tend

to write it first for whatever computer they personally use. And

software sells hardware. Many if not most of the initial sales of

the Apple II came from people who bought one to run VisiCalc. And

why did Bricklin and Frankston write VisiCalc for the Apple II?

Because they personally liked it. They could have chosen any machine

to make into a star.If you want to attract hackers to write software that will sell

your hardware, you have to make it something that they themselves

use. It's not enough to make it "open." It has to be open and

good.And open and good is what Macs are again, finally. The intervening

years have created a situation that is, as far as I know, without

precedent: Apple is popular at the low end and the high end, but

not in the middle. My seventy year old mother has a Mac laptop.

My friends with PhDs in computer science have Mac laptops. [2] And yet

Apple's overall market share is still small.Though unprecedented, I predict this situation is also temporary.So Dad, there's this company called Apple. They make a new kind of

computer that's as well designed as a Bang & Olufsen stereo system,

and underneath is the best Unix machine you can buy. Yes, the price

to earnings ratio is kind of high, but I think a lot of people are

going to want these.

Notes[1] These horrible stickers are much like the intrusive ads popular

on pre-Google search engines. They say to the customer: you are

unimportant. We care about Intel and Microsoft, not you.[2] Y Combinator

is (we hope) visited mostly by

hackers. The proportions of OSes are:

Windows 66.4%, Macintosh 18.8%, Linux 11.4%, and FreeBSD 1.5%.

The Mac number is

a big change from what it would have been five years ago.Italian TranslationRussian TranslationChinese Translation

Writing, Briefly

March 2005

(In the process

of answering an email, I accidentally wrote a tiny essay about writing.

I usually spend weeks on an essay. This one took 67 minutes—23

of writing, and 44 of rewriting.)I think it's far more important to write well than most people

realize. Writing doesn't just communicate ideas; it generates them.

If you're bad at writing and don't like to do it, you'll miss out

on most of the ideas writing would have generated.As for how to write well, here's the short version:

Write a bad version

1 as fast as you can; rewrite it over and over; cut out everything

unnecessary; write in a conversational tone; develop a nose for

bad writing, so you can see and fix it in yours; imitate writers

you like; if you can't get started, tell someone what you plan to

write about, then write down what you said; expect

80% of the ideas in an essay to happen after you start writing it,

and 50% of those you start with to be wrong; be confident enough

to cut; have friends you trust read your stuff and tell you which

bits are confusing or drag; don't (always) make detailed outlines;

mull ideas over for a few days before

writing; carry a small notebook or scrap paper with you; start writing

when you think of the first

sentence; if a deadline

forces you to start before that, just say the most important sentence

first; write about stuff you like; don't try to sound impressive; don't hesitate to change the topic on the fly;

use footnotes to contain digressions; use anaphora to knit

sentences together; read your essays out loud to see (a) where you stumble

over awkward phrases and (b) which bits are boring (the

paragraphs you dread reading); try to tell the

reader something new and useful; work in fairly big quanta of time;

when you restart, begin by rereading what you have so far; when you

finish, leave yourself something easy to start with; accumulate

notes for topics you plan to cover at the bottom of the file; don't

feel obliged to cover any of them; write for a reader who won't

read the essay as carefully as you do, just as pop songs are

designed to sound ok on crappy car radios;

if you say anything mistaken, fix it immediately;

ask friends which sentence you'll regret most; go back and tone

down harsh remarks; publish stuff online, because

an audience makes you write more, and thus generate more

ideas; print out drafts instead of just looking at them

on the screen; use simple, germanic words; learn to distinguish

surprises from digressions; learn to recognize the approach of an

ending, and when one appears, grab it.Russian TranslationJapanese TranslationRomanian TranslationSpanish TranslationGerman TranslationChinese TranslationHungarian TranslationCatalan TranslationDanish TranslationArabic Translation

Undergraduation

Want to start a startup? Get funded by

Y Combinator.

March 2005(Parts of this essay began as replies to students who wrote to

me with questions.)Recently I've had several emails from computer science

undergrads asking what to do in college. I might not

be the best source of advice, because I was a philosophy major in

college. But I took so many CS classes that most CS majors thought

I was one. I was certainly a hacker, at least.HackingWhat should you do in college to become a

good hacker? There are two

main things you can do: become very good at programming, and learn

a lot about specific, cool problems. These turn out to be equivalent,

because each drives you to do the other.The way to be good at programming is to work (a) a lot (b) on hard

problems. And the way to make yourself work on hard problems is

to work on some very engaging project.

Odds are this project won't be a class assignment. My friend Robert

learned a lot by writing network software when he was an

undergrad. One of his projects was to connect Harvard to the

Arpanet; it had been one of the original nodes, but by 1984 the

connection had died. [1] Not only was this

work not for a class, but because he spent all his time on it

and neglected his studies, he was kicked out of

school for a year. [2] It all evened out in the end, and now he's

a professor at MIT. But you'll probably be happier if you don't

go to that extreme; it caused him a lot of worry at the time.Another way to be good at programming is to find other people who

are good at it, and learn what they know. Programmers tend to sort

themselves into tribes according to the type of work they do and

the tools they use, and some tribes are

smarter than others. Look

around you and see what the smart people seem to be working on;

there's usually a reason.Some of the smartest people around you are professors. So one way

to find interesting work is to volunteer as a research assistant.

Professors are especially interested in people who can solve tedious

system-administration type problems for them, so that is a way to

get a foot in the door. What they fear are

flakes and resume padders. It's all too

common for an assistant to result in a net increase in work. So

you have to make it clear you'll mean a net decrease.Don't be put off if they say no. Rejection is almost always less

personal than the rejectee imagines. Just move on to the next.

(This applies to dating too.)Beware, because although most professors are smart, not all of them

work on interesting stuff. Professors have to publish novel results

to advance their careers, but there is more competition in more

interesting areas of research. So what less ambitious professors

do is turn out a series of papers whose conclusions are novel because

no one else cares about them. You're better off avoiding these.I never worked as a research assistant, so I feel a bit dishonest

recommending that route. I learned to program by writing stuff of

my own, particularly by trying to reverse-engineer Winograd's

SHRDLU. I was as obsessed with that program as a mother with a new baby.Whatever the disadvantages of working by yourself, the advantage

is that the project is all your own. You never have to compromise

or ask anyone's permission, and if you have a new idea you can just

sit down and start implementing it.In your own projects you don't have to worry about novelty (as

professors do) or profitability (as businesses do). All that matters

is how hard the project is technically, and that has no correlation

to the nature of the application. "Serious" applications like

databases are often trivial and dull technically (if you ever suffer

from insomnia, try reading the technical literature about databases)

while "frivolous" applications like games are often very sophisticated.

I'm sure there are game companies out there working on products

with more intellectual content than the research at the

bottom nine tenths of university CS departments.If I were in college now I'd probably work on

graphics: a network game, for example, or a tool for 3D animation.

When I was an undergrad there weren't enough cycles around to make

graphics interesting, but it's hard to imagine anything more fun

to work on now.MathWhen I was in college, a lot of the professors believed (or at least

wished) that

computer science was a branch of math. This idea was

strongest at Harvard, where there wasn't even a CS major till the

1980s; till then one had to major in applied math. But it was

nearly as bad at Cornell. When I told the fearsome Professor Conway

that I was interested in AI (a hot topic then), he told me I should

major in math. I'm still not sure whether he thought AI required

math, or whether he thought AI was nonsense and that majoring in

something rigorous would cure me of such stupid ambitions.In fact, the amount of math you need as a hacker is a lot less

than most university departments like to admit. I don't think you

need much more than high school math plus a few concepts from the

theory of computation. (You have to know what an n^2 algorithm is

if you want to avoid writing them.) Unless you're planning to write

math applications, of course. Robotics, for example, is all math.But while you don't literally need math for most kinds of hacking,

in the sense of knowing 1001 tricks for differentiating formulas,

math is very much worth studying for its own sake. It's a

valuable source of metaphors for almost any kind of work.[3] I wish

I'd studied more math in college for that reason.Like a lot of people, I was mathematically abused as a child. I

learned to think of math as a collection of formulas that were

neither beautiful nor had any relation to my life (despite attempts

to translate them into "word problems"), but had to be memorized

in order to do well on tests.One of the most valuable things you could do in college would be

to learn what math is really about. This may not be easy, because

a lot of good mathematicians are bad teachers. And while there are

many popular books on math, few seem good. The best I can think

of are W. W. Sawyer's. And of course Euclid. [4]EverythingThomas Huxley said "Try to learn something about everything and

everything about something." Most universities aim at this

ideal.But what's everything? To me it means, all that people

learn in the course of working honestly on hard problems. All such

work tends to be related, in that ideas and techniques from one

field can often be transplanted successfully to others. Even others

that seem quite distant. For example, I write

essays the same way

I write software: I sit down and blow out a lame version 1 as fast

as I can type, then spend several weeks rewriting it.Working on hard problems is not, by itself, enough. Medieval

alchemists were working on a hard problem, but their approach was

so bogus that there was little

to learn from studying it, except possibly about people's ability

to delude themselves. Unfortunately the sort of AI I was trying

to learn in college had the same flaw: a very hard problem, blithely

approached with hopelessly inadequate techniques. Bold? Closer

to fraudulent.

The social sciences are also fairly bogus, because they're so much

influenced by intellectual fashions. If a

physicist met a colleague

from 100 years ago, he could teach him some new things; if a psychologist

met a colleague from 100 years ago, they'd just get into an

ideological argument.

Yes, of course, you'll learn something by taking a

psychology class. The point is, you'll learn more by taking

a class in another department.The worthwhile departments, in my opinion, are math, the hard

sciences, engineering, history (especially economic and social

history, and the history of science), architecture, and the classics.

A survey course in art history may be worthwhile. Modern literature

is important, but the way to learn about it is just to read. I

don't know enough about music to say.You can skip the social sciences, philosophy, and the various

departments created recently in response to political pressures.

Many of these fields talk about important problems, certainly. But

the way they talk about them is useless. For example, philosophy

talks, among other things, about our obligations to one another;

but you can learn more about this from a wise grandmother or E. B.

White than from an academic philosopher.I speak here from experience. I should probably have been offended

when people laughed at Clinton for saying "It depends on what the

meaning of the word 'is' is." I took about five classes in college

on what the meaning of "is" is.Another way to figure out which fields are worth studying is to

create the dropout graph. For example, I know many people

who switched from math to computer science because they found math

too hard, and no one who did the opposite. People don't do hard

things gratuitously; no one will work on a harder problem unless

it is proportionately (or at least log(n)) more rewarding. So

probably math is more worth studying than computer science. By

similar comparisons you can make a graph of all the departments in

a university. At the bottom you'll find the subjects with least

intellectual content.If you use this method, you'll get roughly the same answer I just

gave.Language courses are an anomaly. I think they're better considered

as extracurricular activities, like pottery classes. They'd be far

more useful when combined with some time living in a country where

the language is spoken. On a whim I studied Arabic as a freshman.

It was a lot of work, and the only lasting benefits were a weird

ability to identify semitic roots and some insights into how people

recognize words.Studio art and creative writing courses are wildcards. Usually

you don't get taught much: you just work (or don't work) on whatever

you want, and then sit around offering "crits" of one another's

creations under the vague supervision of the teacher. But writing and

art are both very hard problems that (some) people work honestly

at, so they're worth doing, especially if you can find a good

teacher.JobsOf course college students have to think about more than just

learning. There are also two practical problems to consider: jobs,

and graduate school.In theory a liberal education is not supposed to supply job training.

But everyone knows this is a bit of a fib. Hackers at every college

learn practical skills, and not by accident.What you should learn to get a job depends on the kind you want.

If you want to work in a big company, learn how to hack

Blub on

Windows. If you want to work at a cool little company or research

lab, you'll do better to learn Ruby on Linux. And if you want to

start your own company, which I think will be more and more common,

master the most powerful tools you can find, because you're going

to be in a race against your competitors, and they'll be your horse.There is not a direct correlation between the skills you should

learn in college and those you'll use in a job. You should aim

slightly high in college.In workouts a football player may bench press 300 pounds, even

though he may never have to exert anything like that much force in

the course of a game. Likewise, if your professors try to make you

learn stuff that's more advanced than you'll need in a job, it may

not just be because they're academics, detached from the real world.

They may be trying to make you lift weights with your brain.The programs you write in classes differ in three critical ways

from the ones you'll write in the real world: they're small; you

get to start from scratch; and the problem is usually artificial

and predetermined. In the real world, programs are bigger, tend

to involve existing code, and often require you to figure out what

the problem is before you can solve it.You don't have to wait to leave (or even enter) college to learn

these skills. If you want to learn how to deal with existing code,

for example, you can contribute to open-source projects. The sort

of employer you want to work for will be as impressed by that as

good grades on class assignments.In existing open-source projects you don't get much practice at

the third skill, deciding what problems to solve. But there's

nothing to stop you starting new projects of your own. And good

employers will be even more impressed

with that.What sort of problem should you try to solve? One way to answer

that is to ask what you need as a user. For example, I stumbled

on a good algorithm for spam filtering because I wanted to stop

getting spam. Now what I wish I had was a mail reader that somehow

prevented my inbox from filling up. I tend to use my inbox as a

todo list. But that's like using a screwdriver to open

bottles; what one really wants is a bottle opener.Grad SchoolWhat about grad school? Should you go? And how do you get into a

good one?In principle, grad school is professional training in research, and

you shouldn't go unless you want to do research as a career. And

yet half the people who get PhDs in CS don't go into research.

I didn't go to grad school to become a professor. I went because

I wanted to learn more.So if you're mainly interested in hacking and you go to grad school,

you'll find a lot of other people who are similarly out of their

element. And if half the people around you are out of their element in the

same way you are, are you really out of your element?There's a fundamental problem in "computer science," and it surfaces

in situations like this. No one is sure what "research" is supposed to be.

A lot

of research is hacking that had to be crammed into the form of an

academic paper to yield one more quantum of publication.So it's kind of misleading to ask whether you'll be at home in grad

school, because very few people are quite at home in computer

science. The whole field is uncomfortable in its own skin. So

the fact that you're mainly interested in hacking shouldn't deter

you from going to grad school. Just be warned you'll have to do a lot of stuff

you don't like.Number one will be your dissertation. Almost everyone hates their

dissertation by the time they're done with it. The

process inherently tends to produce an unpleasant result, like a cake made out

of whole wheat flour and baked for twelve hours. Few dissertations

are read with pleasure, especially by their authors.But thousands before you have suffered through writing a dissertation.

And aside from that, grad school is close to paradise. Many people

remember it as the happiest time of their lives. And nearly all

the rest, including me, remember it as a period that would have

been, if they hadn't had to write a dissertation. [5]The danger with grad school is that you don't see the scary part

upfront. PhD programs start out as college part 2, with several

years of classes. So by the time you face the horror of writing a

dissertation, you're already several years in. If you quit now,

you'll be a grad-school dropout, and you probably won't like that

idea. When Robert got kicked out of grad school for writing the

Internet worm of 1988, I envied him enormously for finding a way out

without the stigma of failure. On the whole, grad school is probably better than most alternatives. You meet a

lot of smart people, and your glum procrastination will at least

be a powerful common bond. And of course you have a PhD at the

end. I forgot about that. I suppose that's worth something.The greatest advantage of a PhD (besides being the union card of

academia, of course) may be that it gives you some baseline confidence.

For example, the Honeywell thermostats in my house have the most

atrocious UI. My mother, who has the same model, diligently spent

a day reading the user's manual to learn how to operate hers. She

assumed the problem was with her. But I can think to myself "If

someone with a PhD in computer science can't understand this

thermostat, it must be badly

designed."If you still want to go to grad school after this equivocal

recommendation, I can give you solid advice about how to get in.

A lot of my friends are CS professors now, so I have the inside

story about admissions. It's quite different from college. At

most colleges, admissions officers decide who gets in. For PhD

programs, the professors do. And they try to do

it well, because the people they admit are going to be working for

them.Apparently only recommendations really matter at the best schools.

Standardized tests count for nothing, and grades for little. The

essay is mostly an opportunity to disqualify yourself by saying

something stupid. The only thing professors

trust is recommendations, preferably from people they know. [6]So if you want to get into a PhD program, the key is to impress

your professors. And from my friends who are professors I know

what impresses them: not merely trying to impress them. They're

not impressed by students who get good grades or want to be their

research assistants so they can get into grad school. They're

impressed by students who get good grades and want to be their

research assistants because they're genuinely interested in the

topic.So the best thing you can do in college, whether you want to get

into grad school or just be good at hacking, is figure out what you

truly like. It's hard to trick professors into letting you into

grad school, and impossible to trick problems into letting you solve

them. College is where faking stops working. From this point,

unless you want to go work for a big company, which is like reverting

to high school, the only way forward is through doing what you

love.Notes

[1] No one seems to have minded, which shows how unimportant

the Arpanet (which became the Internet) was as late as

1984.[2] This is why, when I became an employer, I didn't care

about GPAs. In fact, we actively sought out people

who'd failed out of school. We once put up posters around Harvard

saying "Did you just get kicked out for doing badly in your classes

because you spent all your time working on some project of your

own? Come work for us!" We managed to find a kid who had been,

and he was a great hacker.When Harvard kicks undergrads out for a year, they have to get jobs.

The idea is to show them how awful the real world is, so they'll

understand how lucky they are to be in college. This plan backfired

with the guy who came to work for us, because he had more fun than

he'd had in school, and made more that year from stock options than

any of his professors did in salary. So instead of crawling back

repentant at the end of the year, he took another year off and went

to Europe. He did eventually graduate at about 26.[3] Eric Raymond says the best metaphors for hackers are

in set theory, combinatorics, and graph theory.Trevor Blackwell reminds you to take math classes intended for math majors.

"'Math for engineers' classes sucked mightily. In fact any 'x for

engineers' sucks, where x includes math, law, writing and visual

design."[4] Other highly recommended books: What is Mathematics?, by

Courant and Robbins; Geometry and the Imagination by Hilbert and

Cohn-Vossen.

And for those interested in graphic design,

Byrne's Euclid.

[5] If you wanted to have the perfect life, the thing to do would

be to go to grad school, secretly write your dissertation in the

first year or two, and then just enjoy yourself for the next three

years, dribbling out a chapter at a time. This prospect will make

grad students' mouths water, but I know of no one who's had the

discipline to pull it off.[6] One professor friend says that 15-20% of the grad students they

admit each year are "long shots." But what he means by long shots

are people whose applications are perfect in every way, except

that no one on the admissions committee knows the professors who

wrote the recommendations.So if you want to get into

grad school in the sciences, you need to go to college somewhere with

real research professors. Otherwise you'll seem a risky bet

to admissions committees, no matter how good you are.Which implies

a surprising but apparently inevitable consequence:

little liberal arts colleges are doomed.

Most smart

high school kids at least consider going into the sciences, even

if they ultimately choose not to.

Why go to a college that limits their options?Thanks to Trevor Blackwell, Alex Lewin, Jessica Livingston,

Robert Morris, Eric

Raymond, and several

anonymous CS professors

for reading drafts of this, and to the students whose questions

began it.More Advice for UndergradsJoel Spolsky: Advice for Computer Science College StudentsEric Raymond: How to Become a Hacker

A Unified Theory of VC Suckage

March 2005A couple months ago I got an email from a recruiter asking if I was

interested in being a "technologist in residence" at a new venture

capital fund. I think the idea was to play Karl Rove to the VCs'

George Bush.I considered it for about four seconds. Work for a VC fund? Ick.One of my most vivid memories from our startup is going to visit

Greylock, the famous Boston VCs. They were the most arrogant

people I've met in my life. And I've met a lot of arrogant people.

[1]I'm not alone in feeling this way, of course. Even a VC friend of

mine dislikes VCs. "Assholes," he says.But lately I've been learning more about how the VC world works,

and a few days ago it hit me that there's a reason VCs are the way

they are. It's not so much that the business attracts jerks, or

even that the power they wield corrupts them. The real problem is

the way they're paid.The problem with VC funds is that they're funds. Like the

managers of mutual funds or hedge funds, VCs get paid a percentage

of the money they manage: about 2% a year in management fees,

plus a percentage of the gains. So they want

the fund to be huge-- hundreds of millions of dollars, if possible.

But that means each partner ends up being responsible for investing

a lot of money. And since one person can only manage so many deals,

each deal has to be for multiple millions of dollars.This turns out to explain nearly all the characteristics of VCs

that founders hate.It explains why VCs take so agonizingly long to make up their minds,

and why their due diligence feels like a body cavity search. [2]

With so much at stake, they have to be paranoid.It explains why they steal your ideas. Every founder knows that

VCs will tell your secrets to your competitors if they end up

investing in them. It's not unheard of for VCs to meet you when

they have no intention of funding you, just to pick your brain for

a competitor. This prospect makes naive founders clumsily secretive.

Experienced founders treat it as a cost of doing business. Either

way it sucks. But again, the only reason VCs are so sneaky is the

giant deals they do. With so much at stake, they have to be devious.It explains why VCs tend to interfere in the companies they invest

in. They want to be on your board not just so that they can advise

you, but so that they can watch you. Often they even install a new

CEO. Yes, he may have extensive business experience. But he's

also their man: these newly installed CEOs always play something

of the role of a political commissar in a Red Army unit. With

so much at stake, VCs can't resist micromanaging you.The huge investments themselves are something founders would dislike,

if they realized how damaging they can be. VCs don't invest $x

million because that's the amount you need, but because that's the

amount the structure of their business requires them to invest.

Like steroids, these sudden huge investments can do more harm than

good. Google survived enormous VC funding because it could

legitimately absorb large amounts of money. They had to buy a lot

of servers and a lot of bandwidth to crawl the whole Web. Less

fortunate startups just end up hiring armies of people to sit around

having meetings.In principle you could take a huge VC investment, put it in treasury

bills, and continue to operate frugally. You just try it.And of course giant investments mean giant valuations. They have

to, or there's not enough stock left to keep the founders interested.

You might think a high valuation is a great thing. Many founders

do. But you can't eat paper. You can't benefit from a high valuation

unless you can somehow achieve what those in the business

call a "liquidity event," and the higher

your valuation, the narrower your options for doing that. Many a

founder would be happy to sell his company for $15 million, but VCs

who've just invested at a pre-money valuation of $8 million won't

hear of that. You're rolling the dice again, whether you like it

or not.Back in 1997, one of our competitors raised $20 million in a single

round of VC funding. This was at the time more than the valuation

of our entire company. Was I worried? Not at all: I was delighted.

It was like watching a car you're chasing turn down a street that

you know has no outlet.Their smartest move at that point would have been to take every

penny of the $20 million and use it to buy us. We would have sold.

Their investors would have been furious of course. But I think the

main reason they never considered this was that they never imagined

we could be had so cheap. They probably assumed we were on the

same VC gravy train they were.In fact we only spent about $2 million in our entire existence.

And that gave us flexibility. We could sell ourselves to Yahoo for

$50 million, and everyone was delighted. If our competitor had

done that, the last round of investors would presumably have lost

money. I assume they could have vetoed such a deal. But no one

those days was paying a lot more than Yahoo. So unless their

founders could pull off an IPO (which would be difficult with Yahoo

as a competitor), they had no choice but to ride the thing down.The puffed-up companies that went public during the Bubble didn't

do it just because they were pulled into it by unscrupulous investment

bankers. Most were pushed just as hard from the other side by VCs

who'd invested at high valuations, leaving an IPO as the only way

out. The only people dumber were retail investors. So it was

literally IPO or bust. Or rather, IPO then bust, or just bust.Add up all the evidence of VCs' behavior, and the resulting personality

is not attractive. In fact, it's the classic villain: alternately

cowardly, greedy, sneaky, and overbearing.I used to take it for granted that VCs were like this. Complaining

that VCs were jerks used to seem as naive to me as complaining that

users didn't read the reference manual. Of course VCs were jerks.

How could it be otherwise?But I realize now that they're not intrinsically jerks. VCs are

like car salesmen or bureaucrats: the nature of their work

turns them into jerks.I've met a few VCs I like. Mike Moritz seems a good guy. He even

has a sense of humor, which is almost unheard of among VCs. From

what I've read about John Doerr, he sounds like a good guy too,

almost a hacker. But they work for the very best VC funds. And

my theory explains why they'd tend to be different: just as the

very most popular kids don't have to persecute

nerds, the very best

VCs don't have to act like VCs. They get the pick of all the best

deals. So they don't have to be so paranoid and sneaky, and they

can choose those rare companies, like Google, that will actually

benefit from the giant sums they're compelled to invest.VCs often complain that in their business there's too much money

chasing too few deals. Few realize that this also describes a flaw

in the way funding works at the level of individual firms.Perhaps this was the sort of strategic insight I was supposed to

come up with as a "technologist in residence." If so, the good

news is that they're getting it for free. The bad news is it

means that if you're not one of the very top funds, you're

condemned to be the bad guys.Notes[1] After Greylock booted founder Philip Greenspun out of ArsDigita,

he wrote a hilarious but also very informative

essay about it.

[2] Since most VCs aren't tech guys, the technology side of their

due diligence tends to be like a body cavity search by someone with

a faulty knowledge of human anatomy. After a while we were quite

sore from VCs attempting to probe our nonexistent database orifice.No, we don't use Oracle. We just store the data in files. Our

secret is to use an OS that doesn't lose our data. Which OS?

FreeBSD. Why do you use that instead of Windows NT? Because it's

better and it doesn't cost anything. What, you're using a

freeware OS?How many times that conversation was repeated.

Then when we got to Yahoo, we found they used FreeBSD and stored

their data in files too.Chinese TranslationJapanese Translation

How to Start a Startup

Want to start a startup? Get funded by

Y Combinator.

March 2005(This essay is derived from a talk at the Harvard Computer

Society.)You need three things to create a successful startup: to start with

good people, to make something customers actually want, and to spend

as little money as possible. Most startups that fail do it because

they fail at one of these. A startup that does all three will

probably succeed.And that's kind of exciting, when you think about it, because all

three are doable. Hard, but doable. And since a startup that

succeeds ordinarily makes its founders rich, that implies getting

rich is doable too. Hard, but doable.If there is one message I'd like to get across about startups,

that's it. There is no magically difficult step that requires

brilliance to solve.The IdeaIn particular, you don't need a brilliant

idea to start a startup

around. The way a startup makes money is to offer people better

technology than they have now. But what people have now is often

so bad that it doesn't take brilliance to do better.Google's plan, for example, was simply to create a search site that

didn't suck. They had three new ideas: index more of the Web, use

links to rank search results, and have clean, simple web pages with

unintrusive keyword-based ads. Above all, they were determined to

make a site that was good to use. No doubt there are great technical

tricks within Google, but the overall plan was straightforward.

And while they probably have bigger ambitions now, this alone brings

them a billion dollars a year. [1]There are plenty of other areas that are just as backward as search

was before Google. I can think of several heuristics for generating

ideas for startups, but most reduce to this: look at something

people are trying to do, and figure out how to do it in a way that

doesn't suck.For example, dating sites currently suck far worse than search did

before Google. They all use the same simple-minded model.

They seem to have approached the problem by thinking about how to

do database matches instead of how dating works in the real world.

An undergrad could build something better as a class project. And

yet there's a lot of money at stake. Online dating is a valuable

business now, and it might be worth a hundred times as much if it

worked.An idea for a startup, however, is only a beginning. A lot of

would-be startup founders think the key to the whole process is the

initial idea, and from that point all you have to do is execute.

Venture capitalists know better. If you go to VC firms with a

brilliant idea that you'll tell them about if they sign a nondisclosure

agreement, most will tell you to get lost. That shows how much a

mere idea is worth. The market price is less than the inconvenience

of signing an NDA.Another sign of how little the initial idea is worth is the number

of startups that change their plan en route. Microsoft's original

plan was to make money selling programming languages, of all things.

Their current business model didn't occur to them until IBM dropped

it in their lap five years later.Ideas for startups are worth something, certainly, but the trouble

is, they're not transferrable. They're not something you could

hand to someone else to execute. Their value is mainly as starting

points: as questions for the people who had them to continue thinking

about.What matters is not ideas, but the people who have them. Good

people can fix bad ideas, but good ideas can't save bad people.

PeopleWhat do I mean by good people? One of the best tricks I learned

during our startup was a rule for deciding

who to hire. Could you

describe the person as an animal? It might be hard to translate

that into another language, but I think everyone in the US knows

what it means. It means someone who takes their work a little too

seriously; someone who does what they do so well that they pass

right through professional and cross over into obsessive.What it means specifically depends on the job: a salesperson who

just won't take no for an answer; a hacker who will stay up till

4:00 AM rather than go to bed leaving code with a bug in it; a PR

person who will cold-call New York Times reporters on their cell

phones; a graphic designer who feels physical pain when something

is two millimeters out of place.Almost everyone who worked for us was an animal at what they did.

The woman in charge of sales was so tenacious that I used to feel

sorry for potential customers on the phone with her. You could

sense them squirming on the hook, but you knew there would be no

rest for them till they'd signed up.If you think about people you know, you'll find the animal test is

easy to apply. Call the person's image to mind and imagine the

sentence "so-and-so is an animal." If you laugh, they're not. You

don't need or perhaps even want this quality in big companies, but

you need it in a startup.For programmers we had three additional tests. Was the person

genuinely smart? If so, could they actually get things done? And

finally, since a few good hackers have unbearable personalities,

could we stand to have them around?That last test filters out surprisingly few people. We could bear

any amount of nerdiness if someone was truly smart. What we couldn't

stand were people with a lot of attitude. But most of those weren't

truly smart, so our third test was largely a restatement of the

first.When nerds are unbearable it's usually because they're trying too

hard to seem smart. But the smarter they are, the less pressure

they feel to act smart. So as a rule you can recognize genuinely

smart people by their ability to say things like "I don't know,"

"Maybe you're right," and "I don't understand x well enough."This technique doesn't always work, because people can be influenced

by their environment. In the MIT CS department, there seems to be

a tradition of acting like a brusque know-it-all. I'm told it derives

ultimately from Marvin Minsky, in the same way the classic airline

pilot manner is said to derive from Chuck Yeager. Even genuinely

smart people start to act this way there, so you have to make

allowances.It helped us to have Robert Morris, who is one of the readiest to

say "I don't know" of anyone I've met. (At least, he was before he

became a professor at MIT.) No one dared put on attitude around

Robert, because he was obviously smarter than they were and yet had

zero attitude himself.Like most startups, ours began with a group of friends, and it was

through personal contacts that we got most of the people we hired.

This is a crucial difference between startups and big companies.

Being friends with someone for even a couple days will tell you

more than companies could ever learn in interviews. [2]It's no coincidence that startups start around universities, because

that's where smart people meet. It's not what people learn in

classes at MIT and Stanford that has made technology companies

spring up around them. They could sing campfire songs in the classes

so long as admissions worked the same.If you start a startup, there's a good chance it will be with people

you know from college or grad school. So in theory you ought to

try to make friends with as many smart people as you can in school,

right? Well, no. Don't make a conscious effort to schmooze; that

doesn't work well with hackers.What you should do in college is work on your own projects. Hackers

should do this even if they don't plan to start startups, because

it's the only real way to learn how to program. In some cases you

may collaborate with other students, and this is the best way to

get to know good hackers. The project may even grow into a startup.

But once again, I wouldn't aim too directly at either target. Don't

force things; just work on stuff you like with people you like.Ideally you want between two and four founders. It would be hard

to start with just one. One person would find the moral weight of

starting a company hard to bear. Even Bill Gates, who seems to be

able to bear a good deal of moral weight, had to have a co-founder.

But you don't want so many founders that the company starts to look

like a group photo. Partly because you don't need a lot of people

at first, but mainly because the more founders you have, the worse

disagreements you'll have. When there are just two or three founders,

you know you have to resolve disputes immediately or perish. If

there are seven or eight, disagreements can linger and harden into

factions. You don't want mere voting; you need unanimity.In a technology startup, which most startups are, the founders

should include technical people. During the Internet Bubble there

were a number of startups founded by business people who then went

looking for hackers to create their product for them. This doesn't

work well. Business people are bad at deciding what to do with

technology, because they don't know what the options are, or which

kinds of problems are hard and which are easy. And when business

people try to hire hackers, they can't tell which ones are

good.

Even other hackers have a hard time doing that.

For business people it's roulette.Do the founders of a startup have to include business people? That

depends. We thought so when we started ours, and we asked several

people who were said to know about this mysterious thing called

"business" if they would be the president. But they all said no,

so I had to do it myself. And what I discovered was that business

was no great mystery. It's not something like physics or medicine

that requires extensive study. You just try to get people to pay

you for stuff.I think the reason I made such a mystery of business was that I was

disgusted by the idea of doing it. I wanted to work in the pure,

intellectual world of software, not deal with customers' mundane

problems. People who don't want to get dragged into some kind of

work often develop a protective incompetence at it. Paul Erdos was

particularly good at this. By seeming unable even to cut a grapefruit

in half (let alone go to the store and buy one), he forced other

people to do such things for him, leaving all his time free for

math. Erdos was an extreme case, but most husbands use the same

trick to some degree.Once I was forced to discard my protective incompetence, I found

that business was neither so hard nor so boring as I feared. There

are esoteric areas of business that are quite hard, like tax law

or the pricing of derivatives, but you don't need to know about

those in a startup. All you need to know about business to run a

startup are commonsense things people knew before there were business

schools, or even universities.If you work your way down the Forbes 400 making an x next to the

name of each person with an MBA, you'll learn something important

about business school. After Warren Buffett, you don't hit another

MBA till number 22,

Phil Knight, the CEO of Nike. There are only 5 MBAs in the top

50. What you notice in the Forbes 400 are a lot of people with

technical backgrounds. Bill Gates, Steve Jobs, Larry Ellison,

Michael Dell, Jeff Bezos, Gordon Moore. The rulers of the technology

business tend to come from technology, not business. So if you

want to invest two years in something that will help you succeed

in business, the evidence suggests you'd do better to learn how to

hack than get an MBA. [3]There is one reason you might want to include business people in a

startup, though: because you have to have at least one person willing

and able to focus on what customers want. Some believe only business

people can do this-- that hackers can implement software, but not

design it. That's nonsense. There's nothing about knowing how to

program that prevents hackers from understanding users, or about

not knowing how to program that magically enables business people

to understand them.If you can't understand users, however, you should either learn how

or find a co-founder who can. That is the single most important

issue for technology startups, and the rock that sinks more of them

than anything else.What Customers WantIt's not just startups that have to worry about this. I think most

businesses that fail do it because they don't give customers what

they want. Look at restaurants. A large percentage fail, about a

quarter in the first year. But can you think of one restaurant

that had really good food and went out of business?Restaurants with great food seem to prosper no matter what. A

restaurant with great food can be expensive, crowded, noisy, dingy,

out of the way, and even have bad service, and people will keep

coming. It's true that a restaurant with mediocre food can sometimes

attract customers through gimmicks. But that approach is very

risky. It's more straightforward just to make the food good.It's the same with technology. You hear all kinds of reasons why

startups fail. But can you think of one that had a massively popular

product and still failed?In nearly every failed startup, the real problem was that customers

didn't want the product. For most, the cause of death is listed

as "ran out of funding," but that's only the immediate cause. Why

couldn't they get more funding? Probably because the product was

a dog, or never seemed likely to be done, or both.When I was trying to think of the things every startup needed to

do, I almost included a fourth: get a version 1 out as soon as you

can. But I decided not to, because that's implicit in making

something customers want. The only way to make something customers

want is to get a prototype in front of them and refine it based on

their reactions.The other approach is what I call the "Hail Mary" strategy. You

make elaborate plans for a product, hire a team of engineers to

develop it (people who do this tend to use the term "engineer" for

hackers), and then find after a year that you've spent two million

dollars to develop something no one wants. This was not uncommon

during the Bubble, especially in companies run by business types,

who thought of software development as something terrifying that

therefore had to be carefully planned.We never even considered that approach. As a Lisp hacker, I come

from the tradition of rapid prototyping. I would not claim (at

least, not here) that this is the right way to write every program,

but it's certainly the right way to write software for a startup.

In a startup, your initial plans are almost certain to be wrong in

some way, and your first priority should be to figure out where.

The only way to do that is to try implementing them.Like most startups, we changed our plan on the fly. At first we

expected our customers to be Web consultants. But it turned out

they didn't like us, because our software was easy to use and we hosted

the site. It would be too easy for clients to fire them. We also

thought we'd be able to sign up a lot of catalog companies, because

selling online was a natural extension of their existing business.

But in 1996 that was a hard sell. The middle managers we talked

to at catalog companies saw the Web not as an opportunity, but as

something that meant more work for them.We did get a few of the more adventurous catalog companies. Among

them was Frederick's of Hollywood, which gave us valuable experience

dealing with heavy loads on our servers. But most of our users

were small, individual merchants who saw the Web as an opportunity

to build a business. Some had retail stores, but many only existed

online. And so we changed direction to focus on these users.

Instead of concentrating on the features Web consultants and catalog

companies would want, we worked to make the software easy to use.I learned something valuable from that. It's worth trying very,

very hard to make technology easy to use. Hackers are so used to

computers that they have no idea how horrifying software seems to

normal people. Stephen Hawking's editor told him that every equation

he included in his book would cut sales in half. When you work on

making technology easier to use, you're riding that curve up instead

of down. A 10% improvement in ease of use doesn't just increase

your sales 10%. It's more likely to double your sales.How do you figure out what customers want? Watch them. One of the

best places to do this was at trade shows. Trade shows didn't pay

as a way of getting new customers, but they were worth it as market

research. We didn't just give canned presentations at trade shows.

We used to show people how to build real, working stores. Which

meant we got to watch as they used our software, and talk to them

about what they needed.No matter what kind of startup you start, it will probably be a

stretch for you, the founders, to understand what users want. The

only kind of software you can build without studying users is the

sort for which you are the typical user. But this is just the kind

that tends to be open source: operating systems, programming

languages, editors, and so on. So if you're developing technology

for money, you're probably not going to be developing it for people

like you. Indeed, you can use this as a way to generate ideas for

startups: what do people who are not like you want from technology?When most people think of startups, they think of companies like

Apple or Google. Everyone knows these, because they're big consumer

brands. But for every startup like that, there are twenty more

that operate in niche markets or live quietly down in the infrastructure.

So if you start a successful startup, odds are you'll start one of

those.Another way to say that is, if you try to start the kind of startup

that has to be a big consumer brand, the odds against succeeding

are steeper. The best odds are in niche markets. Since startups

make money by offering people something better than they had before,

the best opportunities are where things suck most. And it would

be hard to find a place where things suck more than in corporate

IT departments. You would not believe the amount of money companies

spend on software, and the crap they get in return. This imbalance

equals opportunity.If you want ideas for startups, one of the most valuable things you

could do is find a middle-sized non-technology company and spend a

couple weeks just watching what they do with computers. Most good

hackers have no more idea of the horrors perpetrated in these places

than rich Americans do of what goes on in Brazilian slums.Start by writing software for smaller companies, because it's easier

to sell to them. It's worth so much to sell stuff to big companies

that the people selling them the crap they currently use spend a

lot of time and money to do it. And while you can outhack Oracle

with one frontal lobe tied behind your back, you can't outsell an

Oracle salesman. So if you want to win through better technology,

aim at smaller customers. [4]They're the more strategically valuable part of the market anyway.

In technology, the low end always eats the high end. It's easier

to make an inexpensive product more powerful than to make a powerful

product cheaper. So the products that start as cheap, simple options

tend to gradually grow more powerful till, like water rising in a

room, they squash the "high-end" products against the ceiling. Sun

did this to mainframes, and Intel is doing it to Sun. Microsoft

Word did it to desktop publishing software like Interleaf and

Framemaker. Mass-market digital cameras are doing it to the expensive

models made for professionals. Avid did it to the manufacturers

of specialized video editing systems, and now Apple is doing it to

Avid. Henry Ford did it to the car makers that preceded

him. If you build the simple, inexpensive option, you'll not only

find it easier to sell at first, but you'll also be in the best

position to conquer the rest of the market.It's very dangerous to let anyone fly under you. If you have the

cheapest, easiest product, you'll own the low end. And if you

don't, you're in the crosshairs of whoever does.Raising MoneyTo make all this happen, you're going to need money. Some startups

have been self-funding-- Microsoft for example-- but most aren't.

I think it's wise to take money from investors. To be self-funding,

you have to start as a consulting company, and it's hard to switch

from that to a product company.Financially, a startup is like a pass/fail course. The way to get

rich from a startup is to maximize the company's chances of succeeding,

not to maximize the amount of stock you retain. So if you can trade

stock for something that improves your odds, it's probably a smart

move.To most hackers, getting investors seems like a terrifying and

mysterious process. Actually it's merely tedious. I'll try to

give an outline of how it works.The first thing you'll need is a few tens of thousands of dollars

to pay your expenses while you develop a prototype. This is called

seed capital. Because so little money is involved, raising seed

capital is comparatively easy-- at least in the sense of getting a

quick yes or no.Usually you get seed money from individual rich people called

"angels." Often they're people who themselves got rich from technology.

At the seed stage, investors don't expect you to have an elaborate

business plan. Most know that they're supposed to decide quickly.

It's not unusual to get a check within a week based on a half-page

agreement.We started Viaweb with $10,000 of seed money from our friend Julian.

But he gave us a lot more than money. He's a former CEO and also

a corporate lawyer, so he gave us a lot of valuable advice about

business, and also did all the legal work of getting us set up as

a company. Plus he introduced us to one of the two

angel investors who supplied our next round of funding.Some angels, especially those with technology backgrounds, may be

satisfied with a demo and a verbal description of what you plan to

do. But many will want a copy of your business plan, if only to

remind themselves what they invested in.Our angels asked for one, and looking back, I'm amazed how much

worry it caused me. "Business plan" has that word "business" in

it, so I figured it had to be something I'd have to read a book

about business plans to write. Well, it doesn't. At this stage,

all most investors expect is a brief description of what you plan

to do and how you're going to make money from it, and the resumes

of the founders. If you just sit down and write out what you've

been saying to one another, that should be fine. It shouldn't take

more than a couple hours, and you'll probably find that writing it

all down gives you more ideas about what to do.For the angel to have someone to make the check out to, you're going

to have to have some kind of company. Merely incorporating yourselves

isn't hard. The problem is, for the company to exist, you have to

decide who the founders are, and how much stock they each have. If

there are two founders with the same qualifications who are both

equally committed to the business, that's easy. But if you have a

number of people who are expected to contribute in varying degrees,

arranging the proportions of stock can be hard. And once you've

done it, it tends to be set in stone.I have no tricks for dealing with this problem. All I can say is,

try hard to do it right. I do have a rule of thumb for recognizing

when you have, though. When everyone feels they're getting a

slightly bad deal, that they're doing more than they should for the

amount of stock they have, the stock is optimally apportioned.There is more to setting up a company than incorporating it, of

course: insurance, business license, unemployment compensation,

various things with the IRS. I'm not even sure what the list is,

because we, ah, skipped all that. When we got real funding near

the end of 1996, we hired a great CFO, who fixed everything

retroactively. It turns out that no one comes and arrests you if

you don't do everything you're supposed to when starting a company.

And a good thing too, or a lot of startups would never get started.

[5]It can be dangerous to delay turning yourself into a company, because

one or more of the founders might decide to split off and start

another company doing the same thing. This does happen. So when

you set up the company, as well as as apportioning the stock, you

should get all the founders to sign something agreeing that everyone's

ideas belong to this company, and that this company is going to be

everyone's only job.[If this were a movie, ominous music would begin here.]While you're at it, you should ask what else they've signed. One

of the worst things that can happen to a startup is to run into

intellectual property problems. We did, and it came closer to

killing us than any competitor ever did.As we were in the middle of getting bought, we discovered that one

of our people had, early on, been bound by an agreement that said

all his ideas belonged to the giant company that was paying for him

to go to grad school. In theory, that could have meant someone

else owned big chunks of our software. So the acquisition came to

a screeching halt while we tried to sort this out. The problem

was, since we'd been about to be acquired, we'd allowed ourselves

to run low on cash. Now we needed to raise more to keep going.

But it's hard to raise money with an IP cloud over your head, because

investors can't judge how serious it is.Our existing investors, knowing that we needed money and had nowhere

else to get it, at this point attempted certain gambits which I

will not describe in detail, except to remind readers that the word

"angel" is a metaphor. The founders thereupon proposed to walk

away from the company, after giving the investors a brief tutorial

on how to administer the servers themselves. And while this was

happening, the acquirers used the delay as an excuse to welch on

the deal.Miraculously it all turned out ok. The investors backed down; we

did another round of funding at a reasonable valuation; the giant

company finally gave us a piece of paper saying they didn't own our

software; and six months later we were bought by Yahoo for much

more than the earlier acquirer had agreed to pay. So we were happy

in the end, though the experience probably took several years off

my life.Don't do what we did. Before you consummate a startup, ask

everyone about their previous IP history.Once you've got a company set up, it may seem presumptuous to go

knocking on the doors of rich people and asking them to invest tens

of thousands of dollars in something that is really just a bunch

of guys with some ideas. But when you look at it from the rich

people's point of view, the picture is more encouraging. Most rich

people are looking for good investments. If you really think you

have a chance of succeeding, you're doing them a favor by letting

them invest. Mixed with any annoyance they might feel about being

approached will be the thought: are these guys the next Google?Usually angels are financially equivalent to founders. They get

the same kind of stock and get diluted the same amount in future

rounds. How much stock should they get? That depends on how

ambitious you feel. When you offer x percent of your company for

y dollars, you're implicitly claiming a certain value for the whole

company. Venture investments are usually described in terms of

that number. If you give an investor new shares equal to 5% of

those already outstanding in return for $100,000, then you've done

the deal at a pre-money valuation of $2 million.How do you decide what the value of the company should be? There

is no rational way. At this stage the company is just a bet. I

didn't realize that when we were raising money. Julian

thought we ought to value the company at several million

dollars. I thought it was preposterous to claim that a couple

thousand lines of code, which was all we had at the time, were worth

several million dollars. Eventually we settled on one millon,

because Julian said no one would invest in a company with a valuation

any lower. [6]What I didn't grasp at the time was that the valuation wasn't just

the value of the code we'd written so far. It was also the value

of our ideas, which turned out to be right, and of all the future

work we'd do, which turned out to be a lot.The next round of funding is the one in which you might deal with

actual

venture capital firms.

But don't wait till you've burned

through your last round of funding to start approaching them. VCs are slow to

make up their minds. They can take months. You don't want to be

running out of money while you're trying to negotiate with them.Getting money from an actual VC firm is a bigger deal than getting

money from angels. The amounts of money involved are larger, millions

usually. So the deals take longer, dilute you more, and impose

more onerous conditions.Sometimes the VCs want to install a new CEO of their own choosing.

Usually the claim is that you need someone mature and experienced,

with a business background. Maybe in some cases this is true. And

yet Bill Gates was young and inexperienced and had no business

background, and he seems to have done ok. Steve Jobs got booted

out of his own company by someone mature and experienced, with a

business background, who then proceeded to ruin the company. So I

think people who are mature and experienced, with a business

background, may be overrated. We used to call these guys "newscasters,"

because they had neat hair and spoke in deep, confident voices, and

generally didn't know much more than they read on the teleprompter.We talked to a number of VCs, but eventually we ended up financing

our startup entirely with angel money. The main reason was that

we feared a brand-name VC firm would stick us with a newscaster as

part of the deal. That might have been ok if he was content to

limit himself to talking to the press, but what if he wanted to

have a say in running the company? That would have led to disaster,

because our software was so complex. We were a company whose whole

m.o. was to win through better technology. The strategic decisions

were mostly decisions about technology, and we didn't need any help

with those.This was also one reason we didn't go public. Back in 1998 our CFO

tried to talk me into it. In those days you could go public as a

dogfood portal, so as a company with a real product and real revenues,

we might have done well. But I feared it would have meant taking

on a newscaster-- someone who, as they say, "can talk Wall Street's

language."I'm happy to see Google is bucking that trend. They didn't talk

Wall Street's language when they did their IPO, and Wall Street

didn't buy. And now Wall Street is collectively kicking itself.

They'll pay attention next time. Wall Street learns new languages

fast when money is involved.You have more leverage negotiating with VCs than you realize. The

reason is other VCs. I know a number of VCs now, and when you talk

to them you realize that it's a seller's market. Even now there

is too much money chasing too few good deals.VCs form a pyramid. At the top are famous ones like Sequoia and

Kleiner Perkins, but beneath those are a huge number you've never

heard of. What they all have in common is that a dollar from them

is worth one dollar. Most VCs will tell you that they don't just

provide money, but connections and advice. If you're talking to

Vinod Khosla or John Doerr or Mike Moritz, this is true. But such

advice and connections can come very expensive. And as you go down

the food chain the VCs get rapidly

dumber. A few steps down from

the top you're basically talking to bankers who've picked up a few

new vocabulary words from reading Wired. (Does your product

use XML?) So I'd advise you to be skeptical about claims

of experience and connections. Basically, a VC is a source of

money. I'd be inclined to go with whoever offered the most money

the soonest with the least strings attached.You may wonder how much to tell VCs. And you should, because some

of them may one day be funding your competitors. I think the best

plan is not to be overtly secretive, but not to tell them everything

either. After all, as most VCs say, they're more interested in the

people than the ideas. The main reason they want to talk about

your idea is to judge you, not the idea. So as long as you seem

like you know what you're doing, you can probably keep a few things

back from them. [7]Talk to as many VCs as you can, even if you don't want their money,

because a) they may be on the board of someone who will buy you,

and b) if you seem impressive, they'll be discouraged from investing

in your competitors. The most efficient way to reach VCs, especially

if you only want them to know about you and don't want their money,

is at the conferences that are occasionally organized for startups

to present to them.Not Spending ItWhen and if you get an infusion of real money from investors, what

should you do with it? Not spend it, that's what. In nearly every

startup that fails, the proximate cause is running out of money.

Usually there is something deeper wrong. But even a proximate cause

of death is worth trying hard to avoid.During the Bubble many startups tried to "get big fast." Ideally

this meant getting a lot of customers fast. But it was easy for

the meaning to slide over into hiring a lot of people fast.Of the two versions, the one where you get a lot of customers fast

is of course preferable. But even that may be overrated. The idea

is to get there first and get all the users, leaving none for

competitors. But I think in most businesses the advantages of being

first to market are not so overwhelmingly great. Google is again

a case in point. When they appeared it seemed as if search was a

mature market, dominated by big players who'd spent millions to

build their brands: Yahoo, Lycos, Excite, Infoseek, Altavista,

Inktomi. Surely 1998 was a little late to arrive at the party.But as the founders of Google knew, brand is worth next to nothing

in the search business. You can come along at any point and make

something better, and users will gradually seep over to you. As

if to emphasize the point, Google never did any advertising. They're

like dealers; they sell the stuff, but they know better than to use

it themselves.The competitors Google buried would have done better to spend those

millions improving their software. Future startups should learn

from that mistake. Unless you're in a market where products are

as undifferentiated as cigarettes or vodka or laundry detergent,

spending a lot on brand advertising is a sign of breakage. And few

if any Web businesses are so undifferentiated. The dating sites

are running big ad campaigns right now, which is all the

more evidence they're ripe for the picking. (Fee, fie, fo, fum, I

smell a company run by marketing guys.)We were compelled by circumstances to grow slowly, and in retrospect

it was a good thing. The founders all learned to do every job in

the company. As well as writing software, I had to do sales and

customer support. At sales I was not very good. I was persistent,

but I didn't have the smoothness of a good salesman. My message

to potential customers was: you'd be stupid not to sell online, and

if you sell online you'd be stupid to use anyone else's software.

Both statements were true, but that's not the way to convince people.I was great at customer support though. Imagine talking to a

customer support person who not only knew everything about the

product, but would apologize abjectly if there was a bug, and then

fix it immediately, while you were on the phone with them. Customers

loved us. And we loved them, because when you're growing slow by

word of mouth, your first batch of users are the ones who were smart

enough to find you by themselves. There is nothing more valuable,

in the early stages of a startup, than smart users. If you listen

to them, they'll tell you exactly how to make a winning product.

And not only will they give you this advice for free, they'll pay

you.We officially launched in early 1996. By the end of that year we

had about 70 users. Since this was the era of "get big fast," I

worried about how small and obscure we were. But in fact we were

doing exactly the right thing. Once you get big (in users or

employees) it gets hard to change your product. That year was

effectively a laboratory for improving our software. By the end

of it, we were so far ahead of our competitors that they never had

a hope of catching up. And since all the hackers had spent many

hours talking to users, we understood online commerce way better

than anyone else.That's the key to success as a startup. There is nothing more

important than understanding your business. You might think that

anyone in a business must, ex officio, understand it. Far from it.

Google's secret

weapon was simply that they understood search. I was working for

Yahoo when Google appeared, and Yahoo didn't understand search. I

know because I once tried to convince the powers that be that we

had to make search better, and I got in reply what was then the

party line about it: that Yahoo was no longer a mere "search engine."

Search was now only a small percentage of our page views, less than

one month's growth, and now that we were established as a "media

company," or "portal," or whatever we were, search could safely be

allowed to wither and drop off, like an umbilical cord.Well, a small fraction of page views they may be, but they are an

important fraction, because they are the page views that Web sessions

start with. I think Yahoo gets that now.Google understands a few other things most Web companies still

don't. The most important is that you should put users before

advertisers, even though the advertisers are paying and users aren't.

One of my favorite bumper stickers reads "if the people lead, the

leaders will follow." Paraphrased for the Web, this becomes "get

all the users, and the advertisers will follow." More generally,

design your product to please users first, and then think about how

to make money from it. If you don't put users first, you leave a

gap for competitors who do.To make something users love, you have to understand them. And the

bigger you are, the harder that is. So I say "get big slow." The

slower you burn through your funding, the more time you have to

learn.The other reason to spend money slowly is to encourage a culture

of cheapness. That's something Yahoo did understand. David Filo's

title was "Chief Yahoo," but he was proud that his unofficial title

was "Cheap Yahoo." Soon after we arrived at Yahoo, we got an email

from Filo, who had been crawling around our directory hierarchy,

asking if it was really necessary to store so much of our data on

expensive RAID drives. I was impressed by that. Yahoo's market

cap then was already in the billions, and they were still worrying

about wasting a few gigs of disk space.When you get a couple million dollars from a VC firm, you tend to

feel rich. It's important to realize you're not. A rich company

is one with large revenues. This money isn't revenue. It's money

investors have given you in the hope you'll be able to generate

revenues. So despite those millions in the bank, you're still poor.For most startups the model should be grad student, not law firm.

Aim for cool and cheap, not expensive and impressive. For us the

test of whether a startup understood this was whether they had Aeron

chairs. The Aeron came out during the Bubble and was very popular

with startups. Especially the type, all too common then, that was

like a bunch of kids playing house with money supplied by VCs. We

had office chairs so cheap that the arms all fell off. This was

slightly embarrassing at the time, but in retrospect the grad-studenty

atmosphere of our office was another of those things we did right

without knowing it.Our offices were in a wooden triple-decker in Harvard Square. It

had been an apartment until about the 1970s, and there was still a

claw-footed bathtub in the bathroom. It must once have been inhabited

by someone fairly eccentric, because a lot of the chinks in the

walls were stuffed with aluminum foil, as if to protect against

cosmic rays. When eminent visitors came to see us, we were a bit

sheepish about the low production values. But in fact that place

was the perfect space for a startup. We felt like our role was to

be impudent underdogs instead of corporate stuffed shirts, and that

is exactly the spirit you want.An apartment is also the right kind of place for developing software.

Cube farms suck for that, as you've probably discovered if you've

tried it. Ever notice how much easier it is to hack at home than

at work? So why not make work more like home?When you're looking for space for a startup, don't feel that it has

to look professional. Professional means doing good work, not

elevators and glass walls. I'd advise most startups to avoid

corporate space at first and just rent an apartment. You want to

live at the office in a startup, so why not have a place designed

to be lived in as your office?Besides being cheaper and better to work in, apartments tend to be

in better locations than office buildings. And for a startup

location is very important. The key to productivity is for people

to come back to work after dinner. Those hours after the phone

stops ringing are by far the best for getting work done. Great

things happen when a group of employees go out to dinner together,

talk over ideas, and then come back to their offices to implement

them. So you want to be in a place where there are a lot of

restaurants around, not some dreary office park that's a wasteland

after 6:00 PM. Once a company shifts over into the model where

everyone drives home to the suburbs for dinner, however late, you've

lost something extraordinarily valuable. God help you if you

actually start in that mode.If I were going to start a startup today, there are only three

places I'd consider doing it: on the Red Line near Central, Harvard,

or Davis Squares (Kendall is too sterile); in Palo Alto on University

or California Aves; and in Berkeley immediately north or south of

campus. These are the only places I know that have the right kind

of vibe.The most important way to not spend money is by not hiring people.

I may be an extremist, but I think hiring people is the worst thing

a company can do. To start with, people are a recurring expense,

which is the worst kind. They also tend to cause you to grow out

of your space, and perhaps even move to the sort of uncool office

building that will make your software worse. But worst of all,

they slow you down: instead of sticking your head in someone's

office and checking out an idea with them, eight people have to

have a meeting about it. So the fewer people you can hire, the

better.During the Bubble a lot of startups had the opposite policy. They

wanted to get "staffed up" as soon as possible, as if you couldn't

get anything done unless there was someone with the corresponding

job title. That's big company thinking. Don't hire people to fill

the gaps in some a priori org chart. The only reason to hire someone

is to do something you'd like to do but can't.If hiring unnecessary people is expensive and slows you down, why

do nearly all companies do it? I think the main reason is that

people like the idea of having a lot of people working for them.

This weakness often extends right up to the CEO. If you ever end

up running a company, you'll find the most common question people

ask is how many employees you have. This is their way of weighing

you. It's not just random people who ask this; even reporters do.

And they're going to be a lot more impressed if the answer is a

thousand than if it's ten.This is ridiculous, really. If two companies have the same revenues,

it's the one with fewer employees that's more impressive. When

people used to ask me how many people our startup had, and I answered

"twenty," I could see them thinking that we didn't count for much.

I used to want to add "but our main competitor, whose ass we regularly

kick, has a hundred and forty, so can we have credit for the larger

of the two numbers?"As with office space, the number of your employees is a choice

between seeming impressive, and being impressive. Any of you who

were nerds in high school know about this

choice. Keep doing it when you start a company.Should You?But should you start a company? Are you the right sort of person

to do it? If you are, is it worth it?More people are the right sort of person to start a startup than

realize it. That's the main reason I wrote this. There could be

ten times more startups than there are, and that would probably be

a good thing.I was, I now realize, exactly the right sort of person to start a

startup. But the idea terrified me at first. I was forced into

it because I was a Lisp hacker. The company

I'd been consulting for seemed to be running into trouble, and there

were not a lot of other companies using Lisp. Since I couldn't

bear the thought of programming in another language (this was 1995,

remember, when "another language" meant C++) the only option seemed

to be to start a new company using Lisp.I realize this sounds far-fetched, but if you're a Lisp hacker

you'll know what I mean. And if the idea of starting a startup

frightened me so much that I only did it out of necessity, there

must be a lot of people who would be good at it but who are too

intimidated to try.So who should start a startup? Someone who is a good hacker, between

about 23 and 38, and who wants to solve the money problem in one

shot instead of getting paid gradually over a conventional working

life.I can't say precisely what a good hacker is. At a first rate

university this might include the top half of computer science

majors. Though of course you don't have to be a CS major to be a

hacker; I was a philosophy major in college.It's hard to tell whether you're a good hacker, especially when

you're young. Fortunately the process of starting startups tends

to select them automatically. What drives people to start startups

is (or should be) looking at existing technology and thinking, don't

these guys realize they should be doing x, y, and z? And that's

also a sign that one is a good hacker.I put the lower bound at 23 not because there's something that

doesn't happen to your brain till then, but because you need to see

what it's like in an existing business before you try running your

own. The business doesn't have to be a startup. I spent a year

working for a software company to pay off my college loans. It was

the worst year of my adult life, but I learned, without realizing

it at the time, a lot of valuable lessons about the software business.

In this case they were mostly negative lessons: don't have a lot

of meetings; don't have chunks of code that multiple people own;

don't have a sales guy running the company; don't make a high-end

product; don't let your code get too big; don't leave finding bugs

to QA people; don't go too long between releases; don't isolate

developers from users; don't move from Cambridge to Route 128; and

so on. [8] But negative lessons are just as valuable as positive

ones. Perhaps even more valuable: it's hard to repeat a brilliant

performance, but it's straightforward to avoid errors. [9]The other reason it's hard to start a company before 23 is that

people won't take you seriously. VCs won't trust you, and will try

to reduce you to a mascot as a condition of funding. Customers

will worry you're going to flake out and leave them stranded. Even

you yourself, unless you're very unusual, will feel your age to

some degree; you'll find it awkward to be the boss of someone much

older than you, and if you're 21, hiring only people younger rather

limits your options.Some people could probably start a company at 18 if they wanted to.

Bill Gates was 19 when he and Paul Allen started Microsoft. (Paul

Allen was 22, though, and that probably made a difference.) So if

you're thinking, I don't care what he says, I'm going to start a

company now, you may be the sort of person who could get away with

it.The other cutoff, 38, has a lot more play in it. One reason I put

it there is that I don't think many people have the physical stamina

much past that age. I used to work till 2:00 or 3:00 AM every

night, seven days a week. I don't know if I could do that now.Also,

startups are a big risk financially. If you try something that

blows up and leaves you broke at 26, big deal; a lot of 26 year

olds are broke. By 38 you can't take so many risks-- especially

if you have kids.My final test may be the most restrictive. Do you actually want

to start a startup? What it amounts to, economically, is compressing

your working life into the smallest possible space. Instead of

working at an ordinary rate for 40 years, you work like hell for

four. And maybe end up with nothing-- though in that case it

probably won't take four years.During this time you'll do little but work, because when you're not

working, your competitors will be. My only leisure activities were

running, which I needed to do to keep working anyway, and about

fifteen minutes of reading a night. I had a girlfriend for a total

of two months during that three year period. Every couple weeks I

would take a few hours off to visit a used bookshop or go to a

friend's house for dinner. I went to visit my family twice.

Otherwise I just worked.Working was often fun, because the people I worked with were some

of my best friends. Sometimes it was even technically interesting.

But only about 10% of the time. The best I can say for the other

90% is that some of it is funnier in hindsight than it seemed then.

Like the time the power went off in Cambridge for about six hours,

and we made the mistake of trying to start a gasoline powered

generator inside our offices. I won't try that again.I don't think the amount of bullshit you have to deal with in a

startup is more than you'd endure in an ordinary working life. It's

probably less, in fact; it just seems like a lot because it's

compressed into a short period. So mainly what a startup buys you

is time. That's the way to think about it if you're trying to

decide whether to start one. If you're the sort of person who would

like to solve the money problem once and for all instead of working

for a salary for 40 years, then a startup makes sense.For a lot of people the conflict is between startups and graduate

school. Grad students are just the age, and just the sort of people,

to start software startups. You may worry that if you do you'll

blow your chances of an academic career. But it's possible to be

part of a startup and stay in grad school, especially at first.

Two of our three original hackers were in grad school the whole

time, and both got their degrees.

There are few sources of energy

so powerful as a procrastinating grad student.If you do have to

leave grad school, in the worst case it won't be for too long. If

a startup fails, it will probably fail quickly enough that you can

return to academic life. And if it succeeds, you may find you no

longer have such a burning desire to be an assistant professor.If you want to do it, do it. Starting a startup is not the great

mystery it seems from outside. It's not something you have to know

about "business" to do. Build something users love, and spend less

than you make. How hard is that?Notes[1] Google's revenues are about two billion a year, but half comes

from ads on other sites.[2] One advantage startups have over established companies is that

there are no discrimination laws about starting businesses. For

example, I would be reluctant to start a startup with a woman

who had small children, or was likely to have them soon. But you're

not allowed to ask prospective employees if they plan to have kids

soon. Believe it or not, under current US law, you're not even

allowed to discriminate on the basis of intelligence. Whereas when

you're starting a company, you can discriminate on any basis you

want about who you start it with.[3] Learning to hack is a lot cheaper than business school, because

you can do it mostly on your own. For the price of a Linux box, a

copy of K&R, and a few hours of advice from your neighbor's fifteen

year old son, you'll be well on your way.[4] Corollary: Avoid starting a startup to sell things to the biggest

company of all, the government. Yes, there are lots of opportunities

to sell them technology. But let someone else start those startups.[5] A friend who started a company in Germany told me they do care

about the paperwork there, and that there's more of it. Which helps

explain why there are not more startups in Germany.[6] At the seed stage our valuation was in principle $100,000, because

Julian got 10% of the company. But this is a very misleading number,

because the money was the least important of the things Julian gave us.[7] The same goes for companies that seem to want to acquire you.

There will be a few that are only pretending to in order to pick

your brains. But you can never tell for sure which these are, so

the best approach is to seem entirely open, but to fail to mention

a few critical technical secrets.[8] I was as bad an employee as this place was a company. I

apologize to anyone who had to work with me there.[9] You could probably write a book about how to succeed in business

by doing everything in exactly the opposite way from the DMV.Thanks to Trevor Blackwell, Sarah Harlin, Jessica Livingston,

and Robert Morris for reading drafts of this essay, and to Steve

Melendez and Gregory Price for inviting me to speak.Domain Name SearchTurkish TranslationHebrew TranslationRussian TranslationChinese TranslationFrench TranslationJapanese TranslationArabic Translation

What You'll Wish You'd Known

January 2005(I wrote this talk for a

high school. I never actually

gave it, because the school authorities vetoed the plan to invite me.)When I said I was speaking at a high school, my friends were curious.

What will you say to high school students? So I asked them, what

do you wish someone had told you in high school? Their answers

were remarkably similar. So I'm going to tell you what we all wish

someone had told us.I'll start by telling you something you don't have to know in high

school: what you want to do with your life. People are always

asking you this, so you think you're supposed to have an answer.

But adults ask this mainly as a conversation starter. They want

to know what sort of person you are, and this question is just to

get you talking. They ask it the way you might poke a hermit crab

in a tide pool, to see what it does.If I were back in high school and someone asked about my plans, I'd

say that my first priority was to learn what the options were. You

don't need to be in a rush to choose your life's work. What you

need to do is discover what you like. You have to work on stuff

you like if you want to be good at what you do.It might seem that nothing would be easier than deciding what you

like, but it turns out to be hard, partly because it's hard to get

an accurate picture of most jobs. Being a doctor is not the way

it's portrayed on TV. Fortunately you can also watch real doctors,

by volunteering in hospitals. [1]But there are other jobs you can't learn about, because no one is

doing them yet. Most of the work I've done in the last ten years

didn't exist when I was in high school. The world changes fast,

and the rate at which it changes is itself speeding up. In such a

world it's not a good idea to have fixed plans.And yet every May, speakers all over the country fire up the Standard

Graduation Speech, the theme of which is: don't give up on your

dreams. I know what they mean, but this is a bad way to put it,

because it implies you're supposed to be bound by some plan you

made early on. The computer world has a name for this: premature

optimization. And it is synonymous with disaster. These speakers

would do better to say simply, don't give up.What they really mean is, don't get demoralized. Don't think that

you can't do what other people can. And I agree you shouldn't

underestimate your potential. People who've done great things tend

to seem as if they were a race apart. And most biographies only

exaggerate this illusion, partly due to the worshipful attitude

biographers inevitably sink into, and partly because, knowing how

the story ends, they can't help streamlining the plot till it seems

like the subject's life was a matter of destiny, the mere unfolding

of some innate genius. In fact I suspect if you had the sixteen

year old Shakespeare or Einstein in school with you, they'd seem

impressive, but not totally unlike your other friends.Which is an uncomfortable thought. If they were just like us, then

they had to work very hard to do what they did. And that's one

reason we like to believe in genius. It gives us an excuse for

being lazy. If these guys were able to do what they did only because

of some magic Shakespeareness or Einsteinness, then it's not our

fault if we can't do something as good.I'm not saying there's no such thing as genius. But if you're

trying to choose between two theories and one gives you an excuse

for being lazy, the other one is probably right.So far we've cut the Standard Graduation Speech down from "don't

give up on your dreams" to "what someone else can do, you can do."

But it needs to be cut still further. There is some variation

in natural ability. Most people overestimate its role, but it does

exist. If I were talking to a guy four feet tall whose ambition

was to play in the NBA, I'd feel pretty stupid saying, you can

do anything if you really try. [2]We need to cut the Standard Graduation Speech down to, "what someone

else with your abilities can do, you can do; and don't underestimate

your abilities." But as so often happens, the closer you get to

the truth, the messier your sentence gets. We've taken a nice,

neat (but wrong) slogan, and churned it up like a mud puddle. It

doesn't make a very good speech anymore. But worse still, it doesn't

tell you what to do anymore. Someone with your abilities? What

are your abilities?UpwindI think the solution is to work in the other direction. Instead

of working back from a goal, work forward from promising situations.

This is what most successful people actually do anyway.In the graduation-speech approach, you decide where you want to be

in twenty years, and then ask: what should I do now to get there?

I propose instead that you don't commit to anything in the future,

but just look at the options available now, and choose those that

will give you the most promising range of options afterward.It's not so important what you work on, so long as you're not wasting

your time. Work on things that interest you and increase your

options, and worry later about which you'll take.Suppose you're a college freshman deciding whether to major in math

or economics. Well, math will give you more options: you can go into

almost any field from math. If you major in math it will be easy

to get into grad school in economics, but if you major in economics

it will be hard to get into grad school in math.Flying a glider is a good metaphor here. Because a glider doesn't

have an engine, you can't fly into the wind without losing a lot

of altitude. If you let yourself get far downwind of good places

to land, your options narrow uncomfortably. As a rule you want to

stay upwind. So I propose that as a replacement for "don't give

up on your dreams." Stay upwind.How do you do that, though? Even if math is upwind of economics,

how are you supposed to know that as a high school student?Well, you don't, and that's what you need to find out. Look for smart people

and hard problems. Smart people tend to clump together, and if you

can find such a clump, it's probably worthwhile to join it. But

it's not straightforward to find these, because there is a lot of

faking going on.To a newly arrived undergraduate, all university departments look

much the same. The professors all seem forbiddingly intellectual

and publish papers unintelligible to outsiders. But while in some

fields the papers are unintelligible because they're full of hard

ideas, in others they're deliberately written in an obscure way to

seem as if they're saying something important. This may seem a

scandalous proposition, but it has been experimentally verified,

in the famous Social Text affair. Suspecting that the papers

published by literary theorists were often just intellectual-sounding

nonsense, a physicist deliberately wrote a paper full of

intellectual-sounding nonsense, and submitted it to a literary

theory journal, which published it.The best protection is always to be working on hard problems.

Writing novels is hard. Reading novels isn't.

Hard means worry: if you're not worrying that

something you're making will come out badly, or that you won't be

able to understand something you're studying, then it isn't hard

enough. There has to be suspense.Well, this seems a grim view of the world, you may think. What I'm

telling you is that you should worry? Yes, but it's not as bad as

it sounds. It's exhilarating to overcome worries. You don't see

faces much happier than people winning gold medals. And you know

why they're so happy? Relief.I'm not saying this is the only way to be happy. Just that some

kinds of worry are not as bad as they sound.AmbitionIn practice, "stay upwind" reduces to "work on hard problems." And

you can start today. I wish I'd grasped that in

high school.Most people like to be good at what they do. In the so-called real

world this need is a powerful force. But high school students

rarely benefit from it, because they're given a fake thing to do.

When I was in high school, I let myself believe that my job was to

be a high school student. And so I let my need to be good at what

I did be satisfied by merely doing well in school.If you'd asked me in high school what the difference was between

high school kids and adults, I'd have said it was that adults had

to earn a living. Wrong. It's that adults take responsibility for

themselves. Making a living is only a small part of it.

Far more important is to take intellectual responsibility for oneself.If I had to go through high school again, I'd treat it like a day

job. I don't mean that I'd slack in school. Working at something

as a day job doesn't mean doing it badly. It means not being defined

by it. I mean I wouldn't think of myself as a high school student,

just as a musician with a day job as a waiter doesn't think of

himself as a waiter. [3] And when I wasn't working at my day job

I'd start trying to do real work.When I ask people what they regret most about high school, they

nearly all say the same thing: that they wasted so much time. If

you're wondering what you're doing now that you'll regret most

later, that's probably it. [4]Some people say this is inevitable — that high school students

aren't capable of getting anything done yet. But I don't think

this is true. And the proof is that you're bored. You probably

weren't bored when you were eight. When you're eight it's called

"playing" instead of "hanging out," but it's the same thing. And

when I was eight, I was rarely bored. Give me a back yard and a

few other kids and I could play all day.The reason this got stale in middle school and high school, I now

realize, is that I was ready for something else. Childhood was

getting old.I'm not saying you shouldn't hang out with your friends — that you

should all become humorless little robots who do nothing but work.

Hanging out with friends is like chocolate cake. You enjoy it more

if you eat it occasionally than if you eat nothing but chocolate

cake for every meal. No matter how much you like chocolate cake,

you'll be pretty queasy after the third meal of it. And that's

what the malaise one feels in high school is: mental queasiness.

[5]You may be thinking, we have to do more than get good grades. We

have to have extracurricular activities. But you know

perfectly well how bogus most of these are. Collecting donations

for a charity is an admirable thing to do, but it's not hard.

It's not getting something done. What I mean by getting something

done is learning how to write well, or how to program computers,

or what life was really like in preindustrial societies, or how to

draw the human face from life. This sort of thing rarely translates

into a line item on a college application.CorruptionIt's dangerous to design your life around getting into college,

because the people you have to impress to get into college are not

a very discerning audience. At most colleges, it's not the professors

who decide whether you get in, but admissions officers, and they

are nowhere near as smart. They're the NCOs of the intellectual

world. They can't tell how smart you are.

The mere existence of prep schools is proof of that.Few parents

would pay so much for their kids to go to a school that didn't

improve their admissions prospects. Prep schools openly say this

is one of their aims. But what that means, if you stop to

think about it, is that they can

hack the admissions process: that they can take the very same kid

and make him seem a more appealing candidate than he would if he

went to the local public school. [6]Right now most of you feel your job in life is to be a promising

college applicant. But that means you're designing your life to

satisfy a process so mindless that there's a whole industry devoted

to subverting it. No wonder you become cynical. The malaise you

feel is the same that a producer of reality TV shows or a tobacco

industry executive feels. And you don't even get paid a lot.So what do you do? What you should not do is rebel. That's what

I did, and it was a mistake. I didn't realize exactly what was

happening to us, but I smelled a major rat. And so I just gave up.

Obviously the world sucked, so why bother?When I discovered that one of our teachers was herself using Cliff's

Notes, it seemed par for the course. Surely it meant nothing to

get a good grade in such a class.In retrospect this was stupid. It was like someone getting fouled

in a soccer game and saying, hey, you fouled me, that's against the

rules, and walking off the field in indignation. Fouls happen.

The thing to do when you get fouled is not to lose your cool. Just

keep playing. By putting you in this situation, society has fouled you. Yes,

as you suspect, a lot of the stuff you learn in your classes is

crap. And yes, as you suspect, the college admissions process is

largely a charade. But like many fouls, this one was unintentional.

[7] So just keep playing.Rebellion is almost as stupid as obedience. In either case you let

yourself be defined by what they tell you to do. The best plan, I

think, is to step onto an orthogonal vector. Don't just do what

they tell you, and don't just refuse to. Instead treat school as

a day job. As day jobs go, it's pretty sweet. You're done at 3

o'clock, and you can even work on your own stuff while you're there.CuriosityAnd what's your real job supposed to be? Unless you're Mozart,

your first task is to figure that out. What are the great things

to work on? Where are the imaginative people? And most importantly,

what are you interested in? The word "aptitude" is misleading,

because it implies something innate. The most powerful sort of

aptitude is a consuming interest in some question, and such interests

are often acquired tastes.A distorted version of this idea has filtered into popular culture

under the name "passion." I recently saw an ad for waiters saying

they wanted people with a "passion for service." The real thing

is not something one could have for waiting on tables. And passion

is a bad word for it. A better name would be curiosity.Kids are curious, but the curiosity I mean has a different shape from kid

curiosity. Kid curiosity is broad and shallow; they ask why at

random about everything. In most adults this curiosity dries up

entirely. It has to: you can't get anything done if you're always

asking why about everything. But in ambitious adults, instead of

drying up, curiosity becomes narrow and deep. The mud flat morphs

into a well.Curiosity turns work into play. For Einstein, relativity wasn't a

book full of hard stuff he had to learn for an exam. It was a

mystery he was trying to solve. So it probably felt like less work

to him to invent it than it would seem to someone now to learn it

in a class.One of the most dangerous illusions you get from school is the idea

that doing great things requires a lot of discipline. Most subjects

are taught in such a boring way that it's only by discipline that

you can flog yourself through them. So I was surprised when, early

in college, I read a quote by Wittgenstein saying that he had no

self-discipline and had never been able to deny himself anything,

not even a cup of coffee.Now I know a number of people who do great work, and it's the same

with all of them. They have little discipline. They're all terrible

procrastinators and find it almost impossible to make themselves

do anything they're not interested in. One still hasn't sent out

his half of the thank-you notes from his wedding, four years ago.

Another has 26,000 emails in her inbox.I'm not saying you can get away with zero self-discipline. You

probably need about the amount you need to go running. I'm often

reluctant to go running, but once I do, I enjoy it. And if I don't

run for several days, I feel ill. It's the same with people who

do great things. They know they'll feel bad if they don't work,

and they have enough discipline to get themselves to their desks

to start working. But once they get started, interest takes over,

and discipline is no longer necessary.Do you think Shakespeare was gritting his teeth and diligently

trying to write Great Literature? Of course not. He was having

fun. That's why he's so good.If you want to do good work, what you need is a great curiosity

about a promising question. The critical moment for Einstein

was when he looked at Maxwell's equations and said, what the hell

is going on here?It can take years to zero in on a productive question, because it

can take years to figure out what a subject is really about. To

take an extreme example, consider math. Most people think they

hate math, but the boring stuff you do in school under the name

"mathematics" is not at all like what mathematicians do.The great mathematician G. H. Hardy said he didn't like math in

high school either. He only took it up because he was better at

it than the other students. Only later did he realize math was

interesting — only later did he start to ask questions instead of

merely answering them correctly.When a friend of mine used to grumble because he had to write a

paper for school, his mother would tell him: find a way to make it

interesting. That's what you need to do: find a question that makes

the world interesting. People who do great things look at the same

world everyone else does, but notice some odd detail that's

compellingly mysterious.And not only in intellectual matters. Henry Ford's great question

was, why do cars have to be a luxury item? What would happen if

you treated them as a commodity? Franz Beckenbauer's was, in effect,

why does everyone have to stay in his position? Why can't defenders

score goals too?NowIf it takes years to articulate great questions, what do you do now,

at sixteen? Work toward finding one. Great questions don't appear

suddenly. They gradually congeal in your head. And what makes

them congeal is experience. So the way to find great questions is

not to search for them — not to wander about thinking, what great

discovery shall I make? You can't answer that; if you could, you'd

have made it.The way to get a big idea to appear in your head is not to hunt for

big ideas, but to put in a lot of time on work that interests you,

and in the process keep your mind open enough that a big idea can

take roost. Einstein, Ford, and Beckenbauer all used this recipe.

They all knew their work like a piano player knows the keys. So

when something seemed amiss to them, they had the confidence to

notice it.Put in time how and on what? Just pick a project that seems

interesting: to master some chunk of material, or to make something,

or to answer some question. Choose a project that will take less

than a month, and make it something you have the means to finish.

Do something hard enough to stretch you, but only just, especially

at first. If you're deciding between two projects, choose whichever

seems most fun. If one blows up in your face, start another. Repeat

till, like an internal combustion engine, the process becomes

self-sustaining, and each project generates the next one. (This

could take years.)It may be just as well not to do a project "for school," if that

will restrict you or make it seem like work. Involve your friends

if you want, but not too many, and only if they're not flakes.

Friends offer moral support (few startups are started by one person),

but secrecy also has its advantages. There's something pleasing

about a secret project. And you can take more risks, because no

one will know if you fail.Don't worry if a project doesn't seem to be on the path to some

goal you're supposed to have. Paths can bend a lot more than you

think. So let the path grow out the project. The most important

thing is to be excited about it, because it's by doing that you

learn.Don't disregard unseemly motivations. One of the most powerful is

the desire to be better than other people at something. Hardy said

that's what got him started, and I think the only unusual thing

about him is that he admitted it. Another powerful motivator is

the desire to do, or know, things you're not supposed to. Closely

related is the desire to do something audacious. Sixteen year olds

aren't supposed to write novels. So if you try, anything you achieve

is on the plus side of the ledger; if you fail utterly, you're doing

no worse than expectations. [8]Beware of bad models. Especially when they excuse laziness. When

I was in high school I used to write "existentialist" short stories

like ones I'd seen by famous writers. My stories didn't have a lot

of plot, but they were very deep. And they were less work to write

than entertaining ones would have been. I should have known that

was a danger sign. And in fact I found my stories pretty boring;

what excited me was the idea of writing serious, intellectual stuff

like the famous writers.Now I have enough experience to realize that those famous writers

actually sucked. Plenty of famous people do; in the short term,

the quality of one's work is only a small component of fame.

I should have been less worried about doing something

that seemed cool, and just done something I liked. That's the

actual road to coolness anyway.A key ingredient in many projects, almost a project on its own, is

to find good books. Most books are bad. Nearly all textbooks are

bad. [9] So don't assume a subject is to be learned from whatever

book on it happens to be closest. You have to search actively for

the tiny number of good books.The important thing is to get out there and do stuff. Instead of

waiting to be taught, go out and learn.Your life doesn't have to be shaped by admissions officers. It

could be shaped by your own curiosity. It is for all ambitious

adults. And you don't have to wait to start. In fact, you don't

have to wait to be an adult. There's no switch inside you that

magically flips when you turn a certain age or graduate from some

institution. You start being an adult when you decide to take

responsibility for your life. You can do that at any age. [10]This may sound like bullshit. I'm just a minor, you may think, I

have no money, I have to live at home, I have to do what adults

tell me all day long. Well, most adults labor under restrictions

just as cumbersome, and they manage to get things done. If you

think it's restrictive being a kid, imagine having kids.The only real difference between adults and high school kids is

that adults realize they need to get things done, and high school

kids don't. That realization hits most people around 23. But I'm

letting you in on the secret early. So get to work. Maybe you can

be the first generation whose greatest regret from high school isn't

how much time you wasted.

Notes[1] A doctor friend warns that even this can give an inaccurate

picture. "Who knew how much time it would take up, how little

autonomy one would have for endless years of training, and how

unbelievably annoying it is to carry a beeper?"[2] His best bet would probably be to become dictator and intimidate

the NBA into letting him play. So far the closest anyone has come

is Secretary of Labor.[3] A day job is one you take to pay the bills so you can do what

you really want, like play in a band, or invent relativity.Treating high school as a day job might actually make it easier for

some students to get good grades. If you treat your classes

as a game, you won't be demoralized if they seem pointless.However bad your classes, you need to get good grades in them to

get into a decent college. And that is worth doing, because

universities are where a lot of the clumps of smart people are these

days.[4] The second biggest regret was caring so much about unimportant

things. And especially about what other people thought of them.I think what they really mean, in the latter case, is caring what

random people thought of them. Adults care just as much what other

people think, but they get to be more selective about the other

people.I have about thirty friends whose opinions I care about,

and the opinion of the rest of the world barely affects me. The

problem in high school is that your peers are chosen for you by

accidents of age and geography, rather than by you based on respect

for their judgement.[5] The key to wasting time is distraction. Without distractions

it's too obvious to your brain that you're not doing anything with

it, and you start to feel uncomfortable. If you want to measure

how dependent you've become on distractions, try this experiment:

set aside a chunk of time on a weekend and sit alone and think.

You can have a notebook to write your thoughts down in, but nothing

else: no friends, TV, music, phone, IM, email, Web, games, books,

newspapers, or magazines. Within an hour most people will feel a

strong craving for distraction.[6] I don't mean to imply that the only function of prep schools

is to trick admissions officers. They also generally provide a

better education. But try this thought experiment: suppose prep

schools supplied the same superior education but had a tiny (.001)

negative effect on college admissions. How many parents would still

send their kids to them?It might also be argued that kids who went to prep schools, because

they've learned more, are better college candidates. But

this seems empirically false. What you learn in even the best high

school is rounding error compared to what you learn in college.

Public school kids arrive at college with a slight disadvantage,

but they start to pull ahead in the sophomore year.(I'm not saying public school kids are smarter than preppies, just

that they are within any given college. That follows necessarily

if you agree prep schools improve kids' admissions prospects.)[7] Why does society foul you? Indifference, mainly. There are

simply no outside forces pushing high school to be good. The air

traffic control system works because planes would crash otherwise.

Businesses have to deliver because otherwise competitors would take

their customers. But no planes crash if your school sucks, and it

has no competitors. High school isn't evil; it's random; but random

is pretty bad.[8] And then of course there is money. It's not a big factor in

high school, because you can't do much that anyone wants. But a

lot of great things were created mainly to make money. Samuel

Johnson said "no man but a blockhead ever wrote except for money."

(Many hope he was exaggerating.)[9] Even college textbooks are bad. When you get to college,

you'll find that (with a few stellar exceptions) the textbooks are

not written by the leading scholars in the field they describe.

Writing college textbooks is unpleasant work, done mostly by people

who need the money. It's unpleasant because the publishers exert

so much control, and there are few things worse than close supervision

by someone who doesn't understand what you're doing. This phenomenon

is apparently

even worse in the production of high school textbooks.[10] Your teachers are always telling you to behave like adults.

I wonder if they'd like it if you did. You may be loud and

disorganized, but you're very docile compared to adults. If you

actually started acting like adults, it would be just as if a bunch

of adults had been transposed into your bodies. Imagine the reaction

of an FBI agent or taxi driver or reporter to being told they had

to ask permission to go the bathroom, and only one person could go

at a time. To say nothing of the things you're taught. If a bunch

of actual adults suddenly found themselves trapped in high school,

the first thing they'd do is form a union and renegotiate all the

rules with the administration.Thanks to Ingrid Bassett, Trevor Blackwell,

Rich Draves, Dan Giffin, Sarah

Harlin, Jessica Livingston, Jackie McDonough, Robert Morris, Mark Nitzberg, Lisa

Randall, and Aaron Swartz for reading drafts of this, and to many

others for talking to me about high school.Why Nerds are UnpopularJapanese TranslationRussian TranslationGeorgian Translation

Made in USA

November 2004(This is a new essay for the Japanese edition of

Hackers

& Painters.

It tries to explain why Americans make some things well

and others badly.)A few years ago an Italian friend of mine travelled by train from

Boston to Providence. She had only been in America for a

couple weeks and hadn't seen much of the country yet. She arrived

looking astonished. "It's so ugly!"People from other rich countries can scarcely imagine

the squalor of the man-made bits of America. In travel books

they show you mostly natural environments: the Grand Canyon,

whitewater rafting, horses in a field. If you see

pictures with man-made things in them, it will be either a

view of the New York skyline shot from a discreet distance,

or a carefully cropped image of a seacoast town in Maine.How can it be, visitors must wonder. How can the richest country

in the world look like this?Oddly enough, it may not be a coincidence. Americans are good

at some things and bad at others. We're good at making

movies and software, and bad at making cars and cities.

And I think we may be good at what we're good at for the same

reason we're bad at what we're bad at. We're impatient.

In America, if you want to do something, you don't worry that

it might come out badly, or upset delicate social balances, or

that people might think you're getting above yourself. If you

want to do something, as Nike says, just do it.This works well in some fields and badly in others. I suspect

it works in movies and software because they're both messy

processes. "Systematic"

is the last word I'd use to describe the way

good programmers write software.

Code is not something they assemble painstakingly after

careful planning, like the pyramids. It's something they

plunge into, working fast and constantly changing their minds,

like a charcoal sketch.In software, paradoxical

as it sounds, good craftsmanship means working fast.

If you work slowly and meticulously,

you merely end up with a very fine implementation of your initial,

mistaken idea.

Working slowly and meticulously is

premature optimization. Better to get a

prototype done fast, and see what new ideas

it gives you.It sounds like making movies works a lot like making software.

Every movie is a Frankenstein, full of imperfections

and usually quite different from what was originally envisioned.

But interesting, and finished fairly quickly. I think we get away with this in movies and software

because they're both malleable mediums. Boldness pays.

And if at the last minute two parts don't quite

fit, you can figure out some hack that will at least conceal

the problem.Not so with cars, or cities. They are all too physical.

If the car business worked like software or movies, you'd

surpass your competitors by making a car that weighed only

fifty pounds, or folded up to the size of a motorcycle when

you wanted to park it. But with physical products there are

more constraints. You don't win by dramatic innovations

so much as by good taste and attention to detail.The trouble is, the very word "taste"

sounds slightly ridiculous to American ears.

It seems pretentious, or frivolous, or even effeminate.

Blue staters think it's "subjective," and red staters

think it's for sissies. So anyone in America

who really cares about design will be sailing upwind.Twenty years ago we used to hear that the problem with

the US car industry was the workers.

We don't hear that any more now that Japanese companies

are building cars in the US. The problem with

American cars is bad design. You can see that just by

looking at them.All that extra sheet metal on the AMC Matador wasn't

added by the workers. The problem

with this car, as with American cars today, is that it was

designed by marketing people instead of designers.Why do the Japanese make better cars than us? Some say it's

because their culture encourages cooperation. That may come

into it. But in this case it seems more to the point that

their culture prizes design and craftsmanship.For centuries the Japanese have made finer things than we

have in the West. When you look at swords they

made in 1200, you just can't believe the date on the label

is right.

Presumably their cars fit together more

precisely than ours for the same reason their joinery always has.

They're obsessed with making things well.Not us.

When we make something in America, our aim is just to get the

job done. Once we reach that point, we take one of two routes.

We can stop there, and have something crude but

serviceable, like a Vise-grip. Or we can improve it,

which usually means encrusting it with gratuitous ornament.

When we want to make a car "better,"

we stick tail fins on it, or make it

longer, or make the

windows smaller, depending on the current fashion.Ditto for houses. In America you can have either a flimsy box banged

together out of two by fours and drywall, or a McMansion-- a

flimsy box banged together out of two by fours and drywall,

but larger, more dramatic-looking, and full of expensive fittings.

Rich people don't get better design or craftsmanship;

they just get a larger, more conspicuous version of the

standard house.We don't especially prize design or craftsmanship here. What

we like is speed, and we're willing to do something in an ugly

way to get it done fast. In some

fields, like software or movies, this is a net win.

But it's not just that software and movies are malleable mediums.

In those businesses, the designers (though they're

not generally called that) have more power.

Software companies, at least successful ones, tend to be run

by programmers. And in the film industry, though producers

may second-guess directors, the director controls most of

what appears on the screen.

And so American software and movies, and Japanese cars, all

have this in common: the people in charge care about

design-- the former because the designers are in charge, and the latter

because the whole culture cares about design.I think most Japanese executives would be horrified at

the idea of making a bad car. Whereas American executives,

in their hearts, still believe the most important thing about

a car is the image it projects.

Make a good car? What's "good?" It's so subjective.

If you want to know how to design a car, ask a focus group.Instead of relying on their own internal design compass

(like Henry Ford did),

American car companies try to make what marketing people

think consumers want. But it isn't working. American cars continue

to lose market share. And the reason is that the customer

doesn't want what he thinks he wants.Letting focus groups design your cars for you

only wins in the short term. In the long term, it pays

to bet on good design. The focus group may say they want the

meretricious feature du jour, but what they want even more is

to imitate sophisticated buyers, and they, though a

small minority, really do care about good design.

Eventually the

pimps and drug dealers notice that the doctors and lawyers

have switched from Cadillac to Lexus, and do the same.Apple is an interesting counterexample to the general

American trend. If you want to buy a nice CD player, you'll

probably buy a Japanese one. But if you want to buy an

MP3 player, you'll probably buy an iPod. What happened?

Why doesn't Sony dominate MP3 players? Because Apple is

in the consumer electronics business now, and unlike

other American companies, they're obsessed with good design.

Or more precisely, their CEO is.I just got an iPod, and it's not just nice. It's

surprisingly nice. For it to surprise me, it must be

satisfying expectations I didn't know I had. No focus

group is going to discover those. Only a great

designer can.Cars aren't the worst thing we make in America.

Where the just-do-it model fails most dramatically is in our cities-- or

rather, exurbs.

If real estate developers operated on a large enough scale, if

they built whole towns, market forces would compel

them to build towns that didn't suck. But they only build a

couple office buildings or suburban streets at a time, and the

result is so depressing that the inhabitants consider it a great

treat to fly to Europe and spend a couple weeks living what

is, for people there, just everyday life. [1]But the just-do-it model does have advantages. It seems the clear

winner for generating wealth and technical innovations

(which are practically the same thing). I think speed is the reason.

It's hard to create wealth by making a commodity. The

real value is in things that are new, and if you want to

be the first to make something, it helps to work fast.

For better or worse, the just-do-it model is fast,

whether you're Dan Bricklin writing the prototype of VisiCalc in

a weekend, or a real estate developer

building a block of shoddy condos in a month.If I had to choose between the just-do-it model and the

careful model, I'd probably choose just-do-it.

But do we have to choose? Could we have it both ways?

Could Americans have nice

places to live without undermining the impatient, individualistic spirit

that makes us good at software? Could other countries

introduce more individualism into their technology companies

and research labs without having it metastasize as strip malls?

I'm optimistic. It's harder to

say about other countries, but in the US, at least, I think

we can have both.Apple is an encouraging example. They've managed to preserve

enough of the impatient, hackerly spirit you need to write

software. And yet when

you pick up a new Apple laptop, well, it doesn't

seem American. It's too perfect. It seems as if it

must have been made by a Swedish or a Japanese company.In many technologies, version 2 has higher resolution. Why

not in design generally? I think we'll gradually see

national characters superseded

by occupational characters: hackers in Japan will be allowed

to behave with a willfulness

that would now seem unJapanese,

and products in America will be designed with an

insistence on taste that would now seem unAmerican.

Perhaps the most successful countries, in the future, will be

those most willing to ignore what are now considered

national characters, and do each kind of work in the way

that works best. Race you.Notes[1] Japanese cities are ugly too, but for different reasons.

Japan is prone to earthquakes, so buildings are traditionally

seen as temporary; there is no grand tradition of city planning

like the one Europeans inherited from Rome. The other cause is

the notoriously corrupt relationship between the government

and construction companies.Thanks to Trevor Blackwell, Barry Eisler, Sarah Harlin,

Shiro Kawai, Jessica Livingston, Jackie McDonough, Robert Morris,

and Eric Raymond

for reading drafts of this.American GothicThe John Rain Books

It's Charisma, Stupid

November 2004, corrected June 2006Occam's razor says we should prefer the simpler of two explanations.

I begin by reminding readers of this principle because I'm about

to propose a theory that will offend both liberals and conservatives.

But Occam's razor means, in effect, that if you want to disagree

with it, you have a hell of a coincidence to explain.Theory: In US presidential elections, the more

charismatic candidate wins.People who write about politics, whether on the left or the right,

have a consistent bias: they take politics seriously. When one

candidate beats another they look for political explanations. The

country is shifting to the left, or the right. And that sort of

shift can certainly be the result of a presidential election, which

makes it easy to believe it was the cause.But when I think about why I voted for Clinton over the first George

Bush, it wasn't because I was shifting to the left. Clinton just

seemed more dynamic. He seemed to want the job more. Bush seemed

old and tired. I suspect it was the same for a lot of voters.Clinton didn't represent any national shift leftward.

[1]

He was

just more charismatic than George Bush or (God help us) Bob Dole.

In 2000 we practically got a controlled experiment to prove it:

Gore had Clinton's policies, but not his charisma, and he suffered

proportionally.

[2]

Same story in 2004. Kerry was smarter and more

articulate than Bush, but rather a stiff. And Kerry lost.As I looked further back, I kept finding the same pattern.

Pundits said Carter beat Ford

because the country distrusted the Republicans after Watergate.

And yet it also happened that Carter was famous for his big grin

and folksy ways, and Ford for being a boring klutz. Four years

later, pundits said the country had lurched to the right. But

Reagan, a former actor, also happened to be even more charismatic

than Carter (whose grin was somewhat less cheery after four stressful

years in office). In 1984 the charisma gap between Reagan and

Mondale was like that between Clinton and Dole, with similar results.

The first George Bush managed to win in 1988, though he would later

be vanquished by one of the most charismatic presidents ever, because

in 1988 he was up against the notoriously uncharismatic

Michael Dukakis.These are the elections I remember personally, but apparently the

same pattern played out in 1964 and 1972. The most recent

counterexample appears to be 1968, when Nixon beat the more charismatic Hubert

Humphrey. But when you examine that election, it tends to support

the charisma theory more than contradict it. As Joe McGinnis

recounts in his famous book The Selling of the President 1968,

Nixon knew he had less charisma than Humphrey, and thus simply

refused to debate him on TV. He knew he couldn't afford to let the

two of them be seen side by side.Now a candidate probably couldn't get away with refusing to debate.

But in 1968 the custom of televised debates was still evolving. In

effect, Nixon won in 1968 because voters were never allowed to see

the real Nixon. All they saw were carefully scripted campaign

spots.Oddly enough, the most recent true counterexample is probably 1960.

Though this election is usually given as an example of the power

of TV, Kennedy apparently would not have won without fraud by party

machines in Illinois and Texas. But TV was still young in 1960;

only 87% of households had it.

[3]

Undoubtedly TV helped Kennedy,

so historians are correct in regarding this election as a

watershed. TV required a new kind of candidate. There would be no

more Calvin Coolidges.The charisma theory may also explain why Democrats tend to lose

presidential elections. The core of the Democrats' ideology seems

to be a belief in government. Perhaps this tends to attract people

who are earnest, but dull. Dukakis, Gore, and Kerry were so similar

in that respect that they might have been brothers. Good thing for

the Democrats that their screen lets through an occasional Clinton,

even if some scandal results.

[4]One would like to believe elections are won and lost on issues, if

only fake ones like Willie Horton. And yet, if they are, we have

a remarkable coincidence to explain. In every presidential election

since TV became widespread, the apparently more charismatic candidate

has won. Surprising, isn't it, that voters' opinions on the issues

have lined up with charisma for 11 elections in a row?The political commentators who come up with shifts to the left or

right in their morning-after analyses are like the financial reporters

stuck writing stories day after day about the random fluctuations

of the stock market. Day ends, market closes up or down, reporter

looks for good or bad news respectively, and writes that the market

was up on news of Intel's earnings, or down on fears of instability

in the Middle East. Suppose we could somehow feed these reporters

false information about market closes, but give them all the other

news intact. Does anyone believe they would notice the anomaly,

and not simply write that stocks were up (or down) on whatever good

(or bad) news there was that day? That they would say, hey, wait

a minute, how can stocks be up with all this unrest in the Middle

East?I'm not saying that issues don't matter to voters. Of course they

do. But the major parties know so well which issues matter how

much to how many voters, and adjust their message so precisely in

response, that they tend to split the difference on the issues,

leaving the election to be decided by the one factor they can't

control: charisma.If the Democrats had been running a candidate as charismatic as

Clinton in the 2004 election, he'd have won. And we'd be reading

that the election was a referendum on the war in Iraq, instead of

that the Democrats are out of touch with evangelical Christians in

middle America.During the 1992 election, the Clinton campaign staff had a big sign

in their office saying "It's the economy, stupid." Perhaps it was

even simpler than they thought.PostscriptOpinions seem to be divided about the charisma theory. Some say

it's impossible, others say it's obvious. This seems a good sign.

Perhaps it's in the sweet spot midway between.As for it being impossible, I reply: here's the data; here's the

theory; theory explains data 100%. To a scientist, at least, that

means it deserves attention, however implausible it seems.You can't believe voters are so superficial that they just choose

the most charismatic guy? My theory doesn't require that. I'm not

proposing that charisma is the only factor, just that it's the only

one left after the efforts of the two parties cancel one

another out.As for the theory being obvious, as far as I know, no one has

proposed it before. Election forecasters are proud when they can

achieve the same results with much more complicated models.Finally, to the people who say that the theory is probably true,

but rather depressing: it's not so bad as it seems. The phenomenon

is like a pricing anomaly; once people realize it's there, it will

disappear. Once both parties realize it's a waste of time to

nominate uncharismatic candidates, they'll tend to nominate only

the most charismatic ones. And if the candidates are equally

charismatic, charisma will cancel out, and elections will be decided

on issues, as political commentators like to think they are now.Notes[1]

As Clinton himself discovered to his surprise when, in one of

his first acts as president, he tried to shift the military leftward.

After a bruising fight he escaped with a face-saving compromise.[2]

True, Gore won the popular vote. But politicians know the electoral

vote decides the election, so that's what they campaign for. If Bush

had been campaigning for the popular vote he would presumably have

got more of it. (Thanks to judgmentalist for this point.)[3]

Source: Nielsen Media Research. Of the remaining 13%, 11 didn't

have TV because they couldn't afford it. I'd argue that the missing

11% were probably also the 11% most susceptible to charisma.[4]

One implication of this theory is that parties shouldn't be too

quick to reject candidates with skeletons in their closets.

Charismatic candidates will tend to have more skeletons than squeaky

clean dullards, but in practice that doesn't seem to lose elections.

The current Bush, for example, probably did more drugs in his

twenties than any preceding president, and yet managed to get elected

with a base of evangelical Christians. All you have to do is say

you've reformed, and stonewall about the details.Thanks to Trevor Blackwell, Maria Daniels, Jessica Livingston,

Jackie McDonough, and Robert Morris for reading drafts of this, and

to Eric Raymond for pointing out that I was wrong about 1968.

Comment on this essay.What Charisma IsPolitics and the Art of ActingJapanese Translation

Bradley's Ghost

November 2004

A lot of people are writing now about

why Kerry lost. Here I want to

examine a more specific question: why were the exit polls so

wrong?In Ohio, which Kerry ultimately

lost 49-51, exit polls gave him a 52-48 victory. And this wasn't just

random error. In every swing state they overestimated the Kerry vote.

In Florida, which Bush ultimately won 52-47, exit polls predicted

a dead heat.(These are not early numbers. They're from about midnight eastern time,

long after polls closed in Ohio and Florida. And yet by the

next afternoon the exit poll numbers online corresponded to the returns.

The only way I can imagine this happening is if those in

charge of the exit polls cooked the books after seeing the actual

returns. But that's another issue.)What happened? The source of the problem may be a variant of

the Bradley Effect.

This term

was invented after Tom Bradley, the black mayor of Los Angeles,

lost an election for governor of California despite a comfortable

lead in the polls. Apparently voters were afraid to say

they planned to vote against him, lest their motives be

(perhaps correctly) suspected.It seems likely that something similar happened in exit polls this year.

In theory, exit polls ought to be very accurate.

You're not asking people what they would do. You're

asking what they just did.How can you get errors asking that? Because some people don't

respond. To get a truly random sample, pollsters ask, say, every

20th person leaving the polling place who they voted for. But not

everyone wants to answer. And the pollsters can't simply ignore

those who won't, or their sample isn't random anymore. So what

they do, apparently, is note down the age and race and sex of the

person, and guess from that who they voted for.This works so long as there is no correlation between who people

vote for and whether they're willing to talk about it. But this

year there may have been. It may be that a significant number of

those who voted for

Bush didn't want to say so.Why not? Because people in the US are more conservative than they're

willing to admit. The values of the elite in this country, at least

at the moment, are NPR values. The average person, as I think both

Republicans and Democrats would agree, is more socially conservative.

But while some openly flaunt the fact that they don't share the

opinions of the elite, others feel a little nervous about it, as

if they had bad table manners.For example, according to current NPR values, you

can't say anything that might be

perceived as disparaging towards homosexuals. To do

so is "homophobic." And yet a large number of Americans are deeply

religious, and the Bible is quite explicit on the subject of

homosexuality. What are they to do? I think what many do is keep

their opinions, but keep them to themselves.They know what they believe, but they also know what they're supposed

to believe.

And so when a stranger (for example, a pollster) asks

them their opinion about something like gay marriage, they will not

always say what they really think.When the values of the elite are liberal, polls will tend to

underestimate the conservativeness of ordinary voters. This seems

to me the leading theory to explain why the exit polls were so

far off this year. NPR values

said one ought to vote for Kerry. So all the people who voted for

Kerry felt virtuous for doing so, and were eager to tell pollsters

they had. No one who voted for Kerry did it as an act of quiet

defiance.Support for a Woman PresidentJapanese Translation

If you liked this, you may also like

Hackers & Painters.

A Version 1.0

October 2004

As E. B. White said, "good writing is rewriting." I didn't

realize this when I was in school. In writing, as in math and

science, they only show you the finished product.

You don't see all the false starts. This gives students a

misleading view of how things get made.Part of the reason it happens is that writers don't want

people to see their mistakes. But I'm willing to let people

see an early draft if it will show how much you have

to rewrite to beat an essay into shape.Below is the oldest version I can find of

The Age of the Essay

(probably the second or third day), with

text that ultimately survived in

red and text that later

got deleted in gray.

There seem to be several categories of cuts: things I got wrong,

things that seem like bragging, flames,

digressions, stretches of awkward prose, and unnecessary words.I discarded more from the beginning. That's

not surprising; it takes a while to hit your stride. There

are more digressions at the start, because I'm not sure where

I'm heading.The amount of cutting is about average. I probably write

three to four words for every one that appears in the final

version of an essay.(Before anyone gets mad at me for opinions expressed here, remember

that anything you see here that's not in the final version is obviously

something I chose not to publish, often because I disagree

with it.)

Recently a friend said that what he liked about

my essays was that they weren't written the way

we'd been taught to write essays in school. You

remember: topic sentence, introductory paragraph,

supporting paragraphs, conclusion. It hadn't

occurred to me till then that those horrible things

we had to write in school were even connected to

what I was doing now. But sure enough, I thought,

they did call them "essays," didn't they?Well, they're not. Those things you have to write

in school are not only not essays, they're one of the

most pointless of all the pointless hoops you have

to jump through in school. And I worry that they

not only teach students the wrong things about writing,

but put them off writing entirely.So I'm going to give the other side of the story: what

an essay really is, and how you write one. Or at least,

how I write one. Students be forewarned: if you actually write

the kind of essay I describe, you'll probably get bad

grades. But knowing how it's really done should

at least help you to understand the feeling of futility

you have when you're writing the things they tell you to.

The most obvious difference between real essays and

the things one has to write in school is that real

essays are not exclusively about English literature.

It's a fine thing for schools to

teach students how to

write. But for some bizarre reason (actually, a very specific bizarre

reason that I'll explain in a moment),

the teaching of

writing has gotten mixed together with the study

of literature. And so all over the country, students are

writing not about how a baseball team with a small budget

might compete with the Yankees, or the role of color in

fashion, or what constitutes a good dessert, but about

symbolism in Dickens.With obvious

results. Only a few people really

care about

symbolism in Dickens. The teacher doesn't.

The students don't. Most of the people who've had to write PhD

disserations about Dickens don't. And certainly

Dickens himself would be more interested in an essay

about color or baseball.How did things get this way? To answer that we have to go back

almost a thousand years. Between about 500 and 1000, life was

not very good in Europe. The term "dark ages" is presently

out of fashion as too judgemental (the period wasn't dark;

it was just different), but if this label didn't already

exist, it would seem an inspired metaphor. What little

original thought there was took place in lulls between

constant wars and had something of the character of

the thoughts of parents with a new baby.

The most amusing thing written during this

period, Liudprand of Cremona's Embassy to Constantinople, is,

I suspect, mostly inadvertantly so.Around 1000 Europe began to catch its breath.

And once they

had the luxury of curiosity, one of the first things they discovered

was what we call "the classics."

Imagine if we were visited

by aliens. If they could even get here they'd presumably know a

few things we don't. Immediately Alien Studies would become

the most dynamic field of scholarship: instead of painstakingly

discovering things for ourselves, we could simply suck up

everything they'd discovered. So it was in Europe in 1200.

When classical texts began to circulate in Europe, they contained

not just new answers, but new questions. (If anyone proved

a theorem in christian Europe before 1200, for example, there

is no record of it.)For a couple centuries, some of the most important work

being done was intellectual archaelogy. Those were also

the centuries during which schools were first established.

And since reading ancient texts was the essence of what

scholars did then, it became the basis of the curriculum.By 1700, someone who wanted to learn about

physics didn't need to start by mastering Greek in order to read Aristotle. But schools

change slower than scholarship: the study of

ancient texts

had such prestige that it remained the backbone of

education

until the late 19th century. By then it was merely a tradition.

It did serve some purposes: reading a foreign language was difficult,

and thus taught discipline, or at least, kept students busy;

it introduced students to

cultures quite different from their own; and its very uselessness

made it function (like white gloves) as a social bulwark.

But it certainly wasn't

true, and hadn't been true for centuries, that students were

serving apprenticeships in the hottest area of scholarship.Classical scholarship had also changed. In the early era, philology

actually mattered. The texts that filtered into Europe were

all corrupted to some degree by the errors of translators and

copyists. Scholars had to figure out what Aristotle said

before they could figure out what he meant. But by the modern

era such questions were answered as well as they were ever

going to be. And so the study of ancient texts became less

about ancientness and more about texts.The time was then ripe for the question: if the study of

ancient texts is a valid field for scholarship, why not modern

texts? The answer, of course, is that the raison d'etre

of classical scholarship was a kind of intellectual archaelogy that

does not need to be done in the case of contemporary authors.

But for obvious reasons no one wanted to give that answer.

The archaeological work being mostly done, it implied that

the people studying the classics were, if not wasting their

time, at least working on problems of minor importance.And so began the study of modern literature. There was some

initial resistance, but it didn't last long.

The limiting

reagent in the growth of university departments is what

parents will let undergraduates study. If parents will let

their children major in x, the rest follows straightforwardly.

There will be jobs teaching x, and professors to fill them.

The professors will establish scholarly journals and publish

one another's papers. Universities with x departments will

subscribe to the journals. Graduate students who want jobs

as professors of x will write dissertations about it. It may

take a good long while for the more prestigious universities

to cave in and establish departments in cheesier xes, but

at the other end of the scale there are so many universities

competing to attract students that the mere establishment of

a discipline requires little more than the desire to do it.High schools imitate universities.

And so once university

English departments were established in the late nineteenth century,

the 'riting component of the 3 Rs

was morphed into English.

With the bizarre consequence that high school students now

had to write about English literature-- to write, without

even realizing it, imitations of whatever

English professors had been publishing in their journals a

few decades before. It's no wonder if this seems to the

student a pointless exercise, because we're now three steps

removed from real work: the students are imitating English

professors, who are imitating classical scholars, who are

merely the inheritors of a tradition growing out of what

was, 700 years ago, fascinating and urgently needed work.Perhaps high schools should drop English and just teach writing.

The valuable part of English classes is learning to write, and

that could be taught better by itself. Students learn better

when they're interested in what they're doing, and it's hard

to imagine a topic less interesting than symbolism in Dickens.

Most of the people who write about that sort of thing professionally

are not really interested in it. (Though indeed, it's been a

while since they were writing about symbolism; now they're

writing about gender.)I have no illusions about how eagerly this suggestion will

be adopted. Public schools probably couldn't stop teaching

English even if they wanted to; they're probably required to by

law. But here's a related suggestion that goes with the grain

instead of against it: that universities establish a

writing major. Many of the students who now major in English

would major in writing if they could, and most would

be better off.It will be argued that it is a good thing for students to be

exposed to their literary heritage. Certainly. But is that

more important than that they learn to write well? And are

English classes even the place to do it? After all,

the average public high school student gets zero exposure to

his artistic heritage. No disaster results.

The people who are interested in art learn about it for

themselves, and those who aren't don't. I find that American

adults are no better or worse informed about literature than

art, despite the fact that they spent years studying literature

in high school and no time at all studying art. Which presumably

means that what they're taught in school is rounding error

compared to what they pick up on their own.Indeed, English classes may even be harmful. In my case they

were effectively aversion therapy. Want to make someone dislike

a book? Force him to read it and write an essay about it.

And make the topic so intellectually bogus that you

could not, if asked, explain why one ought to write about it.

I love to read more than anything, but by the end of high school

I never read the books we were assigned. I was so disgusted with

what we were doing that it became a point of honor

with me to write nonsense at least as good at the other students'

without having more than glanced over the book to learn the names

of the characters and a few random events in it.I hoped this might be fixed in college, but I found the same

problem there. It was not the teachers. It was English.

We were supposed to read novels and write essays about them.

About what, and why? That no one seemed to be able to explain.

Eventually by trial and error I found that what the teacher

wanted us to do was pretend that the story had really taken

place, and to analyze based on what the characters said and did (the

subtler clues, the better) what their motives must have been.

One got extra credit for motives having to do with class,

as I suspect one must now for those involving gender and

sexuality. I learned how to churn out such stuff well enough

to get an A, but I never took another English class.And the books we did these disgusting things to, like those

we mishandled in high school, I find still have black marks

against them in my mind. The one saving grace was that

English courses tend to favor pompous, dull writers like

Henry James, who deserve black marks against their names anyway.

One of the principles the IRS uses in deciding whether to

allow deductions is that, if something is fun, it isn't work.

Fields that are intellectually unsure of themselves rely on

a similar principle. Reading P.G. Wodehouse or Evelyn Waugh or

Raymond Chandler is too obviously pleasing to seem like

serious work, as reading Shakespeare would have been before

English evolved enough to make it an effort to understand him. [sh]

And so good writers (just you wait and see who's still in

print in 300 years) are less likely to have readers turned

against them by clumsy, self-appointed tour guides.

The other big difference between a real essay and the

things

they make you write in school is that a real essay doesn't

take a position and then defend it. That principle,

like the idea that we ought to be writing about literature,

turns out to be another intellectual hangover of long

forgotten origins. It's often mistakenly believed that

medieval universities were mostly seminaries. In fact they

were more law schools. And at least in our tradition

lawyers are advocates: they are

trained to be able to

take

either side of an argument and make as good a case for it

as they can. Whether or not this is a good idea (in the case of prosecutors,

it probably isn't), it tended to pervade

the atmosphere of

early universities. After the lecture the most common form

of discussion was the disputation. This idea

is at least

nominally preserved in our present-day thesis defense-- indeed,

in the very word thesis. Most people treat the words

thesis

and dissertation as interchangeable, but originally, at least,

a thesis was a position one took and the dissertation was

the argument by which one defended it.I'm not complaining that we blur these two words together.

As far as I'm concerned, the sooner we lose the original

sense of the word thesis, the better. For many, perhaps most,

graduate students, it is stuffing a square peg into a round

hole to try to recast one's work as a single thesis. And

as for the disputation, that seems clearly a net lose.

Arguing two sides of a case may be a necessary evil in a

legal dispute, but it's not the best way to get at the truth,

as I think lawyers would be the first to admit.

And yet this principle is built into the very structure of

the essays

they teach you to write in high school. The topic

sentence is your thesis, chosen in advance, the supporting

paragraphs the blows you strike in the conflict, and the

conclusion--- uh, what it the conclusion? I was never sure

about that in high school. If your thesis was well expressed,

what need was there to restate it? In theory it seemed that

the conclusion of a really good essay ought not to need to

say any more than QED.

But when you understand the origins

of this sort of "essay", you can see where the

conclusion comes from. It's the concluding remarks to the

jury.

What other alternative is there? To answer that

we have to

reach back into history again, though this time not so far.

To Michel de Montaigne, inventor of the essay.

He was

doing something quite different from what a

lawyer does,

and

the difference is embodied in the name. Essayer is the French

verb meaning "to try" (the cousin of our word assay),

and an "essai" is an effort.

An essay is something you

write in order

to figure something out.Figure out what? You don't know yet. And so you can't begin with a

thesis, because you don't have one, and may never have

one. An essay doesn't begin with a statement, but with a

question. In a real essay, you don't take a position and

defend it. You see a door that's ajar, and you open it and

walk in to see what's inside.If all you want to do is figure things out, why do you need

to write anything, though? Why not just sit and think? Well,

there precisely is Montaigne's great discovery. Expressing

ideas helps to form them. Indeed, helps is far too weak a

word. 90%

of what ends up in my essays was stuff

I only

thought of when I sat down to write them. That's why I

write them.So there's another difference between essays and

the things

you have to write in school. In school

you are, in theory,

explaining yourself to someone else. In the best case---if

you're really organized---you're just writing it down.

In a real essay you're writing for yourself. You're

thinking out loud.But not quite. Just as inviting people over forces you to

clean up your apartment, writing something that you know

other people will read forces you to think well. So it

does matter to have an audience. The things I've written

just for myself are no good. Indeed, they're bad in

a particular way:

they tend to peter out. When I run into

difficulties, I notice that I

tend to conclude with a few vague

questions and then drift off to get a cup of tea.This seems a common problem.

It's practically the standard

ending in blog entries--- with the addition of a "heh" or an

emoticon, prompted by the all too accurate sense that

something is missing.And indeed, a lot of

published essays peter out in this

same way.

Particularly the sort written by the staff writers of newsmagazines. Outside writers tend to supply

editorials of the defend-a-position variety, which

make a beeline toward a rousing (and

foreordained) conclusion. But the staff writers feel

obliged to write something more

balanced, which in

practice ends up meaning blurry.

Since they're

writing for a popular magazine, they start with the

most radioactively controversial questions, from which

(because they're writing for a popular magazine)

they then proceed to recoil from

in terror.

Gay marriage, for or

against? This group says one thing. That group says

another. One thing is certain: the question is a

complex one. (But don't get mad at us. We didn't

draw any conclusions.)Questions aren't enough. An essay has to come up with answers.

They don't always, of course. Sometimes you start with a

promising question and get nowhere. But those you don't

publish. Those are like experiments that get inconclusive

results. Something you publish ought to tell the reader

something he didn't already know.

But what you tell him doesn't matter, so long as

it's interesting. I'm sometimes accused of meandering.

In defend-a-position writing that would be a flaw.

There you're not concerned with truth. You already

know where you're going, and you want to go straight there,

blustering through obstacles, and hand-waving

your way across swampy ground. But that's not what

you're trying to do in an essay. An essay is supposed to

be a search for truth. It would be suspicious if it didn't

meander.The Meander is a river in Asia Minor (aka

Turkey).

As you might expect, it winds all over the place.

But does it

do this out of frivolity? Quite the opposite.

Like all rivers, it's rigorously following the laws of physics.

The path it has discovered,

winding as it is, represents

the most economical route to the sea.The river's algorithm is simple. At each step, flow down.

For the essayist this translates to: flow interesting.

Of all the places to go next, choose

whichever seems

most interesting.I'm pushing this metaphor a bit. An essayist

can't have

quite as little foresight as a river. In fact what you do

(or what I do) is somewhere between a river and a roman

road-builder. I have a general idea of the direction

I want to go in, and

I choose the next topic with that in mind. This essay is

about writing, so I do occasionally yank it back in that

direction, but it is not all the sort of essay I

thought I was going to write about writing.Note too that hill-climbing (which is what this algorithm is

called) can get you in trouble.

Sometimes, just

like a river,

you

run up against a blank wall. What

I do then is just

what the river does: backtrack.

At one point in this essay

I found that after following a certain thread I ran out

of ideas. I had to go back n

paragraphs and start over

in another direction. For illustrative purposes I've left

the abandoned branch as a footnote.

Err on the side of the river. An essay is not a reference

work. It's not something you read looking for a specific

answer, and feel cheated if you don't find it. I'd much

rather read an essay that went off in an unexpected but

interesting direction than one that plodded dutifully along

a prescribed course.So what's interesting? For me, interesting means surprise.

Design, as Matz

has said, should follow the principle of

least surprise.

A button that looks like it will make a

machine stop should make it stop, not speed up. Essays

should do the opposite. Essays should aim for maximum

surprise.I was afraid of flying for a long time and could only travel

vicariously. When friends came back from faraway places,

it wasn't just out of politeness that I asked them about

their trip.

I really wanted to know. And I found that

the best way to get information out of them was to ask

what surprised them. How was the place different from what

they expected? This is an extremely useful question.

You can ask it of even

the most unobservant people, and it will

extract information they didn't even know they were

recording. Indeed, you can ask it in real time. Now when I go somewhere

new, I make a note of what surprises me about it. Sometimes I

even make a conscious effort to visualize the place beforehand,

so I'll have a detailed image to diff with reality.

Surprises are facts

you didn't already

know.

But they're

more than that. They're facts

that contradict things you

thought you knew. And so they're the most valuable sort of

fact you can get. They're like a food that's not merely

healthy, but counteracts the unhealthy effects of things

you've already eaten.

How do you find surprises? Well, therein lies half

the work of essay writing. (The other half is expressing

yourself well.) You can at least

use yourself as a

proxy for the reader. You should only write about things

you've thought about a lot. And anything you come across

that surprises you, who've thought about the topic a lot,

will probably surprise most readers.For example, in a recent essay I pointed out that because

you can only judge computer programmers by working with

them, no one knows in programming who the heroes should

be.

I

certainly

didn't realize this when I started writing

the

essay, and even now I find it kind of weird. That's

what you're looking for.So if you want to write essays, you need two ingredients:

you need

a few topics that you think about a lot, and you

need some ability to ferret out the unexpected.What should you think about? My guess is that it

doesn't matter. Almost everything is

interesting if you get deeply

enough into it. The one possible exception

are

things

like working in fast food, which

have deliberately had all

the variation sucked out of them.

In retrospect, was there

anything interesting about working in Baskin-Robbins?

Well, it was interesting to notice

how important color was

to the customers. Kids a certain age would point into

the case and say that they wanted yellow. Did they want

French Vanilla or Lemon? They would just look at you

blankly. They wanted yellow. And then there was the

mystery of why the perennial favorite Pralines n' Cream

was so appealing. I'm inclined now to

think it was the salt.

And the mystery of why Passion Fruit tasted so disgusting.

People would order it because of the name, and were always

disappointed. It should have been called In-sink-erator

Fruit.

And there was

the difference in the way fathers and

mothers bought ice cream for their kids.

Fathers tended to

adopt the attitude of

benevolent kings bestowing largesse,

and mothers that of

harried bureaucrats,

giving in to

pressure against their better judgement.

So, yes, there does seem to be material, even in

fast food.What about the other half, ferreting out the unexpected?

That may require some natural ability. I've noticed for

a long time that I'm pathologically observant. ....[That was as far as I'd gotten at the time.]Notes[sh] In Shakespeare's own time, serious writing meant theological

discourses, not the bawdy plays acted over on the other

side of the river among the bear gardens and whorehouses.The other extreme, the work that seems formidable from the moment

it's created (indeed, is deliberately intended to be)

is represented by Milton. Like the Aeneid, Paradise Lost is a

rock imitating a butterfly that happened to get fossilized.

Even Samuel Johnson seems to have balked at this, on the one

hand paying Milton the compliment of an extensive biography,

and on the other writing of Paradise Lost that "none who read it

ever wished it longer."

What the Bubble Got Right

September 2004(This essay is derived from an invited talk at ICFP 2004.)I had a front row seat for the Internet Bubble,

because I worked at Yahoo during 1998 and 1999. One day,

when the stock was trading around $200, I sat down and calculated

what I thought the price should be. The

answer I got was $12. I went to

the next cubicle and told my friend Trevor. "Twelve!" he said.

He tried to sound indignant, but he didn't quite manage it. He

knew as well as I did that our valuation was crazy.Yahoo was a special case. It was not just our price to earnings

ratio that was bogus. Half our earnings were too. Not in

the Enron way, of course. The finance guys seemed

scrupulous about reporting earnings. What made our

earnings bogus was that Yahoo was, in effect, the center of

a Ponzi scheme. Investors looked at Yahoo's earnings

and said to themselves, here is proof that Internet companies can make

money. So they invested in new

startups that promised to be the next Yahoo. And as soon as these startups

got the money, what did they do with it?

Buy millions of dollars worth of advertising on Yahoo to promote

their brand. Result: a capital investment in a startup this

quarter shows up as Yahoo earnings next quarter—stimulating

another round of investments in startups.As in a Ponzi scheme, what seemed to be the returns of this system

were simply the latest round of investments in it.

What made it not a Ponzi scheme was that it was unintentional.

At least, I think it was. The venture capital business is pretty incestuous,

and there were presumably people in a position, if not to create

this situation, to realize what was happening and to milk it.A year later the game was up. Starting in January 2000, Yahoo's

stock price began to crash, ultimately losing 95% of its

value.Notice, though, that even with all the fat trimmed off its market

cap, Yahoo was still worth a lot. Even at the morning-after

valuations of March and April 2001, the people at Yahoo had managed

to create a company worth about $8 billion in just six years.The fact is, despite all the nonsense we heard

during the Bubble about the "new economy," there was a

core of truth. You need

that to get a really big bubble: you need to have something

solid at the center, so that even smart people are sucked in.

(Isaac Newton and Jonathan Swift both lost money

in the South Sea Bubble of 1720.)Now the pendulum has swung the other way. Now anything that

became fashionable during the Bubble is ipso facto unfashionable.

But that's a mistake—an even bigger mistake than believing

what everyone was saying in 1999. Over the long term,

what the Bubble got right will be more important than what

it got wrong.1. Retail VCAfter the excesses of the Bubble, it's now

considered dubious to take companies public before they have earnings.

But there is nothing intrinsically wrong with

that idea. Taking a company public at an early stage is simply

retail VC: instead of going to venture capital firms for the last round of

funding, you go to the public markets.By the end of the Bubble, companies going public with no

earnings were being derided as "concept stocks," as if it

were inherently stupid to invest in them.

But investing in concepts isn't stupid; it's what VCs do,

and the best of them are far from stupid.The stock of a company that doesn't yet have earnings is

worth something.

It may take a while for the market to learn

how to value such companies, just as it had to learn to

value common stocks in the early 20th century. But markets

are good at solving that kind of problem. I wouldn't be

surprised if the market ultimately did a better

job than VCs do now.Going public early will not be the right plan

for every company.

And it can of course be

disruptive—by distracting the management, or by making the early

employees suddenly rich. But just as the market will learn

how to value startups, startups will learn how to minimize

the damage of going public.2. The InternetThe Internet genuinely is a big deal. That was one reason

even smart people were fooled by the Bubble. Obviously

it was going to have a huge effect. Enough of an effect to

triple the value of Nasdaq companies in two years? No, as it

turned out. But it was hard to say for certain at the time. [1]The same thing happened during the Mississippi and South Sea Bubbles.

What drove them was the invention of organized public finance

(the South Sea Company, despite its name, was really a competitor

of the Bank of England). And that did turn out to be

a big deal, in the long run.Recognizing an important trend turns out to be easier than

figuring out how to profit from it. The mistake

investors always seem to make is to take the trend too literally.

Since the Internet was the big new thing, investors supposed

that the more Internettish the company, the better. Hence

such parodies as Pets.Com.In fact most of the money to be made from big trends is made

indirectly. It was not the railroads themselves that

made the most money during the railroad boom, but the companies

on either side, like Carnegie's steelworks, which made the rails,

and Standard Oil, which used railroads to get oil to the East Coast,

where it could be shipped to Europe.I think the Internet will have great effects,

and that what we've seen so far is nothing compared to what's

coming. But most of the winners will only indirectly be

Internet companies; for every Google there will be ten

JetBlues.3. ChoicesWhy will the Internet have great effects? The general

argument is that new forms of communication always do. They happen

rarely (till industrial times there were just speech, writing, and printing),

but when they do, they always cause a big splash.The specific argument, or one of them, is the Internet gives us

more choices. In the "old" economy,

the high cost of presenting information to people meant they

had only a narrow range of options to choose from. The tiny,

expensive pipeline to consumers was tellingly named "the channel."

Control the channel and you

could feed them what you wanted, on your terms. And it

was not just big corporations that depended

on this principle. So, in their way, did

labor unions, the traditional news media,

and the art and literary establishments.

Winning depended not on doing good work, but on gaining control

of some bottleneck.There are signs that this is changing.

Google has over 82 million unique users a month and

annual revenues of about three billion dollars. [2]

And yet have you ever seen

a Google ad?

Something is going on here.Admittedly, Google is an extreme case. It's very easy for

people to switch to a new search engine. It costs little

effort and no money to try a new one, and it's easy to

see if the results are better. And so Google doesn't have

to advertise. In a business like theirs, being the best is

enough.The exciting thing about the Internet is that it's

shifting everything in that direction.

The hard part, if you want to win by making the best stuff,

is the beginning. Eventually everyone

will learn by word of mouth that you're the best,

but how do you survive to that point? And it is in this crucial

stage that the Internet has the most effect. First, the

Internet lets anyone find you at almost zero cost.

Second, it dramatically speeds up the rate at which

reputation spreads by word of mouth. Together these mean that in many

fields the rule will be: Build it, and they will come.

Make something great and put it online.

That is a big change from the recipe for winning in the

past century.4. YouthThe aspect of the Internet Bubble that the press seemed most

taken with was the youth of some of the startup founders.

This too is a trend that will last.

There is a huge standard deviation among 26 year olds. Some

are fit only for entry level jobs, but others are

ready to rule the world if they can find someone to handle

the paperwork for them.A 26 year old may not be very good at managing people or

dealing with the SEC. Those require experience.

But those are also commodities, which can be handed off to

some lieutenant. The most important quality in a CEO is his

vision for the company's future. What will they build next?

And in that department, there are 26 year olds who can

compete with anyone.In 1970 a company president meant someone in his fifties, at

least. If he had technologists working for him, they were

treated like a racing stable: prized, but not powerful. But

as technology has grown more important, the power of nerds

has grown to reflect it. Now it's not enough for a CEO to

have someone smart he can ask about technical matters. Increasingly,

he has to be that person himself.As always, business has clung to old forms. VCs still seem

to want to install a legitimate-looking

talking head as the CEO. But increasingly the founders of

the company are the real powers, and the grey-headed man

installed by the VCs more like a

music group's manager than a general.5. InformalityIn New York, the Bubble had dramatic consequences:

suits went out of fashion. They made one seem old. So in

1998 powerful New York types were suddenly wearing

open-necked shirts and khakis and oval wire-rimmed glasses,

just like guys in Santa Clara.The pendulum has swung back a bit, driven in part by a panicked

reaction by the clothing industry. But I'm betting on the

open-necked shirts. And this is not as frivolous a question

as it might seem. Clothes are important, as all nerds can sense,

though they may not realize it consciously.If you're a nerd, you can understand how important clothes are

by asking yourself how you'd feel about a company

that made you wear a suit and tie to work. The idea sounds

horrible, doesn't it? In fact, horrible far out of proportion

to the mere discomfort of wearing such clothes. A company that

made programmers wear suits would have something deeply wrong

with it.And what would be wrong would be that how one presented oneself

counted more than the quality of one's ideas. That's

the problem with formality. Dressing up is not so much bad in

itself. The problem is the receptor it binds to: dressing

up is inevitably a substitute

for good ideas. It is no coincidence that technically

inept business types are known as "suits."Nerds don't just happen to dress informally. They do it too

consistently. Consciously or not, they dress informally as

a prophylactic measure against stupidity.6. NerdsClothing is only the most visible battleground in the war

against formality. Nerds tend to eschew formality of any sort.

They're not impressed by one's job title, for example,

or any of the other appurtenances of authority.Indeed, that's practically the definition of a nerd. I found

myself talking recently to someone from Hollywood who was planning

a show about nerds. I thought it would be useful if I

explained what a nerd was. What I came up with was: someone who

doesn't expend any effort on marketing himself.A nerd, in other words, is someone who concentrates on substance.

So what's the connection between nerds and technology? Roughly

that you can't fool mother nature. In technical matters, you

have to get the right answers. If your software miscalculates

the path of a space probe, you can't finesse your way out of

trouble by saying that your code is patriotic, or avant-garde,

or any of the other dodges people use in nontechnical

fields.And as technology becomes increasingly important in the

economy, nerd culture is

rising with it. Nerds are already

a lot cooler than they were when I was a kid. When I was in

college in the mid-1980s, "nerd" was still an insult. People

who majored in computer science generally tried to conceal it.

Now women ask me where they can meet nerds. (The answer that

springs to mind is "Usenix," but that would be like drinking

from a firehose.)I have no illusions about why nerd culture is becoming

more accepted. It's not because people are

realizing that substance is more important than marketing.

It's because the nerds are getting

rich. But that is not going

to change.7. OptionsWhat makes the nerds rich, usually, is stock options. Now there

are moves afoot to make it harder for companies to grant

options. To the extent there's some genuine accounting abuse

going on, by all means correct it. But don't kill the golden

goose. Equity is the fuel that drives technical innovation.Options are a good idea because (a) they're fair, and (b) they

work. Someone who goes to work for a company is (one hopes)

adding to its value, and it's only fair to give them a share

of it. And as a purely practical measure, people work a lot

harder when they have options. I've seen that first hand.The fact that a few crooks during the Bubble robbed their

companies by granting themselves options doesn't mean options

are a bad idea. During the railroad boom, some executives

enriched themselves by selling watered stock—by issuing more

shares than they said were outstanding. But that doesn't

make common stock a bad idea. Crooks just use whatever

means are available.If there is a problem with options, it's that they reward

slightly the wrong thing. Not surprisingly, people do what you

pay them to. If you pay them by the hour, they'll work a lot of

hours. If you pay them by the volume of work done, they'll

get a lot of work done (but only as you defined work).

And if you pay them to raise the

stock price, which is what options amount to, they'll raise

the stock price.But that's not quite what you want. What you want is to

increase the actual value of the company, not its market cap.

Over time the two inevitably meet, but not always as quickly

as options vest. Which means options tempt employees, if

only unconsciously, to "pump and dump"—to do things

that will make the company seem valuable.

I found that when I was at Yahoo, I couldn't help thinking,

"how will this sound to investors?" when I should have been

thinking "is this a good idea?"So maybe the standard option deal needs to be tweaked slightly.

Maybe options should be replaced with something tied more

directly to earnings. It's still early days.8. StartupsWhat made the options valuable, for the most part, is

that they were options on the stock of

startups. Startups

were not of course a creation of the Bubble, but they

were more visible during the Bubble than ever before.One thing most people did learn about for the first time

during the Bubble was the startup

created with the intention of selling it.

Originally a

startup meant a small company that hoped to grow into a

big one. But increasingly startups are evolving into a

vehicle for developing technology on spec.As I wrote in

Hackers & Painters, employees seem to be most

productive when they're paid in proportion to the wealth

they generate. And the advantage of a startup—indeed,

almost its raison d'etre—is that it offers something

otherwise impossible to obtain: a way of measuring that.In many businesses, it just makes more sense for companies

to get technology by buying startups rather than developing

it in house. You pay more, but there is less risk,

and risk is what big companies don't want. It makes the

guys developing the technology more accountable, because they

only get paid if they build the winner. And you end up

with better technology, created faster, because things are

made in the innovative atmosphere of startups instead of

the bureaucratic atmosphere of big companies.Our startup, Viaweb, was built to be sold. We were open

with investors about that from the start. And we were

careful to create something that could slot easily into a

larger company. That is the pattern for the future.9. CaliforniaThe Bubble was a California phenomenon. When I showed up

in Silicon Valley in 1998, I felt like an immigrant from

Eastern Europe arriving in America in 1900. Everyone

was so cheerful and healthy and rich. It seemed a new

and improved world.The press, ever eager to exaggerate small trends, now gives

one the impression that Silicon Valley is a ghost town.

Not at all. When I drive down 101 from the airport,

I still feel a buzz of energy, as if there were a giant

transformer nearby. Real estate is still more expensive

than just about anywhere else in the country. The people

still look healthy, and the weather is still fabulous.

The future is there.

(I say "there" because I moved back to the East Coast after

Yahoo. I still wonder if this was a smart idea.)What makes the Bay Area superior is the attitude of the

people. I notice that when I come home to Boston.

The first thing I see when I walk out of the airline terminal

is the fat, grumpy guy in

charge of the taxi line. I brace myself for rudeness:

remember, you're back on the East Coast now.The atmosphere varies from city to city, and fragile

organisms like startups are exceedingly sensitive to such variation.

If it hadn't already been hijacked as a new euphemism

for liberal, the word to describe the atmosphere in

the Bay Area would be "progressive." People there are trying

to build the future.

Boston has MIT and Harvard, but it also has a lot of

truculent, unionized employees like the police who

recently held the Democratic National Convention for

ransom, and a lot of people trying to be

Thurston Howell.

Two sides of an obsolete coin.Silicon Valley may not be the next Paris or London, but it

is at least the next Chicago. For the next fifty years,

that's where new wealth will come from.10. ProductivityDuring the Bubble, optimistic analysts used to justify high

price to earnings ratios by saying that technology was going

to increase productivity dramatically. They were wrong about

the specific companies, but not so wrong about the underlying

principle. I think one of the big trends we'll see in the

coming century is a huge increase in productivity.Or more precisely, a huge increase in variation in

productivity. Technology is a lever. It doesn't add;

it multiplies. If the present range of productivity is

0 to 100, introducing a multiple of 10 increases the range

from 0 to 1000.One upshot of which is that the companies of the future may

be surprisingly small. I sometimes daydream about how big

you could grow a company (in revenues) without ever having

more than ten people. What would happen if you outsourced

everything except product development? If you tried this experiment,

I think you'd be surprised at how far you could get.

As Fred Brooks pointed out, small groups are

intrinsically more productive, because the

internal friction in a group grows as the

square of its size.Till quite recently, running a major company

meant managing an army of workers. Our standards about how

many employees a company should have are still influenced by

old patterns. Startups are perforce small, because they can't

afford to hire a lot of people. But I think it's a big mistake for

companies to loosen their belts as revenues increase. The

question is not whether you can afford the extra salaries.

Can you afford the loss in productivity that comes from making

the company bigger?The prospect of technological leverage will of course raise the

specter of unemployment. I'm surprised people still worry about

this.

After centuries of supposedly job-killing innovations,

the number of jobs is within ten percent of the number of people

who want them. This can't be a coincidence. There must be some

kind of balancing mechanism.What's NewWhen one looks over these trends, is there any overall theme?

There does seem to be: that in the coming century, good ideas

will count for more. That 26

year olds with good ideas will increasingly have an edge over 50

year olds with powerful connections. That doing good work will

matter more than dressing up—or advertising, which is the

same thing for companies. That people

will be rewarded a bit more in proportion to the value of what

they create.If so, this is good news indeed.

Good ideas always tend to win eventually. The problem is,

it can take a very long time.

It took decades for relativity to be accepted, and the

greater part of a century to establish that central planning didn't work.

So even a small increase in the

rate at which good ideas win would be a momentous

change—big enough, probably, to justify a name like

the "new economy."Notes[1] Actually it's hard to say now. As Jeremy Siegel points

out, if the value of a stock is its future earnings, you

can't tell if it was overvalued till you see what the earnings

turn out to be. While certain famous Internet stocks were

almost certainly overvalued in 1999, it is still hard to say for sure

whether, e.g., the Nasdaq index was.Siegel, Jeremy J. "What Is an Asset Price Bubble? An

Operational Definition." European Financial Management,

9:1, 2003.[2] The number of users comes from a 6/03 Nielsen

study quoted on Google's site. (You'd think they'd have

something more recent.) The revenue estimate is based on

revenues of $1.35 billion for the first half of 2004, as

reported in their IPO filing.Thanks to Chris Anderson, Trevor Blackwell, Sarah Harlin, Jessica

Livingston, and Robert Morris for reading drafts of this.The Long TailRussian TranslationJapanese Translation

The Age of the Essay

September 2004Remember the essays you had to write in high school?

Topic sentence, introductory paragraph,

supporting paragraphs, conclusion. The conclusion being,

say, that Ahab in Moby Dick was a Christ-like figure.Oy. So I'm going to try to give the other side of the

story: what an essay really is, and how you write one.

Or at least, how I write one.ModsThe most obvious difference between real essays and

the things one has to write in school is that real

essays are not exclusively about English literature.

Certainly schools should teach students how to

write. But due to a series of historical accidents

the teaching of

writing has gotten mixed together with the study

of literature. And so all over the country students are

writing not about how a baseball team with a small budget

might compete with the Yankees, or the role of color in

fashion, or what constitutes a good dessert, but about

symbolism in Dickens.With the result that writing is made to seem boring and

pointless. Who cares about symbolism in Dickens?

Dickens himself would be more interested in an essay

about color or baseball.How did things get this way? To answer that we have to go back

almost a thousand years. Around 1100, Europe at last began to

catch its breath after centuries of chaos, and once they

had the luxury of curiosity they rediscovered

what we call "the classics." The effect was rather as if

we were visited by beings from another solar system.

These earlier civilizations were so much more sophisticated

that for the next several centuries the main work of

European scholars, in almost every field, was to assimilate

what they knew.During this period the study of ancient texts acquired great

prestige. It seemed the essence of what scholars did. As

European scholarship gained momentum it became less and less important;

by 1350

someone who wanted to learn about science could find better

teachers than Aristotle in his own era. [1]

But schools change slower than scholarship. In the

19th century the study of ancient texts was still the backbone

of the curriculum.The time was then ripe for the question: if the study of

ancient texts is a valid field for scholarship, why not modern

texts? The answer, of course, is that the original raison d'etre

of classical scholarship was a kind of intellectual archaeology that

does not need to be done in the case of contemporary authors.

But for obvious reasons no one wanted to give that answer.

The archaeological work being mostly done, it implied that

those studying the classics were, if not wasting their

time, at least working on problems of minor importance.And so began the study of modern literature. There was a good

deal of resistance at first.

The first courses in English literature

seem to have been offered by the newer colleges, particularly

American ones. Dartmouth, the University of Vermont, Amherst,

and University College, London

taught English literature in the 1820s.

But Harvard didn't have a professor of English literature until

1876, and Oxford not till 1885. (Oxford had a chair of Chinese before

it had one of English.) [2]What tipped the scales, at least in the US, seems to have

been the idea that professors should do research as well

as teach. This idea (along with the PhD, the department, and

indeed the whole concept of the modern university) was imported

from Germany in the late 19th century. Beginning at

Johns Hopkins in 1876, the new model spread rapidly.Writing was one of the casualties. Colleges had long taught

English composition. But how do you do research on composition?

The professors who taught math could be required to do original

math, the professors who taught history could be required to

write scholarly articles about history, but what about the

professors who taught rhetoric or composition? What should they

do research on? The closest thing seemed to be English literature. [3]And so in the late 19th century the teaching of writing was inherited

by English professors. This had two drawbacks:

(a) an expert on literature need not himself be a good writer,

any more than an art historian has to be a good painter, and (b)

the subject of writing now tends to be literature, since that's

what the professor is interested in.High schools imitate universities. The seeds of our miserable

high school experiences were sown in 1892, when

the National Education Association

"formally recommended that literature

and composition be unified in the high school course." [4]

The 'riting component of the 3 Rs then morphed into English,

with the bizarre consequence that high school students now

had to write about English literature-- to write, without

even realizing it, imitations of whatever

English professors had been publishing in their journals a

few decades before.It's no wonder if this seems to the

student a pointless exercise, because we're now three steps

removed from real work: the students are imitating English

professors, who are imitating classical scholars, who are

merely the inheritors of a tradition growing out of what

was, 700 years ago, fascinating and urgently needed work.No DefenseThe other big difference between a real essay and the things

they make you write in school is that a real essay doesn't

take a position and then defend it. That principle,

like the idea that we ought to be writing about literature,

turns out to be another intellectual hangover of long

forgotten origins.It's often mistakenly believed that

medieval universities were mostly seminaries. In fact they

were more law schools. And at least in our tradition

lawyers are advocates, trained to take

either side of an argument and make as good a case for it

as they can.

Whether cause or effect, this spirit pervaded

early universities. The study of rhetoric, the art of arguing

persuasively, was a third of the undergraduate curriculum. [5]

And after the lecture the most common form

of discussion was the disputation. This is at least

nominally preserved in our present-day thesis defense:

most people treat the words thesis

and dissertation as interchangeable, but originally, at least,

a thesis was a position one took and the dissertation was

the argument by which one defended it.Defending a position may be a necessary evil in a

legal dispute, but it's not the best way to get at the truth,

as I think lawyers would be the first to admit. It's not

just that you miss subtleties this way.

The real problem is that you can't change the question.And yet this principle is built into the very structure of

the things they teach you to write in high school. The topic

sentence is your thesis, chosen in advance, the supporting

paragraphs the blows you strike in the conflict, and the

conclusion-- uh, what is the conclusion? I was never sure

about that in high school. It seemed as if we were just

supposed to restate what we said in the first paragraph,

but in different enough words that no one could tell.

Why bother?

But when you understand the origins

of this sort of "essay," you can see where the

conclusion comes from. It's the concluding remarks to the

jury.Good writing should be convincing, certainly, but it

should be convincing because you got the right answers,

not because you did a good job of arguing. When I give a

draft of an essay to friends, there are two things

I want to know: which parts bore them, and which seem

unconvincing. The boring bits can usually be fixed by

cutting. But I don't try to fix the unconvincing bits by

arguing more cleverly. I need to talk the matter over.At the very least I must have explained something badly. In

that case, in the course of the conversation I'll be forced

to come up a with a clearer explanation, which I can just

incorporate in the essay. More often than not I have

to change what I was saying as well.

But the aim is never to be convincing per se.

As the reader gets smarter, convincing and true become identical,

so if I can convince smart readers I must be near the truth.The sort of writing that attempts to persuade may be

a valid (or at least inevitable) form, but it's historically

inaccurate to call it an essay. An essay is

something else.TryingTo understand what a real essay is, we have to

reach back into history again, though this time not so far.

To Michel de Montaigne, who in 1580 published a book of

what he called "essais." He was

doing something quite different from what lawyers do, and

the difference is embodied in the name. Essayer is the French

verb meaning "to try"

and an essai is an attempt. An essay is something you

write to try to figure something out.Figure out what? You don't know yet. And so you can't begin with a

thesis, because you don't have one, and may never have

one. An essay doesn't begin with a statement, but with a

question. In a real essay, you don't take a position and

defend it. You notice a door that's ajar, and you open it and

walk in to see what's inside.If all you want to do is figure things out, why do you need

to write anything, though? Why not just sit and think? Well,

there precisely is Montaigne's great discovery. Expressing

ideas helps to form them. Indeed, helps is far too weak a

word. Most of what ends up in my essays I only

thought of when I sat down to write them. That's why I

write them.In the things you write in school you are, in theory,

merely explaining yourself to the reader.

In a real essay you're writing for yourself.

You're thinking out loud.But not quite.

Just as inviting people over forces you to

clean up your apartment, writing something that

other people will read forces you to think well. So it

does matter to have an audience. The things I've written

just for myself are no good.

They tend to peter out. When I run into

difficulties, I find I conclude with a few vague

questions and then drift off to get a cup of tea.Many published essays peter out in the same way.

Particularly the sort written by the staff writers

of newsmagazines. Outside writers tend to supply

editorials of the defend-a-position variety, which

make a beeline toward a rousing (and

foreordained) conclusion. But the staff writers feel

obliged to write something "balanced."

Since they're writing for a popular magazine, they start with the

most radioactively controversial questions, from which-- because

they're writing for a popular magazine-- they

then proceed to recoil in terror.

Abortion, for or against?

This group says one thing. That group says

another. One thing is certain: the question is a

complex one. (But don't get mad at us. We didn't

draw any conclusions.)The RiverQuestions aren't enough. An essay has to come up with answers.

They don't always, of course. Sometimes you start with a

promising question and get nowhere. But those you don't

publish. Those are like experiments that get inconclusive

results. An essay you publish ought to tell the reader

something he didn't already know.But what you tell him doesn't matter, so long as

it's interesting. I'm sometimes accused of meandering.

In defend-a-position writing that would be a flaw.

There you're not concerned with truth. You already

know where you're going, and you want to go straight there,

blustering through obstacles, and hand-waving

your way across swampy ground. But that's not what

you're trying to do in an essay. An essay is supposed to

be a search for truth. It would be suspicious if it didn't

meander.The Meander (aka Menderes) is a river in Turkey.

As you might expect, it winds all over the place.

But it doesn't do this out of frivolity.

The path it has discovered is the most

economical route to the sea. [6]The river's algorithm is simple. At each step, flow down.

For the essayist this translates to: flow interesting.

Of all the places to go next, choose the most interesting.

One can't have quite as little foresight as a river. I always

know generally what I want to write about.

But not the

specific conclusions I want to reach; from paragraph to

paragraph I let the ideas take their course.This doesn't always work. Sometimes, like a river,

one runs up against a wall. Then I do the same thing the river does:

backtrack. At one point in this essay

I found that after following a certain thread I ran out

of ideas. I had to go back seven paragraphs and start over

in another direction.Fundamentally an essay is a train of thought-- but a cleaned-up

train of thought, as dialogue is cleaned-up conversation.

Real thought, like real conversation, is full of false starts.

It would be exhausting to read. You need to

cut and fill to

emphasize the central thread, like an

illustrator inking over a pencil drawing. But don't

change so much that you lose the spontaneity of the original.Err on the side of the river. An essay is not a reference

work. It's not something you read looking for a specific

answer, and feel cheated if you don't find it. I'd much

rather read an essay that went off in an unexpected but

interesting direction than one that plodded dutifully along

a prescribed course.SurpriseSo what's interesting? For me, interesting means surprise.

Interfaces, as Geoffrey James has said, should follow the principle of

least astonishment. A button that looks like it will make a

machine stop should make it stop, not speed up. Essays

should do the opposite. Essays should aim for maximum

surprise.I was afraid of flying for a long time and could only travel

vicariously. When friends came back from faraway places,

it wasn't just out of politeness that I asked

what they saw. I really wanted to know. And I found

the best way to get information out of them was to ask

what surprised them. How was the place different from what

they expected? This is an extremely useful question.

You can ask it of the most unobservant people, and it will

extract information they didn't even know they were

recording.Surprises are things that you not only didn't know, but that

contradict things you

thought you knew. And so they're the most valuable sort of

fact you can get. They're like a food that's not merely

healthy, but counteracts the unhealthy effects of things

you've already eaten.How do you find surprises? Well, therein lies half

the work of essay writing. (The other half is expressing

yourself well.) The trick is to use yourself as a

proxy for the reader. You should only write about things

you've thought about a lot. And anything you come across

that surprises you, who've thought about the topic a lot,

will probably surprise most readers.For example, in a recent

essay I pointed out that because

you can only judge computer programmers by working with

them, no one knows who the best programmers are overall.

I didn't realize this when I began

that essay, and even now I find it kind of weird. That's

what you're looking for.So if you want to write essays, you need two ingredients:

a few topics you've thought about a lot, and

some ability to ferret out the unexpected.What should you think about? My guess is that it

doesn't matter-- that anything can be interesting if you get deeply

enough into it. One possible exception might be things

that have deliberately had all the variation sucked out of them,

like working in fast food. In retrospect, was there

anything interesting about working at Baskin-Robbins?

Well, it was interesting how important color was

to the customers. Kids a certain age would point into

the case and say that they wanted yellow. Did they want

French Vanilla or Lemon? They would just look at you

blankly. They wanted yellow. And then there was the

mystery of why the perennial favorite Pralines 'n' Cream

was so appealing. (I think now it was the salt.)

And the difference in the way fathers and

mothers bought ice cream for their kids: the fathers

like benevolent kings bestowing largesse, the mothers

harried, giving in to pressure.

So, yes, there does seem to be some material even in

fast food.I didn't notice those things at the time, though. At sixteen

I was about as observant as a lump of rock. I can see more now in

the fragments of memory I preserve of that age than I could see

at the time from having it all happening live, right in front of me.ObservationSo the ability to ferret out the unexpected must not merely be an

inborn one. It must be something you can learn.

How do you learn it?To some extent it's like learning history.

When you first read

history, it's just a whirl of names

and dates.

Nothing seems to stick. But the more you learn, the more hooks you have

for new facts to stick onto-- which means

you accumulate knowledge at an exponential rate. Once you

remember that Normans conquered

England in 1066, it will catch your attention when you hear

that other Normans conquered southern Italy at about the same time.

Which will make you wonder about Normandy, and take note

when a third book mentions that Normans

were not, like most of what is now

called France, tribes that flowed in as the Roman empire collapsed,

but Vikings (norman = north man) who arrived

four centuries later in 911. Which makes

it easier to remember that Dublin was also established by

Vikings in the 840s. Etc, etc squared.Collecting surprises is a similar process.

The more anomalies you've seen, the more easily you'll notice

new ones. Which means, oddly enough, that as you grow older,

life should become more and more surprising. When I was a

kid, I used to think adults had it all figured out.

I had it backwards. Kids are the ones who have it all figured

out. They're just mistaken.When it comes to surprises, the rich get richer. But

(as with wealth) there

may be habits of mind that will help the process along. It's

good to have a habit of asking questions, especially questions

beginning with Why.

But not in the random way that three year

olds ask why. There are an infinite number of questions.

How do you find the fruitful ones?I find it especially

useful to ask why about things that seem wrong.

For example, why should there be a connection between

humor and misfortune? Why do we find it funny when a

character, even one we like, slips on a banana peel?

There's a whole essay's worth of surprises there for sure.If you want to notice things that seem wrong, you'll find a

degree of skepticism helpful. I take it as an axiom

that we're only achieving 1% of what we could.

This helps counteract the rule that gets beaten into our

heads as children: that things are the way they are because

that is how things have to be.

For example, everyone I've talked to while writing this essay

felt the same about

English classes-- that the whole process seemed pointless.

But none of us had the balls at the time to hypothesize that

it was, in fact, all a mistake.

We all thought there was just something we weren't getting.I have a hunch you want to pay attention not just to things

that seem wrong, but things that seem wrong in a humorous way.

I'm always pleased when I see someone laugh as they

read a draft of an essay. But why should I be? I'm aiming

for good ideas. Why should good ideas be funny?

The connection may be surprise.

Surprises make us laugh, and surprises are what

one wants to deliver.I write down things that surprise me in notebooks. I never

actually get around to reading them and using

what I've written, but I do tend to

reproduce the same thoughts later. So the main value

of notebooks may be what writing things down leaves in your

head.People trying to be cool will find themselves at a disadvantage

when collecting surprises. To be surprised is to be mistaken.

And the essence of cool, as any fourteen year old could tell

you, is nil admirari. When you're mistaken, don't

dwell on it; just act like nothing's wrong and maybe no one

will notice.One of the keys to coolness is to avoid situations where

inexperience may make you look foolish.

If you want to find surprises you should do the opposite.

Study lots of different things,

because some of the most interesting surprises are unexpected

connections between different fields. For example,

jam, bacon, pickles, and cheese, which are among the most pleasing

of foods, were all originally intended as methods of preservation.

And so were books and paintings.Whatever you study, include history-- but social and economic

history, not political history. History seems to me so important

that it's misleading to treat it as a mere field of study.

Another way to describe it is all the data we have so far.Among other things, studying history gives one confidence that

there are good ideas waiting to be discovered right under our noses.

Swords evolved during the Bronze Age out of daggers, which

(like their flint predecessors) had a hilt separate from the

blade. Because swords are longer

the hilts kept breaking off. But it took five hundred years

before someone thought of casting hilt and blade as one

piece.DisobedienceAbove all, make a habit of paying

attention to things you're not supposed to, either because

they're "inappropriate,"

or not important, or not what you're

supposed to be working on. If you're curious about something,

trust your instincts.

Follow the threads that attract your

attention. If there's something you're really interested

in, you'll find they have an uncanny way of leading back to

it anyway, just as the conversation of people who are especially

proud of something always tends to lead back to it.For example, I've always been fascinated by comb-overs, especially

the extreme sort that

make a man look as if he's wearing a beret made of his own hair.

Surely this is a lowly sort of thing to be interested in-- the

sort of superficial quizzing

best left to teenage girls. And yet there is something underneath.

The key question, I realized, is how does the comber-over not

see how odd he looks?

And the answer is that he got to look that way incrementally.

What began as combing his hair a little carefully over a

thin patch has gradually, over 20 years, grown into a monstrosity.

Gradualness is very powerful. And that power can be

used for constructive purposes too: just as you can trick

yourself into looking like a freak, you can trick yourself into

creating something so grand that you would never have dared to

plan such a thing. Indeed, this is just how most good

software gets created. You start by writing a stripped-down

kernel (how hard can it be?) and gradually it grows

into a complete operating system. Hence the next leap: could

you do the same thing in painting, or in a novel?See what you can extract from a frivolous question?

If there's one piece of advice I would give about writing essays,

it would be: don't do as you're told.

Don't believe what you're supposed to.

Don't write the

essay readers expect; one learns nothing from

what one expects.

And

don't write the way they taught you to in school.The most important sort of disobedience is to write

essays at all. Fortunately, this sort of disobedience shows

signs of becoming

rampant.

It used to be that only a tiny

number of officially approved writers were allowed to

write essays. Magazines published few of them, and judged

them less by what they said than who wrote them;

a magazine might publish a story by an

unknown writer if it was good enough, but if they published

an essay on x it had to be by someone who was at least

forty and whose job title had x in it. Which is a problem,

because there are a lot of things insiders can't say precisely

because they're insiders.The Internet is changing that.

Anyone can publish an essay on the Web, and it gets judged, as any

writing should, by what it says, not who wrote it.

Who are you to write about x? You are whatever you wrote.Popular magazines made the period between the spread

of literacy and the arrival of TV the golden age of the

short story.

The Web may well make this the golden age of the essay.

And that's certainly not something I realized when

I started writing this.Notes[1] I'm thinking of Oresme (c. 1323-82). But it's hard to pick

a date, because there was a sudden drop-off in scholarship

just as Europeans finished assimilating classical science.

The cause may have been the plague of 1347; the trend in

scientific progress matches the population curve.[2] Parker, William R. "Where Do College English Departments

Come From?" College English 28 (1966-67), pp. 339-351.

Reprinted in Gray, Donald J. (ed). The Department of

English at Indiana University Bloomington 1868-1970. Indiana

University Publications.Daniels, Robert V. The University of Vermont: The First

Two Hundred Years. University of Vermont, 1991.Mueller, Friedrich M. Letter to the Pall Mall

Gazette. 1886/87. Reprinted in Bacon, Alan (ed).

The Nineteenth-Century

History of English Studies. Ashgate, 1998.[3] I'm compressing the story a bit.

At first

literature took a back seat to philology, which (a) seemed more

serious and (b) was popular in Germany, where many of the

leading scholars of that generation had been trained.In some cases the writing teachers were transformed

in situ into English professors.

Francis James Child, who had been Boylston Professor

of Rhetoric at Harvard since 1851,

became in 1876 the university's first professor of English.[4] Parker, op. cit., p. 25.[5] The undergraduate curriculum or trivium (whence

"trivial") consisted of Latin grammar, rhetoric, and logic.

Candidates for masters' degrees went on to study the

quadrivium of arithmetic, geometry, music, and astronomy.

Together these were the seven liberal arts.The study of rhetoric was inherited directly from Rome, where

it was considered the most important

subject. It would not be far from the truth to say that

education in the classical world

meant training landowners' sons

to speak well enough to defend their interests

in political and legal disputes.[6] Trevor Blackwell points out that this

isn't strictly true, because the outside

edges of curves erode faster.

Thanks to Ken Anderson, Trevor Blackwell, Sarah Harlin, Jessica

Livingston, Jackie McDonough, and Robert Morris for reading drafts of

this.Russian TranslationSpanish TranslationJapanese TranslationHungarian TranslationTraditional Chinese Translation

If you liked this, you may also like

Hackers & Painters.

The Python Paradox

August 2004In a recent talk I said something that upset a lot of

people: that you could get smarter programmers to work on

a Python project than you could to work on a Java project.I didn't mean by this that Java programmers are dumb. I

meant that Python programmers are smart. It's a lot of

work to learn a new programming language. And people don't

learn Python because it will get them a job; they learn it

because they genuinely like to program and aren't satisfied with the languages they

already know.Which makes them exactly the kind of programmers

companies should want to hire. Hence what, for lack of a better

name, I'll call the Python paradox: if a company chooses to write

its software in a comparatively esoteric language, they'll be able

to hire better programmers, because they'll attract only those

who cared enough to learn it. And for

programmers the paradox is even more pronounced: the language

to learn, if you want to get a good job, is a language that

people don't learn merely to get a job.Only a few companies have been smart enough to realize this

so far. But there is a kind of selection going on here too: they're

exactly the companies programmers would

most like to work for. Google, for example. When they

advertise Java programming jobs, they also want Python experience.A friend of mine who knows nearly all the widely used languages

uses Python for most of his projects. He says the main reason

is that he likes the way source code looks. That may seem

a frivolous reason to choose one language over another.

But it is not so frivolous as it sounds: when you program,

you spend more time reading code than writing it.

You push blobs of source code around the way a sculptor does

blobs of clay. So a language that makes source code ugly is

maddening to an exacting programmer, as clay full of lumps

would be to a sculptor.At the mention of ugly source code, people will of course think

of Perl. But the superficial ugliness of Perl is not the sort

I mean. Real ugliness is not harsh-looking

syntax, but having to build programs out of the wrong

concepts. Perl may look like a cartoon character swearing,

but there are

cases where it surpasses Python conceptually.So far, anyway. Both languages are of course

moving targets. But they

share, along with Ruby (and Icon, and Joy, and J, and Lisp,

and Smalltalk) the fact that

they're created by, and used by, people who really care about

programming. And those tend to be the ones who do it well.Turkish TranslationJapanese TranslationPortuguese TranslationItalian TranslationPolish TranslationRomanian TranslationRussian TranslationSpanish TranslationFrench TranslationTelugu Translation

If you liked this, you may also like

Hackers & Painters.

Great Hackers

Want to start a startup? Get funded by

Y Combinator.

July 2004(This essay is derived from a talk at Oscon 2004.)

A few months ago I finished a new

book,

and in reviews I keep

noticing words like "provocative'' and "controversial.'' To say

nothing of "idiotic.''I didn't mean to make the book controversial. I was trying to make

it efficient. I didn't want to waste people's time telling them

things they already knew. It's more efficient just to give them

the diffs. But I suppose that's bound to yield an alarming book.EdisonsThere's no controversy about which idea is most controversial:

the suggestion that variation in wealth might not be as big a

problem as we think.I didn't say in the book that variation in wealth was in itself a

good thing. I said in some situations it might be a sign of good

things. A throbbing headache is not a good thing, but it can be

a sign of a good thing-- for example, that you're recovering

consciousness after being hit on the head.Variation in wealth can be a sign of variation in productivity.

(In a society of one, they're identical.) And that

is almost certainly a good thing: if your society has no variation

in productivity, it's probably not because everyone is Thomas

Edison. It's probably because you have no Thomas Edisons.In a low-tech society you don't see much variation in productivity.

If you have a tribe of nomads collecting sticks for a fire, how

much more productive is the best stick gatherer going to be than

the worst? A factor of two? Whereas when you hand people a complex tool

like a computer, the variation in what they can do with

it is enormous.That's not a new idea. Fred Brooks wrote about it in 1974, and

the study he quoted was published in 1968. But I think he

underestimated the variation between programmers. He wrote about productivity in lines

of code: the best programmers can solve a given problem in a tenth

the time. But what if the problem isn't given? In programming, as

in many fields, the hard part isn't solving problems, but deciding

what problems to solve. Imagination is hard to measure, but

in practice it dominates the kind of productivity that's measured

in lines of code.Productivity varies in any field, but there are few in which it

varies so much. The variation between programmers

is so great that it becomes a difference in kind. I don't

think this is something intrinsic to programming, though. In every field,

technology magnifies differences in productivity. I think what's

happening in programming is just that we have a lot of technological

leverage. But in every field the lever is getting longer, so the

variation we see is something that more and more fields will see

as time goes on. And the success of companies, and countries, will

depend increasingly on how they deal with it.If variation in productivity increases with technology, then the

contribution of the most productive individuals will not only be

disproportionately large, but will actually grow with time. When

you reach the point where 90% of a group's output is created by 1%

of its members, you lose big if something (whether Viking raids,

or central planning) drags their productivity down to the average.If we want to get the most out of them, we need to understand these

especially productive people. What motivates them? What do they

need to do their jobs? How do you recognize them? How do you

get them to come and work for you? And then of course there's the

question, how do you become one?More than MoneyI know a handful of super-hackers, so I sat down and thought about

what they have in common. Their defining quality is probably that

they really love to program. Ordinary programmers write code to pay

the bills. Great hackers think of it as something they do for fun,

and which they're delighted to find people will pay them for.Great programmers are sometimes said to be indifferent to money.

This isn't quite true. It is true that all they really care about

is doing interesting work. But if you make enough money, you get

to work on whatever you want, and for that reason hackers are

attracted by the idea of making really large amounts of money.

But as long as they still have to show up for work every day, they

care more about what they do there than how much they get paid for

it.Economically, this is a fact of the greatest importance, because

it means you don't have to pay great hackers anything like what

they're worth. A great programmer might be ten or a hundred times

as productive as an ordinary one, but he'll consider himself lucky

to get paid three times as much. As I'll explain later, this is

partly because great hackers don't know how good they are. But

it's also because money is not the main thing they want.What do hackers want? Like all craftsmen, hackers like good tools.

In fact, that's an understatement. Good hackers find it unbearable

to use bad tools. They'll simply refuse to work on projects with

the wrong infrastructure.At a startup I once worked for, one of the things pinned up on our

bulletin board was an ad from IBM. It was a picture of an AS400,

and the headline read, I think, "hackers despise

it.'' [1]When you decide what infrastructure to use for a project, you're

not just making a technical decision. You're also making a social

decision, and this may be the more important of the two. For

example, if your company wants to write some software, it might

seem a prudent choice to write it in Java. But when you choose a

language, you're also choosing a community. The programmers you'll

be able to hire to work on a Java project won't be as

smart as the

ones you could get to work on a project written in Python.

And the quality of your hackers probably matters more than the

language you choose. Though, frankly, the fact that good hackers

prefer Python to Java should tell you something about the relative

merits of those languages.Business types prefer the most popular languages because they view

languages as standards. They don't want to bet the company on

Betamax. The thing about languages, though, is that they're not

just standards. If you have to move bits over a network, by all

means use TCP/IP. But a programming language isn't just a format.

A programming language is a medium of expression.I've read that Java has just overtaken Cobol as the most popular

language. As a standard, you couldn't wish for more. But as a

medium of expression, you could do a lot better. Of all the great

programmers I can think of, I know of only one who would voluntarily

program in Java. And of all the great programmers I can think of

who don't work for Sun, on Java, I know of zero.Great hackers also generally insist on using open source software.

Not just because it's better, but because it gives them more control.

Good hackers insist on control. This is part of what makes them

good hackers: when something's broken, they need to fix it. You

want them to feel this way about the software they're writing for

you. You shouldn't be surprised when they feel the same way about

the operating system.A couple years ago a venture capitalist friend told me about a new

startup he was involved with. It sounded promising. But the next

time I talked to him, he said they'd decided to build their software

on Windows NT, and had just hired a very experienced NT developer

to be their chief technical officer. When I heard this, I thought,

these guys are doomed. One, the CTO couldn't be a first rate

hacker, because to become an eminent NT developer he would have

had to use NT voluntarily, multiple times, and I couldn't imagine

a great hacker doing that; and two, even if he was good, he'd have

a hard time hiring anyone good to work for him if the project had

to be built on NT. [2]The Final FrontierAfter software, the most important tool to a hacker is probably

his office. Big companies think the function of office space is to express

rank. But hackers use their offices for more than that: they

use their office as a place to think in. And if you're a technology

company, their thoughts are your product. So making hackers work

in a noisy, distracting environment is like having a paint factory

where the air is full of soot.The cartoon strip Dilbert has a lot to say about cubicles, and with

good reason. All the hackers I know despise them. The mere prospect

of being interrupted is enough to prevent hackers from working on

hard problems. If you want to get real work done in an office with

cubicles, you have two options: work at home, or come in early or

late or on a weekend, when no one else is there. Don't companies

realize this is a sign that something is broken? An office

environment is supposed to be something that helps

you work, not something you work despite.Companies like Cisco are proud that everyone there has a cubicle,

even the CEO. But they're not so advanced as they think; obviously

they still view office space as a badge of rank. Note too that

Cisco is famous for doing very little product development in house.

They get new technology by buying the startups that created it-- where

presumably the hackers did have somewhere quiet to work.One big company that understands what hackers need is Microsoft.

I once saw a recruiting ad for Microsoft with a big picture of a

door. Work for us, the premise was, and we'll give you a place to

work where you can actually get work done. And you know, Microsoft

is remarkable among big companies in that they are able to develop

software in house. Not well, perhaps, but well enough.If companies want hackers to be productive, they should look at

what they do at home. At home, hackers can arrange things themselves

so they can get the most done. And when they work at home, hackers

don't work in noisy, open spaces; they work in rooms with doors. They

work in cosy, neighborhoody places with people around and somewhere

to walk when they need to mull something over, instead of in glass

boxes set in acres of parking lots. They have a sofa they can take

a nap on when they feel tired, instead of sitting in a coma at

their desk, pretending to work. There's no crew of people with

vacuum cleaners that roars through every evening during the prime

hacking hours. There are no meetings or, God forbid, corporate

retreats or team-building exercises. And when you look at what

they're doing on that computer, you'll find it reinforces what I

said earlier about tools. They may have to use Java and Windows

at work, but at home, where they can choose for themselves, you're

more likely to find them using Perl and Linux.Indeed, these statistics about Cobol or Java being the most popular

language can be misleading. What we ought to look at, if we want

to know what tools are best, is what hackers choose when they can

choose freely-- that is, in projects of their own. When you ask

that question, you find that open source operating systems already

have a dominant market share, and the number one language is probably

Perl.InterestingAlong with good tools, hackers want interesting projects. What

makes a project interesting? Well, obviously overtly sexy

applications like stealth planes or special effects software would

be interesting to work on. But any application can be interesting

if it poses novel technical challenges. So it's hard to predict

which problems hackers will like, because some become

interesting only when the people working on them discover a new

kind of solution. Before ITA

(who wrote the software inside Orbitz),

the people working on airline fare searches probably thought it

was one of the most boring applications imaginable. But ITA made

it interesting by

redefining the problem in a more ambitious way.I think the same thing happened at Google. When Google was founded,

the conventional wisdom among the so-called portals was that search

was boring and unimportant. But the guys at Google didn't think

search was boring, and that's why they do it so well.This is an area where managers can make a difference. Like a parent

saying to a child, I bet you can't clean up your whole room in

ten minutes, a good manager can sometimes redefine a problem as a

more interesting one. Steve Jobs seems to be particularly good at

this, in part simply by having high standards. There were a lot

of small, inexpensive computers before the Mac. He redefined the

problem as: make one that's beautiful. And that probably drove

the developers harder than any carrot or stick could.They certainly delivered. When the Mac first appeared, you didn't

even have to turn it on to know it would be good; you could tell

from the case. A few weeks ago I was walking along the street in

Cambridge, and in someone's trash I saw what appeared to be a Mac

carrying case. I looked inside, and there was a Mac SE. I carried

it home and plugged it in, and it booted. The happy Macintosh

face, and then the finder. My God, it was so simple. It was just

like ... Google.Hackers like to work for people with high standards. But it's not

enough just to be exacting. You have to insist on the right things.

Which usually means that you have to be a hacker yourself. I've

seen occasional articles about how to manage programmers. Really

there should be two articles: one about what to do if

you are yourself a programmer, and one about what to do if you're not. And the

second could probably be condensed into two words: give up.The problem is not so much the day to day management. Really good

hackers are practically self-managing. The problem is, if you're

not a hacker, you can't tell who the good hackers are. A similar

problem explains why American cars are so ugly. I call it the

design paradox. You might think that you could make your products

beautiful just by hiring a great designer to design them. But if

you yourself don't have good taste,

how are you going to recognize

a good designer? By definition you can't tell from his portfolio.

And you can't go by the awards he's won or the jobs he's had,

because in design, as in most fields, those tend to be driven by

fashion and schmoozing, with actual ability a distant third.

There's no way around it: you can't manage a process intended to

produce beautiful things without knowing what beautiful is. American

cars are ugly because American car companies are run by people with

bad taste.Many people in this country think of taste as something elusive,

or even frivolous. It is neither. To drive design, a manager must

be the most demanding user of a company's products. And if you

have really good taste, you can, as Steve Jobs does, make satisfying

you the kind of problem that good people like to work on.Nasty Little ProblemsIt's pretty easy to say what kinds of problems are not interesting:

those where instead of solving a few big, clear, problems, you have

to solve a lot of nasty little ones. One of the worst kinds of

projects is writing an interface to a piece of software that's

full of bugs. Another is when you have to customize

something for an individual client's complex and ill-defined needs.

To hackers these kinds of projects are the death of a thousand

cuts.The distinguishing feature of nasty little problems is that you

don't learn anything from them. Writing a compiler is interesting

because it teaches you what a compiler is. But writing an interface

to a buggy piece of software doesn't teach you anything, because the

bugs are random. [3] So it's not just fastidiousness that makes good

hackers avoid nasty little problems. It's more a question of

self-preservation. Working on nasty little problems makes you

stupid. Good hackers avoid it for the same reason models avoid

cheeseburgers.Of course some problems inherently have this character. And because

of supply and demand, they pay especially well. So a company that

found a way to get great hackers to work on tedious problems would

be very successful. How would you do it?One place this happens is in startups. At our startup we had

Robert Morris working as a system administrator. That's like having the

Rolling Stones play at a bar mitzvah. You can't hire that kind of

talent. But people will do any amount of drudgery for companies

of which they're the founders. [4]Bigger companies solve the problem by partitioning the company.

They get smart people to work for them by establishing a separate

R&D department where employees don't have to work directly on

customers' nasty little problems. [5] In this model, the research

department functions like a mine. They produce new ideas; maybe

the rest of the company will be able to use them.You may not have to go to this extreme.

Bottom-up programming

suggests another way to partition the company: have the smart people

work as toolmakers. If your company makes software to do x, have

one group that builds tools for writing software of that type, and

another that uses these tools to write the applications. This way

you might be able to get smart people to write 99% of your code,

but still keep them almost as insulated from users as they would

be in a traditional research department. The toolmakers would have

users, but they'd only be the company's own developers. [6]If Microsoft used this approach, their software wouldn't be so full

of security holes, because the less smart people writing the actual

applications wouldn't be doing low-level stuff like allocating

memory. Instead of writing Word directly in C, they'd be plugging

together big Lego blocks of Word-language. (Duplo, I believe, is

the technical term.)ClumpingAlong with interesting problems, what good hackers like is other

good hackers. Great hackers tend to clump together-- sometimes

spectacularly so, as at Xerox Parc. So you won't attract good

hackers in linear proportion to how good an environment you create

for them. The tendency to clump means it's more like the square

of the environment. So it's winner take all. At any given time,

there are only about ten or twenty places where hackers most want to

work, and if you aren't one of them, you won't just have fewer

great hackers, you'll have zero.Having great hackers is not, by itself, enough to make a company

successful. It works well for Google and ITA, which are two of

the hot spots right now, but it didn't help Thinking Machines or

Xerox. Sun had a good run for a while, but their business model

is a down elevator. In that situation, even the best hackers can't

save you.I think, though, that all other things being equal, a company that

can attract great hackers will have a huge advantage. There are

people who would disagree with this. When we were making the rounds

of venture capital firms in the 1990s, several told us that software

companies didn't win by writing great software, but through brand,

and dominating channels, and doing the right deals.They really seemed to believe this, and I think I know why. I

think what a lot of VCs are looking for, at least unconsciously,

is the next Microsoft. And of course if Microsoft is your model,

you shouldn't be looking for companies that hope to win by writing

great software. But VCs are mistaken to look for the next Microsoft,

because no startup can be the next Microsoft unless some other

company is prepared to bend over at just the right moment and be

the next IBM.It's a mistake to use Microsoft as a model, because their whole

culture derives from that one lucky break. Microsoft is a bad data

point. If you throw them out, you find that good products do tend

to win in the market. What VCs should be looking for is the next

Apple, or the next Google.I think Bill Gates knows this. What worries him about Google is

not the power of their brand, but the fact that they have

better hackers. [7]

RecognitionSo who are the great hackers? How do you know when you meet one?

That turns out to be very hard. Even hackers can't tell. I'm

pretty sure now that my friend Trevor Blackwell is a great hacker.

You may have read on Slashdot how he made his

own Segway. The

remarkable thing about this project was that he wrote all the

software in one day (in Python, incidentally).For Trevor, that's

par for the course. But when I first met him, I thought he was a

complete idiot. He was standing in Robert Morris's office babbling

at him about something or other, and I remember standing behind

him making frantic gestures at Robert to shoo this nut out of his

office so we could go to lunch. Robert says he misjudged Trevor

at first too. Apparently when Robert first met him, Trevor had

just begun a new scheme that involved writing down everything about

every aspect of his life on a stack of index cards, which he carried

with him everywhere. He'd also just arrived from Canada, and had

a strong Canadian accent and a mullet.The problem is compounded by the fact that hackers, despite their

reputation for social obliviousness, sometimes put a good deal of

effort into seeming smart. When I was in grad school I used to

hang around the MIT AI Lab occasionally. It was kind of intimidating

at first. Everyone there spoke so fast. But after a while I

learned the trick of speaking fast. You don't have to think any

faster; just use twice as many words to say everything. With this amount of noise in the signal, it's hard to tell good

hackers when you meet them. I can't tell, even now. You also

can't tell from their resumes. It seems like the only way to judge

a hacker is to work with him on something.And this is the reason that high-tech areas

only happen around universities. The active ingredient

here is not so much the professors as the students. Startups grow up

around universities because universities bring together promising young

people and make them work on the same projects. The

smart ones learn who the other smart ones are, and together

they cook up new projects of their own.Because you can't tell a great hacker except by working with him,

hackers themselves can't tell how good they are. This is true to

a degree in most fields. I've found that people who

are great at something are not so much convinced of their own

greatness as mystified at why everyone else seems so incompetent.

But it's particularly hard for hackers to know how good they are,

because it's hard to compare their work. This is easier in most

other fields. In the hundred meters, you know in 10 seconds who's

fastest. Even in math there seems to be a general consensus about

which problems are hard to solve, and what constitutes a good

solution. But hacking is like writing. Who can say which of two

novels is better? Certainly not the authors.With hackers, at least, other hackers can tell. That's because,

unlike novelists, hackers collaborate on projects. When you get

to hit a few difficult problems over the net at someone, you learn

pretty quickly how hard they hit them back. But hackers can't

watch themselves at work. So if you ask a great hacker how good

he is, he's almost certain to reply, I don't know. He's not just

being modest. He really doesn't know.And none of us know, except about people we've actually worked

with. Which puts us in a weird situation: we don't know who our

heroes should be. The hackers who become famous tend to become

famous by random accidents of PR. Occasionally I need to give an

example of a great hacker, and I never know who to use. The first

names that come to mind always tend to be people I know personally,

but it seems lame to use them. So, I think, maybe I should say

Richard Stallman, or Linus Torvalds, or Alan Kay, or someone famous

like that. But I have no idea if these guys are great hackers.

I've never worked with them on anything.If there is a Michael Jordan of hacking, no one knows, including

him.CultivationFinally, the question the hackers have all been wondering about:

how do you become a great hacker? I don't know if it's possible

to make yourself into one. But it's certainly possible to do things

that make you stupid, and if you can make yourself stupid, you

can probably make yourself smart too.The key to being a good hacker may be to work on what you like.

When I think about the great hackers I know, one thing they have

in common is the extreme

difficulty of making them work

on anything they

don't want to. I don't know if this is cause or effect; it may be

both.To do something well you have to love it.

So to the extent you

can preserve hacking as something you love, you're likely to do it

well. Try to keep the sense of wonder you had about programming at

age 14. If you're worried that your current job is rotting your

brain, it probably is.The best hackers tend to be smart, of course, but that's true in

a lot of fields. Is there some quality that's unique to hackers?

I asked some friends, and the number one thing they mentioned was

curiosity.

I'd always supposed that all smart people were curious--

that curiosity was simply the first derivative of knowledge. But

apparently hackers are particularly curious, especially about how

things work. That makes sense, because programs are in effect

giant descriptions of how things work.Several friends mentioned hackers' ability to concentrate-- their

ability, as one put it, to "tune out everything outside their own

heads.'' I've certainly noticed this. And I've heard several

hackers say that after drinking even half a beer they can't program at

all. So maybe hacking does require some special ability to focus.

Perhaps great hackers can load a large amount of context into their

head, so that when they look at a line of code, they see not just

that line but the whole program around it. John McPhee

wrote that Bill Bradley's success as a basketball player was due

partly to his extraordinary peripheral vision. "Perfect'' eyesight

means about 47 degrees of vertical peripheral vision. Bill Bradley

had 70; he could see the basket when he was looking at the floor.

Maybe great hackers have some similar inborn ability. (I cheat by

using a very dense language,

which shrinks the court.)This could explain the disconnect over cubicles. Maybe the people

in charge of facilities, not having any concentration to shatter,

have no idea that working in a cubicle feels to a hacker like having

one's brain in a blender. (Whereas Bill, if the rumors of autism

are true, knows all too well.)One difference I've noticed between great hackers and smart people

in general is that hackers are more

politically incorrect. To the

extent there is a secret handshake among good hackers, it's when they

know one another well enough to express opinions that would get

them stoned to death by the general public. And I can see why

political incorrectness would be a useful quality in programming.

Programs are very complex and, at least in the hands of good

programmers, very fluid. In such situations it's helpful to have

a habit of questioning assumptions.Can you cultivate these qualities? I don't know. But you can at

least not repress them. So here is my best shot at a recipe. If

it is possible to make yourself into a great hacker, the way to do

it may be to make the following deal with yourself: you never have

to work on boring projects (unless your family will starve otherwise),

and in return, you'll never allow yourself to do a half-assed job.

All the great hackers I know seem to have made that deal, though

perhaps none of them had any choice in the matter.Notes

[1] In fairness, I have to say that IBM makes decent hardware. I

wrote this on an IBM laptop.[2] They did turn out to be doomed. They shut down a few months

later.[3] I think this is what people mean when they talk

about the "meaning of life." On the face of it, this seems an

odd idea. Life isn't an expression; how could it have meaning?

But it can have a quality that feels a lot like meaning. In a project

like a compiler, you have to solve a lot of problems, but the problems

all fall into a pattern, as in a signal. Whereas when the problems

you have to solve are random, they seem like noise.

[4] Einstein at one point worked designing refrigerators. (He had equity.)[5] It's hard to say exactly what constitutes research in the

computer world, but as a first approximation, it's software that

doesn't have users.I don't think it's publication that makes the best hackers want to work

in research departments. I think it's mainly not having to have a

three hour meeting with a product manager about problems integrating

the Korean version of Word 13.27 with the talking paperclip.[6] Something similar has been happening for a long time in the

construction industry. When you had a house built a couple hundred

years ago, the local builders built everything in it. But increasingly

what builders do is assemble components designed and manufactured

by someone else. This has, like the arrival of desktop publishing,

given people the freedom to experiment in disastrous ways, but it

is certainly more efficient.[7] Google is much more dangerous to Microsoft than Netscape was.

Probably more dangerous than any other company has ever been. Not

least because they're determined to fight. On their job listing

page, they say that one of their "core values'' is "Don't be evil.''

From a company selling soybean oil or mining equipment, such a

statement would merely be eccentric. But I think all of us in the

computer world recognize who that is a declaration of war on.Thanks to Jessica Livingston, Robert Morris, and Sarah Harlin

for reading earlier versions of this talk.Audio of talkThe Python ParadoxJapanese TranslationRussian TranslationItalian TranslationSpanish Translation

If you liked this, you may also like

Hackers & Painters.

Mind the Gap

May 2004When people care enough about something to do it well, those who

do it best tend to be far better than everyone else. There's a

huge gap between Leonardo and second-rate contemporaries like

Borgognone. You see the same gap between Raymond Chandler and the

average writer of detective novels. A top-ranked professional chess

player could play ten thousand games against an ordinary club player

without losing once.Like chess or painting or writing novels, making money is a very

specialized skill. But for some reason we treat this skill

differently. No one complains when a few people surpass all the

rest at playing chess or writing novels, but when a few people make

more money than the rest, we get editorials saying this is wrong.Why? The pattern of variation seems no different than for any other

skill. What causes people to react so strongly when the skill is

making money?I think there are three reasons we treat making money as different:

the misleading model of wealth we learn as children; the disreputable

way in which, till recently, most fortunes were accumulated; and

the worry that great variations in income are somehow bad for

society. As far as I can tell, the first is mistaken, the second

outdated, and the third empirically false. Could it be that, in a

modern democracy, variation in income is actually a sign of health?The Daddy Model of WealthWhen I was five I thought electricity was created by electric

sockets. I didn't realize there were power plants out there

generating it. Likewise, it doesn't occur to most kids that wealth

is something that has to be generated. It seems to be something

that flows from parents.Because of the circumstances in which they encounter it, children

tend to misunderstand wealth. They confuse it with money. They

think that there is a fixed amount of it. And they think of it as

something that's distributed by authorities (and so should be

distributed equally), rather than something that has to be created

(and might be created unequally).In fact, wealth is not money. Money is just a convenient way of

trading one form of wealth for another. Wealth is the underlying

stuff—the goods and services we buy. When you travel to a

rich or poor country, you don't have to look at people's bank

accounts to tell which kind you're in. You can see

wealth—in buildings and streets, in the clothes and the health

of the people.Where does wealth come from? People make it. This was easier to

grasp when most people lived on farms, and made many of the things

they wanted with their own hands. Then you could see in the house,

the herds, and the granary the wealth that each family created. It

was obvious then too that the wealth of the world was not a fixed

quantity that had to be shared out, like slices of a pie. If you

wanted more wealth, you could make it.This is just as true today, though few of us create wealth directly

for ourselves (except for a few vestigial domestic tasks). Mostly

we create wealth for other people in exchange for money, which we

then trade for the forms of wealth we want.

[1]Because kids are unable to create wealth, whatever they have has

to be given to them. And when wealth is something you're given,

then of course it seems that it should be distributed equally.

[2]

As in most families it is. The kids see to that. "Unfair," they

cry, when one sibling gets more than another.In the real world, you can't keep living off your parents. If you

want something, you either have to make it, or do something of

equivalent value for someone else, in order to get them to give you

enough money to buy it. In the real world, wealth is (except for

a few specialists like thieves and speculators) something you have

to create, not something that's distributed by Daddy. And since

the ability and desire to create it vary from person to person,

it's not made equally.You get paid by doing or making something people want, and those

who make more money are often simply better at doing what people

want. Top actors make a lot more money than B-list actors. The

B-list actors might be almost as charismatic, but when people go

to the theater and look at the list of movies playing, they want

that extra oomph that the big stars have.Doing what people want is not the only way to get money, of course.

You could also rob banks, or solicit bribes, or establish a monopoly.

Such tricks account for some variation in wealth, and indeed for

some of the biggest individual fortunes, but they are not the root

cause of variation in income. The root cause of variation in income,

as Occam's Razor implies, is the same as the root cause of variation

in every other human skill.In the United States, the CEO of a large public company makes about

100 times as much as the average person.

[3]

Basketball players

make about 128 times as much, and baseball players 72 times as much.

Editorials quote this kind of statistic with horror. But I have

no trouble imagining that one person could be 100 times as productive

as another. In ancient Rome the price of slaves varied by

a factor of 50 depending on their skills.

[4]

And that's without

considering motivation, or the extra leverage in productivity that

you can get from modern technology.Editorials about athletes' or CEOs' salaries remind me of early

Christian writers, arguing from first principles about whether the

Earth was round, when they could just walk outside and check.

[5]

How much someone's work is worth is not a policy question. It's

something the market already determines."Are they really worth 100 of us?" editorialists ask. Depends on

what you mean by worth. If you mean worth in the sense of what

people will pay for their skills, the answer is yes, apparently.A few CEOs' incomes reflect some kind of wrongdoing. But are there

not others whose incomes really do reflect the wealth they generate?

Steve Jobs saved a company that was in a terminal decline. And not

merely in the way a turnaround specialist does, by cutting costs;

he had to decide what Apple's next products should be. Few others

could have done it. And regardless of the case with CEOs, it's

hard to see how anyone could argue that the salaries of professional

basketball players don't reflect supply and demand.It may seem unlikely in principle that one individual could really

generate so much more wealth than another. The key to this mystery

is to revisit that question, are they really worth 100 of us?

Would a basketball team trade one of their players for 100

random people? What would Apple's next product look like if you

replaced Steve Jobs with a committee of 100 random people?

[6]

These

things don't scale linearly. Perhaps the CEO or the professional

athlete has only ten times (whatever that means) the skill and

determination of an ordinary person. But it makes all the difference

that it's concentrated in one individual.When we say that one kind of work is overpaid and another underpaid,

what are we really saying? In a free market, prices are determined

by what buyers want. People like baseball more than poetry, so

baseball players make more than poets. To say that a certain kind

of work is underpaid is thus identical with saying that people want

the wrong things.Well, of course people want the wrong things. It seems odd to be

surprised by that. And it seems even odder to say that it's

unjust that certain kinds of work are underpaid.

[7]

Then

you're saying that it's unjust that people want the wrong things.

It's lamentable that people prefer reality TV and corndogs to

Shakespeare and steamed vegetables, but unjust? That seems like

saying that blue is heavy, or that up is circular.The appearance of the word "unjust" here is the unmistakable spectral

signature of the Daddy Model. Why else would this idea occur in

this odd context? Whereas if the speaker were still operating on

the Daddy Model, and saw wealth as something that flowed from a

common source and had to be shared out, rather than something

generated by doing what other people wanted, this is exactly what

you'd get on noticing that some people made much more than others.When we talk about "unequal distribution of income," we should

also ask, where does that income come from?

[8]

Who made the wealth

it represents? Because to the extent that income varies simply

according to how much wealth people create, the distribution may

be unequal, but it's hardly unjust.Stealing ItThe second reason we tend to find great disparities of wealth

alarming is that for most of human history the usual way to accumulate

a fortune was to steal it: in pastoral societies by cattle raiding;

in agricultural societies by appropriating others' estates in times

of war, and taxing them in times of peace.In conflicts, those on the winning side would receive the estates

confiscated from the losers. In England in the 1060s, when William

the Conqueror distributed the estates of the defeated Anglo-Saxon

nobles to his followers, the conflict was military. By the 1530s,

when Henry VIII distributed the estates of the monasteries to his

followers, it was mostly political.

[9]

But the principle was the

same. Indeed, the same principle is at work now in Zimbabwe.In more organized societies, like China, the ruler and his officials

used taxation instead of confiscation. But here too we see the

same principle: the way to get rich was not to create wealth, but

to serve a ruler powerful enough to appropriate it.This started to change in Europe with the rise of the middle class.

Now we think of the middle class as people who are neither rich nor

poor, but originally they were a distinct group. In a feudal

society, there are just two classes: a warrior aristocracy, and the

serfs who work their estates. The middle class were a new, third

group who lived in towns and supported themselves by manufacturing

and trade.Starting in the tenth and eleventh centuries, petty nobles and

former serfs banded together in towns that gradually became powerful

enough to ignore the local feudal lords.

[10]

Like serfs, the middle

class made a living largely by creating wealth. (In port cities

like Genoa and Pisa, they also engaged in piracy.) But unlike serfs

they had an incentive to create a lot of it. Any wealth a serf

created belonged to his master. There was not much point in making

more than you could hide. Whereas the independence of the townsmen

allowed them to keep whatever wealth they created.Once it became possible to get rich by creating wealth, society as

a whole started to get richer very rapidly. Nearly everything we

have was created by the middle class. Indeed, the other two classes

have effectively disappeared in industrial societies, and their

names been given to either end of the middle class. (In the original

sense of the word, Bill Gates is middle class.)But it was not till the Industrial Revolution that wealth creation

definitively replaced corruption as the best way to get rich. In

England, at least, corruption only became unfashionable (and in

fact only started to be called "corruption") when there started to

be other, faster ways to get rich.Seventeenth-century England was much like the third world today,

in that government office was a recognized route to wealth. The

great fortunes of that time still derived more from what we would

now call corruption than from commerce.

[11]

By the nineteenth

century that had changed. There continued to be bribes, as there

still are everywhere, but politics had by then been left to men who

were driven more by vanity than greed. Technology had made it

possible to create wealth faster than you could steal it. The

prototypical rich man of the nineteenth century was not a courtier

but an industrialist.With the rise of the middle class, wealth stopped being a zero-sum

game. Jobs and Wozniak didn't have to make us poor to make themselves

rich. Quite the opposite: they created things that made our lives

materially richer. They had to, or we wouldn't have paid for them.But since for most of the world's history the main route to wealth

was to steal it, we tend to be suspicious of rich people. Idealistic

undergraduates find their unconsciously preserved child's model of

wealth confirmed by eminent writers of the past. It is a case of

the mistaken meeting the outdated."Behind every great fortune, there is a crime," Balzac wrote. Except

he didn't. What he actually said was that a great fortune with no

apparent cause was probably due to a crime well enough executed

that it had been forgotten. If we were talking about Europe in

1000, or most of the third world today, the standard misquotation

would be spot on. But Balzac lived in nineteenth-century France,

where the Industrial Revolution was well advanced. He knew you

could make a fortune without stealing it. After all, he did himself,

as a popular novelist.

[12]Only a few countries (by no coincidence, the richest ones) have

reached this stage. In most, corruption still has the upper hand.

In most, the fastest way to get wealth is by stealing it. And so

when we see increasing differences in income in a rich country,

there is a tendency to worry that it's sliding back toward becoming

another Venezuela. I think the opposite is happening. I think

you're seeing a country a full step ahead of Venezuela.The Lever of TechnologyWill technology increase the gap between rich and poor? It will

certainly increase the gap between the productive and the unproductive.

That's the whole point of technology. With a tractor an energetic

farmer could plow six times as much land in a day as he could with

a team of horses. But only if he mastered a new kind of farming.I've seen the lever of technology grow visibly in my own time. In

high school I made money by mowing lawns and scooping ice cream at

Baskin-Robbins. This was the only kind of work available at the

time. Now high school kids could write software or design web

sites. But only some of them will; the rest will still be scooping

ice cream.I remember very vividly when in 1985 improved technology made it

possible for me to buy a computer of my own. Within months I was

using it to make money as a freelance programmer. A few years

before, I couldn't have done this. A few years before, there was

no such thing as a freelance programmer. But Apple created

wealth, in the form of powerful, inexpensive computers, and programmers

immediately set to work using it to create more.As this example suggests, the rate at which technology increases

our productive capacity is probably exponential, rather than linear.

So we should expect to see ever-increasing variation in individual

productivity as time goes on. Will that increase the gap between

rich and the poor? Depends which gap you mean.Technology should increase the gap in income, but it seems to

decrease other gaps. A hundred years ago, the rich led a different

kind of life from ordinary people. They lived in houses

full of servants, wore elaborately uncomfortable clothes, and

travelled about in carriages drawn by teams of horses which themselves

required their own houses and servants. Now, thanks to technology,

the rich live more like the average person.Cars are a good example of why. It's possible to buy expensive,

handmade cars that cost hundreds of thousands of dollars. But there

is not much point. Companies make more money by building a large

number of ordinary cars than a small number of expensive ones. So

a company making a mass-produced car can afford to spend a lot more

on its design. If you buy a custom-made car, something will always

be breaking. The only point of buying one now is to advertise that

you can.Or consider watches. Fifty years ago, by spending a lot of money

on a watch you could get better performance. When watches had

mechanical movements, expensive watches kept better time. Not any

more. Since the invention of the quartz movement, an ordinary Timex

is more accurate than a Patek Philippe costing hundreds of thousands

of dollars.

[13]

Indeed, as with expensive cars, if you're determined

to spend a lot of money on a watch, you have to put up with some

inconvenience to do it: as well as keeping worse time, mechanical

watches have to be wound.The only thing technology can't cheapen is brand. Which is precisely

why we hear ever more about it. Brand is the residue left as the

substantive differences between rich and poor evaporate. But what

label you have on your stuff is a much smaller matter than having

it versus not having it. In 1900, if you kept a carriage, no one

asked what year or brand it was. If you had one, you were rich.

And if you weren't rich, you took the omnibus or walked. Now even

the poorest Americans drive cars, and it is only because we're so

well trained by advertising that we can even recognize the especially

expensive ones.

[14]The same pattern has played out in industry after industry. If

there is enough demand for something, technology will make it cheap

enough to sell in large volumes, and the mass-produced versions

will be, if not better, at least more convenient.

[15]

And there

is nothing the rich like more than convenience. The rich people I

know drive the same cars, wear the same clothes, have the same kind

of furniture, and eat the same foods as my other friends. Their

houses are in different neighborhoods, or if in the same neighborhood

are different sizes, but within them life is similar. The houses

are made using the same construction techniques and contain much

the same objects. It's inconvenient to do something expensive and

custom.The rich spend their time more like everyone else too. Bertie

Wooster seems long gone. Now, most people who are rich enough not

to work do anyway. It's not just social pressure that makes them;

idleness is lonely and demoralizing.Nor do we have the social distinctions there were a hundred years

ago. The novels and etiquette manuals of that period read now

like descriptions of some strange tribal society. "With respect

to the continuance of friendships..." hints Mrs. Beeton's Book

of Household Management (1880), "it may be found necessary, in

some cases, for a mistress to relinquish, on assuming the responsibility

of a household, many of those commenced in the earlier part of her

life." A woman who married a rich man was expected to drop friends

who didn't. You'd seem a barbarian if you behaved that way today.

You'd also have a very boring life. People still tend to segregate

themselves somewhat, but much more on the basis of education than

wealth.

[16]Materially and socially, technology seems to be decreasing the gap

between the rich and the poor, not increasing it. If Lenin walked

around the offices of a company like Yahoo or Intel or Cisco, he'd

think communism had won. Everyone would be wearing the same clothes,

have the same kind of office (or rather, cubicle) with the same

furnishings, and address one another by their first names instead

of by honorifics. Everything would seem exactly as he'd predicted,

until he looked at their bank accounts. Oops.Is it a problem if technology increases that gap? It doesn't seem

to be so far. As it increases the gap in income, it seems to

decrease most other gaps.Alternative to an AxiomOne often hears a policy criticized on the grounds that it would

increase the income gap between rich and poor. As if it were an

axiom that this would be bad. It might be true that increased

variation in income would be bad, but I don't see how we can say

it's axiomatic.Indeed, it may even be false, in industrial democracies. In a

society of serfs and warlords, certainly, variation in income is a

sign of an underlying problem. But serfdom is not the only cause

of variation in income. A 747 pilot doesn't make 40 times as much

as a checkout clerk because he is a warlord who somehow holds her

in thrall. His skills are simply much more valuable.I'd like to propose an alternative idea: that in a modern society,

increasing variation in income is a sign of health. Technology

seems to increase the variation in productivity at faster than

linear rates. If we don't see corresponding variation in income,

there are three possible explanations: (a) that technical innovation

has stopped, (b) that the people who would create the most wealth

aren't doing it, or (c) that they aren't getting paid for it.I think we can safely say that (a) and (b) would be bad. If you

disagree, try living for a year using only the resources available

to the average Frankish nobleman in 800, and report back to us.

(I'll be generous and not send you back to the stone age.)The only option, if you're going to have an increasingly prosperous

society without increasing variation in income, seems to be (c),

that people will create a lot of wealth without being paid for it.

That Jobs and Wozniak, for example, will cheerfully work 20-hour

days to produce the Apple computer for a society that allows them,

after taxes, to keep just enough of their income to match what they

would have made working 9 to 5 at a big company.Will people create wealth if they can't get paid for it? Only if

it's fun. People will write operating systems for free. But they

won't install them, or take support calls, or train customers to

use them. And at least 90% of the work that even the highest tech

companies do is of this second, unedifying kind.All the unfun kinds of wealth creation slow dramatically in a society

that confiscates private fortunes. We can confirm this empirically.

Suppose you hear a strange noise that you think may be due to a

nearby fan. You turn the fan off, and the noise stops. You turn

the fan back on, and the noise starts again. Off, quiet. On,

noise. In the absence of other information, it would seem the noise

is caused by the fan.At various times and places in history, whether you could accumulate

a fortune by creating wealth has been turned on and off. Northern

Italy in 800, off (warlords would steal it). Northern Italy in

1100, on. Central France in 1100, off (still feudal). England in

1800, on. England in 1974, off (98% tax on investment income).

United States in 1974, on. We've even had a twin study: West

Germany, on; East Germany, off. In every case, the creation of

wealth seems to appear and disappear like the noise of a fan as you

switch on and off the prospect of keeping it.There is some momentum involved. It probably takes at least a

generation to turn people into East Germans (luckily for England).

But if it were merely a fan we were studying, without all the extra

baggage that comes from the controversial topic of wealth, no one

would have any doubt that the fan was causing the noise.If you suppress variations in income, whether by stealing private

fortunes, as feudal rulers used to do, or by taxing them away, as

some modern governments have done, the result always seems to be

the same. Society as a whole ends up poorer.If I had a choice of living in a society where I was materially

much better off than I am now, but was among the poorest, or in one

where I was the richest, but much worse off than I am now, I'd take

the first option. If I had children, it would arguably be immoral

not to. It's absolute poverty you want to avoid, not relative

poverty. If, as the evidence so far implies, you have to have one

or the other in your society, take relative poverty.You need rich people in your society not so much because in spending

their money they create jobs, but because of what they have to do

to get rich. I'm not talking about the trickle-down effect

here. I'm not saying that if you let Henry Ford get rich, he'll

hire you as a waiter at his next party. I'm saying that he'll make

you a tractor to replace your horse.Notes[1]

Part of the reason this subject is so contentious is that some

of those most vocal on the subject of wealth—university

students, heirs, professors, politicians, and journalists—have

the least experience creating it. (This phenomenon will be familiar

to anyone who has overheard conversations about sports in a bar.)Students are mostly still on the parental dole, and have not stopped

to think about where that money comes from. Heirs will be on the

parental dole for life. Professors and politicians live within

socialist eddies of the economy, at one remove from the creation

of wealth, and are paid a flat rate regardless of how hard they

work. And journalists as part of their professional code segregate

themselves from the revenue-collecting half of the businesses they

work for (the ad sales department). Many of these people never

come face to face with the fact that the money they receive represents

wealth—wealth that, except in the case of journalists, someone

else created earlier. They live in a world in which income is

doled out by a central authority according to some abstract notion

of fairness (or randomly, in the case of heirs), rather than given

by other people in return for something they wanted, so it may seem

to them unfair that things don't work the same in the rest of the

economy.(Some professors do create a great deal of wealth for

society. But the money they're paid isn't a quid pro quo.

It's more in the nature of an investment.)[2]

When one reads about the origins of the Fabian Society, it

sounds like something cooked up by the high-minded Edwardian

child-heroes of Edith Nesbit's The Wouldbegoods.[3]

According to a study by the Corporate Library, the median total

compensation, including salary, bonus, stock grants, and the exercise

of stock options, of S&P 500 CEOs in 2002 was $3.65 million.

According to Sports Illustrated, the average NBA player's

salary during the 2002-03 season was $4.54 million, and the average

major league baseball player's salary at the start of the 2003

season was $2.56 million. According to the Bureau of Labor

Statistics, the mean annual wage in the US in 2002 was $35,560.[4]

In the early empire the price of an ordinary adult slave seems

to have been about 2,000 sestertii (e.g. Horace, Sat. ii.7.43).

A servant girl cost 600 (Martial vi.66), while Columella (iii.3.8)

says that a skilled vine-dresser was worth 8,000. A doctor, P.

Decimus Eros Merula, paid 50,000 sestertii for his freedom (Dessau,

Inscriptiones 7812). Seneca (Ep. xxvii.7) reports

that one Calvisius Sabinus paid 100,000 sestertii apiece for slaves

learned in the Greek classics. Pliny (Hist. Nat. vii.39)

says that the highest price paid for a slave up to his time was

700,000 sestertii, for the linguist (and presumably teacher) Daphnis,

but that this had since been exceeded by actors buying their own

freedom.Classical Athens saw a similar variation in prices. An ordinary

laborer was worth about 125 to 150 drachmae. Xenophon (Mem.

ii.5) mentions prices ranging from 50 to 6,000 drachmae (for the

manager of a silver mine).For more on the economics of ancient slavery see:Jones, A. H. M., "Slavery in the Ancient World," Economic History

Review, 2:9 (1956), 185-199, reprinted in Finley, M. I. (ed.),

Slavery in Classical Antiquity, Heffer, 1964.[5]

Eratosthenes (276—195 BC) used shadow lengths in different

cities to estimate the Earth's circumference. He was off by only

about 2%.[6]

No, and Windows, respectively.[7]

One of the biggest divergences between the Daddy Model and

reality is the valuation of hard work. In the Daddy Model, hard

work is in itself deserving. In reality, wealth is measured by

what one delivers, not how much effort it costs. If I paint someone's

house, the owner shouldn't pay me extra for doing it with a toothbrush.It will seem to someone still implicitly operating on the Daddy

Model that it is unfair when someone works hard and doesn't get

paid much. To help clarify the matter, get rid of everyone else

and put our worker on a desert island, hunting and gathering fruit.

If he's bad at it he'll work very hard and not end up with much

food. Is this unfair? Who is being unfair to him?[8]

Part of the reason for the tenacity of the Daddy Model may be

the dual meaning of "distribution." When economists talk about

"distribution of income," they mean statistical distribution. But

when you use the phrase frequently, you can't help associating it

with the other sense of the word (as in e.g. "distribution of alms"),

and thereby subconsciously seeing wealth as something that flows

from some central tap. The word "regressive" as applied to tax

rates has a similar effect, at least on me; how can anything

regressive be good?[9]

"From the beginning of the reign Thomas Lord Roos was an assiduous

courtier of the young Henry VIII and was soon to reap the rewards.

In 1525 he was made a Knight of the Garter and given the Earldom

of Rutland. In the thirties his support of the breach with Rome,

his zeal in crushing the Pilgrimage of Grace, and his readiness to

vote the death-penalty in the succession of spectacular treason

trials that punctuated Henry's erratic matrimonial progress made

him an obvious candidate for grants of monastic property."Stone, Lawrence, Family and Fortune: Studies in Aristocratic

Finance in the Sixteenth and Seventeenth Centuries, Oxford

University Press, 1973, p. 166.[10]

There is archaeological evidence for large settlements earlier,

but it's hard to say what was happening in them.Hodges, Richard and David Whitehouse, Mohammed, Charlemagne and

the Origins of Europe, Cornell University Press, 1983.[11]

William Cecil and his son Robert were each in turn the most

powerful minister of the crown, and both used their position to

amass fortunes among the largest of their times. Robert in particular

took bribery to the point of treason. "As Secretary of State and

the leading advisor to King James on foreign policy, [he] was a

special recipient of favour, being offered large bribes by the Dutch

not to make peace with Spain, and large bribes by Spain to make

peace." (Stone, op. cit., p. 17.)[12]

Though Balzac made a lot of money from writing, he was notoriously

improvident and was troubled by debts all his life.[13]

A Timex will gain or lose about .5 seconds per day. The most

accurate mechanical watch, the Patek Philippe 10 Day Tourbillon,

is rated at -1.5 to +2 seconds. Its retail price is about $220,000.[14]

If asked to choose which was more expensive, a well-preserved

1989 Lincoln Town Car ten-passenger limousine ($5,000) or a 2004

Mercedes S600 sedan ($122,000), the average Edwardian might well

guess wrong.[15]

To say anything meaningful about income trends, you have to

talk about real income, or income as measured in what it can buy.

But the usual way of calculating real income ignores much of the

growth in wealth over time, because it depends on a consumer price

index created by bolting end to end a series of numbers that are

only locally accurate, and that don't include the prices of new

inventions until they become so common that their prices stabilize.So while we might think it was very much better to live in a world

with antibiotics or air travel or an electric power grid than

without, real income statistics calculated in the usual way will

prove to us that we are only slightly richer for having these things.Another approach would be to ask, if you were going back to the

year x in a time machine, how much would you have to spend on trade

goods to make your fortune? For example, if you were going back

to 1970 it would certainly be less than $500, because the processing

power you can get for $500 today would have been worth at least

$150 million in 1970. The function goes asymptotic fairly quickly,

because for times over a hundred years or so you could get all you

needed in present-day trash. In 1800 an empty plastic drink bottle

with a screw top would have seemed a miracle of workmanship.[16]

Some will say this amounts to the same thing, because the rich

have better opportunities for education. That's a valid point. It

is still possible, to a degree, to buy your kids' way into top

colleges by sending them to private schools that in effect hack the

college admissions process.According to a 2002 report by the National Center for Education

Statistics, about 1.7% of American kids attend private, non-sectarian

schools. At Princeton, 36% of the class of 2007 came from such

schools. (Interestingly, the number at Harvard is significantly

lower, about 28%.) Obviously this is a huge loophole. It does at

least seem to be closing, not widening.Perhaps the designers of admissions processes should take a lesson

from the example of computer security, and instead of just assuming

that their system can't be hacked, measure the degree to which it

is.Spanish Translation

How to Make Wealth

Want to start a startup? Get funded by

Y Combinator.

May 2004

(This essay was originally published in Hackers

& Painters.)

If you wanted to get rich, how would you do it? I think your best

bet would be to start or join a startup. That's been a

reliable way to get rich for hundreds of years. The word "startup"

dates from the 1960s, but what happens in one is

very similar to the venture-backed trading voyages of the

Middle Ages.Startups usually involve technology, so much so that the phrase

"high-tech startup" is almost redundant. A startup is a small

company that takes on a hard technical problem.Lots of people get rich knowing nothing more than that.

You don't have to know physics to be a good pitcher. But

I think it could give you an edge to understand the underlying principles.

Why do startups have to be small?

Will a startup inevitably stop being a startup as it

grows larger?

And why do they so often work on

developing new technology? Why are there so many startups

selling new drugs or computer software, and none selling corn oil

or laundry detergent?The PropositionEconomically, you can think of a startup as a way to

compress your whole working life into a few years. Instead

of working at a low intensity for forty years, you work as

hard as you possibly can for four. This pays especially well

in technology, where you earn a premium for working fast.Here is a brief sketch of the economic proposition. If you're

a good hacker in your mid twenties, you can

get a job paying about $80,000 per year. So on average

such a hacker must be

able to do at least $80,000 worth of work per year for the

company just to break even. You could probably

work twice as many hours as a corporate employee, and if

you focus you can probably get three times as much done in

an hour.

[1]

You should get another multiple of two, at

least, by eliminating the drag

of the pointy-haired middle

manager who would be your boss in a big company.

Then there is one more multiple: how much smarter are you

than your job description expects you to be?

Suppose another multiple of three. Combine all these multipliers, and I'm

claiming you could be 36 times more

productive than you're expected to be in a random corporate

job.

[2]

If a fairly good hacker is worth $80,000 a year at a

big company, then a smart

hacker working very hard without any corporate

bullshit to slow him down should be able to do work worth about

$3 million a year.Like all back-of-the-envelope calculations, this one

has a lot of wiggle room. I wouldn't try to

defend the actual numbers. But I stand by the

structure of the calculation. I'm not claiming

the multiplier is precisely 36, but it is certainly more

than 10, and probably rarely as high as 100.If $3 million a year seems

high, remember that we're talking about the limit case:

the case where you not only have zero leisure time

but indeed work so hard that you endanger your health.Startups are not magic. They don't change the laws of

wealth creation. They just represent a point at the far end of the curve.

There is a conservation law at work here: if

you want to make a million dollars, you have to endure a

million dollars' worth of pain.

For example, one way to

make a million dollars would be to work for the

Post Office your whole life, and save every penny of your

salary. Imagine the stress of working for the Post

Office for fifty years. In a startup you compress all

this stress into three or four years. You do tend to get a

certain

bulk discount if you buy the economy-size pain,

but you can't evade the fundamental conservation law.

If starting a startup were easy, everyone would do it.Millions, not BillionsIf $3 million a year seems high to some people, it will seem

low to others. Three million?

How do I get to be a billionaire, like Bill Gates?So let's get Bill Gates out of the way right now. It's not

a good idea to use famous rich people

as examples, because the press only

write about the very richest, and these tend to be outliers.

Bill Gates is a smart, determined, and hardworking man,

but you need more than

that to make as much money as he has. You also need to be

very lucky.There is a large random

factor in the success of any company. So the guys you end

up reading about in the papers are the ones who are very

smart, totally dedicated, and win the lottery.

Certainly Bill is smart and dedicated, but Microsoft also

happens to have been the beneficiary of one of the most spectacular

blunders in the history of business: the licensing deal for

DOS. No doubt Bill did

everything he could to steer IBM into making that blunder,

and he has done an excellent job of exploiting it, but if

there had been one person with a brain on IBM's side,

Microsoft's future would have been very different.

Microsoft at that stage had little leverage over IBM.

They were effectively a component supplier. If IBM had

required an exclusive license, as they should have, Microsoft

would still have signed the deal. It would still have

meant a lot of money for them, and IBM

could easily have gotten an operating system elsewhere.Instead IBM ended up using all its power in the market

to give Microsoft control of the PC standard. From

that point, all Microsoft had to do was execute. They

never had to bet the company on a bold decision. All they

had to do was play hardball with licensees and copy more

innovative products reasonably promptly.If IBM hadn't made this mistake, Microsoft would

still have been a successful company, but it

could not have grown so big so fast.

Bill Gates would be rich, but he'd be somewhere

near the bottom of the Forbes 400 with the other guys his age.There are a lot of ways to get

rich, and this essay is about only one of them. This

essay is about how to make money by creating wealth and

getting paid for it. There are plenty of other ways to

get money, including chance, speculation, marriage, inheritance,

theft, extortion, fraud, monopoly,

graft, lobbying,

counterfeiting, and prospecting. Most of the greatest fortunes

have probably involved several of these.The advantage of creating wealth, as a way to get rich,

is not just that it's more legitimate

(many of the other methods are now illegal)

but that it's more

straightforward. You just have to do something people want.Money Is Not WealthIf you want to create wealth, it will help to understand what it is.

Wealth is not the same thing as money.

[3]

Wealth is as old as

human history. Far older, in fact; ants have wealth.

Money is a comparatively recent invention.Wealth is the fundamental thing. Wealth is stuff we want: food,

clothes, houses, cars, gadgets, travel to interesting places,

and so on. You can have wealth without

having money. If you had a magic machine that

could on command make you a car or cook you dinner or do your

laundry, or do anything else you wanted, you wouldn't need money.

Whereas if you were in the middle of Antarctica, where there is

nothing to buy, it wouldn't matter how much money you had.Wealth is what you want, not money. But if wealth is the important

thing, why does everyone talk about making money? It is

a kind of shorthand: money is a way of moving wealth, and in practice

they are usually interchangeable. But they are not the same thing,

and unless you plan to get rich by counterfeiting, talking about

making money can make it harder to understand how to

make money.Money is a side effect of specialization.

In a specialized society, most of the

things you need, you can't make for yourself. If you want a potato

or a pencil or a place to live, you have to get it from someone

else.How do you get the person who grows the potatoes to give you some?

By giving him something he wants in return. But you can't get

very far by trading things directly with the people who

need them. If you make violins, and none of the local

farmers wants one, how will you eat?The solution societies find, as they get more specialized, is to

make the trade into a two-step process. Instead of trading violins

directly for potatoes, you trade violins for, say, silver,

which you can then trade again for anything else you need. The

intermediate stuff-- the medium of exchange-- can be anything that's

rare and portable. Historically metals have been the most common,

but recently we've been using a medium of exchange, called the dollar,

that doesn't physically exist. It works as a medium of exchange,

however, because its rarity

is guaranteed by the U.S. Government.The advantage of a medium of exchange is that it makes trade work.

The disadvantage is that it tends to obscure what trade really

means. People think that what a business does is make money.

But money is just the intermediate stage-- just

a shorthand-- for whatever people want.

What most businesses really do is make

wealth. They do something people want.

[4]The Pie FallacyA surprising number of people retain from childhood the idea

that there is a fixed amount of wealth in the world.

There is, in any normal family, a fixed amount of money at

any moment. But that's not the same thing.When wealth is talked about in this context, it is often

described as a pie. "You can't make the pie larger,"

say politicians.

When you're

talking about the amount of money in one family's bank

account, or the amount available to a government from one

year's tax revenue, this is true.

If one person gets more, someone else has to get less.I can remember believing, as a child, that if a few

rich people had all the money, it left less for everyone else.

Many people seem to continue to believe something like this

well into adulthood. This fallacy is usually there in the

background when you hear someone talking about how x percent

of the population have y percent of the wealth. If you plan

to start a startup, then whether you realize it or not, you're

planning to disprove the Pie Fallacy.What leads people astray here is the abstraction of

money. Money is not wealth. It's

just something we use to move wealth around.

So although there may be, in certain specific moments (like

your family, this month) a fixed amount of money available to

trade with other people for things you want,

there is not a fixed amount of wealth in the world.

You can make more wealth. Wealth has been getting created and

destroyed (but on balance, created) for all of human history.Suppose you own a beat-up old car.

Instead of sitting on your butt next

summer, you could spend the time restoring your car to pristine condition.

In doing so you create wealth. The world is-- and

you specifically are-- one pristine old car the richer. And not

just in some metaphorical way. If you sell your car,

you'll get more for it.In restoring your old car you have made yourself

richer. You haven't made anyone else poorer. So there is

obviously not a fixed pie. And in fact, when you look at

it this way, you wonder why anyone would think there was.

[5]Kids know, without knowing they know, that they can create

wealth. If you need to give someone a present and don't

have any money, you make one. But kids are so bad at making

things that they consider home-made presents to be a distinct,

inferior, sort of thing to store-bought ones-- a mere expression

of the proverbial thought that counts.

And indeed, the lumpy ashtrays

we made for our parents did not have much of a resale market.CraftsmenThe people most likely to grasp that wealth can be

created are the ones who are good at making things, the craftsmen.

Their hand-made objects become store-bought ones.

But with the rise of industrialization there are fewer and

fewer craftsmen. One of the biggest remaining groups is

computer programmers.A programmer can sit down in front of a computer and

create wealth. A good piece of software is, in itself,

a valuable thing.

There is no manufacturing to confuse the issue. Those

characters you type

are a complete, finished product.

If someone sat down and wrote a web

browser that didn't suck (a fine idea, by the way), the world

would be that much richer.

[5b]Everyone in a company works together to create

wealth, in the sense of making more things people want.

Many of the employees (e.g. the people in the mailroom or

the personnel department) work at one remove from the

actual making of stuff. Not the programmers. They

literally think the product, one line at a time.

And so it's clearer to programmers that wealth is something

that's made, rather than being distributed, like slices of a

pie, by some imaginary Daddy.It's also obvious to programmers that there are huge variations

in the rate at which wealth is created. At Viaweb we had one

programmer who was a sort of monster of productivity.

I remember watching what he did one long day and estimating that

he had added several hundred thousand dollars

to the market value of the company.

A great programmer, on a roll, could

create a million dollars worth of wealth in a couple weeks.

A mediocre programmer over the same period will generate zero or

even negative wealth (e.g. by introducing bugs).This is

why so many of the best programmers are libertarians.

In our world, you sink or swim, and there are no excuses.

When those far removed from the creation of wealth-- undergraduates,

reporters, politicians-- hear

that the richest 5% of the people have

half the total wealth, they tend to think injustice!

An experienced programmer would be more likely to think

is that all? The top 5% of programmers

probably write 99% of the good software.Wealth can be created without being sold. Scientists, till

recently at least, effectively donated the wealth they

created. We are all richer for knowing about penicillin,

because we're less likely to die from infections. Wealth

is whatever people want, and not dying is certainly something

we want. Hackers often donate their work by

writing open source software that anyone can use for free.

I am much the richer for the operating system

FreeBSD, which I'm running on the computer I'm using now,

and so is Yahoo, which runs it on all their servers.What a Job IsIn industrialized countries, people belong to one institution or

another at least until their twenties. After all those years you get

used to the idea of belonging to a group of people who all get up

in the morning, go to some set of buildings, and do things that they

do not, ordinarily, enjoy doing. Belonging to such a group becomes

part of your identity: name, age, role, institution.

If you have to introduce yourself, or

someone else describes you, it will be as something like, John

Smith, age 10, a student at such and such elementary school, or

John Smith, age 20, a student at such and such college.When John Smith finishes school he is expected to get a job. And

what getting a job seems to mean is joining another institution.

Superficially it's a lot like college. You pick the companies you

want to work for and apply to join them. If one likes you, you

become a member of this new group. You get up in the morning and

go to a new set of buildings, and do things that you do not, ordinarily,

enjoy doing. There are a few differences: life is not as much fun,

and you get paid, instead of paying, as you did in college. But

the similarities feel greater than the differences. John Smith is

now John Smith, 22, a software developer at such and such corporation.In fact John Smith's

life has changed more than he realizes. Socially, a company

looks much like college, but the deeper you go into the

underlying reality, the more different it gets.What a company does, and has to do if it wants to continue to

exist, is earn money. And the way most companies make money

is by creating wealth. Companies can be so specialized that this

similarity is concealed, but it is not only manufacturing

companies that create wealth. A big component of wealth is

location.

Remember that magic machine that could

make you cars and cook you dinner and so on? It would not be

so useful if it delivered your dinner to a random location

in central Asia.

If wealth means what people want, companies that move

things also create wealth. Ditto for

many other kinds of companies that don't make anything

physical. Nearly all companies exist to do something people

want.And that's what you do, as well, when you go to work for a company.

But here there is another layer that tends to obscure the underlying

reality. In a company, the work you do is averaged together with

a lot of other people's.

You may not even be aware you're doing something people

want. Your contribution may be indirect. But the company as a

whole must be giving people something they want, or they won't make

any money. And if they are paying you x dollars a year, then on

average you must be contributing at least x dollars a year worth

of work, or the company will be spending more than it makes,

and will go out of business.Someone graduating from college thinks, and is told, that he needs

to get a job, as if the important thing were becoming a member of

an institution. A more direct way to put it would be: you need to

start doing something people want. You don't

need to

join a company to do that. All a company is is a group of people

working together to do something people want. It's doing something people

want that matters, not joining the group.

[6]For most people the

best plan probably is to go to work for some existing

company. But it is a good idea to understand what's happening

when you do this. A job means doing something people want,

averaged together with everyone else in that company.Working HarderThat averaging gets to be a problem.

I think the single biggest problem afflicting large companies is the

difficulty of assigning a value to each person's work.

For the most part they punt. In a

big company you get paid a fairly predictable salary for working

fairly hard. You're expected not to be obviously incompetent or

lazy, but you're not expected to devote your whole life to your

work.It turns out, though, that there are economies of scale in how much of your

life you devote to your work. In the right kind of business,

someone who really devoted himself to work could generate ten or

even a hundred times as much wealth as an average

employee. A programmer, for example, instead of chugging along

maintaining and updating an existing piece of software, could write

a whole new piece of software, and with it create a new source of

revenue.Companies are not set up to reward people who want to do this.

You can't go to your boss and say, I'd like to start working ten

times as hard, so will you please pay me ten times as much? For

one thing, the official fiction is that you are already working as

hard as you can. But a more serious problem is that the company

has no way of measuring the value of your work.Salesmen are an exception. It's easy

to measure how much revenue they generate, and they're

usually paid a percentage of it. If a salesman wants to work harder,

he can just start doing it, and he will automatically

get paid proportionally more.There is one other job besides sales where big companies can

hire first-rate people: in the top management jobs.

And for the same reason: their performance can

be measured. The top managers are

held responsible for the performance of the entire company.

Because an ordinary employee's performance can't usually

be measured, he is not expected to do

more than put in a solid effort. Whereas top management, like

salespeople, have to actually come up with the numbers.

The CEO of a company that tanks cannot plead that he put in

a solid effort. If the company does badly, he's done badly.A company that could pay all its employees so straightforwardly

would be enormously successful. Many employees would work harder

if they could get paid for it. More importantly,

such a company would attract people who wanted to work

especially hard.

It would crush its competitors.Unfortunately, companies can't pay everyone like salesmen. Salesmen

work alone. Most employees' work is tangled together. Suppose

a company makes some kind of consumer gadget. The

engineers build a reliable gadget with all kinds of new features;

the industrial designers design a beautiful case for it; and then

the marketing people convince everyone that

it's something they've got to have. How do you know how much of the

gadget's sales are due to each group's efforts? Or, for that

matter, how much is due to the creators of past gadgets that gave

the company a reputation for quality? There's no way to

untangle all their contributions. Even if you could read the minds

of the consumers, you'd find these factors were all blurred together.If you want to go faster, it's a problem to have your work

tangled together with a large number of other people's. In a

large group, your performance is not separately measurable-- and

the rest of the group slows you down.Measurement and LeverageTo get rich you need to get yourself in a situation with two

things, measurement and leverage. You need to be in a

position where your performance can be measured, or there is

no way to get paid more by doing more. And you have to

have leverage, in the sense that the decisions you make have

a big effect.Measurement alone is not enough. An example of a job with

measurement but not leverage is doing piecework in a

sweatshop. Your performance is measured and you get paid

accordingly, but you have no scope for decisions. The only

decision you get to make is how fast you work, and that

can probably only increase your earnings by a factor

of two or three.An example of a job with both measurement and leverage would

be lead actor in a movie. Your performance can be measured in the

gross of the movie. And you have leverage in the sense that your

performance can make or break it.CEOs also have both measurement and leverage. They're measured,

in that the performance of the company is their performance.

And they have leverage in that their decisions

set the whole company moving in one direction or another.I think everyone who gets rich by their own efforts will be

found to be in a situation with measurement and leverage.

Everyone I can think of does: CEOs, movie stars,

hedge fund managers, professional athletes. A good hint to the

presence of leverage is the possibility of failure.

Upside must be balanced by downside, so if there is

big potential for gain there must also be a terrifying

possibility of loss. CEOs, stars, fund managers, and athletes

all live with the sword hanging over their heads;

the moment they start to suck, they're out. If you're in

a job that feels safe, you are not going to get rich,

because if there is no danger there is almost certainly no leverage.But you don't have to become a CEO or a movie star to

be in a situation with measurement and leverage. All you

need to do is be part of a small group working on a

hard problem.Smallness = MeasurementIf you can't measure the value of the work done by individual

employees, you can get close. You can measure the value

of the work done by small groups.One level at which you can accurately measure the revenue

generated by employees is at the level of the whole company.

When the company is small, you are thereby fairly close to

measuring the contributions of individual employees. A viable

startup might only have ten employees, which puts you within a

factor of ten of measuring individual effort.Starting or joining a startup is thus as close as most

people can get to saying to one's boss, I want to work ten times

as hard, so please pay me ten times as much. There are two

differences: you're not saying it to your boss, but directly to the

customers (for whom your boss is only a proxy after all), and

you're not doing it individually, but along with a small group

of other ambitious people.It will, ordinarily, be a group. Except in a few unusual kinds

of work, like acting or writing books, you can't be a company

of one person.

And the people you work with had better be good, because it's their work that

yours is going to be averaged with.A big company is like a giant galley driven by a thousand rowers.

Two things keep the speed of the

galley down. One is that individual rowers don't see any

result from working harder.

The other is that, in a group of a

thousand people, the average rower is likely to be

pretty average.If you took ten people at random out of the big galley and

put them in a boat by themselves, they could probably go

faster. They would have both carrot and stick to motivate

them. An energetic rower would be encouraged by the thought

that he could have a visible effect on the speed of

the boat. And if someone was lazy, the others would be more likely

to notice and complain.But the real advantage of the ten-man boat shows when

you take the ten best rowers out of the big galley

and put them in a boat together. They will have all

the extra motivation that comes from being in a small group.

But more importantly, by selecting that small a group

you can get the best rowers. Each one will be in

the top 1%. It's a much better deal for them to average

their work together with a small group of their peers than to

average it with everyone.That's the real point of startups. Ideally, you are getting

together with a group of other people who also want to work

a lot harder, and get paid a lot more, than they would in

a big company. And because startups tend to get founded

by self-selecting groups of ambitious people who already

know one another (at least by reputation), the level of

measurement is more precise than you get from smallness alone.

A startup is not merely ten people, but ten people like you.Steve Jobs once said that the success or failure of a startup

depends on the first ten employees. I agree. If

anything, it's more like the first five.

Being small is not, in itself, what makes startups kick butt,

but rather that small groups can be select.

You don't want small in the sense of a

village, but small in the sense of an all-star team.The larger a group, the closer its average member will be to the average

for the population as a whole. So all other things being

equal, a very able person in a big company is probably

getting a bad deal, because his performance is dragged down by

the overall lower performance of the others. Of course,

all other things often are not equal: the able person may

not care about money, or may prefer the stability of a large

company. But a very able person who does care about money

will ordinarily do better to go off and work with a small

group of peers.Technology = LeverageStartups offer anyone a way to be in a situation with

measurement and leverage.

They allow measurement because they're small,

and they offer leverage because they

make money by inventing new technology.What is technology? It's technique. It's the way

we all do things. And when

you discover a new way to do things, its value is multiplied

by all the people who use it. It is the proverbial fishing

rod, rather than the fish. That's the difference between a

startup and a restaurant or a barber shop. You fry eggs or cut

hair one customer at a time. Whereas if

you solve a technical problem that a lot of people care about,

you help everyone who uses your solution.

That's leverage.If you look at history, it seems that most people

who got rich by creating wealth did it by developing

new technology. You just can't fry eggs or cut hair fast enough.

What made the Florentines rich in 1200

was the discovery of new techniques for making the high-tech

product of the time, fine woven cloth. What made the

Dutch rich in 1600 was the discovery of shipbuilding and

navigation techniques that enabled them to dominate the seas

of the Far East.Fortunately there is a natural fit between smallness and

solving hard problems. The leading edge of technology moves

fast. Technology that's valuable today could be worthless

in a couple years. Small companies are more at home in this

world, because they don't have layers of bureaucracy to

slow them down.

Also, technical advances tend to come from unorthodox approaches,

and small companies are less constrained by convention.Big companies can develop technology. They just can't do it

quickly. Their size makes them slow and prevents

them from rewarding employees for the extraordinary

effort required. So in practice big companies only get to develop

technology in fields where large capital requirements prevent startups from

competing with them, like microprocessors, power plants,

or passenger aircraft. And even in those fields they depend heavily

on startups for components and ideas.It's obvious that biotech or software startups exist to solve

hard technical problems, but

I think it will also be found to be true

in businesses that don't seem to be about technology. McDonald's,

for example, grew big by designing a system, the McDonald's

franchise, that could then be reproduced at will all over the

face of the earth. A McDonald's franchise is controlled by rules

so precise that it is practically

a piece of software. Write once, run everywhere.

Ditto for Wal-Mart. Sam Walton got rich not by being a

retailer, but by designing a new kind of store.Use difficulty as a guide not just in selecting the overall

aim of your company, but also at decision points along the way.

At Viaweb one of our rules of thumb was run upstairs.

Suppose you are a little, nimble guy being chased by a big,

fat, bully. You open a door and find yourself in a

staircase. Do you go up or down? I say up. The

bully can probably run downstairs as fast as you can.

Going upstairs his bulk will be more of a disadvantage.

Running upstairs is hard for you but even harder for him.What this meant in practice was that we deliberately sought

hard problems. If there were two features we could add to our

software, both equally valuable in proportion to their difficulty,

we'd always take the harder one. Not just because it was

more valuable, but because it was harder.

We delighted in forcing bigger, slower competitors

to follow us over difficult ground.

Like guerillas, startups prefer the difficult terrain of the

mountains, where the troops of the central government

can't follow. I can remember times when we were just

exhausted after wrestling all day with some horrible technical

problem. And I'd be delighted, because something that was

hard for us would be impossible for our competitors.This is not just a good way to run a startup. It's what

a startup is.

Venture capitalists know about this and have a phrase for it:

barriers to entry. If you go to a VC with a new

idea and ask him to invest in it, one of the first things

he'll ask is, how hard would this be for someone else to

develop? That is, how much difficult ground

have you put between yourself and potential pursuers?

[7]

And you had better have a convincing explanation of why

your technology would be hard to duplicate. Otherwise as

soon as some big company becomes aware of it, they'll make

their own, and with their brand name, capital, and

distribution clout, they'll take away your market overnight.

You'd be like guerillas caught in the open field by regular

army forces.One way to put up barriers to entry is through patents.

But patents may not provide much protection.

Competitors commonly find ways to work around a patent.

And if they can't, they

may simply violate it and invite you to sue them.

A big company is not afraid to be sued; it's an everyday thing

for them. They'll make sure that suing them is expensive and

takes a long time.

Ever heard of Philo Farnsworth? He invented

television. The reason you've never

heard of him is that his company was not the one to make

money from it.

[8]

The company that did was RCA, and

Farnsworth's reward for his efforts was a decade of

patent litigation.Here, as so often, the best defense is a good offense. If

you can develop technology that's simply too hard for

competitors to duplicate, you don't need to rely on other

defenses. Start by picking a hard problem, and

then at every decision point, take the harder choice.

[9]The Catch(es)If it were simply a matter of working harder than

an ordinary employee and getting paid proportionately, it would

obviously be a good deal to start a startup. Up to a point it

would be more fun. I don't think many people

like the slow pace of big companies, the interminable meetings,

the water-cooler conversations, the clueless middle managers,

and so on.Unfortunately there are a couple catches. One is that you

can't choose the point on the curve that you want to inhabit.

You can't decide, for example, that you'd like to work just

two or three times as hard, and get paid that much more. When

you're running a startup, your competitors decide how

hard you work. And they pretty much all make the same decision:

as hard as you possibly can.The other catch is that the payoff is only on average proportionate

to your productivity. There is, as I said before, a large

random multiplier in the success of any company. So in

practice the deal is not that you're 30 times as productive and get

paid 30 times as much. It is that you're 30 times as productive,

and get paid between zero and a thousand times as much.

If the mean is 30x, the median is probably zero.

Most startups tank, and not just the dogfood

portals we all heard about during

the Internet Bubble. It's common for a startup

to be developing a genuinely good product, take slightly

too long to do it, run out of money, and have to shut down.A startup is like a mosquito. A bear can absorb a hit and a crab

is armored against one, but a mosquito is designed for one thing:

to score. No energy is wasted on defense. The defense of mosquitos,

as a species, is that there are a lot of them, but this is little

consolation to the individual mosquito.Startups, like mosquitos, tend to be an all-or-nothing proposition.

And you don't generally know which of the two you're going to

get till the last minute.

Viaweb came close to tanking several times. Our trajectory

was like a sine wave. Fortunately we got bought at

the top of the cycle, but it was damned close. While we were

visiting Yahoo in California to talk about selling the company

to them, we had to borrow a conference room to reassure

an investor who was about to back out of a new round of funding

that we needed to stay alive.The all-or-nothing aspect of startups was not something we wanted.

Viaweb's hackers were all extremely risk-averse.

If there had been some way just to work super hard and get

paid for it, without having a lottery mixed in, we would have

been delighted. We would have much preferred a 100% chance of

$1 million to a 20% chance of $10 million, even though

theoretically the second is worth twice as much. Unfortunately,

there is not currently any space in the business world where

you can get the first deal.The closest you can get is by

selling your startup in the early stages, giving up upside

(and risk) for a smaller but guaranteed payoff. We had a

chance to do this, and stupidly, as we then thought, let it slip by.

After that we became comically eager to sell.

For the next year or so,

if anyone expressed the slightest curiosity about Viaweb

we would try to sell them the company. But there were no takers,

so we had to keep going.It would have been a bargain to

buy us at an early stage, but companies doing acquisitions are not

looking for bargains. A company big enough to acquire

startups will be big enough to be fairly conservative, and

within the company the people in charge of acquisitions will

be among the more conservative, because they are likely to be

business school types who joined the company late.

They would rather overpay for a safe choice. So

it is easier to sell an established startup, even at a large

premium, than an early-stage one.Get UsersI think it's a good idea to get bought, if you can. Running a

business is different from growing one.

It is just as well to let a big company take over once you reach

cruising altitude. It's

also financially wiser, because selling allows you to diversify.

What would you think of a financial advisor who put all his

client's assets into one volatile stock?How do you get bought? Mostly by doing the same things

you'd do if you didn't intend to sell the company. Being

profitable, for example. But getting bought is also an art

in its own right, and one that we spent a lot of time trying

to master.Potential buyers will

always delay if they can. The hard part about getting

bought is getting them to act. For most people, the most powerful motivator

is not the hope of gain, but the fear of loss. For potential

acquirers, the most powerful motivator is the prospect that

one of their competitors will buy you. This, as we found,

causes CEOs to take red-eyes.

The second biggest is the worry that, if they don't buy you

now, you'll continue to grow rapidly and will cost more to

acquire later, or even become a competitor.In both cases, what it all comes down to is users.

You'd think that a company about to buy you would do a lot of

research and decide for themselves how valuable your technology

was. Not at all. What they go by is the number of users you

have.In effect, acquirers assume the customers know who has the

best technology. And this is not as stupid as it sounds. Users

are the only real proof that you've created wealth. Wealth is

what people want, and if people aren't using your software,

maybe it's not just because you're bad at marketing. Maybe it's

because you haven't made what they want.Venture capitalists have a list of danger signs to watch out for.

Near the top is the company run by techno-weenies who are

obsessed with solving interesting technical problems, instead

of making users happy. In a startup, you're not just trying to

solve problems. You're trying to solve problems that

users care about.So I think you should make users the test, just as

acquirers do. Treat a startup as an optimization problem

in which performance is measured by number of users. As anyone

who has tried to optimize software knows, the key is measurement.

When you try to guess where your program is slow, and what would

make it faster, you almost always guess wrong.Number of users may not be the perfect test, but it will

be very close. It's what acquirers care about. It's what

revenues depend on.

It's what makes competitors unhappy.

It's what impresses reporters, and potential

new users. Certainly it's a better test than your a priori

notions of what problems are important to solve, no matter how

technically adept you are.Among other things, treating a startup as an optimization

problem will help you avoid another

pitfall that VCs worry about, and rightly-- taking a long time

to develop a product. Now we can recognize this as something

hackers already know to avoid: premature optimization. Get a version

1.0 out there as soon as you can. Until you have some users to

measure, you're optimizing based on guesses.The ball you need to keep your eye on here is the underlying

principle that wealth is what people want. If you plan to get

rich by creating wealth, you have to know what people want.

So few businesses really pay attention to making customers happy.

How often do you walk into a store, or call a company on the

phone, with a feeling of dread in the back of your mind?

When you hear "your call is important to us, please stay on

the line," do you think, oh good, now everything will be all right?A restaurant can afford to serve the occasional burnt dinner.

But in technology, you cook one thing and that's what everyone

eats. So any difference between what people want and what

you deliver is multiplied.

You please or annoy

customers wholesale. The closer you can get to what they want,

the more wealth you generate.Wealth and PowerMaking wealth is not the only way to get rich. For most of

human history it has not even been the most common. Until

a few centuries ago,

the main sources of wealth were mines, slaves and serfs,

land, and cattle,

and the only ways to acquire these rapidly were by inheritance,

marriage, conquest, or confiscation.

Naturally wealth had a bad reputation.Two things changed. The first was the rule of law. For most of the world's

history, if you did somehow accumulate a fortune, the ruler or his

henchmen

would find a way to steal it.

But in medieval Europe something new happened.

A new class of merchants and manufacturers

began to collect in towns.

[10]

Together they were able to withstand the local feudal

lord. So

for the first time in our history, the bullies stopped stealing the

nerds' lunch money.

This was naturally a great incentive,

and possibly indeed the main cause of the second big change,

industrialization.A great deal has been written about the causes of the Industrial

Revolution. But surely a necessary, if not sufficient, condition

was that people who made fortunes be able to enjoy them in peace.

[11]

One piece of evidence is what happened to countries

that tried to return to the old model, like the Soviet

Union, and to a lesser extent Britain under the labor

governments of the 1960s and early 1970s. Take away the incentive

of wealth, and technical innovation grinds to a halt.Remember what a startup is, economically:

a way of saying, I want to work faster. Instead of accumulating

money slowly by being paid a regular wage for fifty years, I

want to get it over with as soon as possible. So governments

that forbid you to accumulate wealth are in effect decreeing

that you work slowly. They're willing to let you earn $3 million over

fifty years, but they're not willing to let you work so hard that

you can do it in two. They are like

the corporate boss that you can't go to and say, I want to work

ten times as hard, so please pay me ten times a much.

Except this is not a boss you can escape by starting your own

company.The problem with working slowly is not just that technical

innovation happens slowly. It's that it tends not to happen at all.

It's only when you're deliberately looking for hard problems,

as a way to use speed to the greatest advantage, that you take

on this kind of project. Developing new technology is a

pain in the ass. It is, as Edison said, one percent

inspiration and ninety-nine percent perspiration.

Without the incentive of wealth, no one wants to do it.

Engineers will work on sexy projects like fighter planes and moon

rockets for ordinary salaries, but more mundane technologies

like light bulbs or semiconductors have to be developed by entrepreneurs.Startups

are not just something that happened in Silicon Valley in

the last couple decades. Since it became possible to

get rich by creating wealth, everyone who has done it has

used essentially the same recipe: measurement and leverage,

where measurement comes from working with a small

group, and leverage from developing new techniques.

The recipe was the same in Florence in 1200 as it is

in Santa Clara today.Understanding this may help to answer an important question:

why Europe grew so powerful.

Was it something about the geography of

Europe? Was it that Europeans are somehow racially superior?

Was it their religion? The answer (or at least

the proximate cause) may be that the

Europeans

rode on the crest of a powerful new idea: allowing those who

made a lot of money to keep it.Once you're allowed to do that,

people who want to get rich can do it by generating

wealth instead of stealing it.

The resulting technological growth translates not only

into wealth but into military power. The theory that led to

the stealth plane was developed by a Soviet mathematician.

But because the Soviet Union didn't have a computer industry,

it remained for them a theory;

they didn't have hardware capable of executing the calculations

fast enough to design an actual airplane.In that respect the Cold War teaches the same lesson as

World War II and, for that matter, most wars in recent history.

Don't let a ruling

class of warriors and politicians squash the entrepreneurs.

The same recipe that makes individuals rich

makes countries powerful. Let the nerds keep their lunch

money, and you rule the world.Notes[1]

One valuable thing you tend to get only in startups is

uninterruptability. Different kinds of

work have different time quanta. Someone proofreading a

manuscript

could probably be interrupted every fifteen minutes

with little loss of productivity. But the time quantum for

hacking is very long: it might take an hour just to load

a problem into your head. So the

cost of having someone from personnel

call you about a form you forgot to fill out can be huge.This is why hackers give you such a baleful stare as they

turn from their screen to answer your question. Inside

their heads a giant house of cards is tottering.The mere possibility of being interrupted deters hackers

from starting hard projects. This is why they

tend to work late at night, and why it's next to impossible

to write great software in a cubicle (except late at night).One great advantage of startups is that they don't yet have

any of the people who interrupt you. There is no personnel

department, and thus no form nor anyone to call you about it.[2]

Faced with the idea that people working for startups might be

20 or 30 times as productive as those working for large companies,

executives at large companies will naturally wonder, how could

I get the people working for me to do that? The answer is

simple: pay them to.Internally most companies are run like Communist states.

If you believe in free markets, why not turn your company into one?Hypothesis: A company will be maximally profitable when each

employee is paid in proportion to the wealth they generate.[3]

Until recently even governments sometimes didn't grasp the

distinction between money and wealth. Adam

Smith (Wealth of Nations, v:i) mentions several

that tried to preserve their

"wealth" by forbidding the export of gold or silver.

But having more of the medium of exchange would not make

a country richer; if you have more money chasing the same

amount of material wealth, the only result is higher prices.[4]

There are many senses of the word "wealth," not all of

them material. I'm not trying to make a deep philosophical

point here about which

is the true kind. I'm writing about one specific,

rather technical sense of the word "wealth." What

people will give you money for.

This is an interesting sort of wealth to study, because

it is the kind that prevents you from starving.

And what people will give you money for depends on them,

not you.When you're starting a business,

it's easy to slide into thinking that customers

want what you do. During the Internet Bubble I talked

to a woman who, because she liked the outdoors, was

starting an "outdoor portal." You know what

kind of business you should start if you like

the outdoors? One to recover data from crashed hard disks.What's the connection? None at all. Which is precisely my point.

If you want

to create wealth (in the narrow technical sense of not

starving) then you should be especially skeptical about any

plan that centers on things you like doing.

That is where your idea of what's valuable is least

likely to coincide with other people's.[5]

In the average car restoration you probably do make everyone

else microscopically poorer, by doing a small amount of damage to

the environment. While environmental costs should be taken

into account, they don't

make wealth a zero-sum game. For example, if you repair

a machine that's broken because a part has come unscrewed,

you create wealth with no environmental cost.[5b]

This essay was written before Firefox.[6]

Many people feel confused and depressed in

their early twenties. Life seemed so much more fun in college.

Well, of course it was. Don't be fooled by the surface similarities.

You've gone from guest to servant.

It's possible to have fun in this new world.

Among other things, you now get to go behind the doors that say

"authorized personnel only."

But the change is a shock at first, and all the worse

if you're not consciously aware of it.[7]

When VCs asked us how long it would take another startup

to duplicate our software, we used to reply that they probably

wouldn't be able to at all. I think this made us seem naive,

or liars.[8]

Few technologies have one clear inventor. So as

a rule, if you know the "inventor" of something

(the telephone, the assembly line, the airplane,

the light bulb, the transistor) it is because their

company made money from it, and the company's PR people worked

hard to spread the story. If you don't know who invented

something (the automobile, the television, the computer,

the jet engine, the laser), it's because other companies

made all the money.[9]

This is a good plan for life in general.

If you have two choices, choose the harder.

If you're trying to decide whether to go out running or

sit home and watch TV, go running.

Probably the reason this trick works so well is that

when you have two choices and one is harder, the

only reason you're even considering the other is laziness.

You know in the back of your mind what's the right thing

to do, and this trick merely forces you to acknowledge it.[10]

It is probably no accident that the middle class

first appeared in northern Italy and the low countries,

where there were no strong central governments. These two

regions were the richest of their time and became the twin

centers from which Renaissance civilization radiated.

If they no longer play that role, it is because

other places, like the United States, have been truer to the

principles they discovered.[11]

It may indeed be a sufficient condition. But if so, why didn't

the Industrial Revolution happen earlier? Two possible (and

not incompatible) answers: (a) It did.

The Industrial Revolution was one in a series.

(b) Because in medieval towns, monopolies

and guild regulations initially slowed the development of new means

of production.

Comment on this essay.Russian TranslationArabic TranslationSpanish Translation

You'll find this essay and 14 others in

Hackers & Painters.

The Word "Hacker"

April 2004To the popular press, "hacker" means someone who breaks

into computers. Among programmers it means a good programmer.

But the two meanings are connected. To programmers,

"hacker" connotes mastery in the most literal sense: someone

who can make a computer do what he wants—whether the computer

wants to or not.To add to the confusion, the noun "hack" also has two senses. It can

be either a compliment or an insult. It's called a hack when

you do something in an ugly way. But when you do something

so clever that you somehow beat the system, that's also

called a hack. The word is used more often in the former than

the latter sense, probably because ugly solutions are more

common than brilliant ones.Believe it or not, the two senses of "hack" are also

connected. Ugly and imaginative solutions have something in

common: they both break the rules. And there is a gradual

continuum between rule breaking that's merely ugly (using

duct tape to attach something to your bike) and rule breaking

that is brilliantly imaginative (discarding Euclidean space).Hacking predates computers. When he

was working on the Manhattan Project, Richard Feynman used to

amuse himself by breaking into safes containing secret documents.

This tradition continues today.

When we were in grad school, a hacker friend of mine who spent too much

time around MIT had

his own lock picking kit.

(He now runs a hedge fund, a not unrelated enterprise.)It is sometimes hard to explain to authorities why one would

want to do such things.

Another friend of mine once got in trouble with the government for

breaking into computers. This had only recently been declared

a crime, and the FBI found that their usual investigative

technique didn't work. Police investigation apparently begins with

a motive. The usual motives are few: drugs, money, sex,

revenge. Intellectual curiosity was not one of the motives on

the FBI's list. Indeed, the whole concept seemed foreign to

them.Those in authority tend to be annoyed by hackers'

general attitude of disobedience. But that disobedience is

a byproduct of the qualities that make them good programmers.

They may laugh at the CEO when he talks in generic corporate

newspeech, but they also laugh at someone who tells them

a certain problem can't be solved.

Suppress one, and you suppress the other.This attitude is sometimes affected. Sometimes young programmers

notice the eccentricities of eminent hackers and decide to

adopt some of their own in order to seem smarter.

The fake version is not merely

annoying; the prickly attitude of these posers

can actually slow the process of innovation.But even factoring in their annoying eccentricities,

the disobedient attitude of hackers is a net win. I wish its

advantages were better understood.For example, I suspect people in Hollywood are

simply mystified by

hackers' attitudes toward copyrights. They are a perennial

topic of heated discussion on Slashdot.

But why should people who program computers

be so concerned about copyrights, of all things?Partly because some companies use mechanisms to prevent

copying. Show any hacker a lock and his first thought is

how to pick it. But there is a deeper reason that

hackers are alarmed by measures like copyrights and patents.

They see increasingly aggressive measures to protect

"intellectual property"

as a threat to the intellectual

freedom they need to do their job.

And they are right.It is by poking about inside current technology that

hackers get ideas for the next generation. No thanks,

intellectual homeowners may say, we don't need any

outside help. But they're wrong.

The next generation of computer technology has

often—perhaps more often than not—been developed by outsiders.In 1977 there was no doubt some group within IBM developing

what they expected to be

the next generation of business computer. They were mistaken.

The next generation of business computer was

being developed on entirely different lines by two long-haired

guys called Steve in a garage in Los Altos. At about the

same time, the powers that be

were cooperating to develop the

official next generation operating system, Multics.

But two guys who thought Multics excessively complex went off

and wrote their own. They gave it a name that

was a joking reference to Multics: Unix.The latest intellectual property laws impose

unprecedented restrictions on the sort of poking around that

leads to new ideas. In the past, a competitor might use patents

to prevent you from selling a copy of something they

made, but they couldn't prevent you from

taking one apart to see how it worked. The latest

laws make this a crime. How are we

to develop new technology if we can't study current

technology to figure out how to improve it?Ironically, hackers have brought this on themselves.

Computers are responsible for the problem. The control systems

inside machines used to be physical: gears and levers and cams.

Increasingly, the brains (and thus the value) of products is

in software. And by this I mean software in the general sense:

i.e. data. A song on an LP is physically stamped into the

plastic. A song on an iPod's disk is merely stored on it.Data is by definition easy to copy. And the Internet

makes copies easy to distribute. So it is no wonder

companies are afraid. But, as so often happens, fear has

clouded their judgement. The government has responded

with draconian laws to protect intellectual property.

They probably mean well. But

they may not realize that such laws will do more harm

than good.Why are programmers so violently opposed to these laws?

If I were a legislator, I'd be interested in this

mystery—for the same reason that, if I were a farmer and suddenly

heard a lot of squawking coming from my hen house one night,

I'd want to go out and investigate. Hackers are not stupid,

and unanimity is very rare in this world.

So if they're all squawking,

perhaps there is something amiss.Could it be that such laws, though intended to protect America,

will actually harm it? Think about it. There is something

very American about Feynman breaking into safes during

the Manhattan Project. It's hard to imagine the authorities

having a sense of humor about such things over

in Germany at that time. Maybe it's not a coincidence.Hackers are unruly. That is the essence of hacking. And it

is also the essence of Americanness. It is no accident

that Silicon Valley

is in America, and not France, or Germany,

or England, or Japan. In those countries, people color inside

the lines.I lived for a while in Florence. But after I'd been there

a few months I realized that what I'd been unconsciously hoping

to find there was back in the place I'd just left.

The reason Florence is famous is that in 1450, it was New York.

In 1450 it was filled with the kind of turbulent and ambitious

people you find now in America. (So I went back to America.)It is greatly to America's advantage that it is

a congenial atmosphere for the right sort of unruliness—that

it is a home not just for the smart, but for smart-alecks.

And hackers are invariably smart-alecks. If we had a national

holiday, it would be April 1st. It says a great deal about

our work that we use the same word for a brilliant or a

horribly cheesy solution. When we cook one up we're not

always 100% sure which kind it is. But as long as it has

the right sort of wrongness, that's a promising sign.

It's odd that people

think of programming as precise and methodical. Computers

are precise and methodical. Hacking is something you do

with a gleeful laugh.In our world some of the most characteristic solutions

are not far removed from practical

jokes. IBM was no doubt rather surprised by the consequences

of the licensing deal for DOS, just as the hypothetical

"adversary" must be when Michael Rabin solves a problem by

redefining it as one that's easier to solve.Smart-alecks have to develop a keen sense of how much they

can get away with. And lately hackers

have sensed a change

in the atmosphere.

Lately hackerliness seems rather frowned upon.To hackers the recent contraction in civil liberties seems

especially ominous. That must also mystify outsiders.

Why should we care especially about civil

liberties? Why programmers, more than

dentists or salesmen or landscapers?Let me put the case in terms a government official would appreciate.

Civil liberties are not just an ornament, or a quaint

American tradition. Civil liberties make countries rich.

If you made a graph of

GNP per capita vs. civil liberties, you'd notice a definite

trend. Could civil liberties really be a cause, rather

than just an effect? I think so. I think a society in which

people can do and say what they want will also tend to

be one in which the most efficient solutions win, rather than

those sponsored by the most influential people.

Authoritarian countries become corrupt;

corrupt countries become poor; and poor countries are weak.

It seems to me there is

a Laffer curve for government power, just as for

tax revenues. At least, it seems likely enough that it

would be stupid to try the experiment and find out. Unlike

high tax rates, you can't repeal totalitarianism if it

turns out to be a mistake.This is why hackers worry. The government spying on people doesn't

literally make programmers write worse code. It just leads

eventually to a world in which bad ideas win. And because

this is so important to hackers, they're especially sensitive

to it. They can sense totalitarianism approaching from a

distance, as animals can sense an approaching

thunderstorm.It would be ironic if, as hackers fear, recent measures

intended to protect national security and intellectual property

turned out to be a missile aimed right at what makes

America successful. But it would not be the first time that

measures taken in an atmosphere of panic had

the opposite of the intended effect.There is such a thing as Americanness.

There's nothing like living abroad to teach you that.

And if you want to know whether something will nurture or squash

this quality, it would be hard to find a better focus

group than hackers, because they come closest of any group

I know to embodying it. Closer, probably, than

the men running our government,

who for all their talk of patriotism

remind me more of Richelieu or Mazarin

than Thomas Jefferson or George Washington.When you read what the founding fathers had to say for

themselves, they sound more like hackers.

"The spirit of resistance to government,"

Jefferson wrote, "is so valuable on certain occasions, that I wish

it always to be kept alive."Imagine an American president saying that today.

Like the remarks of an outspoken old grandmother, the sayings of

the founding fathers have embarrassed generations of

their less confident successors. They remind us where we come from.

They remind us that it is the people who break rules that are

the source of America's wealth and power.Those in a position to impose rules naturally want them to be

obeyed. But be careful what you ask for. You might get it.Thanks to Ken Anderson, Trevor Blackwell, Daniel Giffin,

Sarah Harlin, Shiro Kawai, Jessica Livingston, Matz,

Jackie McDonough, Robert Morris, Eric Raymond, Guido van Rossum,

David Weinberger, and

Steven Wolfram for reading drafts of this essay.

(The image shows Steves Jobs and Wozniak

with a "blue box."

Photo by Margret Wozniak. Reproduced by permission of Steve

Wozniak.)Portuguese TranslationHebrew TranslationRomanian Translation

You'll find this essay and 14 others in

Hackers & Painters.

What You Can't Say

January 2004

Have you ever seen an old photo of yourself and

been embarrassed at the way you looked? Did we actually

dress like that? We did. And we had no idea how

silly we looked.

It's the nature of fashion to be invisible, in the

same way the movement of the earth is invisible to all

of us riding on it.What scares me is that there are moral fashions too.

They're just as arbitrary, and just as invisible to most people.

But they're much more dangerous.

Fashion is mistaken for good design;

moral fashion is mistaken for good.

Dressing oddly gets you laughed at. Violating

moral fashions can get you fired, ostracized, imprisoned, or

even killed.If you could travel back in a time machine, one thing

would be true no matter where you went: you'd have to watch

what you said.

Opinions we consider harmless could have

gotten you in big trouble.

I've already said at least one thing that would have gotten me in big

trouble in most of Europe in the seventeenth century,

and did get Galileo in big trouble when he said

it that the earth moves. [1]

It seems to be a constant throughout history: In every

period, people believed things that were just ridiculous,

and believed them so strongly that you would have gotten in

terrible trouble for saying otherwise.Is our time any different?

To anyone who has read any amount of history, the answer is

almost certainly no. It would be a remarkable coincidence if ours

were the first era to get everything just right.It's tantalizing to think we believe

things that people in the future will find ridiculous.

What would someone coming back to visit us in a time machine

have to be careful not to say?

That's what I want to study here.

But

I want to do more than just shock everyone with

the heresy du jour. I want to find general

recipes for discovering what you can't say, in any era.The Conformist TestLet's start with a test:

Do you have any opinions that you would be reluctant to express

in front of a group of your peers?If the answer is no,

you might want to stop and think about that. If everything

you believe is something you're supposed to believe, could

that possibly be a coincidence? Odds are it isn't. Odds are

you just think what you're told.The other alternative would be that you independently considered

every question and came up with the exact same answers that

are now considered acceptable. That seems unlikely, because

you'd also have to make the same mistakes. Mapmakers

deliberately put slight mistakes in their maps so they can

tell when someone copies them. If another map has the same

mistake, that's very convincing evidence.Like every other era in history, our moral map almost certainly

contains a few mistakes. And anyone who makes the same mistakes

probably didn't do it by accident. It would be

like someone claiming they had independently decided in

1972 that bell-bottom jeans were a good idea.If you believe everything you're supposed to now, how can

you be sure you wouldn't also have believed everything you

were supposed to if you had grown up among the plantation

owners of the pre-Civil War South, or in Germany in the 1930s or

among the Mongols in 1200, for that matter? Odds are you

would have.Back in the era of terms like "well-adjusted," the idea

seemed to be that there was something wrong with

you if you thought things you didn't dare say out loud.

This seems backward. Almost certainly, there

is something wrong with you if you don't think things

you don't dare say out loud.TroubleWhat can't we say? One way to find these ideas is simply to look

at things people do say, and get in trouble for. [2]Of course, we're not just looking for things we can't say.

We're looking for things we can't say that are true, or at least

have enough chance of being true that the question

should remain open. But many of the

things people get in trouble for saying probably

do make it over this second, lower threshold. No one

gets in trouble for saying

that 2 + 2 is 5, or that people in Pittsburgh are ten feet tall.

Such obviously false statements might be treated as jokes, or

at worst as evidence of insanity, but they are not likely to

make anyone mad. The statements that make people mad are

the ones they worry might be believed.

I suspect the statements that make people maddest

are those they worry might be true.If Galileo had said that people in Padua were ten feet tall,

he would have been regarded as a harmless eccentric. Saying

the earth orbited the sun was another matter. The church knew

this would set people thinking.Certainly, as we look back on the past, this rule of thumb works

well. A lot of the statements people got in trouble for seem

harmless now. So it's likely that visitors from the

future would agree with at least some of the statements that

get people in trouble today. Do we have no Galileos? Not

likely.To find them,

keep track of opinions that get

people in trouble, and start asking, could this be true?

Ok, it may be heretical (or whatever modern equivalent), but

might it also be true?HeresyThis won't get us all the answers, though. What if no one

happens to have gotten in trouble for a particular idea yet?

What if some idea would be so radioactively controversial that

no one would dare express it in public? How can we find these too?Another approach is to follow that word, heresy. In every period

of history, there seem to have been labels that got applied to

statements to shoot them down before anyone had a chance to ask

if they were true or not. "Blasphemy", "sacrilege", and "heresy"

were such

labels for a good part of western history, as in more recent times

"indecent", "improper", and "unamerican" have been. By now these

labels have lost their sting. They always do.

By now they're mostly used ironically.

But in their time,

they had real force.The word "defeatist", for example, has no particular political

connotations now.

But in Germany in 1917 it was a weapon, used by Ludendorff in

a purge of those who favored a negotiated peace.

At the start of World War II it was used

extensively by Churchill and his supporters to silence their

opponents.

In 1940, any argument against Churchill's aggressive policy was "defeatist".

Was it right or wrong? Ideally, no one got far enough to ask

that.

We have such labels today, of course, quite a lot of them,

from the all-purpose "inappropriate" to the dreaded "divisive."

In any period, it should be easy to figure out what such labels are,

simply by looking at what people call ideas they disagree

with besides untrue. When a politician says his opponent is

mistaken, that's a straightforward criticism, but when he

attacks a statement as "divisive" or "racially insensitive"

instead of arguing that it's false, we should start paying

attention.So another way to figure out which of our taboos future generations

will laugh at is to start with the

labels. Take a label "sexist", for example and try to think

of some ideas that would be called that. Then for each ask, might

this be true?Just start listing ideas at random? Yes, because they

won't really be random. The ideas that come to mind first

will be the most plausible ones. They'll be things you've already

noticed but didn't let yourself think.In 1989 some clever researchers tracked

the eye movements of radiologists as they scanned chest images for

signs of lung cancer. [3] They found that even when the radiologists

missed a cancerous lesion, their eyes had usually paused at the site of it.

Part of their brain knew there was something there; it just

didn't percolate all the way up into conscious knowledge.

I think many interesting heretical thoughts are already mostly

formed in our minds. If we turn off our self-censorship

temporarily, those will be the first to emerge.Time and SpaceIf we could look into the future it would be obvious which

of our taboos they'd laugh at.

We can't do that, but we can do something almost as good: we can

look into the past. Another way to figure out what we're

getting wrong is to look at what used to be acceptable

and is now unthinkable.Changes between the past and the present sometimes do represent

progress. In a field like physics,

if we disagree with past generations it's because we're

right and they're wrong. But this becomes rapidly less true as

you move away from the certainty of the hard sciences. By the time

you get to social questions, many changes are just fashion.

The age of consent fluctuates like hemlines.We may imagine that we are a great deal smarter and more virtuous than

past generations, but the more history you read, the less likely

this seems. People in past times were much like us. Not heroes,

not barbarians. Whatever their ideas were, they were ideas

reasonable people could believe.So here is another source of interesting heresies. Diff present

ideas against those of various past cultures, and see what you

get. [4]

Some will be

shocking by present standards. Ok, fine; but which might also be true?You don't have to look into the past to find big differences.

In our own time, different societies have wildly varying ideas

of what's ok and what isn't.

So you can try diffing other cultures' ideas against ours as well.

(The best way to do that is to visit them.)

Any idea that's considered harmless in a significant

percentage of times and places, and yet is taboo in ours,

is a candidate for something we're mistaken

about.For example, at the high water mark of political correctness

in the early 1990s, Harvard distributed to its

faculty and staff a brochure saying, among other things, that it

was inappropriate to compliment a colleague or student's

clothes. No more "nice shirt."

I think this principle is rare among the world's cultures, past or present.

There are probably more where it's considered especially

polite to compliment someone's clothing than where it's considered

improper.

Odds are this is, in a mild form, an example of one of

the taboos a visitor from the future would

have to be careful to avoid if he happened to set his time machine for

Cambridge, Massachusetts, 1992. [5]PrigsOf course, if they have time machines in the future they'll

probably have a separate reference manual just for Cambridge.

This has always been a fussy place, a town of i dotters and

t crossers, where you're liable to get both your grammar and

your ideas corrected in the same conversation. And that

suggests another way to find taboos. Look for prigs,

and see what's inside their heads.Kids' heads are repositories of all our taboos.

It seems fitting to us that kids' ideas should be bright and clean.

The picture we give them of the world is

not merely simplified, to suit their developing minds,

but sanitized as well, to suit our

ideas of what kids ought to think. [6]You can see this on a small scale in the matter of

dirty words. A lot of my friends are starting to have children

now, and they're all trying

not to use words like

"fuck" and "shit" within baby's hearing, lest baby start using

these words too.

But these

words are part of the language, and adults use them all the

time. So parents are giving their kids an inaccurate idea of

the language by not using

them. Why do they do this? Because they don't think it's

fitting that kids should use the whole language. We like

children to seem innocent. [7]Most adults, likewise, deliberately give kids a misleading

view of the world.

One of the most obvious

examples is Santa Claus. We think it's cute for little kids to

believe in Santa Claus. I myself think it's cute for little

kids to believe in Santa Claus. But one wonders, do we tell

them this stuff for their sake, or for ours?I'm not arguing for or against this idea here. It is probably

inevitable that parents should want to dress up their kids'

minds in cute little baby outfits. I'll probably do it myself.

The important thing for our purposes is that, as a result,

a well brought-up teenage kid's brain is a more

or less complete collection of all our taboos and in mint

condition, because they're untainted by experience.

Whatever we think that will later turn out to be ridiculous,

it's almost certainly inside that head.How do we get at these ideas? By the following thought experiment.

Imagine a kind of latter-day Conrad character

who has worked for a time as a mercenary in Africa, for a time

as a doctor in Nepal, for a time as the manager of a

nightclub in Miami. The specifics don't matter just

someone who has

seen a lot. Now imagine comparing what's inside this guy's head

with what's inside the head

of a well-behaved sixteen year old girl from

the suburbs. What does he think that

would shock her?

He knows the world; she knows, or at least embodies, present

taboos. Subtract one from the other, and the result is what

we can't say.

MechanismI can think of one more way to figure out what we can't

say: to look at how taboos are created. How do moral

fashions arise, and why are they adopted?

If we can understand this mechanism, we

may be able to see it at work in our own time.Moral fashions don't seem to be created the way ordinary

fashions are. Ordinary fashions seem to arise by accident when

everyone imitates the whim of some influential person.

The fashion for broad-toed shoes in

late fifteenth century Europe began because Charles VIII of

France had six toes on one foot. The fashion for the

name Gary began when the actor Frank Cooper adopted the name

of a tough mill town in Indiana. Moral fashions more often

seem to be created deliberately. When there's something we

can't say, it's often because some group doesn't want us to.The prohibition will be strongest when the group is nervous.

The irony of Galileo's situation was that he got in trouble

for repeating Copernicus's ideas. Copernicus himself didn't.

In fact, Copernicus was a canon of a cathedral, and dedicated his

book to the pope. But by Galileo's time the church was in

the throes of the Counter-Reformation and was much more

worried about unorthodox ideas.To launch a taboo, a group has to be poised halfway between

weakness and power. A confident group doesn't need taboos

to protect it. It's not considered improper to

make disparaging remarks about Americans, or the English.

And yet a group has to be powerful enough to enforce a

taboo. Coprophiles, as of this writing, don't seem to be

numerous or energetic enough to have had their

interests promoted to a lifestyle.I suspect the biggest source of moral taboos will turn out to

be power struggles in which one side only barely has

the upper hand. That's where you'll find a group

powerful enough to enforce taboos, but weak enough to need them.Most struggles, whatever they're really about, will be cast

as struggles between competing ideas.

The English Reformation was at bottom a struggle for wealth and power,

but it ended up being

cast as a struggle to preserve the souls

of Englishmen from the corrupting influence of Rome.

It's easier to get people to fight for an idea.

And whichever side wins, their

ideas will also be considered to have triumphed, as if God

wanted to signal his agreement by selecting that side as the victor.We often like to think of World War II as a triumph

of freedom over totalitarianism. We conveniently forget that

the Soviet Union was also one of the winners.I'm not saying that struggles are never about ideas,

just that they will always be made to seem to be about

ideas, whether they are or not. And just as there is nothing

so unfashionable as the last, discarded fashion, there is

nothing so wrong as the principles of the most recently

defeated opponent.

Representational art is only now

recovering from the approval of both Hitler and Stalin. [8]Although moral fashions tend to arise from different sources

than fashions in clothing, the mechanism of their adoption seems

much the same. The early adopters will be driven by ambition:

self-consciously cool people who want to distinguish themselves

from the common herd. As the fashion becomes established they'll

be joined by a second, much larger group, driven by fear. [9] This

second group adopt the fashion not because they want to stand

out but because they are afraid of standing out.So if you want to figure out what we can't say, look at the

machinery of fashion and try to predict what it would make

unsayable. What groups are powerful but nervous, and what

ideas would they like to suppress? What ideas were tarnished by

association when they ended up on the losing side of a recent

struggle? If a self-consciously cool person wanted to differentiate

himself from preceding fashions (e.g. from his parents),

which of their ideas would he tend to reject?

What are conventional-minded people afraid of saying?This technique won't find us all the things we can't say.

I can think of some that aren't the result of

any recent struggle. Many of our taboos are rooted

deep in the past. But this approach, combined with the

preceding four, will turn up a good number of unthinkable

ideas.WhySome would ask, why would one want to do this? Why deliberately

go poking around among nasty, disreputable ideas? Why look

under rocks?I do it, first of all, for the same reason I did look under

rocks as a kid: plain curiosity. And I'm especially curious about

anything that's forbidden. Let me see and decide for myself.Second, I do it because I don't like the idea of being mistaken.

If, like other eras, we believe things that will later seem ridiculous,

I want to know what they are so that I, at least, can avoid

believing them.Third, I do it because it's good for the brain. To do good work

you need a brain that can go anywhere. And you especially need a

brain that's in the habit of going where it's not supposed to.Great work tends to grow out of ideas

that others have overlooked, and no idea is so overlooked as one that's

unthinkable.

Natural selection, for example.

It's so simple. Why didn't anyone think of it before? Well,

that is all too obvious. Darwin himself was careful to tiptoe

around the implications of his theory. He wanted to spend his

time thinking about biology, not arguing with people who accused

him of being an atheist.In the sciences, especially, it's a great advantage to be able to

question assumptions.

The m.o. of scientists, or at least of the

good ones, is precisely that: look for places where

conventional wisdom is broken, and then try to pry apart the

cracks and see what's underneath. That's where new theories come

from.A good scientist, in other words, does not merely ignore

conventional wisdom, but makes a special effort to break it.

Scientists go looking for trouble.

This should be the m.o. of any scholar, but

scientists seem much more willing to look under rocks. [10]Why? It could

be that the scientists are simply smarter; most physicists could,

if necessary, make it through a PhD program in French literature,

but few professors of French literature could make it through

a PhD program in physics. Or it could be because it's clearer

in the sciences whether theories are true or false, and this

makes scientists bolder. (Or it could be that, because it's

clearer in the sciences whether theories are true or false, you

have to be smart to get jobs as a scientist, rather than just a

good politician.)Whatever the reason, there seems a clear correlation between

intelligence and willingness to consider shocking ideas.

This isn't just because smart people actively work to find holes in

conventional thinking. I think conventions also have

less hold over them to start with.

You can see that in the

way they dress.It's not only in the sciences that heresy pays off.

In any competitive field, you can

win big by seeing things that others daren't.

And in every

field there are probably heresies few dare utter. Within

the US car industry there is a lot of hand-wringing now

about declining market share.

Yet the cause is so obvious that any observant outsider could

explain it in a second: they make bad cars. And they have for

so long that by now the US car brands are antibrands something

you'd buy a car despite, not because of. Cadillac stopped

being the Cadillac of cars in about 1970. And yet I suspect

no one dares say this. [11] Otherwise these companies would have

tried to fix the problem.Training yourself to think unthinkable thoughts has advantages

beyond the thoughts themselves. It's like stretching.

When you stretch before running, you put your body into positions

much more extreme

than any it will assume during the run.

If you can think things

so outside the box that they'd make people's hair stand on end,

you'll have no trouble with the small trips outside the box that

people call innovative.Pensieri StrettiWhen you find something you can't say, what do you do with it?

My advice is, don't say it. Or at least, pick your battles.Suppose in the future there is a movement to ban

the color yellow. Proposals to paint anything yellow are

denounced as "yellowist", as is anyone suspected of liking the

color. People who like orange are tolerated but viewed with

suspicion. Suppose you realize there is nothing

wrong with yellow. If you go around saying this, you'll be

denounced as a yellowist too, and you'll find yourself having a

lot of arguments with anti-yellowists.

If your aim in life is to rehabilitate the color yellow, that may

be what you want.

But if you're mostly interested in

other questions, being labelled as a yellowist will just be

a distraction. Argue with idiots, and you become an idiot.The most important thing is to be able to think what you

want, not to say what you want. And if you feel you have to

say everything you think, it may inhibit you from thinking

improper thoughts. I think it's better to follow the opposite

policy. Draw a sharp line between your thoughts and your

speech. Inside your head, anything is allowed.

Within my head I make a point of encouraging the most outrageous

thoughts I can imagine.

But, as in

a secret society, nothing that happens within the building

should be told to outsiders. The first rule of Fight

Club is, you do not talk about Fight Club.When Milton was going to visit Italy in the 1630s,

Sir Henry Wootton, who had been ambassador to Venice, told him

his motto should be

"i pensieri stretti & il viso sciolto." Closed thoughts

and an open face. Smile at everyone, and don't tell them

what you're thinking. This was wise advice.

Milton was an argumentative fellow, and the Inquisition

was a bit restive at that time. But I think the difference

between Milton's situation and ours is only a matter of

degree.

Every era has its heresies, and if you don't get imprisoned for them you

will at least get in enough trouble that it becomes a complete

distraction.I admit it seems cowardly to keep quiet.

When I read about the harassment to which

the Scientologists subject their critics [12], or that pro-Israel groups

are "compiling dossiers" on those who speak out against Israeli

human rights abuses [13], or about people being sued for

violating the DMCA [14], part of me wants

to say, "All right, you bastards, bring it on."

The problem is, there are so many things you can't say.

If you said them all you'd

have no time left for your real work.

You'd have to turn into Noam Chomsky. [15]The trouble with keeping your thoughts secret, though,

is that you lose the advantages of discussion. Talking

about an idea leads to more ideas.

So the optimal plan, if you can manage it,

is to have a few trusted

friends you can speak openly to. This is not just a

way to develop ideas; it's also a good

rule of thumb for choosing friends. The people

you can say heretical things to without getting jumped on

are also the most interesting to know.Viso Sciolto?I don't think we need

the viso sciolto so much as the pensieri stretti.

Perhaps the best policy is to make it plain that you don't

agree with whatever zealotry is current in your time, but

not to be too specific about what you disagree with. Zealots

will try to draw you out, but you don't have to answer them.

If they try to force you to treat a question on their

terms by asking "are you with us or against us?" you can

always just answer "neither".Better still, answer "I haven't decided."

That's what Larry Summers

did when a group tried to put

him in this position. Explaining himself later, he said

"I don't do litmus tests." [16]

A lot of the

questions people get hot about are actually quite complicated.

There is no prize for getting the answer quickly.If the anti-yellowists seem to be getting out of hand and

you want to fight back, there are ways

to do it without getting yourself accused of being a

yellowist. Like skirmishers in

an ancient army, you want to avoid directly engaging the

main body of the enemy's troops. Better to harass them

with arrows from a distance.One way to do this is to ratchet the debate up one level of

abstraction.

If you argue against censorship in general, you can avoid being

accused of whatever heresy is contained

in the book or film that someone is trying to censor.

You can attack labels with meta-labels: labels that refer

to the use of labels to prevent discussion.

The spread of the term "political correctness" meant the beginning of

the end of political correctness, because it enabled one to

attack the phenomenon as a whole without being accused of any

of the specific heresies it sought to suppress.Another way to counterattack is with metaphor. Arthur Miller

undermined the House Un-American Activities Committee

by writing a play, "The Crucible," about the Salem witch trials.

He never referred directly to the committee and so gave them

no way to reply.

What could HUAC do, defend the Salem witch trials? And yet

Miller's metaphor stuck so well that to this day the activities

of the committee are often described as a "witch-hunt."Best of all, probably, is humor. Zealots, whatever their

cause, invariably lack a sense of humor.

They can't reply in kind to jokes.

They're as unhappy on the territory of

humor as a mounted knight on a skating rink.

Victorian prudishness, for example, seems to have been defeated

mainly by treating it as a joke. Likewise its reincarnation as

political correctness.

"I am glad that I

managed to write 'The Crucible,'" Arthur Miller wrote,

"but looking back I have often wished I'd

had the temperament to do an absurd comedy, which is what the

situation deserved." [17]ABQA Dutch friend says

I should use Holland as an example of a tolerant society.

It's true they have a long tradition of

comparative open-mindedness. For centuries the low countries were the place

to go to say things you couldn't say anywhere else,

and this helped to make the region a center of scholarship and industry

(which have been closely tied for longer than most people realize).

Descartes, though claimed by the French, did much of his thinking in

Holland.And yet, I wonder. The Dutch seem to live their lives up to their

necks in rules and regulations. There's so much you can't do there;

is there really nothing

you can't say?Certainly the fact that they value open-mindedness is no guarantee.

Who thinks they're not open-minded? Our hypothetical prim miss from

the suburbs thinks she's open-minded. Hasn't she been

taught to be? Ask anyone, and they'll say the same thing: they're

pretty open-minded, though they draw the line at things that are really

wrong. (Some tribes

may avoid "wrong" as

judgemental, and may instead use a more neutral sounding euphemism

like "negative" or "destructive".)When people are bad at math, they know it, because they get the

wrong answers on tests. But when people are bad at open-mindedness

they don't know it. In fact they tend to think the opposite.

Remember, it's the nature of fashion to be invisible. It wouldn't

work otherwise. Fashion doesn't

seem like fashion to someone in the grip of it. It just seems like

the right thing to do. It's only by looking from a distance that

we see oscillations in people's idea of the right thing to do, and

can identify them as fashions.Time gives us such distance for free. Indeed, the arrival of new

fashions makes old fashions easy to see, because they

seem so ridiculous by contrast. From one end of a pendulum's

swing, the other end seems especially far away.To see fashion in your own time, though, requires a conscious effort.

Without time to give you distance, you have to create distance yourself.

Instead of being part of the mob, stand

as far away from it as you can and watch what it's

doing. And pay especially close attention whenever an idea is being

suppressed. Web filters for children and employees often ban

sites containing pornography, violence, and hate speech. What

counts as pornography and violence? And what, exactly, is

"hate speech?" This sounds like a phrase out of 1984.Labels like that are probably the biggest external clue.

If a statement is false,

that's the worst thing you can say about it. You don't

need to say that it's heretical. And if it isn't false, it

shouldn't be suppressed. So when you see statements being

attacked as x-ist or y-ic (substitute your current values of

x and y), whether in 1630 or 2030, that's a sure sign that

something is wrong. When you hear such labels being used,

ask why.Especially if you hear yourself using them. It's not just

the mob you need to learn to watch from a distance. You need to be

able to watch your own thoughts from a distance. That's not

a radical idea, by the way; it's the main difference between

children and adults. When a child gets angry because he's

tired, he doesn't know what's happening. An adult can

distance himself enough from the

situation to say "never mind, I'm just tired." I don't

see why one couldn't, by a similar process, learn to

recognize and discount the effects of moral fashions.You have to take that extra step if you want to think clearly.

But it's harder, because now you're working against social customs

instead of with them. Everyone encourages you to grow up to the

point where you can discount your own bad moods. Few encourage

you to continue to the point where you can discount society's bad

moods.How can you see the wave, when you're the water? Always be

questioning. That's the only defence. What can't you say? And why?NotesThanks to Sarah Harlin, Trevor Blackwell, Jessica Livingston,

Robert Morris, Eric Raymond and Bob van der Zwaan for reading drafts of this

essay, and to Lisa Randall, Jackie McDonough, Ryan Stanley and Joel Rainey

for conversations about heresy.

Needless to say they bear no blame for opinions

expressed in it, and especially for opinions not

expressed in it.Re: What You Can't SayLabelsJapanese TranslationFrench TranslationGerman TranslationDutch TranslationRomanian TranslationHebrew TranslationTurkish TranslationChinese TranslationButtonsA Civic Duty to AnnoyThe Perils of ObedienceAliens Cause Global WarmingHays CodeStratagem 32Conspiracy TheoriesMark Twain: Corn-pone OpinionsA Blacklist for "Excuse Makers"What You Can't Say Will Hurt You

Filters that Fight Back

August 2003

We may be able to improve the accuracy of Bayesian spam filters

by having them follow links to see what's

waiting at the other end. Richard Jowsey of

death2spam now does

this in borderline cases, and reports that it works well.Why only do it in borderline cases? And why only do it once?As I mentioned in Will Filters Kill Spam?,

following all the urls in

a spam would have an amusing side-effect. If popular email clients

did this in order to filter spam, the spammer's servers

would take a serious pounding. The more I think about this,

the better an idea it seems. This isn't just amusing; it

would be hard to imagine a more perfectly targeted counterattack

on spammers.So I'd like to suggest an additional feature to those

working on spam filters: a "punish" mode which,

if turned on, would spider every url

in a suspected spam n times, where n could be set by the user. [1]As many people have noted, one of the problems with the

current email system is that it's too passive. It does

whatever you tell it. So far all the suggestions for fixing

the problem seem to involve new protocols. This one

wouldn't.If widely used, auto-retrieving spam filters would make

the email system rebound. The huge volume of the

spam, which has so far worked in the spammer's favor,

would now work against him, like a branch snapping back in

his face. Auto-retrieving spam filters would drive the

spammer's

costs up,

and his sales down: his bandwidth usage

would go through the roof, and his servers would grind to a

halt under the load, which would make them unavailable

to the people who would have responded to the spam.Pump out a million emails an hour, get a

million hits an hour on your servers.

We would want to ensure that this is only done to

suspected spams. As a rule, any url sent to millions of

people is likely to be a spam url, so submitting every http

request in every email would work fine nearly all the time.

But there are a few cases where this isn't true: the urls

at the bottom of mails sent from free email services like

Yahoo Mail and Hotmail, for example.To protect such sites, and to prevent abuse, auto-retrieval

should be combined with blacklists of spamvertised sites.

Only sites on a blacklist would get crawled, and

sites would be blacklisted

only after being inspected by humans. The lifetime of a spam

must be several hours at least, so

it should be easy to update such a list in time to

interfere with a spam promoting a new site. [2]High-volume auto-retrieval would only be practical for users

on high-bandwidth

connections, but there are enough of those to cause spammers

serious trouble. Indeed, this solution neatly

mirrors the problem. The problem with spam is that in

order to reach a few gullible people the spammer sends

mail to everyone. The non-gullible recipients

are merely collateral damage. But the non-gullible majority

won't stop getting spam until they can stop (or threaten to

stop) the gullible

from responding to it. Auto-retrieving spam filters offer

them a way to do this.Would that kill spam? Not quite. The biggest spammers

could probably protect their servers against auto-retrieving

filters. However, the easiest and cheapest way for them

to do it would be to include working unsubscribe links in

their mails. And this would be a necessity for smaller fry,

and for "legitimate" sites that hired spammers to promote

them. So if auto-retrieving filters became widespread,

they'd become auto-unsubscribing filters.In this scenario, spam would, like OS crashes, viruses, and

popups, become one of those plagues that only afflict people

who don't bother to use the right software.

Notes[1] Auto-retrieving filters will have to follow redirects,

and should in some cases (e.g. a page that just says

"click here") follow more than one level of links.

Make sure too that

the http requests are indistinguishable from those of

popular Web browsers, including the order and referrer.If the response

doesn't come back within x amount of time, default to

some fairly high spam probability.Instead of making n constant, it might be a good idea to

make it a function of the number of spams that have been

seen mentioning the site. This would add a further level of

protection against abuse and accidents.[2] The original version of this article used the term

"whitelist" instead of "blacklist". Though they were

to work like blacklists, I preferred to call them whitelists

because it might make them less vulnerable to legal attack.

This just seems to have confused readers, though.There should probably be multiple blacklists. A single point

of failure would be vulnerable both to attack and abuse.

Thanks to Brian Burton, Bill Yerazunis, Dan Giffin,

Eric Raymond, and Richard Jowsey for reading drafts of this.FFB FAQJapanese TranslationA Perl FFBLycos DDoS@Home

Hackers and Painters

May 2003(This essay is derived from a guest lecture at Harvard, which incorporated

an earlier talk at Northeastern.)When I finished grad school in computer science I went

to art school to study painting. A lot of people seemed surprised

that someone interested in computers would also be interested in painting.

They seemed to think that

hacking and painting were very different kinds of work-- that

hacking was cold, precise, and methodical, and that

painting was the frenzied expression of some primal urge.Both of these images are wrong. Hacking and painting have a

lot in common. In fact, of all the different types of people I've

known, hackers and painters are among the most alike.What hackers and painters have in common is that they're

both makers. Along with composers, architects, and writers,

what hackers and painters are trying to do is make good things.

They're not doing research per se, though if in the course of

trying to make good things they discover some new technique,

so much the better.I've never liked the term "computer science." The main

reason I don't like it is that there's no such thing.

Computer science is a

grab bag of tenuously related areas thrown together

by an accident of history, like Yugoslavia.

At one end you have people who are really mathematicians,

but call what they're doing computer science so they can get DARPA grants.

In the middle you have people working on

something like the natural history of computers-- studying the

behavior of algorithms for routing data through

networks, for example. And then at the other extreme you

have the hackers, who are trying to

write interesting software, and for whom computers are just a

medium of expression, as concrete is for architects or

paint for painters. It's as if

mathematicians, physicists, and architects all had to be in

the same department.Sometimes what the hackers do is called "software engineering,"

but this term is just as misleading.

Good software designers are no more engineers than architects are.

The border between architecture and engineering is not sharply

defined, but it's there.

It falls between what and how: architects decide what to do,

and engineers figure out how to do it.What and how should not be kept too separate. You're

asking for trouble if you try to decide what to do without

understanding how to do it.

But hacking can certainly be more than just deciding how to

implement some spec. At its best, it's creating the spec-- though

it turns out the best way to do that is to implement it.Perhaps one day

"computer science" will, like Yugoslavia, get broken up into its

component parts. That might be a good thing. Especially if it

meant independence for my native land, hacking.Bundling all these different types of work together in one

department may be convenient administratively, but it's confusing

intellectually. That's the other reason I don't like the name

"computer science." Arguably the people in the middle are doing

something like an experimental science. But the people at either

end, the hackers and the mathematicians, are not actually doing science.The mathematicians don't seem bothered by this. They happily

set to work proving theorems like the other mathematicians

over in the math department, and probably soon stop noticing

that the building they work in says ``computer science'' on the

outside. But for the hackers this label is a problem.

If what they're doing is called science, it makes them feel they

ought to be acting scientific.

So instead of doing what they really want to do, which is

to design beautiful software, hackers in universities and

research labs feel they ought to be writing research papers.In the best case, the papers are just a formality. Hackers write

cool software, and then write a paper about it, and the paper

becomes a proxy for the achievement represented by the software.

But often this mismatch causes problems. It's easy to

drift away from building beautiful things toward building ugly

things that make more suitable subjects for research papers.Unfortunately, beautiful things don't always make the

best subjects for papers.

Number one, research must be original-- and

as anyone who has written a PhD dissertation knows, the way to

be sure that you're exploring virgin territory is to to stake

out a piece of ground that no one wants. Number two, research must be

substantial-- and awkward systems yield meatier papers,

because you can write about the obstacles you have to overcome

in order to get things done. Nothing yields meaty problems like

starting with the wrong assumptions. Most of AI is an example

of this rule; if you assume that knowledge can be represented

as a list of predicate logic expressions whose arguments represent

abstract concepts, you'll have a lot of

papers to write about how to make this work. As Ricky Ricardo

used to say, "Lucy, you got a lot of explaining to do."The way to create something beautiful is often to make subtle

tweaks to something that already exists, or to combine existing

ideas in a slightly new way. This kind of work is hard to

convey in a research paper.So why do universities and research labs continue to judge

hackers by publications?

For the same reason that "scholastic aptitude"

gets measured by simple-minded standardized tests, or

the productivity of programmers gets measured in lines of code.

These tests

are easy to apply, and there is nothing so tempting as an easy test

that kind of works.Measuring what hackers are actually trying to do, designing

beautiful software, would be much more difficult. You need

a good sense of design to judge

good design. And

there is no correlation, except possibly

a negative

one, between people's ability to recognize good

design and their confidence that they can.The only external test is time. Over time, beautiful

things tend to thrive, and ugly

things tend to get discarded. Unfortunately, the amounts of time

involved can be longer than human lifetimes. Samuel Johnson

said it took a hundred years for a writer's reputation to

converge. You have to wait for the writer's

influential friends to die, and then for all their followers

to die.I think hackers just have to resign themselves to having a large random

component in their reputations. In this they are no different

from other makers. In fact, they're lucky by comparison.

The influence of fashion is not nearly so great in hacking as it

is in painting.There are worse things than having people misunderstand your

work. A worse danger is that you

will yourself misunderstand your work. Related fields are

where you go looking for ideas. If you find yourself in the computer science

department, there is a natural temptation to believe, for example,

that hacking is the applied version of what theoretical computer

science is the theory of. All

the time I was in graduate school I had an uncomfortable feeling

in the back of my mind that I ought to know more theory,

and that it was very remiss of me to have forgotten all that

stuff within three weeks of the final exam.Now I realize I was

mistaken. Hackers need to understand the theory of computation

about as much as painters need to understand paint chemistry.

You need to know how to calculate time and

space complexity and about

Turing completeness. You might also want to remember at

least the concept of a state machine, in case you have to write

a parser or a regular expression library. Painters in fact

have to remember a good deal more about paint chemistry than

that.I've found that the best sources of ideas

are not the other fields that have the word "computer" in

their names, but the other fields inhabited by makers.

Painting has been a much richer source of ideas than the

theory of computation.For example, I was taught in college

that one ought to figure out a program

completely on paper

before even going near a computer. I found that I did not

program this way. I found that I liked to program

sitting in front of a computer, not a piece of paper. Worse

still, instead of patiently writing out a complete program

and assuring myself it was correct, I tended to just spew

out code that was hopelessly broken, and gradually beat it into

shape. Debugging, I was taught, was a kind of final pass where

you caught typos and oversights. The way I worked, it

seemed like programming consisted of debugging.For a long time I felt bad about this, just as I once

felt bad that I didn't hold my pencil the way they taught me

to in elementary school.

If I had only looked over at

the other makers, the painters or the architects, I would

have realized that there was a name for what I was doing:

sketching. As far as I can tell, the

way they taught me to program in college was all wrong.

You should figure out programs as you're writing them,

just as writers and painters and architects do.Realizing this has real implications for software design.

It means that a programming language should, above all, be

malleable. A programming language is for

thinking of

programs, not for expressing programs you've already thought

of. It should be a pencil, not a pen. Static typing would

be a fine idea if people actually did write programs the way

they taught me to in college. But that's not how any of the

hackers I know write programs. We need a language that lets us

scribble and smudge and smear, not a language where you have

to sit with a teacup of types balanced on your knee and make

polite conversation with a strict old aunt of a compiler.While we're on the subject of static typing, identifying with

the makers will save us from another problem that afflicts

the sciences: math envy. Everyone in the sciences

secretly believes that mathematicians are smarter than they are.

I think mathematicians also believe this. At any rate,

the result is that scientists tend to make their

work look as mathematical as possible. In a field like

physics this probably doesn't do much harm, but the further you

get from the natural sciences, the more of a problem it

becomes.A page of formulas just looks so impressive.

(Tip: for extra impressiveness, use Greek variables.) And

so there is a great temptation to work on problems you

can treat formally, rather than problems that are, say,

important.If hackers identified with other makers, like writers and

painters, they wouldn't feel tempted to do

this. Writers and painters don't suffer from math envy.

They feel as if they're doing something completely unrelated.

So are hackers, I think.If universities and research labs keep hackers from doing

the kind of work they want to do,

perhaps the place for them is in companies.

Unfortunately, most companies won't let hackers do what they

want either. Universities and research labs force hackers

to be scientists, and companies force them to be engineers.I only discovered this myself quite recently. When Yahoo bought

Viaweb, they asked me what I wanted to do. I had never

liked the business side very much, and said that I just wanted to

hack. When I got to Yahoo, I found that what hacking meant

to them was implementing software, not designing it. Programmers

were seen as technicians who translated the visions (if

that is the word) of product managers into code.This seems to be the

default plan in big companies. They do it because

it decreases the standard deviation of the outcome.

Only a small percentage of hackers can actually design software,

and it's hard for the

people running a company to pick these out. So instead of

entrusting the future of the software to

one brilliant hacker, most companies set things up so that it is

designed by committee, and the hackers merely

implement the design.If you want to make money at some point, remember this,

because this is one of the reasons startups win. Big companies want

to decrease the standard deviation of design outcomes because they

want to avoid disasters. But when you damp oscillations, you

lose the high points as well as the low. This is not a problem for

big companies, because they don't win by making great

products. Big companies win by sucking less than other big companies.So if you can figure out a way to get in a

design war with a company big enough that its software is

designed by product managers, they'll never be able to keep up

with you. These opportunities are not easy to find, though.

It's hard to engage a big company in a design war,

just as it's hard to engage an opponent inside a castle in hand

to hand combat. It would be pretty easy to write a better

word processor than Microsoft Word, for example, but Microsoft,

within the castle of their operating system monopoly,

probably wouldn't even notice if you did.The place to fight design wars is in new markets, where no one

has yet managed to establish any fortifications. That's where

you can win big by taking the bold approach to design, and

having the same people both design and implement the product.

Microsoft themselves did this at the start. So did Apple.

And Hewlett-Packard. I suspect almost every successful startup

has.So one way to build great software is to start your own

startup. There are two problems with this, though. One is

that in a startup you have to do so much besides write software.

At Viaweb I considered myself lucky if I

got to hack a quarter of the time. And the things I had to

do the other three quarters of the time ranged from tedious

to terrifying. I have a benchmark for this, because I

once had to leave a board meeting to have

some cavities filled. I remember sitting back in the

dentist's chair, waiting for the drill, and feeling like

I was on vacation.The other problem with startups is that there is not much

overlap between the kind of software that makes money and the

kind that's interesting to write. Programming languages

are interesting to write, and Microsoft's first product was

one, in fact, but no one will pay for programming languages

now. If you want to make money, you tend to be forced to work

on problems that are too nasty for anyone to solve for free.All makers face this problem. Prices are

determined by supply and demand, and there is just not as much

demand for things that are fun to work on as there is for

things that solve the mundane problems of individual customers.

Acting in off-Broadway plays just doesn't pay as well as

wearing a gorilla suit in someone's booth at a

trade show. Writing novels doesn't pay as well as writing

ad copy for garbage disposals.

And hacking programming languages doesn't pay as well

as figuring out how to connect some company's

legacy database to their Web server.I think the answer to this problem, in the case of software,

is a concept known to nearly all makers: the day job.

This phrase began with musicians, who

perform at night. More generally, it means that you have one

kind of work you do for money, and another for love.Nearly all makers have day jobs early in their careers.

Painters and writers notoriously do. If you're lucky

you can get a day job that's closely

related to your real work. Musicians often

seem to work in record stores. A hacker working on some

programming language or operating system might likewise be able to

get a day job using it. [1]When I say that the answer is for hackers to have day jobs,

and work on beautiful software on the side, I'm not proposing

this as a new idea. This is what open-source hacking is all

about. What I'm saying is that open-source is probably the right

model, because it has been independently confirmed by all the

other makers.It seems surprising to me that any employer would be reluctant

to let hackers work on open-source projects.

At Viaweb, we would have been reluctant to hire anyone

who didn't. When we interviewed

programmers, the main

thing we cared about was what kind of software they

wrote in their spare time.

You can't do anything really well unless

you love it, and if you love to hack you'll inevitably

be working on projects of your own. [2]Because hackers are makers rather than scientists,

the right place to look for metaphors is not in the

sciences, but among other kinds of makers. What else can painting

teach us about hacking?One thing we can learn, or at least confirm, from the

example of painting is how to learn to hack. You learn to

paint mostly by doing it.

Ditto for hacking. Most hackers don't learn to hack by

taking college courses in programming. They learn to hack

by writing programs of their own at age thirteen. Even in

college classes, you learn to hack mostly by hacking. [3]Because painters leave a trail of work behind them, you

can watch them learn by doing. If you look at the work

of a painter in chronological order, you'll find that each

painting builds on things that have been learned in previous

ones. When there's something in

a painting that works very well, you can usually find version

1 of it in a smaller form in some earlier painting.I think most makers work this way. Writers and architects seem

to as well. Maybe it would be good for hackers

to act more like painters, and regularly start over from scratch,

instead of continuing to work for years on one project, and

trying to incorporate all their later ideas as revisions.The fact that hackers learn to hack by doing it is another

sign of how different hacking is from the sciences. Scientists

don't learn science by doing it, but by doing labs and problem sets.

Scientists start out doing work that's perfect, in the sense

that they're just trying to reproduce work someone else has

already done for them.

Eventually, they get

to the point where they can do original work.

Whereas hackers, from the start, are doing original work; it's

just very bad. So hackers start original, and get good, and

scientists start good, and get original.

The other way makers learn is from examples.

For a painter, a museum is a reference library of techniques.

For hundreds of years it has been part of the traditional

education of painters to copy the works of the great masters,

because copying forces you to look closely

at the way a painting is made.Writers do this too.

Benjamin Franklin learned to write by summarizing the points

in the essays of Addison and Steele and then trying to

reproduce them. Raymond Chandler did the same thing

with detective stories.Hackers, likewise, can learn to program by looking at

good programs-- not just at what they do, but the source

code too. One of the less publicized benefits

of the open-source movement is that it has made it easier

to learn to program. When I learned to program, we had to rely

mostly on examples in books. The one big chunk of

code available then was Unix, but even this was not

open source. Most of the people who read the source

read it in illicit photocopies of John Lions' book, which

though written in 1977 was not allowed to be published

until 1996.Another example we can take from painting is the way that

paintings are created by gradual refinement. Paintings usually

begin with a sketch.

Gradually the details get filled in.

But it is not merely a process of filling in. Sometimes

the original plans turn out to be mistaken.

Countless paintings,

when you look at them in xrays, turn out to have limbs that

have been moved or facial features that have been readjusted.Here's a case where we can learn from painting. I think hacking

should work this way too. It's unrealistic

to expect that the specifications for a program will be

perfect. You're

better off if you admit this up front, and write programs in

a way that allows specifications to change on the fly.(The structure of large companies makes this hard for them

to do, so here is another place where startups have an advantage.)Everyone by now presumably knows about the danger of premature

optimization. I think we should be just as worried about

premature design-- deciding too early what

a program should do.The right tools can help us avoid

this danger.

A good programming language should, like oil paint, make it

easy to change your mind. Dynamic typing is a win here because

you don't have to

commit to specific data representations up front.

But the key to flexibility, I think, is to make the language

very abstract.

The easiest program to change is one that's very short.

This sounds like a paradox, but a great painting

has to be better than it has to be.

For example, when Leonardo

painted the portrait of Ginevra de Benci

in the National Gallery, he put a juniper bush behind her head.

In it he carefully

painted each individual leaf. Many painters might have thought,

this is just something to put in the background to frame

her head. No one will look that closely at it.Not Leonardo. How hard he worked on part of a painting didn't

depend at all on how closely he expected anyone to look at it.

He was like Michael Jordan. Relentless.Relentlessness wins because, in the aggregate, unseen details

become visible.

When people walk by the portrait of Ginevra de Benci,

their attention is often immediately arrested by it,

even before they look at the label and notice that it says

Leonardo da Vinci. All those unseen details combine to produce

something that's just stunning, like a thousand barely audible

voices all singing in tune.Great software, likewise, requires a fanatical devotion to

beauty. If you look inside good software, you find that

parts no one is ever supposed to see are beautiful too.

I'm not claiming I write great software, but I

know that when it comes to code I behave in a way that would

make me eligible for prescription drugs if I approached everyday

life the same way.

It drives me crazy to see code that's badly indented,

or that uses ugly variable names.If a hacker were a mere implementor, turning a spec into code, then

he could just work his way through it from one end to the other like

someone digging a ditch. But if the hacker is a creator, we have

to take inspiration into account.In hacking, like painting,

work comes in cycles. Sometimes you get excited about some

new project and you want to work sixteen hours a day on it.

Other times nothing seems interesting.To do good work you have to take these cycles into

account, because they're affected by how you react to them.

When you're driving a

car with a manual transmission on a hill, you have to back off

the clutch sometimes to avoid stalling. Backing

off can likewise prevent ambition from stalling.

In both painting and hacking there are some

tasks that are terrifyingly ambitious, and others that are

comfortingly routine. It's a good idea to save some easy

tasks for moments when you would otherwise stall.In hacking, this can literally mean saving up bugs.

I like debugging: it's the

one time that hacking is as straightforward as

people think it is. You have a

totally constrained problem, and all you have to do is solve

it. Your program is supposed to do x. Instead it does y.

Where does it go wrong? You know you're going to win

in the end. It's as relaxing as painting a wall.The example of painting can teach us not only how to manage our

own work, but how to work together. A lot of the

great art of the past is the work of multiple hands, though

there may only be one name on the wall next to it in the

museum. Leonardo was an apprentice in the workshop of

Verrocchio and painted one of the angels in his Baptism of

Christ. This sort of thing was the rule, not the exception.

Michelangelo was considered especially dedicated for insisting

on painting all the figures on the ceiling of the Sistine

Chapel himself.As far as I know, when painters worked together on a painting,

they never worked on the same parts. It was common

for the master to paint the principal figures and for assistants

to paint the others and the background. But you never had

one guy painting over the work of another.I think this is the right model for collaboration in software

too. Don't push it too far. When a piece of code is

being hacked by three or four different people, no one of whom

really owns it, it will end up being like a common-room. It will

tend to feel bleak and abandoned, and accumulate cruft.

The right

way to collaborate, I think, is to divide projects into sharply

defined modules, each with a definite owner, and with interfaces

between them that are as carefully designed and, if possible,

as articulated as programming languages.Like painting, most software is intended for

a human audience. And so hackers, like painters, must have

empathy to do really great work. You have to be able to see

things from the user's point of view.When I was a kid I was always being told to look at things from

someone else's point of view. What this always meant in

practice was to do what someone else wanted, instead of what

I wanted. This of course gave empathy a bad name, and I made a

point of not cultivating it.Boy, was I wrong. It turns out that looking at things from

other people's point of view is practically the secret of

success. It doesn't necessarily mean being self-sacrificing.

Far from it. Understanding how someone else sees things

doesn't imply that you'll act in his interest; in some

situations-- in war, for example-- you want to do exactly

the opposite. [4]Most makers make things for a human audience.

And to engage an audience you have to understand what they need.

Nearly all the greatest paintings are paintings of people,

for example, because people are what people are interested in.Empathy is probably the single most important difference

between a good hacker and a great one. Some hackers

are quite smart, but when it comes to empathy are

practically solipsists. It's hard for such

people to design great software [5], because they can't

see things from the user's point of view.One way to tell how good people are at empathy is to watch

them explain a technical question to someone without a technical

background. We probably all know people who, though otherwise smart,

are just comically bad at this. If someone asks them at

a dinner party what a programming language is, they'll

say something like ``Oh, a high-level language is what

the compiler uses as input to generate object code.''

High-level language? Compiler? Object code? Someone who

doesn't know what a programming language is obviously doesn't

know what these things are, either.Part of what software has to do is explain itself. So to

write good software you have to understand how little users

understand.

They're going to walk up to the software with no preparation, and

it had better do what they guess it will, because they're

not going to read the manual. The best system I've ever seen

in this respect was the original Macintosh, in 1985.

It did what software almost never does: it just worked. [6]Source code, too, should explain itself. If I could get people to

remember just one quote about programming, it would be the

one at the beginning of Structure and Interpretation of Computer

Programs.

Programs should be written for people to read, and

only incidentally for machines to execute.

You need to have

empathy not just for your users, but for your readers. It's in

your interest, because you'll be one of them.

Many a hacker has written a program only to

find on returning to it six months later that he has no idea

how it works. I know several people who've sworn off Perl after

such experiences. [7]Lack of empathy is associated with intelligence, to the point

that there is even something of a fashion for it in some places.

But I don't think there's any correlation.

You can do well in math and

the natural sciences without having to learn empathy, and people in these

fields tend to be smart, so the two qualities have come to be

associated. But there are plenty of dumb people who are bad at

empathy too. Just listen to the people who call in with questions on

talk shows. They ask whatever it is they're asking in

such a roundabout way

that the hosts often have to rephrase the question for them.So, if hacking works like painting and writing, is it as cool?

After all, you only get one life.

You might as well spend it working on something great.Unfortunately, the question is hard to answer. There is always

a big time lag in prestige. It's like light from a distant star.

Painting has prestige now because of great work people did five hundred

years ago. At the time, no one thought

these paintings were as important as we do today. It would have

seemed very odd to people at the time that Federico da Montefeltro,

the Duke of Urbino, would one day be known mostly as the guy

with the strange nose in a painting

by Piero della Francesca.So while I admit that hacking doesn't seem as cool as painting now,

we should remember that painting itself didn't seem as cool in

its glory days as it does now.What we can say with some confidence is that these are the glory

days of hacking. In most fields the great work is done early on.

The paintings made between 1430 and 1500 are still unsurpassed.

Shakespeare appeared just as professional theater was being born,

and pushed the medium

so far that every playwright since has had to live in his shadow.

Albrecht Durer did the same thing with engraving, and Jane Austen

with the novel.Over and over we see the same pattern. A new medium appears, and

people are so excited about it that they explore most of its

possibilities in the first couple generations. Hacking seems

to be in this phase now.Painting was not, in Leonardo's time, as cool as his work

helped make it.

How cool hacking turns out to be will depend on what we can

do with this new medium.

Notes[1] The greatest damage that photography has done

to painting may be the fact that it killed the best day job.

Most of the great painters in history supported

themselves by painting portraits. [2] I've been told that Microsoft discourages

employees from contributing to open-source projects, even in

their spare time.

But so many of the best hackers work on open-source

projects now that the main effect of this policy may be

to ensure that they won't be able to hire any first-rate

programmers.[3] What you learn about programming in college is much like

what you learn about books or clothes or dating: what bad taste you

had in high school.[4] Here's an example of applied empathy.

At Viaweb, if we couldn't decide between two alternatives, we'd

ask, what would our competitors hate most? At one point a

competitor added a feature to their software that was basically

useless, but since it was one of few they had that we didn't, they

made much of it in the trade press.

We could have tried to explain that the feature was useless,

but we decided it would annoy our competitor more if we

just implemented it ourselves, so we hacked together our own

version that afternoon.[5] Except text editors and compilers. Hackers don't need empathy to

design these, because they are themselves typical users.[6] Well, almost. They overshot the available RAM somewhat,

causing much inconvenient disk swapping, but this could be fixed

within a few months by buying an additional disk drive.[7] The way to make programs easy to read is not to

stuff them with comments. I would take Abelson and Sussman's

quote a step further. Programming languages should be designed

to express algorithms, and only incidentally to tell computers

how to execute them. A good programming language

ought to be better for explaining software than English.

You should only

need comments when there is some kind of kludge you need to warn

readers about, just as on a road there are only

arrows on parts with unexpectedly sharp curves.

Thanks to Trevor Blackwell, Robert Morris, Dan Giffin, and Lisa

Randall for reading drafts of this, and to Henry Leitner

and Larry Finkelstein for inviting me to speak.Japanese TranslationSpanish TranslationGerman TranslationPortuguese TranslationCzech TranslationWhy Good Design Comes from Bad DesignKnuth: Computer Programming as an Art

You'll find this essay and 14 others in

Hackers & Painters.

If Lisp is So Great

May 2003If Lisp is so great, why don't more people use it? I was

asked this question by a student in the audience at a

talk I gave recently. Not for the first time, either.In languages, as in so many things, there's not much

correlation between popularity and quality. Why does

John Grisham (King of Torts sales rank, 44) outsell

Jane Austen (Pride and Prejudice sales rank, 6191)?

Would even Grisham claim that it's because he's a better

writer?Here's the first sentence of Pride and Prejudice:

It is a truth universally acknowledged, that a single man

in possession of a good fortune must be in want of a

wife.

"It is a truth universally acknowledged?" Long words for

the first sentence of a love story.Like Jane Austen, Lisp looks hard. Its syntax, or lack

of syntax, makes it look completely unlike

the languages

most people are used to. Before I learned Lisp, I was afraid

of it too. I recently came across a notebook from 1983

in which I'd written:

I suppose I should learn Lisp, but it seems so foreign.

Fortunately, I was 19 at the time and not too resistant to learning

new things. I was so ignorant that learning

almost anything meant learning new things.People frightened by Lisp make up other reasons for not

using it. The standard

excuse, back when C was the default language, was that Lisp

was too slow. Now that Lisp dialects are among

the faster

languages available, that excuse has gone away.

Now the standard excuse is openly circular: that other languages

are more popular.(Beware of such reasoning. It gets you Windows.)Popularity is always self-perpetuating, but it's especially

so in programming languages. More libraries

get written for popular languages, which makes them still

more popular. Programs often have to work with existing programs,

and this is easier if they're written in the same language,

so languages spread from program to program like a virus.

And managers prefer popular languages, because they give them

more leverage over developers, who can more easily be replaced.Indeed, if programming languages were all more or less equivalent,

there would be little justification for using any but the most

popular. But they aren't all equivalent, not by a long

shot. And that's why less popular languages, like Jane Austen's

novels, continue to survive at all. When everyone else is reading

the latest John Grisham novel, there will always be a few people

reading Jane Austen instead.Japanese TranslationRomanian TranslationSpanish Translation

The Hundred-Year Language

April 2003(This essay is derived from a keynote talk at PyCon 2003.)It's hard to predict what

life will be like in a hundred years. There are only a few

things we can say with certainty. We know that everyone will

drive flying cars,

that zoning laws will be relaxed to allow buildings

hundreds of stories tall, that it will be dark most of the

time, and that women will all be trained in the martial arts.

Here I want to zoom in on one detail of this

picture. What kind of programming language will they use to

write the software controlling those flying cars?This is worth thinking about not so

much because we'll actually get to use these languages as because,

if we're lucky, we'll use languages on the path from this

point to that.I think that, like species, languages will form evolutionary trees,

with dead-ends branching off all over. We can see this

happening already.

Cobol, for all its sometime popularity, does not seem to have any

intellectual descendants. It is an evolutionary dead-end-- a

Neanderthal language.I predict a similar fate for Java. People

sometimes send me mail saying, "How can you say that Java

won't turn out to be a successful language? It's already

a successful language." And I admit that it is, if you

measure success by shelf space taken up by books on it

(particularly individual books on it), or by

the number of undergrads who believe they have to

learn it to get a job. When I say Java won't

turn out to be a successful language, I mean something more

specific: that Java

will turn out to be an evolutionary dead-end, like Cobol.This is just a guess. I may be wrong. My point here is not to dis Java,

but to raise the issue of evolutionary

trees and get people asking, where on the tree is language X?

The reason to ask this question isn't just so that

our ghosts can say, in a

hundred years, I told you so. It's because staying close to

the main branches is a useful heuristic for finding languages that will

be good to program in now.At any given time, you're probably happiest on

the main branches of an evolutionary tree.

Even when there were still plenty of Neanderthals,

it must have sucked to be one. The

Cro-Magnons would have been constantly coming over and

beating you up and stealing your food.The reason I want to

know what languages will be like in a hundred years is so that

I know what branch of the tree to bet on now.The evolution of languages differs from the evolution of species

because branches can converge. The Fortran branch, for example,

seems to be merging with the descendants

of Algol. In theory this is possible for species too, but it's

not likely to have happened to any bigger than a cell.Convergence

is more likely for languages partly because the space of

possibilities is smaller, and partly because mutations

are not random. Language designers deliberately incorporate

ideas from other languages.It's especially useful for language designers to think

about where the evolution of programming languages is likely

to lead, because they can steer accordingly.

In that case, "stay on a main branch" becomes more than a

way to choose a good language.

It becomes a heuristic for making the right decisions about

language design.Any programming language can be divided into

two parts: some set of fundamental operators that play the role

of axioms, and the rest of the language, which could in principle

be written in terms of these fundamental operators.I think the fundamental operators are the most important factor in a

language's long term survival. The rest you can change. It's

like the rule that in buying a house you should consider

location first of all. Everything else you can fix later, but you

can't fix the location.I think it's important not just that the axioms be well chosen,

but that there be few of them. Mathematicians have always felt

this way about axioms-- the fewer, the better-- and I think they're

onto something.At the very least, it has to be a useful exercise to look closely

at the core of a language to see if there are any axioms that

could be weeded out. I've found in my long career as a slob that

cruft breeds cruft, and I've seen this happen in software as

well as under beds and in the corners of rooms.I have a hunch that

the main branches of the evolutionary tree pass through the languages

that have the smallest, cleanest cores.

The more of a language you can write in itself,

the better.Of course, I'm making a big assumption in even asking what

programming languages will be like in a hundred years.

Will we even be writing programs in a hundred years? Won't

we just tell computers what we want them to do?There hasn't been a lot of progress in that department

so far.

My guess is that a hundred years from now people will

still tell computers what to do using programs we would recognize

as such. There may be tasks that we

solve now by writing programs and which in a hundred years

you won't have to write programs to solve, but I think

there will still be a good deal of

programming of the type that we do today.It may seem presumptuous to think anyone can predict what

any technology will look like in a hundred years. But

remember that we already have almost fifty years of history behind us.

Looking forward a hundred years is a graspable idea

when we consider how slowly languages have evolved in the

past fifty.Languages evolve slowly because they're not really technologies.

Languages are notation. A program is a formal description of

the problem you want a computer to solve for you. So the rate

of evolution in programming languages is more like the

rate of evolution in mathematical notation than, say,

transportation or communications.

Mathematical notation does evolve, but not with the giant

leaps you see in technology.Whatever computers are made of in a hundred years, it seems

safe to predict they will be much faster than

they are now. If Moore's Law continues to put out, they will be 74

quintillion (73,786,976,294,838,206,464) times faster. That's kind of

hard to imagine. And indeed, the most likely prediction in the

speed department may be that Moore's Law will stop working.

Anything that is supposed to double every eighteen months seems

likely to run up against some kind of fundamental limit eventually.

But I have no trouble believing that computers will be very much

faster. Even if they only end up being a paltry million

times faster, that should change the ground rules for programming

languages substantially. Among other things, there

will be more room for what

would now be considered slow languages, meaning languages

that don't yield very efficient code.And yet some applications will still demand speed.

Some of the problems we want to solve with

computers are created by computers; for example, the

rate at which you have to process video images depends

on the rate at which another computer can

generate them. And there is another class of problems

which inherently have an unlimited capacity to soak up cycles:

image rendering, cryptography, simulations.If some applications can be increasingly inefficient while

others continue to demand all the speed the hardware can

deliver, faster computers will mean that languages have

to cover an ever wider range of efficiencies. We've seen

this happening already. Current implementations of some

popular new languages are shockingly wasteful by the

standards of previous decades.This isn't just something that happens with programming

languages. It's a general historical trend. As technologies improve,

each generation can do things that the previous generation

would have considered wasteful. People thirty years ago would

be astonished at how casually we make long distance phone calls.

People a hundred years ago would be even more astonished that

a package would one day travel from Boston to New York via Memphis.I can already tell you what's going to happen to all those extra

cycles that faster hardware is going to give us in the

next hundred years. They're nearly all going to be wasted.I learned to program when computer power was scarce.

I can remember taking all the spaces out of my Basic programs

so they would fit into the memory of a 4K TRS-80. The

thought of all this stupendously inefficient software

burning up cycles doing the same thing over and over seems

kind of gross to me. But I think my intuitions here are wrong. I'm

like someone who grew up poor, and can't bear to spend money

even for something important, like going to the doctor.Some kinds of waste really are disgusting. SUVs, for example, would

arguably be gross even if they ran on a fuel which would never

run out and generated no pollution. SUVs are gross because they're

the solution to a gross problem. (How to make minivans look more

masculine.)

But not all waste is bad. Now that we have the infrastructure

to support it, counting the minutes of your long-distance

calls starts to seem niggling. If you have the

resources, it's more elegant to think of all phone calls as

one kind of thing, no matter where the other person is.There's good waste, and bad waste. I'm interested

in good waste-- the kind where, by spending more, we can get

simpler designs. How will we take advantage of the opportunities

to waste cycles that we'll get from new, faster hardware?The desire for speed is so deeply engrained in us, with

our puny computers, that it will take a conscious effort

to overcome it. In language design, we should be consciously seeking out

situations where we can trade efficiency for even the

smallest increase in convenience.Most data structures exist because of speed. For example,

many languages today have both strings and lists. Semantically, strings

are more or less a subset of lists in which the elements are

characters. So why do you need a separate data type?

You don't, really. Strings only

exist for efficiency. But it's lame to clutter up the semantics

of the language with hacks to make programs run faster.

Having strings in a language seems to be a case of

premature optimization.If we think of the core of a language as a set of axioms,

surely it's gross to have additional axioms that add no expressive

power, simply for the sake of efficiency. Efficiency is

important, but I don't think that's the right way to get it.The right way to solve that problem, I think, is to separate

the meaning of a program from the implementation details.

Instead of having both lists and strings, have just lists,

with some way to give the compiler optimization advice that

will allow it to lay out strings as contiguous bytes if

necessary.Since speed doesn't matter in most of a program, you won't

ordinarily need to bother with

this sort of micromanagement.

This will be more and more true as computers get faster.Saying less about implementation should also make programs

more flexible.

Specifications change while a program is being written, and this is not

only inevitable, but desirable.The word "essay" comes

from the French verb "essayer", which means "to try".

An essay, in the original sense, is something you

write to try to figure something out. This happens in

software too. I think some of the best programs were essays,

in the sense that the authors didn't know when they started

exactly what they were trying to write.Lisp hackers already know about the value of being flexible

with data structures. We tend to write the first version of

a program so that it does everything with lists. These

initial versions can be so shockingly inefficient that it

takes a conscious effort not to think about what they're

doing, just as, for me at least, eating a steak requires a

conscious effort not to think where it came from.What programmers in a hundred years will be looking for, most of

all, is a language where you can throw together an unbelievably

inefficient version 1 of a program with the least possible

effort. At least, that's how we'd describe it in present-day

terms. What they'll say is that they want a language that's

easy to program in.Inefficient software isn't gross. What's gross is a language

that makes programmers do needless work. Wasting programmer time

is the true inefficiency, not wasting machine time. This will

become ever more clear as computers get faster.I think getting rid of strings is already something we

could bear to think about. We did it in Arc, and it seems

to be a win; some operations that would be awkward to

describe as regular expressions can be described

easily as recursive functions.How far will this flattening of data structures go? I can think

of possibilities that shock even me, with my conscientiously broadened

mind. Will we get rid of arrays, for example? After all, they're

just a subset of hash tables where the keys are vectors of

integers. Will we replace hash tables themselves with lists?There are more shocking prospects even than that. The Lisp

that McCarthy described in 1960, for example, didn't

have numbers. Logically, you don't need to have a separate notion

of numbers, because you can represent them as lists: the integer

n could be represented as a list of n elements. You can do math this

way. It's just unbearably inefficient.No one actually proposed implementing numbers as lists in

practice. In fact, McCarthy's 1960 paper was not, at the time,

intended to be implemented at all. It was a theoretical exercise,

an attempt to create a more elegant alternative to the Turing

Machine. When someone did, unexpectedly, take this paper and

translate it into a working Lisp interpreter, numbers certainly

weren't represented as lists; they were represented in binary,

as in every other language.Could a programming language go so far as to get rid of numbers

as a fundamental data type? I ask this not so much as a serious

question as as a way to play chicken with the future. It's like

the hypothetical case of an irresistible force meeting an

immovable object-- here, an unimaginably inefficient

implementation meeting unimaginably great resources.

I don't see why not. The future is pretty long. If there's

something we can do to decrease the number of axioms in the core

language, that would seem to be the side to bet on as t approaches

infinity. If the idea still seems unbearable in a hundred years,

maybe it won't in a thousand.Just to be clear about this, I'm not proposing that all numerical

calculations would actually be carried out using lists. I'm proposing

that the core language, prior to any additional notations about

implementation, be defined this way. In practice any program

that wanted to do any amount of math would probably represent

numbers in binary, but this would be an optimization, not part of

the core language semantics.Another way to burn up cycles is to have many layers of

software between the application and the hardware. This too is

a trend we see happening already: many recent languages are

compiled into byte code. Bill Woods once told me that,

as a rule of thumb, each layer of interpretation costs a

factor of 10 in speed. This extra cost buys you flexibility.The very first version of Arc was an extreme case of this sort

of multi-level slowness, with corresponding benefits. It

was a classic "metacircular" interpreter written

on top of Common Lisp, with a definite family resemblance

to the eval function defined in McCarthy's original Lisp paper.

The whole thing was only a couple hundred lines of

code, so it was very easy to understand and change. The

Common Lisp we used, CLisp, itself runs on top

of a byte code interpreter. So here we had two levels of

interpretation, one of them (the top one) shockingly inefficient,

and the language was usable. Barely usable, I admit, but

usable.Writing software as multiple layers is a powerful technique

even within applications. Bottom-up programming means writing

a program as a series of layers, each of which serves as a

language for the one above. This approach tends to yield

smaller, more flexible programs. It's also the best route to

that holy grail, reusability. A language is by definition

reusable. The more

of your application you can push down into a language for writing

that type of application, the more of your software will be

reusable.Somehow the idea of reusability got attached

to object-oriented programming in the 1980s, and no amount of

evidence to the contrary seems to be able to shake it free. But

although some object-oriented software is reusable, what makes

it reusable is its bottom-upness, not its object-orientedness.

Consider libraries: they're reusable because they're language,

whether they're written in an object-oriented style or not.I don't predict the demise of object-oriented programming, by the

way. Though I don't think it has much to offer good programmers,

except in certain specialized domains, it is irresistible to

large organizations. Object-oriented programming

offers a sustainable way to write spaghetti code. It lets you accrete

programs as a series of patches.

Large organizations

always tend to develop software this way, and I expect this

to be as true in a hundred years as it is today.

As long as we're talking about the future, we had better

talk about parallel computation, because that's where this

idea seems to live. That is, no matter when you're talking, parallel

computation seems to be something that is going to happen

in the future.Will the future ever catch up with it? People have been

talking about parallel computation as something imminent

for at least 20

years, and it hasn't affected programming practice much so far.

Or hasn't it? Already

chip designers have to think about it, and so must

people trying to write systems software on multi-cpu computers.The real question is, how far up the ladder of abstraction will

parallelism go?

In a hundred years will it affect even application programmers? Or

will it be something that compiler writers think about, but

which is usually invisible in the source code of applications?One thing that does seem likely is that most opportunities for

parallelism will be wasted. This is a special case of my more

general prediction that most of the extra computer power we're

given will go to waste. I expect that, as with the stupendous

speed of the underlying hardware, parallelism will be something

that is available if you ask for it explicitly, but ordinarily

not used. This implies that the kind of parallelism we have in

a hundred years will not, except in special applications, be

massive parallelism. I expect for

ordinary programmers it will be more like being able to fork off

processes that all end up running in parallel.And this will, like asking for specific implementations of data

structures, be something that you do fairly late in the life of a

program, when you try to optimize it. Version 1s will ordinarily

ignore any advantages to be got from parallel computation, just

as they will ignore advantages to be got from specific representations

of data.Except in special kinds of applications, parallelism won't

pervade the programs that are written in a hundred years. It would be

premature optimization if it did.How many programming languages will there

be in a hundred years? There seem to be a huge number of new

programming languages lately. Part of the reason is that

faster hardware has allowed programmers to make different

tradeoffs between speed and convenience, depending on the

application. If this is a real trend, the hardware we'll

have in a hundred years should only increase it.And yet there may be only a few widely-used languages in a

hundred years. Part of the reason I say this

is optimism: it seems that, if you did a really good job,

you could make a language that was ideal for writing a

slow version 1, and yet with the right optimization advice

to the compiler, would also yield very fast code when necessary.

So, since I'm optimistic, I'm going to predict that despite

the huge gap they'll have between acceptable and maximal

efficiency, programmers in a hundred years will have languages

that can span most of it.As this gap widens, profilers will become increasingly important.

Little attention is paid to profiling now. Many people still

seem to believe that the way to get fast applications is to

write compilers that generate fast code. As the gap between

acceptable and maximal performance widens, it will become

increasingly clear that the way to get fast applications is

to have a good guide from one to the other.When I say there may only be a few languages, I'm not including

domain-specific "little languages". I think such embedded languages

are a great idea, and I expect them to proliferate. But I expect

them to be written as thin enough skins that users can see

the general-purpose language underneath.Who will design the languages of the future? One of the most exciting

trends in the last ten years has been the rise of open-source

languages like Perl, Python, and Ruby.

Language design is being taken over by hackers. The results

so far are messy, but encouraging. There are some stunningly

novel ideas in Perl, for example. Many are stunningly bad, but

that's always true of ambitious efforts. At its current rate

of mutation, God knows what Perl might evolve into in a hundred

years.It's not true that those who can't do, teach (some of the best

hackers I know are professors), but it is true that there are a

lot of things that those who teach can't do. Research imposes

constraining caste restrictions. In any academic

field there are topics that are ok to work on and others that

aren't. Unfortunately the distinction between acceptable and

forbidden topics is usually based on how intellectual

the work sounds when described in research papers, rather than

how important it is for getting good results. The extreme case

is probably literature; people studying literature rarely

say anything that would be of the slightest use to those

producing it.Though the situation is better in the sciences,

the overlap between the kind of work you're allowed to do and the

kind of work that yields good languages is distressingly small.

(Olin Shivers has grumbled eloquently

about this.) For example, types seem to be an inexhaustible source

of research papers, despite the fact that static typing

seems to preclude true macros-- without which, in my opinion, no

language is worth using.The trend is not merely toward languages being developed

as open-source projects rather than "research", but toward

languages being designed by the application programmers who need

to use them, rather than by compiler writers. This seems a good

trend and I expect it to continue.

Unlike physics in a hundred years, which is almost necessarily

impossible to predict, I think it may be possible in principle

to design a language now that would appeal to users in a hundred

years.One way to design a language is to just write down the program

you'd like to be able to write, regardless of whether there

is a compiler that can translate it or hardware that can run it.

When you do this you can assume unlimited resources. It seems

like we ought to be able to imagine unlimited resources as well

today as in a hundred years.What program would one like to write? Whatever is least work.

Except not quite: whatever would be least work if your ideas about

programming weren't already influenced by the languages you're

currently used to. Such influence can be so pervasive that

it takes a great effort to overcome it. You'd think it would

be obvious to creatures as lazy as us how to express a program

with the least effort. In fact, our ideas about what's possible

tend to be so limited by whatever language we think in that

easier formulations of programs seem very surprising. They're

something you have to discover, not something you naturally

sink into.One helpful trick here

is to use the length of the program as an approximation for

how much work it is to write. Not the length in characters,

of course, but the length in distinct syntactic elements-- basically,

the size of the parse tree. It may not be quite true that

the shortest program is the least work to write, but it's

close enough that you're better off aiming for the solid

target of brevity than the fuzzy, nearby one of least work.

Then the algorithm for language design becomes: look at a program

and ask, is there any way to write this that's shorter?In practice, writing programs in an imaginary hundred-year

language will work to varying degrees depending

on how close you are to the core. Sort routines you can

write now. But it would be

hard to predict now what kinds of libraries might be needed in

a hundred years. Presumably many libraries will be for domains that

don't even exist yet. If SETI@home works, for example, we'll

need libraries for communicating with aliens. Unless of course

they are sufficiently advanced that they already communicate

in XML.At the other extreme, I think you might be able to design the

core language today. In fact, some might argue that it was already

mostly designed in 1958.If the hundred year language were available today, would we

want to program in it? One way to answer this question is to

look back. If present-day programming languages had been available

in 1960, would anyone have wanted to use them?In some ways, the answer is no. Languages today assume

infrastructure that didn't exist in 1960. For example, a language

in which indentation is significant, like Python, would not

work very well on printer terminals. But putting such problems

aside-- assuming, for example, that programs were all just

written on paper-- would programmers of the 1960s have liked

writing programs in the languages we use now?I think so.

Some of the less imaginative ones,

who had artifacts of early languages built into their ideas of

what a program was, might have had trouble. (How can you manipulate

data without doing pointer arithmetic? How can you implement

flow charts without gotos?) But I think the smartest programmers

would have had no trouble making the most of present-day

languages, if they'd had them.If we had the hundred-year language now, it would at least make a

great pseudocode. What about using it to write software?

Since the hundred-year language

will need to generate fast code for some applications, presumably

it could generate code efficient enough to run acceptably well

on our hardware. We might have to give more optimization advice

than users in a hundred years, but it still might be a net win.Now we have two ideas that, if you combine them, suggest interesting

possibilities: (1) the hundred-year language could, in principle, be

designed today, and (2) such a language, if it existed, might be good to

program in today. When you see these ideas laid out like that,

it's hard not to think, why not try writing the hundred-year language

now?When you're working on language design, I think it is good to

have such a target and to keep it consciously in mind. When you

learn to drive, one of the principles they teach you is to

align the car not by lining up the hood with the stripes painted

on the road, but by aiming at some point in the distance. Even

if all you care about is what happens in the next ten feet, this

is the right answer. I

think we can and should do the same thing with programming languages.

NotesI believe Lisp Machine Lisp was the first language to embody

the principle that declarations (except those of dynamic variables)

were merely optimization advice,

and would not change the meaning of a correct program. Common Lisp

seems to have been the first to state this explicitly.Thanks to Trevor Blackwell, Robert Morris, and Dan Giffin for

reading drafts of this, and to Guido van Rossum, Jeremy Hylton, and the

rest of the Python crew for inviting me to speak at PyCon.Japanese Translation

You'll find this essay and 14 others in

Hackers & Painters.

Why Nerds are Unpopular

February 2003When we were in junior high school, my friend Rich and I made a map

of the school lunch tables according to popularity. This was easy

to do, because kids only ate lunch with others of about the same

popularity. We graded them from A to E. A tables were full of

football players and cheerleaders and so on. E tables contained the

kids with mild cases of Down's Syndrome, what in the language of

the time we called "retards."We sat at a D table, as low as you could get without looking

physically different. We were not being especially candid to grade

ourselves as D. It would have taken a deliberate lie to say otherwise.

Everyone in the school knew exactly how popular everyone else was,

including us.My stock gradually rose during high school. Puberty finally arrived;

I became a decent soccer player; I started a scandalous underground

newspaper. So I've seen a good part of the popularity landscape.I know a lot of people who were nerds in school, and they all tell

the same story: there is a strong correlation between being smart

and being a nerd, and an even stronger inverse correlation between

being a nerd and being popular. Being smart seems to make you

unpopular.Why? To someone in school now, that may seem an odd question to

ask. The mere fact is so overwhelming that it may seem strange to

imagine that it could be any other way. But it could. Being smart

doesn't make you an outcast in elementary school. Nor does it harm

you in the real world. Nor, as far as I can tell, is the problem

so bad in most other countries. But in a typical American secondary

school, being smart is likely to make your life difficult. Why?

The key to this mystery is to rephrase the question slightly. Why

don't smart kids make themselves popular? If they're so smart, why

don't they figure out how popularity works and beat the system,

just as they do for standardized tests?One argument says that this would be impossible, that the smart

kids are unpopular because the other kids envy them for being smart,

and nothing they could do could make them popular. I wish. If the

other kids in junior high school envied me, they did a great job

of concealing it. And in any case, if being smart were really an

enviable quality, the girls would have broken ranks. The guys that

guys envy, girls like.In the schools I went to, being smart just didn't matter much. Kids

didn't admire it or despise it. All other things being equal, they

would have preferred to be on the smart side of average rather than the

dumb side, but intelligence counted far less than, say, physical

appearance, charisma, or athletic ability.So if intelligence in itself is not a factor in popularity, why are

smart kids so consistently unpopular? The answer, I think, is that

they don't really want to be popular.If someone had told me that at the time, I would have laughed at

him. Being unpopular in school makes kids miserable, some of them

so miserable that they commit suicide. Telling me that I didn't

want to be popular would have seemed like telling someone dying of

thirst in a desert that he didn't want a glass of water. Of course

I wanted to be popular.But in fact I didn't, not enough. There was something else I wanted

more: to be smart. Not simply to do well in school, though that

counted for something, but to design beautiful rockets, or to write

well, or to understand how to program computers. In general, to

make great things.At the time I never tried to separate my wants and weigh them

against one another. If I had, I would have seen that being smart

was more important. If someone had offered me the chance to be

the most popular kid in school, but only at the price of being of

average intelligence (humor me here), I wouldn't have taken it.Much as they suffer from their unpopularity, I don't think many

nerds would. To them the thought of average intelligence is unbearable.

But most kids would take that deal. For half of them, it would be

a step up. Even for someone in the eightieth percentile (assuming,

as everyone seemed to then, that intelligence is a scalar), who

wouldn't drop thirty points in exchange for being loved and admired

by everyone?And that, I think, is the root of the problem. Nerds serve two

masters. They want to be popular, certainly, but they want even

more to be smart. And popularity is not something you can do in

your spare time, not in the fiercely competitive environment of an

American secondary school.

Alberti, arguably the archetype of the Renaissance Man, writes that

"no art, however minor, demands less than total dedication if you

want to excel in it."

I wonder if anyone in the world works harder

at anything than American school kids work at popularity. Navy SEALs

and neurosurgery residents seem slackers by comparison. They

occasionally take vacations; some even have hobbies. An American

teenager may work at being popular every waking hour, 365 days a

year.I don't mean to suggest they do this consciously. Some of them truly

are little Machiavellis, but what I really mean here is that teenagers

are always on duty as conformists.For example, teenage kids pay a great deal of attention to clothes.

They don't consciously dress to be popular. They dress to look good.

But to who? To the other kids. Other kids' opinions become their

definition of right, not just for clothes, but for almost everything

they do, right down to the way they walk. And so every effort they

make to do things "right" is also, consciously or not, an effort

to be more popular.Nerds don't realize this. They don't realize that it takes work to

be popular. In general, people outside some very demanding field

don't realize the extent to which success depends on constant (though

often unconscious) effort. For example, most people seem to consider

the ability to draw as some kind of innate quality, like being tall.

In fact, most people who "can draw" like drawing, and have spent

many hours doing it; that's why they're good at it. Likewise, popular

isn't just something you are or you aren't, but something you make

yourself.The main reason nerds are unpopular is that they have other things

to think about. Their attention is drawn to books or the natural

world, not fashions and parties. They're like someone trying to

play soccer while balancing a glass of water on his head. Other

players who can focus their whole attention on the game beat them

effortlessly, and wonder why they seem so incapable.Even if nerds cared as much as other kids about popularity, being

popular would be more work for them. The popular kids learned to

be popular, and to want to be popular, the same way the nerds learned

to be smart, and to want to be smart: from their parents. While the

nerds were being trained to get the right answers, the popular kids

were being trained to please.

So far I've been finessing the relationship between smart and nerd,

using them as if they were interchangeable. In fact it's only the

context that makes them so. A nerd is someone who isn't socially

adept enough. But "enough" depends on where you are. In a typical

American school, standards for coolness are so high (or at least,

so specific) that you don't have to be especially awkward to look

awkward by comparison.Few smart kids can spare the attention that popularity requires.

Unless they also happen to be good-looking, natural athletes, or

siblings of popular kids, they'll tend to become nerds. And that's

why smart people's lives are worst between, say, the ages of eleven

and seventeen. Life at that age revolves far more around popularity

than before or after.Before that, kids' lives are dominated by their parents, not by

other kids. Kids do care what their peers think in elementary school,

but this isn't their whole life, as it later becomes.Around the age of eleven, though, kids seem to start treating their

family as a day job. They create a new world among themselves, and

standing in this world is what matters, not standing in their family.

Indeed, being in trouble in their family can win them points in the

world they care about.The problem is, the world these kids create for themselves is at

first a very crude one. If you leave a bunch of eleven-year-olds

to their own devices, what you get is Lord of the Flies. Like

a lot of American kids, I read this book in school. Presumably it

was not a coincidence. Presumably someone wanted to point out to

us that we were savages, and that we had made ourselves a cruel and

stupid world. This was too subtle for me. While the book seemed

entirely believable, I didn't get the additional message. I wish

they had just told us outright that we were savages and our world

was stupid.

Nerds would find their unpopularity more bearable if it merely

caused them to be ignored. Unfortunately, to be unpopular in school

is to be actively persecuted.Why? Once again, anyone currently in school might think this a

strange question to ask. How could things be any other way? But

they could be. Adults don't normally persecute nerds. Why do teenage

kids do it?Partly because teenagers are still half children, and many

children are just intrinsically cruel. Some torture nerds for the

same reason they pull the legs off spiders. Before you develop a

conscience, torture is amusing.Another reason kids persecute nerds is to make themselves feel

better. When you tread water, you lift yourself up by pushing water

down. Likewise, in any social hierarchy, people unsure of their own

position will try to emphasize it by maltreating those they think

rank below. I've read that this is why poor whites in the United

States are the group most hostile to blacks.But I think the main reason other kids persecute nerds is that it's

part of the mechanism of popularity. Popularity is only partially

about individual attractiveness. It's much more about alliances.

To become more popular, you need to be constantly doing things that

bring you close to other popular people, and nothing brings people

closer than a common enemy.Like a politician who wants to distract voters from bad times at

home, you can create an enemy if there isn't a real one. By singling

out and persecuting a nerd, a group of kids from higher in the

hierarchy create bonds between themselves. Attacking an outsider

makes them all insiders. This is why the worst cases of bullying

happen with groups. Ask any nerd: you get much worse treatment from

a group of kids than from any individual bully, however sadistic.If it's any consolation to the nerds, it's nothing personal. The

group of kids who band together to pick on you are doing the same

thing, and for the same reason, as a bunch of guys who get together

to go hunting. They don't actually hate you. They just need something

to chase.Because they're at the bottom of the scale, nerds are a safe target

for the entire school. If I remember correctly, the most popular

kids don't persecute nerds; they don't need to stoop to such things.

Most of the persecution comes from kids lower down, the nervous

middle classes.The trouble is, there are a lot of them. The distribution of

popularity is not a pyramid, but tapers at the bottom like a pear.

The least popular group is quite small. (I believe we were the only

D table in our cafeteria map.) So there are more people who want

to pick on nerds than there are nerds.As well as gaining points by distancing oneself from unpopular kids,

one loses points by being close to them. A woman I know says that

in high school she liked nerds, but was afraid to be seen talking

to them because the other girls would make fun of her. Unpopularity

is a communicable disease; kids too nice to pick on nerds will still

ostracize them in self-defense.It's no wonder, then, that smart kids tend to be unhappy in middle

school and high school. Their other interests leave them little

attention to spare for popularity, and since popularity resembles

a zero-sum game, this in turn makes them targets for the whole

school. And the strange thing is, this nightmare scenario happens

without any conscious malice, merely because of the shape of the

situation.

For me the worst stretch was junior high, when kid culture was new

and harsh, and the specialization that would later gradually separate

the smarter kids had barely begun. Nearly everyone I've talked to

agrees: the nadir is somewhere between eleven and fourteen.In our school it was eighth grade, which was ages twelve and thirteen

for me. There was a brief sensation that year when one of our

teachers overheard a group of girls waiting for the school bus, and

was so shocked that the next day she devoted the whole class to an

eloquent plea not to be so cruel to one another.It didn't have any noticeable effect. What struck me at the time

was that she was surprised. You mean she doesn't know the kind of

things they say to one another? You mean this isn't normal?It's important to realize that, no, the adults don't know what the

kids are doing to one another. They know, in the abstract, that

kids are monstrously cruel to one another, just as we know in the

abstract that people get tortured in poorer countries. But, like

us, they don't like to dwell on this depressing fact, and they don't

see evidence of specific abuses unless they go looking for it.Public school teachers are in much the same position as prison

wardens. Wardens' main concern is to keep the prisoners on the

premises. They also need to keep them fed, and as far as possible

prevent them from killing one another. Beyond that, they want to

have as little to do with the prisoners as possible, so they leave

them to create whatever social organization they want. From what

I've read, the society that the prisoners create is warped, savage,

and pervasive, and it is no fun to be at the bottom of it.In outline, it was the same at the schools I went to. The most

important thing was to stay on the premises. While there, the

authorities fed you, prevented overt violence, and made some effort

to teach you something. But beyond that they didn't want to have

too much to do with the kids. Like prison wardens, the teachers

mostly left us to ourselves. And, like prisoners, the culture we

created was barbaric.

Why is the real world more hospitable to nerds? It might seem that

the answer is simply that it's populated by adults, who are too

mature to pick on one another. But I don't think this is true.

Adults in prison certainly pick on one another. And so, apparently,

do society wives; in some parts of Manhattan, life for women sounds

like a continuation of high school, with all the same petty intrigues.I think the important thing about the real world is not that it's

populated by adults, but that it's very large, and the things you

do have real effects. That's what school, prison, and ladies-who-lunch

all lack. The inhabitants of all those worlds are trapped in little

bubbles where nothing they do can have more than a local effect.

Naturally these societies degenerate into savagery. They have no

function for their form to follow.When the things you do have real effects, it's no longer enough

just to be pleasing. It starts to be important to get the right

answers, and that's where nerds show to advantage. Bill Gates will

of course come to mind. Though notoriously lacking in social skills,

he gets the right answers, at least as measured in revenue.The other thing that's different about the real world is that it's

much larger. In a large enough pool, even the smallest minorities

can achieve a critical mass if they clump together. Out in the real

world, nerds collect in certain places and form their own societies

where intelligence is the most important thing. Sometimes the current

even starts to flow in the other direction: sometimes, particularly

in university math and science departments, nerds deliberately

exaggerate their awkwardness in order to seem smarter. John Nash

so admired Norbert Wiener that he adopted his habit of touching the

wall as he walked down a corridor.

As a thirteen-year-old kid, I didn't have much more experience of

the world than what I saw immediately around me. The warped little

world we lived in was, I thought, the world. The world seemed cruel

and boring, and I'm not sure which was worse.Because I didn't fit into this world, I thought that something must

be wrong with me. I didn't realize that the reason we nerds didn't

fit in was that in some ways

we were a step ahead. We were already thinking about

the kind of things that matter in the real world, instead of spending

all our time playing an exacting but mostly pointless game like the

others.We were a bit like an adult would be if he were thrust back into

middle school. He wouldn't know the right clothes to wear, the right

music to like, the right slang to use. He'd seem to the kids a

complete alien. The thing is, he'd know enough not to care what

they thought. We had no such confidence.A lot of people seem to think it's good for smart kids to be thrown

together with "normal" kids at this stage of their lives. Perhaps.

But in at least some cases the reason the nerds don't fit in really

is that everyone else is crazy. I remember sitting in the audience

at a "pep rally" at my high school, watching as the cheerleaders

threw an effigy of an opposing player into the audience to be torn

to pieces. I felt like an explorer witnessing some bizarre tribal

ritual.

If I could go back and give my thirteen year old self some advice,

the main thing I'd tell him would be to stick his head up and look

around. I didn't really grasp it at the time, but the whole world

we lived in was as fake as a Twinkie. Not just school, but the

entire town. Why do people move to suburbia? To have kids! So no

wonder it seemed boring and sterile. The whole place was a giant

nursery, an artificial town created explicitly for the purpose of

breeding children.Where I grew up, it felt as if there was nowhere to go, and nothing

to do. This was no accident. Suburbs are deliberately designed to

exclude the outside world, because it contains things that could

endanger children.And as for the schools, they were just holding pens within this

fake world. Officially the purpose of schools is to teach kids. In

fact their primary purpose is to keep kids locked up in one

place for a big chunk of the day so adults can get things done. And

I have no problem with this: in a specialized industrial society,

it would be a disaster to have kids running around loose.What bothers me is not that the kids are kept in prisons, but that

(a) they aren't told about it, and (b) the prisons are run mostly

by the inmates. Kids are sent off to spend six years memorizing

meaningless facts in a world ruled by a caste of giants who run

after an oblong brown ball, as if this were the most natural thing

in the world. And if they balk at this surreal cocktail, they're

called misfits.

Life in this twisted world is stressful for the kids. And not just

for the nerds. Like any war, it's damaging even to the winners.Adults can't avoid seeing that teenage kids are tormented. So why

don't they do something about it? Because they blame it on puberty.

The reason kids are so unhappy, adults tell themselves, is that

monstrous new chemicals, hormones, are now coursing through their

bloodstream and messing up everything. There's nothing wrong with

the system; it's just inevitable that kids will be miserable at

that age.This idea is so pervasive that even the kids believe it, which

probably doesn't help. Someone who thinks his feet naturally hurt

is not going to stop to consider the possibility that he is wearing

the wrong size shoes.I'm suspicious of this theory that thirteen-year-old kids are

intrinsically messed up. If it's physiological, it should be

universal. Are Mongol nomads all nihilists at thirteen? I've read

a lot of history, and I have not seen a single reference

to this supposedly universal fact before the twentieth century.

Teenage apprentices in the Renaissance seem to have been cheerful

and eager. They got in fights and played tricks on one another of

course (Michelangelo had his nose broken by a bully), but they

weren't crazy.As far as I can tell, the concept of the hormone-crazed teenager

is coeval with suburbia. I don't think this is a coincidence. I

think teenagers are driven crazy by the life they're made to lead.

Teenage apprentices in the Renaissance were working dogs. Teenagers

now are neurotic lapdogs. Their craziness is the craziness of the

idle everywhere.

When I was in school, suicide was a constant topic among the smarter

kids. No one I knew did it, but several planned to, and

some may have tried. Mostly this was just a pose. Like other

teenagers, we loved the dramatic, and suicide seemed very dramatic.

But partly it was because our lives were at times genuinely miserable.Bullying was only part of the problem. Another problem, and possibly

an even worse one, was that we never had anything real to work on.

Humans like to work; in most of the world, your work is your identity.

And all the work we did was

pointless, or seemed so at the time.At best it was practice for real work we might do far in the future,

so far that we didn't even know at the time what we were practicing

for. More often it was just an arbitrary series of hoops to jump

through, words without content designed mainly for testability.

(The three main causes of the Civil War were....

Test: List the three main causes of the Civil War.)And there was no way to opt out. The adults had agreed among

themselves that this was to be the route to college. The only way

to escape this empty life was to submit to it.

Teenage kids used to have a more active role in society. In

pre-industrial times, they were all apprentices of one sort or

another, whether in shops or on farms or even on warships. They

weren't left to create their own societies. They were junior members

of adult societies.Teenagers seem to have respected adults more then, because

the adults were the visible experts in the skills they were trying

to learn. Now most kids have little idea what their parents do in

their distant offices, and see no connection (indeed, there is

precious little) between schoolwork and the work they'll do as

adults.And if teenagers respected adults more, adults also had more use

for teenagers. After a couple years' training, an apprentice could

be a real help. Even the newest apprentice could be made to carry

messages or sweep the workshop.Now adults have no immediate use for teenagers. They would be in

the way in an office. So they drop them off at school on their way

to work, much as they might drop the dog off at a kennel if they

were going away for the weekend.What happened? We're up against a hard one here. The cause of this

problem is the same as the cause of so many present ills: specialization.

As jobs become more specialized, we have to train longer for them.

Kids in pre-industrial times started working at about 14 at

the latest; kids on farms, where most people lived, began far

earlier. Now kids who go to college don't start working full-time

till 21 or 22. With some degrees, like MDs and PhDs, you may not

finish your training till 30.Teenagers now are useless, except as cheap labor in industries like

fast food, which evolved to exploit precisely this fact. In almost

any other kind of work, they'd be a net loss. But they're also too

young to be left unsupervised. Someone has to watch over them, and

the most efficient way to do this is to collect them together in

one place. Then a few adults can watch all of them.If you stop there, what you're describing is literally a prison,

albeit a part-time one. The problem is, many schools practically

do stop there. The stated purpose of schools is to educate the kids.

But there is no external pressure to do this well. And so most

schools do such a bad job of teaching that the kids don't really

take it seriously-- not even the smart kids. Much of the time we

were all, students and teachers both, just going through the motions.In my high school French class we were supposed to read Hugo's Les

Miserables. I don't think any of us knew French well enough to make

our way through this enormous book. Like the rest of the class, I

just skimmed the Cliff's Notes. When we were given a test on the

book, I noticed that the questions sounded odd. They were full of

long words that our teacher wouldn't have used. Where had these

questions come from? From the Cliff's Notes, it turned out. The

teacher was using them too. We were all just pretending.There are certainly great public school teachers. The energy and

imagination of my fourth grade teacher, Mr. Mihalko, made that

year something his students still talk about, thirty years later.

But teachers like him were individuals swimming

upstream. They couldn't fix the system.

In almost any group of people you'll find hierarchy.

When groups of adults form in the real world, it's generally for

some common purpose, and the leaders end up being those who are best

at it. The problem with most schools is, they have no purpose.

But hierarchy there must be.

And so the kids make one out of nothing.We have a phrase to describe what happens when rankings have to be

created without any meaningful criteria. We say that the situation

degenerates into a popularity contest. And that's exactly what

happens in most American schools.

Instead of depending on some real test, one's rank

depends mostly on one's ability to increase one's rank. It's

like the court of Louis XIV. There is no external opponent, so the

kids become one another's opponents.When there is some real external test of skill, it isn't painful

to be at the bottom of the hierarchy. A rookie on a football team

doesn't resent the skill of the veteran; he hopes to be like him

one day and is happy to have the chance to learn from him.

The veteran may in turn feel a sense of

noblesse oblige.

And most importantly, their status depends on how well they

do against opponents, not on whether they can push the other down.Court hierarchies are another thing entirely. This type of society

debases anyone who enters it. There is neither admiration at the

bottom, nor noblesse oblige at the top. It's kill or be killed.This is the sort of society that gets created

in American

secondary schools. And it happens because these schools have no

real purpose beyond keeping the kids all in one place for a certain

number of hours each day. What I didn't realize at the time, and

in fact didn't realize till very recently, is that the twin horrors

of school life, the cruelty and the boredom, both have the same

cause.

The mediocrity of American public schools has worse consequences

than just making kids unhappy for six years. It breeds a rebelliousness

that actively drives kids away from the things they're supposed to

be learning.Like many nerds, probably, it was years after high school before I

could bring myself to read anything we'd been assigned then.

And I lost more than books. I mistrusted words like "character" and

"integrity" because they had been so debased by adults. As they

were used then, these words all seemed to mean the same thing:

obedience. The kids who got praised for these qualities tended to

be at best dull-witted prize bulls, and at worst facile schmoozers.

If that was what character and integrity were, I wanted no part of

them.The word I most misunderstood was "tact." As used by adults, it

seemed to mean keeping your mouth shut.

I assumed it was derived from the same root as

"tacit" and "taciturn," and that it literally meant being quiet. I

vowed that I would never be tactful; they were never going to shut

me up. In fact, it's derived from the same root as "tactile," and

what it means is to have a deft touch. Tactful is the opposite of

clumsy. I don't think I learned this until college.Nerds aren't the only losers in the popularity rat race. Nerds are

unpopular because they're distracted. There are other kids who

deliberately opt out because they're so disgusted with the whole

process.Teenage kids, even rebels, don't like to be alone, so when kids opt

out of the system, they tend to do it as a group. At the schools I

went to, the focus of rebellion was drug use, specifically marijuana.

The kids in this tribe wore black concert t-shirts and were called

"freaks."Freaks and nerds were allies, and there was a good deal of overlap

between them. Freaks were on the whole smarter than other kids,

though never studying (or at least never appearing to) was an

important tribal value. I was more in the nerd camp, but I was

friends with a lot of freaks.They used drugs, at least at first, for the social bonds they

created. It was something to do together, and because the drugs

were illegal, it was a shared badge of rebellion.I'm not claiming that bad schools are the whole reason kids get

into trouble with drugs. After a while, drugs have their own momentum.

No doubt some of the freaks ultimately used drugs to escape from

other problems-- trouble at home, for example. But, in my school

at least, the reason most kids started using drugs was rebellion.

Fourteen-year-olds didn't start smoking pot because they'd heard

it would help them forget their problems. They started because they

wanted to join a different tribe.Misrule breeds rebellion; this is not a new idea. And yet the

authorities still for the most part act as if drugs were themselves

the cause of the problem.

The real problem is the emptiness of school life. We won't see

solutions till adults realize that. The adults who

may realize it first are the ones who were themselves nerds in

school. Do you want your kids to be as unhappy in eighth grade as

you were? I wouldn't. Well, then, is there anything we can do to

fix things? Almost certainly. There is nothing inevitable about the

current system. It has come about mostly by default.Adults, though, are busy. Showing up for school plays is one thing.

Taking on the educational bureaucracy is another. Perhaps a few

will have the energy to try to change things. I suspect the hardest

part is realizing that you can.Nerds still in school should not hold their breath. Maybe one day

a heavily armed force of adults will show up in helicopters to

rescue you, but they probably won't be coming this month. Any

immediate improvement in nerds' lives is probably going to have to

come from the nerds themselves.Merely understanding the situation they're in should make it less

painful. Nerds aren't losers. They're just playing a different game,

and a game much closer to the one played in the real world. Adults

know this. It's hard to find successful adults now who don't claim

to have been nerds in high school.It's important for nerds to realize, too, that school is not life.

School is a strange, artificial thing, half sterile and half feral.

It's all-encompassing, like life, but it isn't the real thing. It's

only temporary, and if you look, you can see beyond it even while

you're still in it.If life seems awful to kids, it's neither because hormones are

turning you all into monsters (as your parents believe), nor because

life actually is awful (as you believe). It's because the adults,

who no longer have any economic use for you, have abandoned you to

spend years cooped up together with nothing real to do. Any society

of that type is awful to live in.

You don't have

to look any further to explain why teenage kids are unhappy.I've said some harsh things in this essay, but really the thesis

is an optimistic one-- that several problems we take for granted

are in fact not insoluble after all. Teenage kids are not inherently

unhappy monsters. That should be encouraging news to kids and adults

both.

Thanks to Sarah Harlin, Trevor Blackwell, Robert Morris,

Eric Raymond, and Jackie Weicker for reading drafts of this essay,

and Maria Daniels for scanning photos.Re: Why Nerds are UnpopularGateway High School, 1981Japanese TranslationFrench TranslationMy War With BrianButtonsPortuguese TranslationSpanish Translation

Better Bayesian Filtering

January 2003(This article was given as a talk at the 2003 Spam Conference.

It describes the work I've done to improve the performance of

the algorithm described in A Plan for Spam,

and what I plan to do in the future.)The first discovery I'd like to present here is an algorithm for

lazy evaluation of research papers. Just

write whatever you want and don't cite any previous work, and

indignant readers will send you references to all the papers you

should have cited. I discovered this algorithm

after ``A Plan for Spam'' [1] was on Slashdot.Spam filtering is a subset of text classification,

which is a well established field, but the first papers about

Bayesian

spam filtering per se seem to have been two

given at the same conference in 1998,

one by Pantel and Lin [2],

and another by a group from

Microsoft Research [3].When I heard about this work I was a bit surprised. If

people had been onto Bayesian filtering four years ago,

why wasn't everyone using it?

When I read the papers I found out why. Pantel and Lin's filter was the

more effective of the two, but it

only caught 92% of spam, with 1.16% false positives.When I tried writing a Bayesian spam filter,

it caught 99.5% of spam with less than .03% false

positives [4].

It's always alarming when two people

trying the same experiment get widely divergent results.

It's especially alarming here because those two sets of numbers

might yield opposite conclusions.

Different users have different requirements, but I think for

many people a filtering rate of 92% with 1.16% false positives means

that filtering is not an acceptable solution, whereas

99.5% with less than .03% false positives means that it is.So why did we get such different numbers?

I haven't tried to reproduce Pantel and Lin's results, but

from reading the paper I see five things that probably account

for the difference.One is simply that they trained their filter on very little

data: 160 spam and 466 nonspam mails.

Filter performance should still be climbing with data

sets that small. So their numbers may not even be an accurate

measure of the performance of their algorithm, let alone of

Bayesian spam filtering in general.But I think the most important difference is probably

that they ignored message headers. To anyone who has worked

on spam filters, this will seem a perverse decision.

And yet in the very first filters I tried writing, I ignored the

headers too. Why? Because I wanted to keep the problem neat.

I didn't know much about mail headers then, and they seemed to me

full of random stuff. There is a lesson here for filter

writers: don't ignore data. You'd think this lesson would

be too obvious to mention, but I've had to learn it several times.Third, Pantel and Lin stemmed the tokens, meaning they reduced e.g. both

``mailing'' and ``mailed'' to the root ``mail''. They may

have felt they were forced to do this by the small size

of their corpus, but if so this is a kind of premature

optimization.Fourth, they calculated probabilities differently.

They used all the tokens, whereas I only

use the 15 most significant. If you use all the tokens

you'll tend to miss longer spams, the type where someone tells you their life

story up to the point where they got rich from some multilevel

marketing scheme. And such an algorithm

would be easy for spammers to spoof: just add a big

chunk of random text to counterbalance the spam terms.Finally, they didn't bias against false positives.

I think

any spam filtering algorithm ought to have a convenient

knob you can twist to decrease the

false positive rate at the expense of the filtering rate.

I do this by counting the occurrences

of tokens in the nonspam corpus double.

I don't think it's a good idea to treat spam filtering as

a straight text classification problem. You can use

text classification techniques, but solutions can and should

reflect the fact that the text is email, and spam

in particular. Email is not just text; it has structure.

Spam filtering is not just classification, because

false positives are so much worse than false negatives

that you should treat them as a different kind of error.

And the source of error is not just random variation, but

a live human spammer working actively to defeat your filter.TokensAnother project I heard about

after the Slashdot article was Bill Yerazunis'

CRM114 [5].

This is the counterexample to the design principle I

just mentioned. It's a straight text classifier,

but such a stunningly effective one that it manages to filter

spam almost perfectly without even knowing that's

what it's doing.Once I understood how CRM114 worked, it seemed

inevitable that I would eventually have to move from filtering based

on single words to an approach like this. But first, I thought,

I'll see how far I can get with single words. And the answer is,

surprisingly far.Mostly I've been working on smarter tokenization. On

current spam, I've been able to achieve filtering rates that

approach CRM114's. These techniques are mostly orthogonal to Bill's;

an optimal solution might incorporate both.``A Plan for Spam'' uses a very simple

definition of a token. Letters, digits, dashes, apostrophes,

and dollar signs are constituent characters, and everything

else is a token separator. I also ignored case.Now I have a more complicated definition of a token:

Case is preserved. Exclamation points are constituent characters. Periods and commas are constituents if they occur

between two digits. This lets me get ip addresses

and prices intact. A price range like $20-25 yields two tokens,

$20 and $25. Tokens that occur within the

To, From, Subject, and Return-Path lines, or within urls,

get marked accordingly. E.g. ``foo'' in the Subject line

becomes ``Subject\*foo''. (The asterisk could

be any character you don't allow as a constituent.)

Such measures increase the filter's vocabulary, which

makes it more discriminating. For example, in the current

filter, ``free'' in the Subject line

has a spam probability of 98%, whereas the same token

in the body has a spam probability of only 65%.Here are some of the current probabilities [6]:

Subject\*FREE 0.9999

free!! 0.9999

To\*free 0.9998

Subject\*free 0.9782

free! 0.9199

Free 0.9198

Url\*free 0.9091

FREE 0.8747

From\*free 0.7636

free 0.6546

In the Plan for Spam filter, all these tokens would have had the

same probability, .7602. That filter recognized about 23,000

tokens. The current one recognizes about 187,000.The disadvantage of having a larger universe of tokens

is that there is more

chance of misses.

Spreading your corpus out over more tokens

has the same effect as making it smaller.

If you consider exclamation points as

constituents, for example, then you could end up

not having a spam probability for free with seven exclamation

points, even though you know that free with just two

exclamation points has a probability of 99.99%.One solution to this is what I call degeneration. If you

can't find an exact match for a token,

treat it as if it were a less specific

version. I consider terminal exclamation

points, uppercase letters, and occurring in one of the

five marked contexts as making a token more specific.

For example, if I don't find a probability for

``Subject\*free!'', I look for probabilities for

``Subject\*free'', ``free!'', and ``free'', and take whichever one

is farthest from .5.Here are the alternatives [7]

considered if the filter sees ``FREE!!!'' in the

Subject line and doesn't have a probability for it.

Subject\*Free!!!

Subject\*free!!!

Subject\*FREE!

Subject\*Free!

Subject\*free!

Subject\*FREE

Subject\*Free

Subject\*free

FREE!!!

Free!!!

free!!!

FREE!

Free!

free!

FREE

Free

free

If you do this, be sure to consider versions with initial

caps as well as all uppercase and all lowercase. Spams

tend to have more sentences in imperative mood, and in

those the first word is a verb. So verbs with initial caps

have higher spam probabilities than they would in all

lowercase. In my filter, the spam probability of ``Act''

is 98% and for ``act'' only 62%.If you increase your filter's vocabulary, you can end up

counting the same word multiple times, according to your old

definition of ``same''.

Logically, they're not the

same token anymore. But if this still bothers you, let

me add from experience that the words you seem to be

counting multiple times tend to be exactly the ones you'd

want to.Another effect of a larger vocabulary is that when you

look at an incoming mail you find more interesting tokens,

meaning those with probabilities far from .5. I use the

15 most interesting to decide if mail is spam.

But you can run into a problem when you use a fixed number

like this. If you find a lot of maximally interesting tokens,

the result can end up being decided by whatever random factor

determines the ordering of equally interesting tokens.

One way to deal with this is to treat some

as more interesting than others.For example, the

token ``dalco'' occurs 3 times in my spam corpus and never

in my legitimate corpus. The token ``Url\*optmails''

(meaning ``optmails'' within a url) occurs 1223 times.

And yet, as I used to calculate probabilities for tokens,

both would have the same spam probability, the threshold of .99.That doesn't feel right. There are theoretical

arguments for giving these two tokens substantially different

probabilities (Pantel and Lin do), but I haven't tried that yet.

It does seem at least that if we find more than 15 tokens

that only occur in one corpus or the other, we ought to

give priority to the ones that occur a lot. So now

there are two threshold values. For tokens that occur only

in the spam corpus, the probability is .9999 if they

occur more than 10 times and .9998 otherwise. Ditto

at the other end of the scale for tokens found

only in the legitimate corpus.I may later scale token probabilities substantially,

but this tiny amount of scaling at least ensures that

tokens get sorted the right way.Another possibility would be to consider not

just 15 tokens, but all the tokens over a certain

threshold of interestingness. Steven Hauser does this

in his statistical spam filter [8].

If you use a threshold, make it very high, or

spammers could spoof you by packing messages with

more innocent words.Finally, what should one do

about html? I've tried the whole spectrum of options, from

ignoring it to parsing it all. Ignoring html is a bad idea,

because it's full of useful spam signs. But if you parse

it all, your filter might degenerate into a mere html

recognizer. The most effective approach

seems to be the middle course, to notice some tokens but not

others. I look at a, img, and font tags, and ignore the

rest. Links and images you should certainly look at, because

they contain urls.I could probably be smarter about dealing with html, but I

don't think it's worth putting a lot of time into this.

Spams full of html are easy to filter. The smarter

spammers already avoid it. So

performance in the future should not depend much on how

you deal with html.PerformanceBetween December 10 2002 and January 10 2003 I got about

1750 spams.

Of these, 4 got through. That's a filtering

rate of about 99.75%.Two of the four spams I missed got through because they

happened to use words that occur often in my legitimate

email.The third was one of those that exploit

an insecure cgi script to send mail to third parties.

They're hard to filter based just

on the content because the headers are innocent and

they're careful about the words they use. Even so I can

usually catch them. This one squeaked by with a

probability of .88, just under the threshold of .9.Of course, looking at multiple token sequences

would catch it easily. ``Below is the result of

your feedback form'' is an instant giveaway.The fourth spam was what I call

a spam-of-the-future, because this is what I expect spam to

evolve into: some completely neutral

text followed by a url. In this case it was was from

someone saying they had finally finished their homepage

and would I go look at it. (The page was of course an

ad for a porn site.)If the spammers are careful about the headers and use a

fresh url, there is nothing in spam-of-the-future for filters

to notice. We can of course counter by sending a

crawler to look at the page. But that might not be necessary.

The response rate for spam-of-the-future must

be low, or everyone would be doing it.

If it's low enough,

it won't pay for spammers to send it, and we won't

have to work too hard on filtering it.Now for the really shocking news: during that same one-month

period I got three false positives.In a way it's

a relief to get some false positives. When I wrote ``A Plan

for Spam'' I hadn't had any, and I didn't know what they'd

be like. Now that I've had a few, I'm relieved to find

they're not as bad as I feared.

False positives yielded by statistical

filters turn out to be mails that sound a lot like spam, and

these tend to be the ones you would least mind missing [9].Two of the false positives were newsletters

from companies I've bought things from. I never

asked to receive them, so arguably they

were spams, but I count them as false positives because

I hadn't been deleting them as spams before. The reason

the filters caught them was that both companies in

January switched to commercial email senders

instead of sending the mails from their own servers,

and both the headers and the bodies became much spammier.The third false positive was a bad one, though. It was

from someone in Egypt and written in all uppercase. This was

a direct result of making tokens case sensitive; the Plan

for Spam filter wouldn't have caught it.It's hard to say what the overall false positive rate is,

because we're up in the noise, statistically.

Anyone who has worked on filters (at least, effective filters) will

be aware of this problem.

With some emails it's

hard to say whether they're spam or not, and these are

the ones you end up looking at when you get filters

really tight. For example, so far the filter has

caught two emails that were sent to my address because

of a typo, and one sent to me in the belief that I was

someone else. Arguably, these are neither my spam

nor my nonspam mail.Another false positive was from a vice president at Virtumundo.

I wrote to them pretending to be a customer,

and since the reply came back through Virtumundo's

mail servers it had the most incriminating

headers imaginable. Arguably this isn't a real false

positive either, but a sort of Heisenberg uncertainty

effect: I only got it because I was writing about spam

filtering.Not counting these, I've had a total of five false positives

so far, out of about 7740 legitimate emails, a rate of .06%.

The other two were a notice that something I bought

was back-ordered, and a party reminder from Evite.I don't think this number can be trusted, partly

because the sample is so small, and partly because

I think I can fix the filter not to catch

some of these.False positives seem to me a different kind of error from

false negatives.

Filtering rate is a measure of performance. False

positives I consider more like bugs. I approach improving the

filtering rate as optimization, and decreasing false

positives as debugging.So these five false positives are my bug list. For example,

the mail from Egypt got nailed because the uppercase text

made it look to the filter like a Nigerian spam.

This really is kind of a bug. As with

html, the email being all uppercase is really conceptually one

feature, not one for each word. I need to handle case in a

more sophisticated way.So what to make of this .06%? Not much, I think. You could

treat it as an upper bound, bearing in mind the small sample size.

But at this stage it is more a measure of the bugs

in my implementation than some intrinsic false positive rate

of Bayesian filtering.FutureWhat next? Filtering is an optimization problem,

and the key to optimization is profiling. Don't

try to guess where your code is slow, because you'll

guess wrong. Look at where your code is slow,

and fix that. In filtering, this translates to:

look at the spams you miss, and figure out what you

could have done to catch them.For example, spammers are now working aggressively to

evade filters, and one of the things they're doing is

breaking up and misspelling words to prevent filters from

recognizing them. But working on this is not my first

priority, because I still have no trouble catching these

spams [10].There are two kinds of spams I currently do

have trouble with.

One is the type that pretends to be an email from

a woman inviting you to go chat with her or see her profile on a dating

site. These get through because they're the one type of

sales pitch you can make without using sales talk. They use

the same vocabulary as ordinary email.The other kind of spams I have trouble filtering are those

from companies in e.g. Bulgaria offering contract programming

services. These get through because I'm a programmer too, and

the spams are full of the same words as my real mail.I'll probably focus on the personal ad type first. I think if

I look closer I'll be able to find statistical differences

between these and my real mail. The style of writing is

certainly different, though it may take multiword filtering

to catch that.

Also, I notice they tend to repeat the url,

and someone including a url in a legitimate mail wouldn't do that [11].The outsourcing type are going to be hard to catch. Even if

you sent a crawler to the site, you wouldn't find a smoking

statistical gun.

Maybe the only answer is a central list of

domains advertised in spams [12]. But there can't be that

many of this type of mail. If the only

spams left were unsolicited offers of contract programming

services from Bulgaria, we could all probably move on to

working on something else.Will statistical filtering actually get us to that point?

I don't know. Right now, for me personally, spam is

not a problem. But spammers haven't yet made a serious

effort to spoof statistical filters. What will happen when they do?I'm not optimistic about filters that work at the

network level [13].

When there is a static obstacle worth getting past, spammers

are pretty efficient at getting past it. There

is already a company called Assurance Systems that will

run your mail through Spamassassin and tell you whether

it will get filtered out.Network-level filters won't be completely useless.

They may be enough to kill all the "opt-in"

spam, meaning spam from companies like Virtumundo and

Equalamail who claim that they're really running opt-in lists.

You can filter those based just on the headers, no

matter what they say in the body. But anyone willing to

falsify headers or use open relays, presumably including

most porn spammers, should be able to get some message past

network-level filters if they want to. (By no means the

message they'd like to send though, which is something.)The kind of filters I'm optimistic about are ones that

calculate probabilities based on each individual user's mail.

These can be much more effective, not only in

avoiding false positives, but in filtering too: for example,

finding the recipient's email address base-64 encoded anywhere in

a message is a very good spam indicator.But the real advantage of individual filters is that they'll all be

different. If everyone's filters have different probabilities,

it will make the spammers' optimization loop, what programmers

would call their edit-compile-test cycle, appallingly slow.

Instead of just tweaking a spam till it gets through a copy of

some filter they have on their desktop, they'll have to do a

test mailing for each tweak. It would be like programming in

a language without an interactive toplevel,

and I wouldn't wish that

on anyone.Notes[1]

Paul Graham. ``A Plan for Spam.'' August 2002.

http://paulgraham.com/spam.html.Probabilities in this algorithm are

calculated using a degenerate case of Bayes' Rule. There are

two simplifying assumptions: that the probabilities

of features (i.e. words) are independent, and that we know

nothing about the prior probability of an email being

spam.The first assumption is widespread in text classification.

Algorithms that use it are called ``naive Bayesian.''The second assumption I made because the proportion of spam in

my incoming mail fluctuated so much from day to day (indeed,

from hour to hour) that the overall prior ratio seemed

worthless as a predictor. If you assume that P(spam) and

P(nonspam) are both .5, they cancel out and you can

remove them from the formula.If you were doing Bayesian filtering in a situation where

the ratio of spam to nonspam was consistently very high or

(especially) very low, you could probably improve filter

performance by incorporating prior probabilities. To do

this right you'd have to track ratios by time of day, because

spam and legitimate mail volume both have distinct daily

patterns.[2]

Patrick Pantel and Dekang Lin. ``SpamCop-- A Spam

Classification & Organization Program.'' Proceedings of AAAI-98

Workshop on Learning for Text Categorization.[3]

Mehran Sahami, Susan Dumais, David Heckerman and Eric Horvitz.

``A Bayesian Approach to Filtering Junk E-Mail.'' Proceedings of AAAI-98

Workshop on Learning for Text Categorization.[4] At the time I had zero false positives out of about 4,000

legitimate emails. If the next legitimate email was

a false positive, this would give us .03%. These false positive

rates are untrustworthy, as I explain later. I quote

a number here only to emphasize that whatever the false positive rate

is, it is less than 1.16%.

[5] Bill Yerazunis. ``Sparse Binary Polynomial Hash Message

Filtering and The CRM114 Discriminator.'' Proceedings of 2003

Spam Conference.[6] In ``A Plan for Spam'' I used thresholds of .99 and .01.

It seems justifiable to use thresholds proportionate to the

size of the corpora. Since I now have on the order of 10,000 of each

type of mail, I use .9999 and .0001.[7] There is a flaw here I should probably fix. Currently,

when ``Subject\*foo'' degenerates to just ``foo'', what that means is

you're getting the stats for occurrences of ``foo'' in

the body or header lines other than those I mark.

What I should do is keep track of statistics for ``foo''

overall as well as specific versions, and degenerate from

``Subject\*foo'' not to ``foo'' but to ``Anywhere\*foo''. Ditto for

case: I should degenerate from uppercase to any-case, not

lowercase.It would probably be a win to do this with prices

too, e.g. to degenerate from ``$129.99'' to ``$--9.99'', ``$--.99'',

and ``$--''.You could also degenerate from words to their stems,

but this would probably only improve filtering rates early on

when you had small corpora.[8] Steven Hauser. ``Statistical Spam Filter Works for Me.''

http://www.sofbot.com.[9] False positives are not all equal, and we should remember

this when comparing techniques for stopping spam.

Whereas many of the false positives caused by filters

will be near-spams that you wouldn't mind missing,

false positives caused by blacklists, for example, will be just

mail from people who chose the wrong ISP. In both

cases you catch mail that's near spam, but for blacklists nearness

is physical, and for filters it's textual.

[10] If spammers get good enough at obscuring tokens

for this to be a problem, we can respond by simply removing

whitespace, periods, commas, etc. and using a dictionary to

pick the words out of the resulting sequence.

And of course finding words this way that weren't visible in

the original text would in itself be evidence of spam.Picking out the words won't be trivial. It will require

more than just reconstructing word boundaries; spammers

both add (``xHot nPorn cSite'') and omit (``P#rn'') letters.

Vision research may be useful here, since human vision is

the limit that such tricks will approach.[11]

In general, spams are more repetitive than regular email.

They want to pound that message home. I currently don't

allow duplicates in the top 15 tokens, because

you could get a false positive if the sender happens to use

some bad word multiple times. (In my current filter, ``dick'' has

a spam probabilty of .9999, but it's also a name.)

It seems we should at least notice duplication though,

so I may try allowing up to two of each token, as Brian Burton does in

SpamProbe.[12] This is what approaches like Brightmail's will

degenerate into once spammers are pushed into using mad-lib

techniques to generate everything else in the message.[13]

It's sometimes argued that we should be working on filtering

at the network level, because it is more efficient. What people

usually mean when they say this is: we currently filter at the

network level, and we don't want to start over from scratch.

But you can't dictate the problem to fit your solution.Historically, scarce-resource arguments have been the losing

side in debates about software design.

People only tend to use them to justify choices

(inaction in particular) made for other reasons.Thanks to Sarah Harlin, Trevor Blackwell, and

Dan Giffin for reading drafts of this paper, and to Dan again

for most of the infrastructure that this filter runs on.Related:A Plan for SpamPlan for Spam FAQ2003 Spam Conference ProceedingsJapanese TranslationChinese TranslationTest of These Suggestions

Design and Research

January 2003(This article is derived from a keynote talk at the fall 2002 meeting

of NEPLS.)Visitors to this country are often surprised to find that

Americans like to begin a conversation by asking "what do you do?"

I've never liked this question. I've rarely had a

neat answer to it. But I think I have finally solved the problem.

Now, when someone asks me what I do, I look them straight

in the eye and say "I'm designing a

new dialect of Lisp."

I recommend this answer to anyone who doesn't like being asked what

they do. The conversation will turn immediately to other topics.I don't consider myself to be doing research on programming languages.

I'm just designing one, in the same way that someone might design

a building or a chair or a new typeface.

I'm not trying to discover anything new. I just want

to make a language that will be good to program in. In some ways,

this assumption makes life a lot easier.The difference between design and research seems to be a question

of new versus good. Design doesn't have to be new, but it has to

be good. Research doesn't have to be good, but it has to be new.

I think these two paths converge at the top: the best design

surpasses its predecessors by using new ideas, and the best research

solves problems that are not only new, but actually worth solving.

So ultimately we're aiming for the same destination, just approaching

it from different directions.What I'm going to talk about today is what your target looks like

from the back. What do you do differently when you treat

programming languages as a design problem instead of a research topic?The biggest difference is that you focus more on the user.

Design begins by asking, who is this

for and what do they need from it? A good architect,

for example, does not begin by creating a design that he then

imposes on the users, but by studying the intended users and figuring

out what they need.Notice I said "what they need," not "what they want." I don't mean

to give the impression that working as a designer means working as

a sort of short-order cook, making whatever the client tells you

to. This varies from field to field in the arts, but

I don't think there is any field in which the best work is done by

the people who just make exactly what the customers tell them to.The customer is always right in

the sense that the measure of good design is how well it works

for the user. If you make a novel that bores everyone, or a chair

that's horribly uncomfortable to sit in, then you've done a bad

job, period. It's no defense to say that the novel or the chair

is designed according to the most advanced theoretical principles.And yet, making what works for the user doesn't mean simply making

what the user tells you to. Users don't know what all the choices

are, and are often mistaken about what they really want.The answer to the paradox, I think, is that you have to design

for the user, but you have to design what the user needs, not simply

what he says he wants.

It's much like being a doctor. You can't just treat a patient's

symptoms. When a patient tells you his symptoms, you have to figure

out what's actually wrong with him, and treat that.This focus on the user is a kind of axiom from which most of the

practice of good design can be derived, and around which most design

issues center.If good design must do what the user needs, who is the user? When

I say that design must be for users, I don't mean to imply that good

design aims at some kind of

lowest common denominator. You can pick any group of users you

want. If you're designing a tool, for example, you can design it

for anyone from beginners to experts, and what's good design

for one group might be bad for another. The point

is, you have to pick some group of users. I don't think you can

even talk about good or bad design except with

reference to some intended user.You're most likely to get good design if the intended users include

the designer himself. When you design something

for a group that doesn't include you, it tends to be for people

you consider to be less sophisticated than you, not more sophisticated.That's a problem, because looking down on the user, however benevolently,

seems inevitably to corrupt the designer.

I suspect that very few housing

projects in the US were designed by architects who expected to live

in them. You can see the same thing

in programming languages. C, Lisp, and Smalltalk were created for

their own designers to use. Cobol, Ada, and Java, were created

for other people to use.If you think you're designing something for idiots, the odds are

that you're not designing something good, even for idiots.

Even if you're designing something for the most sophisticated

users, though, you're still designing for humans. It's different

in research. In math you

don't choose abstractions because they're

easy for humans to understand; you choose whichever make the

proof shorter. I think this is true for the sciences generally.

Scientific ideas are not meant to be ergonomic.Over in the arts, things are very different. Design is

all about people. The human body is a strange

thing, but when you're designing a chair,

that's what you're designing for, and there's no way around it.

All the arts have to pander to the interests and limitations

of humans. In painting, for example, all other things being

equal a painting with people in it will be more interesting than

one without. It is not merely an accident of history that

the great paintings of the Renaissance are all full of people.

If they hadn't been, painting as a medium wouldn't have the prestige

that it does.Like it or not, programming languages are also for people,

and I suspect the human brain is just as lumpy and idiosyncratic

as the human body. Some ideas are easy for people to grasp

and some aren't. For example, we seem to have a very limited

capacity for dealing with detail. It's this fact that makes

programing languages a good idea in the first place; if we

could handle the detail, we could just program in machine

language.Remember, too, that languages are not

primarily a form for finished programs, but something that

programs have to be developed in. Anyone in the arts could

tell you that you might want different mediums for the

two situations. Marble, for example, is a nice, durable

medium for finished ideas, but a hopelessly inflexible one

for developing new ideas.A program, like a proof,

is a pruned version of a tree that in the past has had

false starts branching off all over it. So the test of

a language is not simply how clean the finished program looks

in it, but how clean the path to the finished program was.

A design choice that gives you elegant finished programs

may not give you an elegant design process. For example,

I've written a few macro-defining macros full of nested

backquotes that look now like little gems, but writing them

took hours of the ugliest trial and error, and frankly, I'm still

not entirely sure they're correct.We often act as if the test of a language were how good

finished programs look in it.

It seems so convincing when you see the same program

written in two languages, and one version is much shorter.

When you approach the problem from the direction of the

arts, you're less likely to depend on this sort of

test. You don't want to end up with a programming

language like marble.For example, it is a huge win in developing software to

have an interactive toplevel, what in Lisp is called a

read-eval-print loop. And when you have one this has

real effects on the design of the language. It would not

work well for a language where you have to declare

variables before using them, for example. When you're

just typing expressions into the toplevel, you want to be

able to set x to some value and then start doing things

to x. You don't want to have to declare the type of x

first. You may dispute either of the premises, but if

a language has to have a toplevel to be convenient, and

mandatory type declarations are incompatible with a

toplevel, then no language that makes type declarations

mandatory could be convenient to program in.In practice, to get good design you have to get close, and stay

close, to your users. You have to calibrate your ideas on actual

users constantly, especially in the beginning. One of the reasons

Jane Austen's novels are so good is that she read them out loud to

her family. That's why she never sinks into self-indulgently arty

descriptions of landscapes,

or pretentious philosophizing. (The philosophy's there, but it's

woven into the story instead of being pasted onto it like a label.)

If you open an average "literary" novel and imagine reading it out loud

to your friends as something you'd written, you'll feel all too

keenly what an imposition that kind of thing is upon the reader.In the software world, this idea is known as Worse is Better.

Actually, there are several ideas mixed together in the concept of

Worse is Better, which is why people are still arguing about

whether worse

is actually better or not. But one of the main ideas in that

mix is that if you're building something new, you should get a

prototype in front of users as soon as possible.The alternative approach might be called the Hail Mary strategy.

Instead of getting a prototype out quickly and gradually refining

it, you try to create the complete, finished, product in one long

touchdown pass. As far as I know, this is a

recipe for disaster. Countless startups destroyed themselves this

way during the Internet bubble. I've never heard of a case

where it worked.What people outside the software world may not realize is that

Worse is Better is found throughout the arts.

In drawing, for example, the idea was discovered during the

Renaissance. Now almost every drawing teacher will tell you that

the right way to get an accurate drawing is not to

work your way slowly around the contour of an object, because errors will

accumulate and you'll find at the end that the lines don't meet.

Instead you should draw a few quick lines in roughly the right place,

and then gradually refine this initial sketch.In most fields, prototypes

have traditionally been made out of different materials.

Typefaces to be cut in metal were initially designed

with a brush on paper. Statues to be cast in bronze

were modelled in wax. Patterns to be embroidered on tapestries

were drawn on paper with ink wash. Buildings to be

constructed from stone were tested on a smaller scale in wood.What made oil paint so exciting, when it

first became popular in the fifteenth century, was that you

could actually make the finished work from the prototype.

You could make a preliminary drawing if you wanted to, but you

weren't held to it; you could work out all the details, and

even make major changes, as you finished the painting.You can do this in software too. A prototype doesn't have to

be just a model; you can refine it into the finished product.

I think you should always do this when you can. It lets you

take advantage of new insights you have along the way. But

perhaps even more important, it's good for morale.Morale is key in design. I'm surprised people

don't talk more about it. One of my first

drawing teachers told me: if you're bored when you're

drawing something, the drawing will look boring.

For example, suppose you have to draw a building, and you

decide to draw each brick individually. You can do this

if you want, but if you get bored halfway through and start

making the bricks mechanically instead of observing each one,

the drawing will look worse than if you had merely suggested

the bricks.Building something by gradually refining a prototype is good

for morale because it keeps you engaged. In software, my

rule is: always have working code. If you're writing

something that you'll be able to test in an hour, then you

have the prospect of an immediate reward to motivate you.

The same is true in the arts, and particularly in oil painting.

Most painters start with a blurry sketch and gradually

refine it.

If you work this way, then in principle

you never have to end the day with something that actually

looks unfinished. Indeed, there is even a saying among

painters: "A painting is never finished, you just stop

working on it." This idea will be familiar to anyone who

has worked on software.Morale is another reason that it's hard to design something

for an unsophisticated user. It's hard to stay interested in

something you don't like yourself. To make something

good, you have to be thinking, "wow, this is really great,"

not "what a piece of shit; those fools will love it."Design means making things for humans. But it's not just the

user who's human. The designer is human too.Notice all this time I've been talking about "the designer."

Design usually has to be under the control of a single person to

be any good. And yet it seems to be possible for several people

to collaborate on a research project. This seems to

me one of the most interesting differences between research and

design.There have been famous instances of collaboration in the arts,

but most of them seem to have been cases of molecular bonding rather

than nuclear fusion. In an opera it's common for one person to

write the libretto and another to write the music. And during the Renaissance,

journeymen from northern

Europe were often employed to do the landscapes in the

backgrounds of Italian paintings. But these aren't true collaborations.

They're more like examples of Robert Frost's

"good fences make good neighbors." You can stick instances

of good design together, but within each individual project,

one person has to be in control.I'm not saying that good design requires that one person think

of everything. There's nothing more valuable than the advice

of someone whose judgement you trust. But after the talking is

done, the decision about what to do has to rest with one person.Why is it that research can be done by collaborators and

design can't? This is an interesting question. I don't

know the answer. Perhaps,

if design and research converge, the best research is also

good design, and in fact can't be done by collaborators.

A lot of the most famous scientists seem to have worked alone.

But I don't know enough to say whether there

is a pattern here. It could be simply that many famous scientists

worked when collaboration was less common.Whatever the story is in the sciences, true collaboration

seems to be vanishingly rare in the arts. Design by committee is a

synonym for bad design. Why is that so? Is there some way to

beat this limitation?I'm inclined to think there isn't-- that good design requires

a dictator. One reason is that good design has to

be all of a piece. Design is not just for humans, but

for individual humans. If a design represents an idea that

fits in one person's head, then the idea will fit in the user's

head too.Related:Japanese TranslationTaste for MakersRomanian TranslationSpanish Translation

A Plan for Spam

Like to build things? Try Hacker

News.

August 2002(This article describes the spam-filtering techniques

used in the spamproof web-based mail reader we

built to exercise Arc. An

improved algorithm is described in Better

Bayesian Filtering.)I think it's possible to stop spam, and that

content-based filters are the way to do it.

The Achilles heel of the spammers is their message.

They can circumvent any other barrier you set up. They have so far, at

least. But they have to deliver their message, whatever it

is. If we can write software that recognizes their messages,

there is no way they can get around that.\_ \_ \_To the recipient, spam is easily recognizable. If you hired

someone to read your mail and discard the spam, they would

have little trouble doing it. How much do we have

to do, short of AI, to automate this process?I think we will be able to solve the problem with fairly

simple algorithms. In fact, I've found that you can filter

present-day spam acceptably well using nothing more than a

Bayesian combination of the spam probabilities of individual

words. Using a slightly tweaked (as described below) Bayesian

filter, we now miss less than 5 per 1000 spams, with 0 false positives.The statistical approach is not usually the first one people

try when they write spam filters. Most hackers' first instinct is

to try to write software that recognizes individual properties of

spam. You look at spams

and you think, the gall of these guys to try sending me mail

that begins "Dear Friend" or has a subject line that's all

uppercase and ends in eight exclamation points. I can filter

out that stuff with about one line of code.And so you do,

and in the beginning it works. A few simple rules will take

a big bite out of your incoming spam. Merely looking

for the word "click" will catch 79.7% of the

emails in my spam corpus, with only 1.2% false positives.I spent about six months writing software that looked for

individual spam features before I tried the statistical

approach. What I found was that recognizing that last few

percent of spams got very hard, and that as I

made the filters stricter I got more false positives.False positives are innocent emails that get mistakenly

identified as spams.

For most users,

missing legitimate email is

an order of magnitude worse than receiving spam, so a

filter that yields false positives is like an acne cure

that carries a risk of death to the patient.The more spam a user gets, the less

likely he'll be to notice one innocent mail sitting in his

spam folder. And strangely enough, the better your spam filters get,

the more dangerous false positives become, because when the

filters are really good, users will be more likely to

ignore everything they catch.I don't know why I avoided trying the statistical approach

for so long. I think it was because I got addicted to

trying to identify spam features myself, as if I were playing

some kind of competitive game with the spammers. (Nonhackers

don't often realize this, but most hackers are very competitive.)

When I did try statistical analysis, I

found immediately that it was much cleverer than I had been.

It discovered, of course, that terms like "virtumundo" and

"teens" were good indicators of spam. But it also

discovered that "per" and "FL" and "ff0000" are good

indicators of spam. In fact, "ff0000" (html for bright red)

turns out to be as good an indicator of spam as any

pornographic term.\_ \_ \_Here's a sketch of how I do statistical filtering. I start

with one corpus of spam and one of nonspam mail. At the

moment each one has about 4000 messages in it. I scan

the entire text, including headers and embedded html

and javascript, of each message in each corpus.

I currently consider alphanumeric characters,

dashes, apostrophes, and dollar signs to be part of tokens,

and everything else to be a token separator. (There is

probably room for improvement here.) I ignore tokens that

are all digits, and I also ignore html comments, not even

considering them as token separators.I count the number

of times each token (ignoring case, currently) occurs in

each corpus. At this stage I end up with two large hash

tables, one for each corpus, mapping tokens to number

of occurrences.Next I create a third hash table, this time mapping

each token to the probability that an email containing it is a spam,

which I calculate as follows [1]:

(let ((g (\* 2 (or (gethash word good) 0)))

(b (or (gethash word bad) 0)))

(unless (< (+ g b) 5)

(max .01

(min .99 (float (/ (min 1 (/ b nbad))

(+ (min 1 (/ g ngood))

(min 1 (/ b nbad)))))))))

where word is the token whose probability we're

calculating, good and bad are the hash tables

I created in the first step, and ngood and nbad

are the number of nonspam and spam messages respectively.I explained this as code to show a couple of important details.

I want to bias the probabilities slightly to avoid false

positives, and by trial and error I've found that a good

way to do it is to double all the numbers in good.

This helps to distinguish between words that occasionally

do occur in legitimate email and words that almost never do.

I only consider words that occur more than five times in

total (actually, because of the doubling, occurring three

times in nonspam mail would be enough). And then there is

the question of what probability to assign to words that

occur in one corpus but not the other. Again by trial and

error I chose .01 and .99. There may be room for tuning

here, but as the corpus grows such tuning will happen

automatically anyway.The especially observant will notice that while I consider

each corpus to be a single long stream of text for purposes

of counting occurrences, I use the number of emails in

each, rather than their combined length, as the divisor

in calculating spam probabilities. This adds another

slight bias to protect against false positives.When new mail arrives, it is scanned into tokens, and

the most interesting fifteen tokens, where interesting is

measured by how far their spam probability is from a

neutral .5, are used to calculate the probability that

the mail is spam. If probs

is a list of the fifteen individual probabilities, you

calculate the

combined probability thus:

(let ((prod (apply #'\* probs)))

(/ prod (+ prod (apply #'\* (mapcar #'(lambda (x)

(- 1 x))

probs)))))

One question that arises in

practice is what probability to assign to a word you've

never seen, i.e. one that doesn't occur in the hash table

of word probabilities. I've found, again by trial and

error, that .4 is a good number to use. If you've never

seen a word before, it is probably fairly innocent; spam

words tend to be all too familiar.There are examples of this algorithm being applied to

actual emails in an appendix at the end.I treat mail as spam if the algorithm above gives it a

probability of more than .9 of being spam. But in practice

it would not matter much where I put this threshold, because

few probabilities end up in the middle of the range.\_ \_ \_One great advantage of the statistical approach is that you

don't have to read so many spams. Over the past six months,

I've read literally thousands of spams, and it is really

kind of demoralizing. Norbert Wiener said if you compete

with slaves you become a slave, and there is something

similarly degrading about competing with spammers. To

recognize individual spam features you have to try to get

into the mind of the spammer, and frankly I want to spend

as little time inside the minds of spammers as possible.But the real advantage of the Bayesian approach, of course,

is that you know what

you're measuring. Feature-recognizing filters like

SpamAssassin assign a spam "score" to email. The Bayesian

approach assigns an actual probability. The problem with

a "score" is that no one knows what it means. The user

doesn't know what it means, but worse still, neither does

the developer of the filter. How many points should an

email get for having the word "sex" in it? A probability

can of course be mistaken, but there is little ambiguity

about what it means, or how evidence should be combined

to calculate it. Based on my corpus, "sex" indicates

a .97 probability of the containing email being a spam,

whereas "sexy" indicates .99 probability.

And Bayes' Rule, equally unambiguous, says that an email

containing both words would, in the (unlikely)

absence of any other evidence, have a 99.97% chance of

being a spam.Because it is measuring probabilities, the Bayesian approach

considers all the evidence in the email, both good and bad.

Words that occur disproportionately rarely

in spam (like "though" or "tonight" or "apparently")

contribute as much to decreasing the probability as

bad words like "unsubscribe" and "opt-in" do to

increasing it. So an otherwise innocent email that happens

to include the word "sex" is not going to get tagged as spam.Ideally, of course, the probabilities should be calculated

individually for each user. I get a lot of email containing

the word "Lisp", and (so far) no spam that does. So a word

like that is effectively a kind of password for sending

mail to me. In my earlier spam-filtering software, the user

could set up a list of such words and mail containing

them would automatically get past the filters. On my

list I put words like "Lisp" and also my zipcode, so

that (otherwise rather spammy-sounding) receipts from

online orders would get through. I thought I was being

very clever, but I found that the Bayesian filter did the

same thing for me, and moreover discovered of a lot of words I

hadn't thought of.When I said at the start that our filters let through less than

5 spams per 1000 with 0 false positives, I'm talking about

filtering my mail based on a corpus of my mail. But these

numbers are not misleading, because that is the approach I'm

advocating: filter each user's mail based on the spam and

nonspam mail he receives. Essentially, each user should

have two delete buttons, ordinary delete and delete-as-spam.

Anything deleted as spam goes into the spam corpus,

and everything else goes into the nonspam corpus.You could start

users with a seed filter, but ultimately each user should have

his own per-word probabilities based on the actual mail he

receives. This (a) makes the filters more effective, (b) lets

each user decide their own precise definition of spam,

and (c) perhaps best of all makes it hard for spammers

to tune mails to get through the filters. If a lot of the

brain of the filter is in the individual databases, then

merely tuning spams to get through the seed filters

won't guarantee anything about how well they'll get through

individual users' varying and much more trained filters.Content-based spam filtering is often combined with a whitelist,

a list of senders whose mail can be accepted with no filtering.

One easy way to build such a

whitelist is to keep a list of every address the user has

ever sent mail to. If a mail reader has a delete-as-spam

button then you could also add the from address

of every email the user has deleted as ordinary trash.I'm an advocate of whitelists, but more as a way to save

computation than as a way to improve filtering. I used to think that

whitelists would make filtering easier, because you'd

only have to filter email from people you'd never heard

from, and someone sending you mail for the first time is

constrained by convention in what they can say to you.

Someone you already know might send you an email talking about sex,

but someone sending you mail for the first time would not

be likely to. The problem is, people can have more than one

email address, so a new from-address doesn't guarantee that

the sender is writing to you for the first time.

It is not unusual

for an old friend (especially if he is a hacker) to suddenly

send you an email with a new from-address, so you can't

risk false positives by filtering mail from unknown

addresses especially stringently.In a sense, though, my filters do themselves embody a kind

of whitelist (and blacklist) because they are based on

entire messages, including the headers. So to that

extent they "know" the email addresses of trusted senders

and even the routes by which mail gets from them to me.

And they know the same about spam, including the server

names, mailer versions, and protocols.\_ \_ \_If I thought that I could keep up current rates of spam

filtering, I would consider this problem solved. But it

doesn't mean much to be able to filter out most present-day

spam, because spam evolves.

Indeed, most

antispam techniques so far have been like pesticides that

do nothing more than create a new, resistant strain of bugs.I'm more hopeful about Bayesian filters, because they evolve

with the spam. So as spammers start using "c0ck"

instead of "cock" to evade simple-minded spam filters

based on individual words, Bayesian filters automatically

notice. Indeed, "c0ck" is far more damning evidence than

"cock", and Bayesian filters know precisely how much more.Still, anyone who proposes a plan for spam filtering has to

be able to answer the question: if the spammers knew

exactly what you were doing,

how well could they get past you? For example, I think that if

checksum-based spam filtering becomes a serious obstacle,

the spammers will just

switch to mad-lib techniques for generating message bodies.To beat Bayesian filters, it would not be enough for spammers

to make their emails unique or to stop using individual

naughty words. They'd have to make their mails indistinguishable

from your ordinary mail. And this I think would severely

constrain them. Spam is mostly sales

pitches, so unless your regular mail is all sales pitches,

spams will inevitably have a different character. And

the spammers would also, of course, have to change (and keep

changing) their whole infrastructure, because otherwise

the headers would look as bad to the Bayesian filters as ever,

no matter what they did to the message body. I don't know

enough about the infrastructure that spammers use to know

how hard it would be to make the headers look innocent, but

my guess is that it would be even harder than making the

message look innocent.Assuming they could solve the problem of the headers,

the spam of the future will probably look something like

this:

Hey there. Thought you should check out the following:

http://www.27meg.com/foo

because that is about as much sales pitch as content-based

filtering will leave the spammer room to make. (Indeed, it

will be hard even to get this past filters, because if everything

else in the email is neutral, the spam probability will hinge on

the url, and it will take some effort to make that look neutral.)Spammers range from businesses running so-called

opt-in lists who don't even try to conceal their identities,

to guys who hijack mail servers to send out spams promoting

porn sites. If we use filtering to whittle their

options down to mails like the one above, that should

pretty much put the spammers on the "legitimate" end of

the spectrum out of business; they feel obliged

by various state laws to include boilerplate about why

their spam is not spam, and how to cancel your

"subscription," and that kind of text is easy to

recognize.(I used to think it was naive to believe that stricter laws

would decrease spam. Now I think that while stricter laws

may not decrease the amount of spam that spammers send,

they can certainly help filters to decrease the amount of

spam that recipients actually see.)All along the spectrum, if you restrict the sales pitches spammers

can make, you will inevitably tend to put them out of

business. That word business is an important one to

remember. The spammers are businessmen. They send spam because

it works. It works because although the response rate

is abominably low (at best 15 per million, vs 3000 per

million for a catalog mailing), the cost, to them, is

practically nothing. The cost is enormous for the recipients,

about 5 man-weeks for each million recipients who spend

a second to delete the spam, but the spammer

doesn't have to pay that.Sending spam does cost the spammer something, though. [2]

So the lower we can get the

response rate-- whether by filtering, or by using filters to force

spammers to dilute their pitches-- the fewer businesses will find it

worth their while to send spam.The reason the spammers use the kinds of

sales

pitches that they do is to increase response rates.

This is possibly even more disgusting

than getting inside the mind of a spammer,

but let's take a quick look inside the mind of someone

who responds to a spam. This person is either

astonishingly credulous or deeply in denial about their

sexual interests. In either case, repulsive or

idiotic as the spam seems to us, it is exciting

to them. The spammers wouldn't say these things if they

didn't sound exciting. And "thought you

should check out the following" is just not going to

have nearly the pull with the spam recipient as

the kinds of things that spammers say now.

Result: if it can't contain exciting sales pitches,

spam becomes less effective as a marketing vehicle,

and fewer businesses want to use it.That is the big win in the end. I started writing spam

filtering software because I didn't want have to look at

the stuff anymore.

But if we get good enough at filtering

out spam, it will stop working, and the spammers

will actually stop sending it.\_ \_ \_Of all the approaches to fighting spam, from software to laws,

I believe Bayesian filtering will be the single most

effective. But I also

think that the more different kinds of antispam efforts

we undertake, the better, because any measure that

constrains spammers will tend to make filtering easier.

And even within the world of content-based filtering, I think

it will be a good thing if there are many different kinds

of software being used simultaneously. The more different

filters there are, the harder it will be for

spammers to tune spams to get through them.

Appendix: Examples of FilteringHere is an example of a spam that arrived while I was writing

this article. The fifteen most interesting words in this spam are:

qvp0045

indira

mx-05

intimail

$7500

freeyankeedom

cdo

bluefoxmedia

jpg

unsecured

platinum

3d0

qves

7c5

7c266675

The words are a mix of stuff from the headers and from the

message body, which is typical of spam. Also typical of spam

is that every one of these words has a spam probability,

in my database, of .99. In fact there are more than fifteen words

with probabilities of .99, and these are just the first

fifteen seen.Unfortunately that makes this email a boring example of

the use of Bayes' Rule. To see an interesting variety of

probabilities we have to look at this actually quite

atypical spam.The fifteen most interesting words in this spam, with their probabilities,

are:

madam 0.99

promotion 0.99

republic 0.99

shortest 0.047225013

mandatory 0.047225013

standardization 0.07347802

sorry 0.08221981

supported 0.09019077

people's 0.09019077

enter 0.9075001

quality 0.8921298

organization 0.12454646

investment 0.8568143

very 0.14758544

valuable 0.82347786

This time the evidence is a mix of good and bad. A word like

"shortest" is almost as much evidence for innocence as a

word like "madam" or "promotion" is for guilt. But still the

case for guilt is stronger. If you combine these numbers

according to Bayes' Rule, the resulting probability is .9027."Madam" is obviously from spams beginning

"Dear Sir or Madam." They're not very common, but the

word "madam" never occurs in my legitimate email, and

it's all about the ratio."Republic" scores high because

it often shows up in Nigerian scam emails, and also occurs once

or twice in spams referring to Korea and South Africa.

You might say that it's

an accident that it thus helps identify this spam. But I've

found when examining spam probabilities that there are

a lot of these accidents, and they have an uncanny tendency to

push things in the right direction rather than the wrong one.

In this case, it is not entirely a coincidence that the word

"Republic" occurs in Nigerian scam emails and this spam.

There is a whole class of dubious business propositions involving

less developed countries, and these in turn are more likely

to have names that specify explicitly (because they aren't) that they

are republics.[3]On the other hand, "enter" is a genuine miss. It occurs

mostly in unsubscribe instructions, but here is used in a

completely innocent way. Fortunately the statistical approach is

fairly robust, and can tolerate quite a lot of misses

before the results start to be thrown off.For comparison,

here is an example of that rare bird, a spam that

gets through the filters. Why? Because by sheer chance it happens

to be loaded with words that occur in my actual email:

perl 0.01

python 0.01

tcl 0.01

scripting 0.01

morris 0.01

graham 0.01491078

guarantee 0.9762507

cgi 0.9734398

paul 0.027040077

quite 0.030676773

pop3 0.042199217

various 0.06080265

prices 0.9359873

managed 0.06451222

difficult 0.071706355

There are a couple pieces of good news here. First, this mail

probably wouldn't get through the filters of someone who didn't

happen to specialize in programming languages and have a good

friend called Morris. For the average user, all the top five words here

would be neutral and would not contribute to the spam probability.Second, I think filtering based on word pairs

(see below) might well

catch this one: "cost effective", "setup fee", "money back" -- pretty

incriminating stuff. And of course if they continued to spam me

(or a network I was part of), "Hostex" itself would be

recognized as a spam term.Finally, here is an innocent email.

Its fifteen most interesting words are as follows:

continuation 0.01

describe 0.01

continuations 0.01

example 0.033600237

programming 0.05214485

i'm 0.055427782

examples 0.07972858

color 0.9189189

localhost 0.09883721

hi 0.116539136

california 0.84421706

same 0.15981844

spot 0.1654587

us-ascii 0.16804294

what 0.19212411

Most of the words here indicate the mail is an innocent one.

There are two bad smelling words, "color"

(spammers love colored fonts) and "California"

(which occurs in testimonials and also in menus in

forms), but they are not enough to outweigh obviously

innocent words like "continuation" and "example".It's interesting that "describe" rates as so thoroughly

innocent. It hasn't occurred in a

single one of my 4000 spams. The data turns out to be

full of such surprises. One of the things you learn

when you analyze spam texts is how

narrow a subset of the language spammers operate in. It's

that fact, together with the equally characteristic vocabulary

of any individual user's mail, that makes Bayesian filtering

a good bet.Appendix: More IdeasOne idea that I haven't tried yet is to filter based on

word pairs, or even triples, rather than individual words.

This should yield a much sharper estimate of the probability.

For example, in my current database, the word "offers"

has a probability of .96. If you based the probabilities

on word pairs, you'd end up with "special offers"

and "valuable offers" having probabilities of .99

and, say, "approach offers" (as in "this approach offers")

having a probability of .1 or less.The reason I haven't done this is that filtering based on

individual words already works so well. But it does

mean that there is room to tighten the filters if spam

gets harder to detect.

(Curiously, a filter based on word pairs would be

in effect a Markov-chaining text generator running

in reverse.)Specific spam features (e.g. not seeing the recipient's

address in the to: field) do of course have value in

recognizing spam. They can be considered in this

algorithm by treating them as virtual words. I'll probably

do this in future versions, at least for a handful of the

most egregious spam indicators. Feature-recognizing

spam filters are right in many details; what they lack

is an overall discipline for combining evidence.Recognizing nonspam features may be more important than

recognizing spam features. False positives are such a

worry that they demand extraordinary measures. I will

probably in future versions add a second level of testing

designed specifically to avoid false positives. If a

mail triggers this second level of filters it will be accepted

even if its spam probability is above the threshold.I don't expect this second level of filtering to be Bayesian.

It will inevitably

be not only ad hoc, but based on guesses, because the number of

false positives will not tend to be large enough to notice patterns.

(It is just as well, anyway, if a backup system doesn't rely on the same

technology as the primary system.)Another thing I may try in the future is to focus extra attention

on specific parts of the email. For example, about 95% of current

spam includes the url of a site they want

you to visit. (The remaining 5% want you to call a phone number,

reply by email or to a US mail address, or in a few

cases to buy a certain stock.) The url is in such cases

practically enough by itself to determine whether the email

is spam.Domain names differ from the rest of the text in

a (non-German) email in that they often consist of several

words stuck together. Though computationally expensive

in the general case, it might be worth trying to

decompose them. If a filter has never seen the

token "xxxporn" before it will have an individual spam

probability of .4, whereas "xxx" and "porn" individually

have probabilities (in my corpus) of .9889 and .99

respectively, and a combined probability of .9998.I expect decomposing domain names to become more

important as spammers are gradually forced to stop using

incriminating words in the text of their messages. (A url

with an ip address is of course an extremely incriminating sign,

except in the mail of a few sysadmins.)It might be a good idea to have a cooperatively maintained

list of urls promoted by spammers. We'd need a trust metric

of the type studied by Raph Levien to prevent malicious

or incompetent submissions, but if we had such a thing it

would provide a boost to any filtering software. It would

also be a convenient basis for boycotts.Another way to test dubious urls would be to send out a

crawler to look at the site before the user looked at the

email mentioning it. You could use a Bayesian filter to

rate the site just as you would an email, and whatever

was found on the site could be included in calculating

the probability of the email being a spam. A url that led

to a redirect would of course be especially suspicious.One cooperative project that I think really would be a good

idea would be to accumulate a giant corpus of spam. A large,

clean corpus is the key to making Bayesian filtering work

well. Bayesian filters could actually use the corpus as

input. But such a corpus would be useful for other kinds

of filters too, because it could be used to test them.Creating such a corpus poses some technical problems. We'd

need trust metrics to prevent malicious or incompetent

submissions, of course. We'd also need ways of erasing

personal information (not just to-addresses and ccs, but

also e.g. the arguments to unsubscribe urls, which often

encode the to-address) from mails in the corpus. If anyone

wants to take on this project, it would be a good thing for

the world.Appendix: Defining SpamI think there is a rough

consensus on what spam is, but it would be useful to have

an explicit definition. We'll need to do this if we want to establish

a central corpus of spam, or even to compare spam filtering

rates meaningfully.To start with, spam is not unsolicited commercial email.

If someone in my neighborhood heard that I was looking for an old

Raleigh three-speed in good condition, and sent me an email

offering to sell me one, I'd be delighted, and yet this

email would be both commercial and unsolicited. The

defining feature of spam (in fact, its raison d'etre)

is not that it is unsolicited, but that it is automated.It is merely incidental, too, that spam is usually commercial.

If someone started sending mass email to support some political

cause, for example, it would be just as much spam as email

promoting a porn site.I propose we define spam as unsolicited automated email.

This definition thus includes some email

that many legal definitions of spam don't. Legal definitions

of spam, influenced presumably by lobbyists, tend to exclude

mail sent by companies that have an "existing relationship" with

the recipient. But buying something from a company, for

example, does not imply that you have solicited

ongoing email from them.

If I order something from an online

store, and they then send me a stream of spam, it's still

spam.Companies sending spam often give you a way to "unsubscribe,"

or ask you to go to their site and change your "account

preferences" if you want to stop getting spam. This is

not enough to stop the mail from being spam. Not opting out

is not the same as opting in. Unless the

recipient explicitly checked a clearly labelled box (whose

default was no) asking to receive the email, then it is spam.In some business relationships, you do implicitly solicit

certain kinds of mail. When you order online, I think you

implicitly solicit a receipt, and notification when the

order ships.

I don't mind when Verisign sends me mail warning that

a domain name is about to expire (at least, if they are the

actual

registrar for it). But when Verisign sends me

email offering a FREE Guide to Building My

E-Commerce Web Site, that's spam.

Notes:[1] The examples in this article are translated

into Common Lisp for, believe it or not, greater accessibility.

The application described here is one that we wrote in order to

test a new Lisp dialect called Arc that is

not yet released.[2] Currently the lowest rate seems to be about $200 to send a million spams.

That's very cheap, 1/50th of a cent per spam.

But filtering out 95%

of spam, for example, would increase the spammers' cost to reach

a given audience by a factor of 20. Few can have

margins big enough to absorb that.[3] As a rule of thumb, the more qualifiers there are before the

name of a country, the more corrupt the rulers. A

country called The Socialist People's Democratic Republic

of X is probably the last place in the world you'd want to live.

Thanks to Sarah Harlin for reading drafts of this; Daniel Giffin (who is

also writing the production Arc interpreter) for several good ideas about

filtering and for creating our mail infrastructure; Robert Morris,

Trevor Blackwell and Erann Gat for many discussions about spam; Raph

Levien for advice about trust metrics; and Chip Coldwell

and Sam Steingold for advice about statistics.

You'll find this essay and 14 others in

Hackers & Painters.

More Info:Plan for Spam FAQBetter Bayesian FilteringFilters that Fight BackWill Filters Kill Spam?Japanese TranslationSpanish TranslationChinese TranslationProbabilitySpam is DifferentFilters vs. BlacklistsTrust MetricsFiltering ResearchMicrosoft PatentSlashdot ArticleThe Wrong WayLWN: Filter ComparisonCRM114 gets 99.87%

Revenge of the Nerds

Want to start a startup? Get funded by

Y Combinator.

May 2002

"We were after the C++ programmers. We managed to drag a

lot of them about halfway to Lisp."- Guy Steele, co-author of the Java spec

In the software business there is an ongoing

struggle between the pointy-headed academics, and another

equally formidable force, the pointy-haired bosses. Everyone

knows who the pointy-haired boss is, right? I think most

people in the technology world not only recognize this

cartoon character, but know the actual person in their company

that he is modelled upon.The pointy-haired boss miraculously combines two qualities

that are common by themselves, but rarely seen together:

(a) he knows nothing whatsoever about technology, and

(b) he has very strong opinions about it.Suppose, for example, you need to write a piece of software.

The pointy-haired boss has no idea how this software

has to work, and can't tell one programming language from

another, and yet he knows what language you should write it in.

Exactly. He thinks you should write it in Java.Why does he think this? Let's

take a look inside the brain of the pointy-haired boss. What

he's thinking is something like this. Java is a standard.

I know it must be, because I read about it in the press all the time.

Since it is a standard, I won't get in trouble for using it.

And that also means there will always be lots of Java programmers,

so if the programmers working for me now quit, as programmers

working for me mysteriously always do, I can easily replace

them.Well, this doesn't sound that unreasonable. But it's all

based on one unspoken assumption, and that assumption

turns out to be false. The pointy-haired boss believes that all

programming languages are pretty much equivalent.

If that were true, he would be right on

target. If languages are all equivalent, sure, use whatever

language everyone else is using.But all languages are not equivalent, and I think I can prove

this to you without even getting into the differences between them.

If you asked the pointy-haired boss in 1992 what language

software should be written in, he would have answered with as

little hesitation as he does today. Software should be

written in C++. But if languages are all equivalent, why should the

pointy-haired boss's opinion ever change? In fact, why should

the developers of Java have even bothered to create a new

language?Presumably, if you create a new language, it's because you think

it's better in some way than what people already had. And in fact, Gosling

makes it clear in the first Java white paper that Java

was designed to fix some problems with C++.

So there you have it: languages are not all equivalent.

If you follow the

trail through the pointy-haired boss's brain to Java and then

back through Java's history to its origins, you end up holding

an idea that contradicts the assumption you started with.So, who's right? James Gosling, or the pointy-haired boss?

Not surprisingly, Gosling is right. Some languages are better,

for certain problems, than others. And you know, that raises some

interesting questions. Java was designed to be better, for certain

problems, than C++. What problems? When is Java better and

when is C++? Are there situations where other languages are

better than either of them?Once you start considering this question, you have opened a

real can of worms. If the pointy-haired boss had to think

about the problem in its full complexity, it would make his

brain explode. As long as he considers all languages

equivalent, all he has to do is choose the one

that seems to have the most momentum, and since that is more

a question of fashion than technology, even he

can probably get the right answer.

But if languages vary, he suddenly

has to solve two simultaneous equations, trying to find

an optimal balance between two things he knows nothing

about: the relative suitability of the twenty or so leading

languages for the problem he needs to solve, and the odds of

finding programmers, libraries, etc. for each.

If that's what's on the other side of the door, it

is no surprise that the pointy-haired boss doesn't want to open it.The disadvantage of believing that all programming languages

are equivalent is that it's not true. But the advantage is

that it makes your life a lot simpler.

And I think that's the main reason the idea is so widespread.

It is a comfortable idea.We know that Java must be pretty good, because it is the

cool, new programming language. Or is it? If you look at the world of

programming languages from a distance, it looks like Java is

the latest thing. (From far enough away, all you can see is

the large, flashing billboard paid for by Sun.)

But if you look at this world

up close, you find that there are degrees of coolness. Within

the hacker subculture, there is another language called Perl

that is considered a lot cooler than Java. Slashdot, for

example, is generated by Perl. I don't think you would find

those guys using Java Server Pages. But there is another,

newer language, called Python, whose users tend to look down on Perl,

and more waiting in the wings.If you look at these languages in order, Java, Perl, Python,

you notice an interesting pattern. At least, you notice this

pattern if you are a Lisp hacker. Each one is progressively

more like Lisp. Python copies even features

that many Lisp hackers consider to be mistakes.

You could translate simple Lisp programs into Python line for line.

It's 2002, and programming languages have almost caught up

with 1958.Catching Up with MathWhat I mean is that

Lisp was first discovered by John McCarthy in 1958,

and popular programming languages are only now

catching up with the ideas he developed then.Now, how could that be true? Isn't computer technology something

that changes very rapidly? I mean, in 1958, computers were

refrigerator-sized behemoths with the processing power of

a wristwatch. How could any technology that old even be

relevant, let alone superior to the latest developments?I'll tell you how. It's because Lisp was not really

designed to be a programming language, at least not in the sense

we mean today. What we mean by a programming language is

something we use to tell a computer what to do. McCarthy

did eventually intend to develop a programming language in

this sense, but the Lisp that we actually ended up with was based

on something separate that he did as a

theoretical exercise-- an effort

to define a more convenient alternative to the Turing Machine.

As McCarthy said later,

Another way to show that Lisp was neater than Turing machines

was to write a universal Lisp function

and show that it is briefer and more comprehensible than the

description of a universal Turing machine.

This was the Lisp function eval...,

which computes the value of

a Lisp expression....

Writing eval required inventing a notation representing Lisp

functions as Lisp data, and such a notation

was devised for the purposes of the paper with no thought that

it would be used to express Lisp programs in practice.

What happened next was that, some time in late 1958, Steve Russell,

one of McCarthy's

grad students, looked at this definition of eval and realized

that if he translated it into machine language, the result

would be a Lisp interpreter.This was a big surprise at the time.

Here is what McCarthy said about it later in an interview:

Steve Russell said, look, why don't I program this eval..., and

I said to him, ho, ho, you're confusing theory with practice,

this eval is intended for reading, not for

computing. But he went ahead and did it. That is, he compiled the eval

in my paper into [IBM] 704 machine

code, fixing bugs, and then advertised this as a Lisp interpreter,

which it certainly was. So at that point Lisp

had essentially the form that it has today....

Suddenly, in a matter of weeks I think, McCarthy found his theoretical

exercise transformed into an actual programming language-- and a

more powerful one than he had intended.So the short explanation of why this 1950s language is not

obsolete is that it was not technology but math, and

math doesn't get stale. The right thing to compare Lisp

to is not 1950s hardware, but, say, the Quicksort

algorithm, which was discovered in 1960 and is still

the fastest general-purpose sort.There is one other language still

surviving from the 1950s, Fortran, and it represents the

opposite approach to language design. Lisp was a

piece of theory that unexpectedly got turned into a

programming language. Fortran was developed intentionally as

a programming language, but what we would now consider a

very low-level one.Fortran I, the language that was

developed in 1956, was a very different animal from present-day

Fortran. Fortran I was pretty much assembly

language with math. In some ways it was less

powerful than more recent assembly languages; there were no

subroutines, for example, only branches.

Present-day Fortran is now arguably closer to Lisp than to

Fortran I.Lisp and Fortran were the trunks of two separate evolutionary trees,

one rooted in math and one rooted in machine architecture.

These two trees have been converging ever since.

Lisp started out powerful, and over the next twenty years

got fast. So-called mainstream languages started out

fast, and over the next forty years gradually got more powerful,

until now the most advanced

of them are fairly close to Lisp.

Close, but they are still missing a few things....What Made Lisp DifferentWhen it was first developed, Lisp embodied nine new

ideas. Some of these we now take for granted, others are

only seen in more advanced languages, and two are still

unique to Lisp. The nine ideas are, in order of their

adoption by the mainstream,

Conditionals. A conditional is an if-then-else

construct. We take these for granted now, but Fortran I

didn't have them. It had only a conditional goto

closely based on the underlying machine instruction. A function type. In Lisp, functions are

a data type just like integers or strings.

They have a literal representation, can be stored in variables,

can be passed as arguments, and so on. Recursion. Lisp was the first programming language to

support it. Dynamic typing. In Lisp, all variables

are effectively pointers. Values are what

have types, not variables, and assigning or binding

variables means copying pointers, not what they point to. Garbage-collection. Programs composed of expressions. Lisp programs are

trees of expressions, each of which returns a value.

This is in contrast to Fortran

and most succeeding languages, which distinguish between

expressions and statements.It was natural to have this

distinction in Fortran I because

you could not nest statements. And

so while you needed expressions for math to work, there was

no point in making anything else return a value, because

there could not be anything waiting for it.This limitation

went away with the arrival of block-structured languages,

but by then it was too late. The distinction between

expressions and statements was entrenched. It spread from

Fortran into Algol and then to both their descendants. A symbol type. Symbols are effectively pointers to strings

stored in a hash table. So

you can test equality by comparing a pointer,

instead of comparing each character. A notation for code using trees of symbols and constants. The whole language there all the time. There is

no real distinction between read-time, compile-time, and runtime.

You can compile or run code while reading, read or run code

while compiling, and read or compile code at runtime.Running code at read-time lets users reprogram Lisp's syntax;

running code at compile-time is the basis of macros; compiling

at runtime is the basis of Lisp's use as an extension

language in programs like Emacs; and reading at runtime

enables programs to communicate using s-expressions, an

idea recently reinvented as XML.

When Lisp first appeared, these ideas were far

removed from ordinary programming practice, which was

dictated largely by the hardware available in the late 1950s.

Over time, the default language, embodied

in a succession of popular languages, has

gradually evolved toward Lisp. Ideas 1-5 are now widespread.

Number 6 is starting to appear in the mainstream.

Python has a form of 7, though there doesn't seem to be

any syntax for it.As for number 8, this may be the most interesting of the

lot. Ideas 8 and 9 only became part of Lisp

by accident, because Steve Russell implemented

something McCarthy had never intended to be implemented.

And yet these ideas turn out to be responsible for

both Lisp's strange appearance and its most distinctive

features. Lisp looks strange not so much because

it has a strange syntax as because it has no syntax;

you express programs directly in the parse trees that

get built behind the scenes when other languages are

parsed, and these trees are made

of lists, which are Lisp data structures.Expressing the language in its own data structures turns

out to be a very powerful feature. Ideas 8 and 9

together mean that you

can write programs that write programs. That may sound

like a bizarre idea, but it's an everyday thing in Lisp.

The most common way to do it is with something called a

macro.The term "macro" does not mean in Lisp what it means in other

languages.

A Lisp macro can be anything from an abbreviation

to a compiler for a new language.

If you want to really understand Lisp,

or just expand your programming horizons, I would

learn more about macros.Macros (in the Lisp sense) are still, as far as

I know, unique to Lisp.

This is partly because in order to have macros you

probably have to make your language look as strange as

Lisp. It may also be because if you do add that final

increment of power, you can no

longer claim to have invented a new language, but only

a new dialect of Lisp.I mention this mostly

as a joke, but it is quite true. If you define

a language that has car, cdr, cons, quote, cond, atom,

eq, and

a notation for functions expressed as lists, then you

can build all the rest of Lisp out of it. That is in

fact the defining quality of Lisp: it was in order to

make this so that McCarthy gave Lisp the shape it has.Where Languages MatterSo suppose Lisp does represent a kind of limit

that mainstream languages are approaching asymptotically-- does

that mean you should actually use it to write software?

How much do you lose by using a less powerful language?

Isn't it wiser, sometimes, not to be

at the very edge of innovation?

And isn't popularity to some extent

its own justification? Isn't the pointy-haired boss right,

for example, to want to use a language for which he can easily

hire programmers?There are, of course, projects where the choice of programming

language doesn't matter much. As a

rule, the more demanding the application, the more

leverage you get from using a powerful language. But

plenty of projects are not demanding at all.

Most programming probably consists of writing

little glue programs, and for

little glue programs you

can use any language that you're already

familiar with and that has good libraries for whatever you

need to do. If you just need to feed data from one

Windows app to another, sure, use Visual Basic.You can write little glue programs in Lisp too

(I use it as a desktop calculator), but the biggest win

for languages like Lisp is at the other end of

the spectrum, where you need to write sophisticated

programs to solve hard problems in the face of fierce competition.

A good example is the

airline fare search program that ITA Software licenses to

Orbitz. These

guys entered a market already dominated by two big,

entrenched competitors, Travelocity and Expedia, and

seem to have just humiliated them technologically.The core of ITA's application is a 200,000 line Common Lisp program

that searches many orders of magnitude more possibilities

than their competitors, who apparently

are still using mainframe-era programming techniques.

(Though ITA is also in a sense

using a mainframe-era programming language.)

I have never seen any of ITA's code, but according to

one of their top hackers they use a lot of macros,

and I am not surprised to hear it.Centripetal ForcesI'm not saying there is no cost to using uncommon

technologies. The pointy-haired boss is not completely

mistaken to worry about this. But because he doesn't understand

the risks, he tends to magnify them.I can think of three problems that could arise from using

less common languages. Your programs might not work well with

programs written in other languages. You might have fewer

libraries at your disposal. And you might have trouble

hiring programmers.How much of a problem is each of these? The importance of

the first varies depending on whether you have control

over the whole system. If you're writing software that has

to run on a remote user's machine on top of a buggy,

closed operating system (I mention no names), there may be

advantages to writing your application in the

same language as the OS.

But if you control the whole system and

have the source code of all the parts, as ITA presumably does, you

can use whatever languages you want. If

any incompatibility arises, you can fix it yourself.In server-based applications you can

get away with using the most advanced technologies,

and I think this is the main

cause of what Jonathan Erickson calls the "programming language

renaissance." This is why we even hear about new

languages like Perl and Python. We're not hearing about these

languages because people are using them to write Windows

apps, but because people are using them on servers. And as

software shifts

off the desktop and onto servers (a future even

Microsoft seems resigned to), there will be less

and less pressure to use middle-of-the-road technologies.As for libraries, their importance also

depends on the application. For less demanding problems,

the availability of libraries can outweigh the intrinsic power

of the language. Where is the breakeven point? Hard to say

exactly, but wherever it is, it is short of anything you'd

be likely to call an application. If a company considers

itself to be in the software business, and they're writing

an application that will be one of their products,

then it will probably involve several hackers and take at

least six months to write. In a project of that

size, powerful languages probably start to outweigh

the convenience of pre-existing libraries.The third worry of the pointy-haired boss, the difficulty

of hiring programmers, I think is a red herring. How many

hackers do you need to hire, after all? Surely by now we

all know that software is best developed by teams of less

than ten people. And you shouldn't have trouble hiring

hackers on that scale for any language anyone has ever heard

of. If you can't find ten Lisp hackers, then your company is

probably based in the wrong city for developing software.In fact, choosing a more powerful language probably decreases the

size of the team you need, because (a) if you use a more powerful

language you probably won't need as many hackers,

and (b) hackers who work in more advanced languages are likely

to be smarter.I'm not saying that you won't get a lot of pressure to use

what are perceived as "standard" technologies. At Viaweb

(now Yahoo Store),

we raised some eyebrows among VCs and potential acquirers by

using Lisp. But we also raised eyebrows by using

generic Intel boxes as servers instead of

"industrial strength" servers like Suns, for using a

then-obscure open-source Unix variant called FreeBSD instead

of a real commercial OS like Windows NT, for ignoring

a supposed e-commerce standard called

SET that no one now

even remembers, and so on.You can't let the suits make technical decisions for you.

Did it

alarm some potential acquirers that we used Lisp? Some, slightly,

but if we hadn't used Lisp, we wouldn't have been

able to write the software that made them want to buy us.

What seemed like an anomaly to them was in fact

cause and effect.If you start a startup, don't design your product to please

VCs or potential acquirers. Design your product to please

the users. If you win the users, everything else will

follow. And if you don't, no one will care

how comfortingly orthodox your technology choices were.The Cost of Being AverageHow much do you lose by using a less powerful language?

There is actually some data out there about that.The most convenient measure of power is probably

code size.

The point of high-level

languages is to give you bigger abstractions-- bigger bricks,

as it were, so you don't need as many to build

a wall of a given size.

So the more powerful

the language, the shorter the program (not simply in

characters, of course, but in distinct elements).How does a more powerful language enable you to write

shorter programs? One technique you can use, if the language will

let you, is something called

bottom-up programming. Instead of

simply writing your application in the base language, you

build on top of the base language a language for writing

programs like yours, then write your program

in it. The combined code can be much shorter than if you

had written your whole program in the base language-- indeed,

this is how most compression algorithms work.

A bottom-up program should be easier to modify as well,

because in many cases the language layer won't have to change

at all.Code size is important, because the time it takes

to write a program depends mostly on its length.

If your program would be three times as long in another

language, it will take three times as long to write-- and

you can't get around this by hiring more people, because

beyond a certain size new hires are actually a net lose.

Fred Brooks described this phenomenon in his famous

book The Mythical Man-Month, and everything I've seen

has tended to confirm what he said.So how much shorter are your programs if you write them in

Lisp? Most of the numbers I've heard for Lisp

versus C, for example, have been around 7-10x.

But a recent article about ITA in

New

Architect magazine said that

"one line of Lisp can replace 20 lines of C," and since

this article was full of quotes from ITA's president, I

assume they got this number from ITA. If so then

we can put some faith in it; ITA's software includes a lot

of C and C++ as well as Lisp, so they are speaking from

experience.My guess is that these multiples aren't even constant.

I think they increase when

you face harder problems and also when you have smarter

programmers. A really good hacker can squeeze more

out of better tools.As one data point on the curve, at any rate,

if you were to compete with ITA and

chose to write your software in C, they would be able to develop

software twenty times faster than you.

If you spent a year on a new feature, they'd be able to

duplicate it in less than three weeks. Whereas if they spent

just three months developing something new, it would be

five years before you had it too.And you know what? That's the best-case scenario.

When you talk about code-size ratios, you're implicitly assuming

that you can actually write the program in the weaker language.

But in fact there are limits on what programmers can do.

If you're trying to solve a hard problem with a language that's

too low-level, you reach a point where there is just too

much to keep in your head at once.So when I say it would take ITA's imaginary

competitor five years to duplicate something ITA could

write in Lisp in three months, I mean five years

if nothing goes wrong. In fact, the way things work in

most companies, any

development project that would take five years is

likely never to get finished at all.I admit this is an extreme case. ITA's hackers seem to

be unusually smart, and C is a pretty low-level language.

But in a competitive market, even a differential of two or

three to one would

be enough to guarantee that you'd always be behind.A RecipeThis is the kind of possibility that the pointy-haired boss

doesn't even want to think about. And so most of them don't.

Because, you know, when it comes down to it, the pointy-haired

boss doesn't mind if his company gets their ass kicked, so

long as no one can prove it's his fault.

The safest plan for him personally

is to stick close to the center of the herd.Within large organizations, the phrase used to

describe this approach is "industry best practice."

Its purpose is to shield the pointy-haired

boss from responsibility: if he chooses

something that is "industry best practice," and the company

loses, he can't be blamed. He didn't choose, the industry did.I believe this term was originally used to describe

accounting methods and so on. What it means, roughly,

is don't do anything weird. And in accounting that's

probably a good idea. The terms "cutting-edge" and

"accounting" do not sound good together. But when you import

this criterion into decisions about technology, you start

to get the wrong answers.Technology often should be

cutting-edge. In programming languages, as Erann Gat

has pointed out, what "industry best practice" actually

gets you is not the best, but merely the

average. When a decision causes you to develop software at

a fraction of the rate of more aggressive competitors,

"best practice" is a misnomer.

So here we have two pieces of information that I think are

very valuable. In fact, I know it from my own experience.

Number 1, languages vary in power. Number 2, most managers

deliberately ignore this. Between them, these two facts

are literally a recipe for making money. ITA is an example

of this recipe in action.

If you want to win in a software

business, just take on the hardest problem you can find,

use the most powerful language you can get, and wait for

your competitors' pointy-haired bosses to revert to the mean.

Appendix: PowerAs an illustration of what I mean about the relative power

of programming languages, consider the following problem.

We want to write a function that generates accumulators-- a

function that takes a number n, and

returns a function that takes another number i and

returns n incremented by i.(That's incremented by, not plus. An accumulator

has to accumulate.)In Common Lisp this would be

(defun foo (n)

(lambda (i) (incf n i)))

and in Perl 5,

sub foo {

my ($n) = @\_;

sub {$n += shift}

}

which has more elements than the Lisp version because

you have to extract parameters manually in Perl.In Smalltalk the code is slightly longer than in Lisp

foo: n

|s|

s := n.

^[:i| s := s+i. ]

because although in general lexical variables work, you can't

do an assignment to a parameter, so you have to create a

new variable s.In Javascript the example is, again, slightly longer, because

Javascript retains

the distinction between statements and

expressions, so you need explicit return statements

to return values:

function foo(n) {

return function (i) {

return n += i } }

(To be fair, Perl also retains

this distinction, but deals with it in typical Perl fashion

by letting you omit returns.)If you try to translate the Lisp/Perl/Smalltalk/Javascript code into

Python you run into some limitations. Because Python

doesn't fully support lexical variables,

you have to create a data structure to hold the value of n.

And although

Python does have a function data type, there is no

literal representation for one (unless the body is

only a single expression) so you need to create a named

function to return. This is what you end up with:

def foo(n):

s = [n]

def bar(i):

s[0] += i

return s[0]

return bar

Python users might legitimately ask why they can't

just write

def foo(n):

return lambda i: return n += i

or even

def foo(n):

lambda i: n += i

and my guess is that they probably will, one day.

(But if they don't want to wait for Python to evolve the rest

of the way into Lisp, they could always just...)

In OO languages, you can, to a limited extent, simulate

a closure (a function that refers to variables defined in

enclosing scopes) by defining a class with one method

and a field to replace each variable from an enclosing

scope. This makes the programmer do the kind of code

analysis that would be done by the compiler in a language

with full support for lexical scope, and it won't work

if more than one function refers to the same variable,

but it is enough in simple cases like this.Python experts seem to agree that this is the

preferred way to solve the problem in Python, writing

either

def foo(n):

class acc:

def \_\_init\_\_(self, s):

self.s = s

def inc(self, i):

self.s += i

return self.s

return acc(n).inc

or

class foo:

def \_\_init\_\_(self, n):

self.n = n

def \_\_call\_\_(self, i):

self.n += i

return self.n

I include these because I wouldn't want Python

advocates to say I was misrepresenting the language,

but both seem to me more complex than the first

version. You're doing the same thing, setting up

a separate place to hold the accumulator; it's just

a field in an object instead of the head of a list.

And the use of these special,

reserved field names, especially \_\_call\_\_, seems

a bit of a hack.In the rivalry between Perl and Python, the claim of the

Python hackers seems to be that

that Python is a more elegant alternative to Perl, but what

this case shows is that power is the ultimate elegance:

the Perl program is simpler (has fewer elements), even if the

syntax is a bit uglier.How about other languages? In the other languages

mentioned in this talk-- Fortran, C, C++, Java, and

Visual Basic-- it is not clear whether you can actually

solve this problem.

Ken Anderson says that the following code is about as close

as you can get in Java:

public interface Inttoint {

public int call(int i);

}

public static Inttoint foo(final int n) {

return new Inttoint() {

int s = n;

public int call(int i) {

s = s + i;

return s;

}};

}

This falls short of the spec because it only works for

integers. After many email exchanges with Java hackers,

I would say that writing a properly polymorphic version

that behaves like the preceding examples is somewhere

between damned awkward and impossible. If anyone wants to

write one I'd be very curious to see it, but I personally

have timed out.It's not literally true that you can't solve this

problem in other languages, of course. The fact

that all these languages are Turing-equivalent means

that, strictly speaking, you can write any program in

any of them. So how would you do it? In the limit case,

by writing a Lisp

interpreter in the less powerful language.That sounds like a joke, but it happens so often to

varying degrees in large programming projects that

there is a name for the phenomenon, Greenspun's Tenth

Rule:

Any sufficiently

complicated C or Fortran program contains an ad hoc

informally-specified bug-ridden slow implementation of half of

Common Lisp.

If you try to solve a

hard problem, the question is not whether you will use

a powerful enough language, but whether you will (a)

use a powerful language, (b) write a de facto interpreter

for one, or (c) yourself become a human compiler for one.

We see this already

begining to happen in the Python example, where we are

in effect simulating the code that a compiler

would generate to implement a lexical variable.This practice is not only common, but institutionalized. For example,

in the OO world you hear a good deal about

"patterns".

I wonder if these patterns are not sometimes evidence of case (c),

the human compiler, at work. When I see patterns in my programs,

I consider it a sign of trouble. The shape of a program

should reflect only the problem it needs to solve.

Any other regularity in the code is a sign, to me at

least, that I'm using abstractions that aren't powerful

enough-- often that I'm generating by hand the

expansions of some macro that I need to write.Notes

The IBM 704 CPU was about the size of a refrigerator,

but a lot heavier. The CPU weighed 3150 pounds,

and the 4K of RAM was in a separate

box weighing another 4000 pounds. The

Sub-Zero 690, one of the largest household refrigerators,

weighs 656 pounds. Steve Russell also wrote the first (digital) computer

game, Spacewar, in 1962. If you want to trick a pointy-haired boss into letting you

write software in Lisp, you could try telling him it's XML. Here is the accumulator generator in other Lisp dialects:

Scheme: (define (foo n)

(lambda (i) (set! n (+ n i)) n))

Goo: (df foo (n) (op incf n \_)))

Arc: (def foo (n) [++ n \_])

Erann Gat's sad tale about

"industry best practice" at JPL inspired me to address

this generally misapplied phrase. Peter Norvig found that

16 of the 23 patterns in Design Patterns were

"invisible

or simpler" in Lisp. Thanks to the many people who answered my questions about

various languages and/or read drafts of this, including

Ken Anderson, Trevor Blackwell, Erann Gat, Dan Giffin, Sarah Harlin,

Jeremy Hylton, Robert Morris, Peter Norvig, Guy Steele, and Anton

van Straaten.

They bear no blame for any opinions expressed.

Related:Many people have responded to this talk,

so I have set up an additional page to deal with the issues they have

raised: Re: Revenge of the Nerds.It also set off an extensive and often useful discussion on the

LL1

mailing list. See particularly the mail by Anton van Straaten on semantic

compression.Some of the mail on LL1 led me to try to go deeper into the subject

of language power in Succinctness is Power.A larger set of canonical implementations of the accumulator

generator benchmark are collected together on their own page.Japanese Translation, Spanish

Translation,

Chinese Translation

You'll find this essay and 14 others in

Hackers & Painters.

Succinctness is Power

May 2002

"The quantity of meaning compressed into a small space by

algebraic signs, is another circumstance that facilitates

the reasonings we are accustomed to carry on by their aid."- Charles Babbage, quoted in Iverson's Turing Award Lecture

In the discussion about issues raised by Revenge

of the Nerds on the LL1 mailing list, Paul Prescod wrote

something that stuck in my mind.

Python's goal is regularity and readability, not succinctness.

On the face of it, this seems a rather damning thing to claim about a

programming language. As far as I can tell, succinctness = power.

If so, then substituting, we get

Python's goal is regularity and readability, not power.

and this doesn't seem a tradeoff (if it is a tradeoff)

that you'd want to make.

It's not far from saying that Python's goal is not to be effective

as a programming language.Does succinctness = power? This seems to me an important question,

maybe the most important question for anyone interested in

language design, and one that it would be useful to confront

directly. I don't feel sure yet that the answer is a simple yes, but it seems

a good hypothesis to begin with.HypothesisMy hypothesis is that succinctness is power, or is close enough

that except in pathological examples you can treat them as

identical.It seems to me that succinctness is what programming languages are

for. Computers would be just as happy to be told what to

do directly in machine language. I think that the main

reason we take the trouble to develop high-level languages is to

get leverage, so that we can say (and more importantly, think)

in 10 lines of a high-level language what would require 1000

lines of machine language. In other words,

the main point of high-level languages is to make source code smaller.If smaller source code is the purpose of high-level languages, and

the power of something is how well it achieves its purpose, then

the measure of the power of a programming language is how small it

makes your programs.Conversely, a language that doesn't make your programs small is

doing a bad job of what programming languages are supposed to

do, like a knife that doesn't cut well, or printing that's illegible.

MetricsSmall in what sense though? The most common measure of code size is

lines of code. But I think that this metric is the most common because

it is the easiest to measure. I don't think anyone really believes

it is the true test of the length of a program. Different

languages have different conventions for how much you should put

on a line; in C a lot of lines have nothing on them but a delimiter or two.Another easy test is the number of characters in a

program, but this is not very good either; some languages (Perl,

for example) just

use shorter identifiers than others.I think a better measure of the size of a program would be the

number of elements, where an element is anything that

would be a distinct node if you drew a tree representing the

source code. The name of

a variable or function is an element;

an integer or a floating-point number is an element;

a segment of literal text is an element;

an element of a pattern, or a format directive, is an element;

a new block is an element. There are borderline cases

(is -5 two elements or one?) but I think most of them are the

same for every language, so they don't affect comparisons much.This metric needs fleshing out, and

it could require interpretation in the case of specific languages,

but I think it tries to measure the right thing, which is the

number of parts a program has. I think the tree you'd draw in this

exercise is what you have to make in your head in order to

conceive of the program, and so its size is proportionate to the

amount of work you have to do to write or read it.DesignThis kind of metric would allow us to compare different languages,

but that is not, at least for me, its main value. The main value

of the succinctness test is as a guide in designing languages.

The most useful comparison between languages is between two

potential variants of the same language. What can I do in the

language to make programs shorter?If the conceptual load of

a program is proportionate to its complexity, and a given programmer

can tolerate a fixed conceptual load, then this is the same as asking,

what can I do to enable programmers to get the most done? And

that seems to me identical to asking, how can I design a good

language?(Incidentally, nothing makes it more patently obvious that the old

chestnut "all languages are equivalent" is false than designing

languages. When you are designing a new language, you're constantly

comparing two languages-- the language if I did x, and if I didn't-- to

decide which is better. If this were really a meaningless question,

you might as well flip a coin.)Aiming for succinctness seems a good way to find new ideas.

If you can do something that makes many

different programs shorter, it is probably not a coincidence: you have

probably discovered a useful new abstraction. You might even be

able to write a program to help by searching

source code for repeated patterns. Among other languages, those

with a reputation for succinctness would be the ones to look to for

new ideas: Forth, Joy, Icon.ComparisonThe first person to write about these issues, as far as I know, was

Fred Brooks in the Mythical Man Month. He wrote

that programmers seemed to generate about the same

amount of code per day regardless of the language.

When I first read this in my early twenties,

it was a big surprise to me and seemed to have huge implications.

It meant that (a) the only way to get software written faster was to

use a more succinct language, and (b) someone who took the

trouble to do this could leave competitors who didn't in the dust.Brooks' hypothesis, if it's true, seems to be at the very heart of hacking.

In the years since, I've paid close attention to any evidence I could

get on the question, from formal studies to anecdotes about individual

projects. I have seen nothing to contradict him.I have not yet seen evidence that seemed to me conclusive,

and I don't expect to. Studies

like Lutz Prechelt's comparison of programming languages, while

generating the kind of results I expected, tend to use problems that

are too short to be meaningful tests. A better test of a language is

what happens in programs that take a month to write. And the only

real test, if you believe as I do that the main purpose of a language

is to be good to think in (rather than just to tell a computer what to

do once you've thought of it) is what new things you can write in it.

So any language comparison where

you have to meet a predefined spec is testing slightly the wrong

thing.The true test of a language is how well you can discover

and solve new problems, not

how well you can use it to solve a problem someone else has

already formulated. These two are quite different criteria.

In art, mediums like embroidery and mosaic work well if you

know beforehand what you want to make, but are absolutely lousy if

you don't. When you want to discover the image as you make it--

as you have to do with anything as complex as an image of a

person, for example-- you need to use a more fluid medium like pencil or

ink wash or oil paint. And indeed, the way tapestries and mosaics are made in

practice is to make a painting first, then copy it. (The word

"cartoon" was originally used to describe a painting intended for

this purpose).What this means is that we are never likely to have accurate comparisons

of the relative power of programming languages. We'll have precise

comparisons, but not accurate ones. In particular, explicit studies

for the purpose of comparing languages,

because they will probably use small problems, and will necessarily use

predefined problems, will tend to underestimate the power of the

more powerful languages.Reports from the field, though they will necessarily be less precise than

"scientific" studies, are likely to be more meaningful. For example,

Ulf Wiger of Ericsson did a

study that

concluded that Erlang was 4-10x

more succinct than C++, and proportionately faster to develop

software in:

Comparisons between Ericsson-internal development projects indicate

similar line/hour productivity, including all phases of software development,

rather independently of which language (Erlang, PLEX, C, C++, or Java)

was used. What differentiates the different languages then becomes source

code volume.

The study also deals explictly with a point that was

only implicit in Brooks' book (since he measured lines of debugged code):

programs written in more powerful languages tend to have fewer bugs.

That becomes an end in itself, possibly more important than programmer

productivity, in applications like network switches.The Taste TestUltimately, I think you have to go with your gut. What does it feel

like to program in the language? I think the way to find (or design)

the best language is to become hypersensitive to how well a language

lets you think, then choose/design the language that feels best. If

some language feature is awkward or restricting, don't worry, you'll

know about it.Such hypersensitivity will come at a cost. You'll find that you can't

stand programming in clumsy languages. I find it unbearably

restrictive to program in languages without macros, just as someone used

to dynamic typing finds it unbearably restrictive to have to go back to

programming in a language where you have to declare the type of

every variable, and can't make a list of objects of different types.

I'm not the only one. I know many Lisp hackers that this has happened

to. In fact, the most accurate measure of the relative power of programming

languages might be the percentage of people who know the language

who will take any job where they get to use that language, regardless

of the application domain.RestrictivenessI think most hackers know what it means for a language to feel restrictive.

What's happening when you feel that? I think it's the same feeling

you get when the street you want to take is blocked off, and you have to

take a long detour to get where you wanted to go. There is something

you want to say, and the language won't let you.What's really going on here, I think, is that a restrictive language is

one that isn't succinct enough. The problem is not simply that you can't

say what you planned to. It's that the detour the language makes you

take is longer. Try this thought experiment. Suppose there were

some program you wanted to write, and the language wouldn't let you

express it the way you planned to, but instead forced you to write the

program in some other way that was shorter. For me at least,

that wouldn't feel very restrictive. It would be like the street you

wanted to take being blocked off, and the policeman at the

intersection directing you to a shortcut instead of a detour. Great!I think most (ninety percent?) of

the feeling of restrictiveness comes from being forced to make the program

you write in the language longer than one you have in your head.

Restrictiveness is mostly lack of succinctness.

So when a language feels restrictive, what that (mostly) means is that it isn't

succinct enough, and when a language isn't succinct, it will

feel restrictive.ReadabilityThe quote I began with mentions two other qualities, regularity and

readability. I'm not sure what regularity is, or what advantage,

if any, code that is regular and readable has over code that is merely

readable. But I think I know what is meant by readability, and I think

it is also related to succinctness.We have to be careful here to distinguish between the readability of

an individual line of code and the readability of the whole program.

It's the second that matters. I agree that a line of Basic is likely to be

more readable than a line of Lisp. But a program written in Basic is

is going to have more lines than the same program written in

Lisp (especially once you cross over into Greenspunland). The

total effort of reading the Basic program will surely be greater.

total effort = effort per line x number of lines

I'm not as sure that readability is directly proportionate to succinctness

as I am that power is, but certainly succinctness is a factor

(in the mathematical sense; see equation above) in readability.

So it may not even be meaningful to say that the goal of a language is

readability, not succinctness; it could be like saying the goal was readability,

not readability.What readability-per-line does mean, to the user encountering the language

for the first time, is that source code will look unthreatening. So

readability-per-line

could be a good marketing decision, even if it is a bad design

decision. It's isomorphic to the very successful technique of letting

people pay in installments: instead of frightening them with a high

upfront price, you tell them the low monthly payment. Installment plans

are a net lose for the buyer, though, as mere readability-per-line probably

is for the programmer.

The buyer is going to make a lot of those low, low payments; and

the programmer is going to read a lot of those individually readable lines.This tradeoff predates programming languages. If you're used to reading

novels and newspaper articles, your first experience of reading a math

paper can be dismaying. It could take half an hour to read a single page.

And yet, I am pretty sure that the notation is not the problem, even though

it may feel like it is. The math paper is hard to read

because the ideas are hard. If you expressed the same ideas in prose

(as mathematicians had to do before they evolved succinct notations),

they wouldn't be any easier to read, because the paper would grow to the

size of a book.To What Extent?A number of people have rejected

the idea that succinctness = power. I think it would be more useful, instead

of simply arguing that they are the same or aren't, to ask:

to what extent does succinctness = power?

Because clearly succinctness is

a large part of what higher-level languages are for. If it is not all they're

for, then what else are they for, and how important, relatively, are these

other functions?I'm not proposing this just to make the debate more civilized. I really

want to know the answer. When, if ever, is a language too succinct for

its own good?The hypothesis I began with was that, except in pathological examples,

I thought succinctness could be considered identical with power. What

I meant was that in any language anyone would design, they

would be identical, but that if someone wanted to design a language

explicitly to disprove this hyphothesis, they could probably do it. I'm

not even sure of that, actually.Languages, not ProgramsWe should be clear that we are talking about the succinctness

of languages, not of individual programs.

It certainly is possible for individual programs to be written too densely.I wrote about this in On Lisp. A complex macro

may have to save many times its own length to be justified. If writing

some hairy macro could save you ten lines of code every time you use it,

and the macro is itself ten lines of code, then you get a net saving in

lines if you use it more than once. But that could still be a bad move,

because macro definitions are harder to read than ordinary code. You

might have to use the macro ten or twenty times before it yielded a net

improvement in readability.I'm sure every language has such tradeoffs (though I suspect the stakes

get higher as the language gets more powerful). Every programmer must

have seen code that some clever person has made marginally shorter

by using dubious programming tricks.So there is no argument about that-- at least, not from me. Individual

programs can certainly be too succinct for their own good. The question

is, can a language be? Can a language compel programmers to write

code that's short (in elements) at the expense of overall readability?One reason it's hard to imagine a language being too succinct is that if

there were some excessively compact way to phrase something, there would

probably also be a longer way. For example, if you felt Lisp programs using

a lot of macros or higher-order functions were too dense, you could, if you

preferred, write code that was isomorphic to Pascal. If you

don't want to express factorial in Arc as a call to a higher-order function

(rec zero 1 \* 1-)

you can also write out a recursive definition:

(rfn fact (x) (if (zero x) 1 (\* x (fact (1- x)))))

Though I can't off the top of my head think of any examples, I am interested

in the question of whether a language could be too succinct. Are there languages

that force you to write code in a way that is crabbed and incomprehensible?

If anyone has examples, I would be very interested to see them.(Reminder: What I'm looking for are programs that are very dense according

to the metric of "elements" sketched above, not merely programs that are

short because delimiters can be omitted and everything has a one-character name.)

Japanese TranslationRussian TranslationLutz Prechelt: Comparison of Seven LanguagesErann Gat: Lisp vs. JavaPeter Norvig Tries Prechelt's TestMatthias Felleisen: Expressive Power of LanguagesKragen Sitaker: Redundancy and PowerForthJoyIconJK

What Languages Fix

Kevin Kelleher suggested an interesting way to compare programming

languages: to describe each in terms of the problem it

fixes. The surprising thing is how many, and how well, languages can be

described this way.

Algol: Assembly language is too low-level.Pascal: Algol doesn't have enough data types.Modula: Pascal is too wimpy for systems programming.

Simula: Algol isn't good enough at simulations.Smalltalk: Not everything in Simula is an object.Fortran: Assembly language is too low-level.Cobol: Fortran is scary.PL/1: Fortran doesn't have enough data types.Ada: Every existing language is missing something.Basic: Fortran is scary.APL: Fortran isn't good enough at manipulating arrays.J: APL requires its own character set.C: Assembly language is too low-level.C++: C is too low-level.Java: C++ is a kludge. And Microsoft is going to crush us.C#: Java is controlled by Sun.

Lisp: Turing Machines are an awkward way to describe computation.Scheme: MacLisp is a kludge.T: Scheme has no libraries.Common Lisp: There are too many dialects of Lisp.Dylan: Scheme has no libraries, and Lisp syntax is scary.

Perl: Shell scripts/awk/sed are not enough like programming languages.Python: Perl is a kludge.Ruby: Perl is a kludge, and Lisp syntax is scary.Prolog: Programming is not enough like logic.

Japanese TranslationFrench TranslationPortuguese Translation

Taste for Makers

February 2002

"...Copernicus'

aesthetic objections to [equants] provided one essential

motive for his rejection of the Ptolemaic system...."- Thomas Kuhn, The Copernican Revolution"All of us had been trained by Kelly Johnson and believed

fanatically in his insistence that an airplane that looked

beautiful would fly the same way."- Ben Rich, Skunk Works"Beauty is the first test: there is no permanent place in this

world for ugly mathematics."- G. H. Hardy, A Mathematician's Apology

I was talking recently to a friend who teaches

at MIT. His field is hot now and

every year he is inundated by applications from

would-be graduate students. "A lot of them seem smart,"

he said. "What I can't tell is whether they have any kind

of taste."Taste. You don't hear that word much now.

And yet we still need the underlying

concept, whatever we call it. What my friend meant was

that he wanted students who were not just good technicians,

but who could use their technical knowledge to

design beautiful things.Mathematicians call good work "beautiful,"

and so, either now or in the past, have

scientists, engineers, musicians, architects, designers,

writers, and painters.

Is it just a coincidence that they used the same word, or is

there some overlap in what they meant? If there

is an overlap, can we use one field's discoveries

about beauty to help us in another?For those of us who design things, these are not just

theoretical questions. If there is such a thing as

beauty, we need to be able to recognize it. We need

good taste to make good things.

Instead of

treating beauty as an airy abstraction, to be either blathered

about or avoided depending on how one feels about airy

abstractions, let's try considering it as a practical question:

how do you make good stuff?If you mention taste nowadays, a lot of people will tell

you that "taste is subjective."

They believe this because it really feels that

way to them. When they like something, they have no idea

why. It could be because it's beautiful, or because their

mother had one, or because they saw a movie star with one

in a magazine, or because they know it's expensive.

Their thoughts are a tangle of unexamined impulses.Most of us are encouraged, as children, to leave this tangle

unexamined. If you make fun of your little brother for

coloring people green in his coloring book, your

mother is likely to tell you something like "you like to

do it your way and he likes to do it his way."Your mother at this point is not trying to teach you

important truths about aesthetics. She's trying to get

the two of you to stop bickering.Like many of the half-truths adults tell us, this one

contradicts other things they tell us. After dinning

into you that taste is merely a matter of personal preference,

they take you to the museum and tell you that you should

pay attention because Leonardo is a great artist.What goes through the kid's head at this point? What does

he think "great artist" means? After having been

told for years that everyone just likes to do

things their own way, he is

unlikely to head straight for the conclusion that a great

artist is someone whose work is better than the others'.

A far more likely theory, in his Ptolemaic model of

the universe, is that a great artist is something that's

good for you, like broccoli, because someone said so in a book.Saying that taste is just personal preference is a good way

to prevent disputes. The trouble is, it's not true.

You feel this when you start to design things.Whatever job people do, they naturally want to do better.

Football players

like to win games. CEOs like to increase earnings. It's

a matter of pride, and a real pleasure, to get better at

your job. But if

your job is to design things, and there is no such thing

as beauty, then there is no way to get better at your job.

If taste is just personal preference, then everyone's is

already perfect: you like whatever you like, and that's it.As in any job, as you continue to design things, you'll get

better at it. Your tastes will change. And, like anyone

who gets better at their job, you'll know you're getting

better. If so,

your old tastes were

not merely different, but worse. Poof goes the axiom that

taste can't be wrong.Relativism is fashionable at the moment, and that may hamper

you from thinking about taste, even as yours grows.

But if you come out of the closet and admit, at least to yourself,

that there is such a thing as good and bad design, then you

can start to study good design in detail.

How has

your taste changed? When you made mistakes, what

caused you to make them? What have other people learned about

design?Once you start to examine the question, it's surprising how

much different fields' ideas of beauty have in common. The same

principles of good design crop up again and again.Good design is simple. You hear this from math to

painting. In math it means that a shorter proof tends to be

a better one. Where axioms are concerned, especially,

less is more. It means much the same thing in programming.

For architects and designers it means that beauty should

depend on a few carefully chosen structural elements

rather than a profusion of superficial ornament. (Ornament

is not in itself bad, only when it's camouflage on insipid

form.) Similarly, in painting, a

still life of a few carefully observed and solidly

modelled objects will tend to be more interesting than a

stretch of flashy

but mindlessly repetitive painting of, say, a lace collar.

In writing it means: say what you mean

and say it briefly.It seems strange to have to emphasize simplicity.

You'd think simple would be the default. Ornate

is more work. But something seems to come over people

when they try to be creative. Beginning writers adopt

a pompous tone that doesn't sound anything like the way

they speak. Designers trying to be artistic resort to

swooshes and curlicues. Painters discover that they're expressionists.

It's all evasion.

Underneath

the long words or the "expressive" brush strokes, there

is not much going on, and that's frightening.When you're

forced to be simple, you're forced to face the real problem.

When you can't deliver ornament, you have to deliver

substance.Good design is timeless.

In math, every proof is timeless unless it contains a mistake.

So what does Hardy mean when he says there is no permanent

place for ugly mathematics? He means the same thing Kelly Johnson did:

if something is ugly, it can't be the best solution. There

must be a better one, and eventually

someone will discover it.Aiming at timelessness is a way to make

yourself find the best answer:

if you can imagine someone surpassing you, you should do it yourself.

Some of the greatest masters did this so well that they

left little room for those who came after.

Every engraver since Durer has had to live in his shadow.Aiming at timelessness is also a way to evade

the grip of fashion. Fashions almost by definition

change with time, so if you can make something that

will still look good far into the future, then its

appeal must derive more from merit and less from fashion.Strangely enough, if you want to make something that will

appeal to future generations, one way to do it is to

try to appeal to past generations. It's hard to guess what

the future will be like, but we can be sure it will be

like the past in caring nothing for present fashions.

So if you can make something that appeals to people today

and would also have appealed to people in 1500, there is a good

chance it will appeal to people in 2500.Good design solves the right problem. The typical

stove has four burners arranged in a square, and a dial

to control each. How do you arrange the dials? The

simplest answer is to put them in a row. But this is a

simple answer to the wrong question.

The dials are for humans to use, and if you put them in a row,

the unlucky human will have to stop and think each time

about which dial matches which burner. Better to arrange the dials

in a square like the burners.A lot of bad design is industrious, but misguided.

In the mid twentieth century there was a vogue for

setting text in sans-serif fonts.

These fonts are closer to the pure, underlying letterforms.

But in text that's not the problem you're trying to solve.

For legibility it's more important that letters be easy

to tell apart.

It may look Victorian, but a Times Roman lowercase g is

easy to tell from a lowercase y.Problems can be improved as well as solutions.

In software, an intractable problem can usually be replaced

by an equivalent one that's easy to solve.

Physics progressed faster as the problem became

predicting observable behavior, instead of reconciling it

with scripture.Good design is suggestive.

Jane Austen's novels contain almost no

description; instead of telling you how

everything looks, she tells her story so well that you

envision the scene for yourself.

Likewise, a painting that suggests is usually more engaging

than one that tells. Everyone makes up their own story about the

Mona Lisa.In architecture and design, this

principle means that a building or object should let you

use it how you want: a good building, for example, will

serve as a backdrop for whatever life people want to lead in it, instead

of making them live as if they were executing a program

written by the architect.In software, it means you should give users a few

basic elements that they can combine as they wish, like Lego.

In math it means a proof that

becomes the basis for a lot of new work is

preferable to a proof that was difficult,

but doesn't lead to future discoveries; in the

sciences generally, citation is considered a rough

indicator of merit.Good design is often slightly funny. This one

may not always be true. But Durer's

engravings

and Saarinen's

womb chair and the

Pantheon and the

original Porsche 911 all seem

to me slightly funny. Godel's incompleteness theorem

seems like a practical joke.I think it's because humor is related to strength.

To have a sense of humor is to be strong:

to keep one's sense of humor is to shrug off misfortunes,

and to lose one's sense of humor is to be wounded by them.

And so the mark-- or at least the prerogative-- of strength

is not to take

oneself too seriously.

The confident will often, like

swallows, seem to be making fun of the whole process slightly,

as Hitchcock does in his films or Bruegel in his paintings-- or

Shakespeare, for that matter.Good design may not have to be funny, but it's hard to

imagine something that could be called humorless also being

good design.Good design is hard. If you look at the people who've

done great work, one thing they all seem to have in common is that they

worked very hard. If you're not working hard,

you're probably wasting your time.Hard problems call for great

efforts. In math, difficult proofs require ingenious solutions,

and those tend to be interesting. Ditto in engineering.When you

have to climb a mountain you toss everything unnecessary

out of your pack. And so an architect who has to build

on a difficult site, or a small budget, will find that he

is forced to produce an elegant design. Fashions and

flourishes get knocked aside by the difficult business

of solving the problem at all.Not every kind of hard is good. There is good pain and bad pain.

You want the kind of pain you get from going running, not the

kind you get from stepping on a nail.

A difficult

problem could be good for a designer, but a fickle client or unreliable

materials would not be.In art, the highest place has traditionally been given to

paintings of people. There is something to this tradition,

and not just because pictures of faces get to press

buttons in our brains that other pictures don't. We are

so good at looking at faces that we force anyone who

draws them to work hard to satisfy us. If you

draw a tree and you change the angle of a branch

five degrees, no one will know. When you change the angle

of someone's eye five degrees, people notice.When Bauhaus designers adopted Sullivan's "form follows function,"

what they meant was, form should follow function. And

if function is hard enough, form is forced to follow it,

because there is no effort to spare for error. Wild animals

are beautiful because they have hard lives.Good design looks easy. Like great athletes,

great designers make it look easy. Mostly this is

an illusion. The easy, conversational tone of good

writing comes only on the eighth rewrite.In science and engineering, some of the greatest

discoveries seem so simple that you say to yourself,

I could have thought of that. The discoverer is

entitled to reply, why didn't you?Some Leonardo heads are just a few lines. You look

at them and you think, all you have to do is get eight

or ten lines in the right place and you've made this beautiful

portrait. Well, yes, but you have to get them in

exactly the right place. The slightest error

will make the whole thing collapse.Line drawings are in fact the most difficult visual

medium, because they demand near perfection.

In math terms, they are a closed-form solution; lesser

artists literally solve the same problems by successive

approximation. One of the reasons kids give up drawing

at ten or so is that they decide to start

drawing like grownups, and one of the first things

they try is a line drawing of a face. Smack!In most fields the appearance of ease seems to come with

practice. Perhaps what practice does is train your

unconscious mind to handle tasks that used to

require conscious thought. In some cases

you literally train your body. An expert pianist can

play notes faster than the brain can send signals to

his hand.

Likewise an artist, after a while, can

make visual perception flow in through his eye and

out through his hand as automatically as someone tapping his foot to

a beat.When people talk about being in

"the zone," I think what they mean is that the

spinal cord has the situation under control.

Your spinal cord is less hesitant, and

it frees conscious thought for the hard problems.

Good design uses symmetry.

I think symmetry may just

be one way to achieve simplicity, but it's important enough

to be mentioned on its own.

Nature uses it a lot, which is a good sign.There are two kinds of symmetry, repetition and recursion.

Recursion means repetition in subelements, like the

pattern of veins in a leaf.Symmetry is unfashionable in some fields now, in reaction to

excesses in the past. Architects started consciously

making buildings asymmetric in Victorian times and by the

1920s asymmetry was an explicit premise of modernist architecture.

Even these buildings only tended to be asymmetric

about major axes, though; there were hundreds of minor symmetries.In writing you find symmetry at every level, from the phrases

in a sentence to the plot of a novel. You find the same

in music and art.

Mosaics (and some Cezannes) get extra visual punch by making

the whole picture out of the same atoms. Compositional

symmetry yields some of the most memorable paintings,

especially when two halves react to one another, as in

the Creation of Adam or

American Gothic.In math and engineering, recursion, especially, is a big win.

Inductive proofs are wonderfully short. In software,

a problem that can be solved by recursion is nearly always

best solved that way. The Eiffel Tower looks striking partly

because it is a recursive solution, a tower on a tower.The danger of symmetry, and repetition especially, is that

it can be used as a substitute for thought.Good design resembles nature. It's not so much that

resembling nature is intrinsically good as that nature

has had a long time to work on the

problem. It's a good sign when your answer resembles nature's.It's not cheating to copy.

Few would deny that a story should be like life.

Working from life is a valuable tool in painting too, though its

role has often been misunderstood.

The aim is not simply to make a record.

The point of painting from life is

that it gives your mind something to chew on: when your

eyes are looking at something, your hand will do more

interesting work.Imitating nature also works in engineering. Boats have

long had spines and ribs like an animal's ribcage.

In some cases we may have to wait for better technology:

early aircraft designers were mistaken to

design aircraft that looked like birds, because they didn't

have materials or power sources light enough (the Wrights' engine

weighed 152 lbs. and

generated only 12 hp.) or control systems sophisticated

enough for machines that flew like birds, but I could

imagine little unmanned reconnaissance planes flying

like birds in fifty years.Now that we have enough computer power, we can imitate nature's

method as well as its results. Genetic algorithms may let us

create things too complex to design in the ordinary

sense.Good design is redesign. It's rare to get things right

the first time. Experts expect to throw away some early work.

They plan for plans to change.It takes confidence to throw work away. You have to be able

to think, there's more where that came from.

When people first start drawing, for example,

they're often reluctant to redo parts that aren't

right; they feel they've been lucky to get that far,

and if they try to redo something, it will turn out worse. Instead

they convince themselves that the drawing is not that bad,

really-- in fact, maybe they meant it to look that way.Dangerous territory, that; if anything you should

cultivate dissatisfaction.

In Leonardo's drawings there are often five

or six attempts to get a line right.

The distinctive back of the Porsche

911 only appeared in the redesign of an awkward

prototype.

In Wright's early plans for the

Guggenheim,

the right half was a ziggurat; he inverted it to get the

present shape.Mistakes are natural. Instead of treating them

as disasters, make them easy to acknowledge and easy to fix.

Leonardo more or less invented the sketch, as a

way to make drawing bear a greater weight of exploration.

Open-source software has fewer bugs because it admits the

possibility of bugs.It helps to have a medium that makes change easy.

When oil paint replaced tempera in the fifteenth century,

it helped

painters to deal with difficult subjects like the human

figure because, unlike tempera, oil can be blended and overpainted.

Good design can copy. Attitudes to copying

often make a round trip. A novice

imitates without knowing it; next he tries

consciously to be original; finally, he decides it's

more important to be right than original.Unknowing imitation is almost a recipe for bad design.

If you don't know where your ideas are coming from,

you're probably imitating an imitator.

Raphael so pervaded mid-nineteenth century taste that almost

anyone who tried to draw was imitating him, often at several

removes.

It was this, more than Raphael's own work, that bothered

the Pre-Raphaelites.The ambitious are not content to imitate. The

second phase in the growth of taste is a conscious

attempt at originality.I think the

greatest masters go on to achieve a kind of selflessness.

They just want to get the right answer, and if part of the

right answer has already been discovered by someone else,

that's no reason not to use it.

They're confident enough to take from anyone without

feeling that their own vision will be lost in the process.

Good design is often strange. Some of the very best work

has an uncanny quality: Euler's

Formula,

Bruegel's

Hunters in the Snow, the

SR-71, Lisp. They're not just

beautiful, but strangely beautiful.I'm not sure why. It may just be my own stupidity. A

can-opener must seem miraculous to a dog. Maybe if I were smart

enough it would seem the most natural thing in the world that

ei\*pi = -1. It is after all necessarily true.Most of the qualities I've mentioned are things that can be

cultivated, but I don't think it works to cultivate strangeness.

The best you can do is not squash it if it starts to appear.

Einstein didn't try to make relativity strange.

He tried to make it true, and the truth turned out to be strange.At an art school where I once studied, the students wanted

most of all to develop a personal style.

But if you just try to make good things, you'll

inevitably do it in a distinctive way, just as each person

walks in a distinctive way. Michelangelo was not trying

to paint like Michelangelo. He was just trying to paint

well; he couldn't help painting like Michelangelo.The only style worth having is the one you can't help.

And this is especially true for strangeness. There is no

shortcut to it. The Northwest Passage that the Mannerists,

the Romantics, and two generations of American high school

students have searched for does not seem to exist. The

only way to get there is to go through good and come out

the other side.

Good design happens in chunks. The inhabitants

of fifteenth century Florence included Brunelleschi, Ghiberti,

Donatello, Masaccio, Filippo Lippi,

Fra Angelico, Verrocchio, Botticelli, Leonardo, and Michelangelo.

Milan at the time was as big as Florence.

How many fifteenth century Milanese artists can you name?Something was happening in Florence in the fifteenth century.

And it can't have been heredity, because it isn't happening now.

You have to assume that whatever

inborn ability Leonardo and Michelangelo had, there were

people born in Milan with just as much. What happened to

the Milanese Leonardo?There are roughly a thousand times

as many people alive in the US right now as lived in

Florence during the fifteenth century. A thousand Leonardos

and a thousand Michelangelos walk among us.

If DNA ruled, we should be greeted daily by artistic

marvels. We aren't, and the reason is that to make Leonardo

you need more than his innate ability. You also need Florence

in 1450.Nothing is more powerful

than a community of talented people working on related

problems. Genes count for little by comparison: being a genetic

Leonardo was not enough to compensate for having been born

near Milan instead of Florence.

Today we move around more, but great work still comes

disproportionately from a few hotspots:

the Bauhaus, the Manhattan Project, the New Yorker,

Lockheed's Skunk Works, Xerox Parc.At any given time there are a

few hot topics and a few groups doing great work on them,

and it's nearly impossible to do

good work yourself if you're too far removed from one

of these centers. You can push or pull these trends

to some extent, but you can't break away from them.

(Maybe you can, but the Milanese Leonardo couldn't.)

Good design is often daring. At every period

of history, people have believed things that were just

ridiculous, and believed them so strongly that you risked

ostracism or even violence by saying otherwise.If our own time were any different, that would be remarkable.

As far as I can tell it isn't.This problem afflicts not just every

era, but in some degree every field.

Much Renaissance art was in its time considered shockingly secular:

according to Vasari, Botticelli repented and gave up painting, and

Fra Bartolommeo and Lorenzo di Credi actually burned some of their

work.

Einstein's theory of relativity offended many contemporary physicists,

and was not fully accepted for decades-- in France, not until the

1950s.Today's experimental error is tomorrow's new theory. If

you want to discover great new things, then instead of turning

a blind eye to the places where conventional wisdom and

truth don't quite meet, you should pay particular attention

to them.As a practical matter, I think it's easier to see ugliness

than to imagine beauty. Most of the people who've made beautiful

things seem to have done it by fixing something that they

thought ugly. Great work usually seems to happen because someone sees

something and thinks, I could do better than that. Giotto

saw traditional Byzantine madonnas painted according to a

formula that had satisfied everyone for centuries, and to him

they looked wooden and unnatural.

Copernicus was so troubled by a hack that all his contemporaries

could tolerate that he felt there must be a better solution.Intolerance for ugliness is not in itself enough. You have to

understand a field well before you develop a good nose for

what needs fixing. You have to do your homework. But as

you become expert in a field, you'll start to hear little

voices saying, What a hack! There must be a better way.

Don't ignore those voices. Cultivate them. The recipe for

great work is: very exacting taste, plus the ability

to gratify it.NotesSullivan

actually said "form ever follows function," but

I think the usual misquotation is closer to what modernist

architects meant.

Stephen G. Brush, "Why was Relativity Accepted?"

Phys. Perspect. 1 (1999) 184-214.Japanese TranslationChinese TranslationSlovenian TranslationGerman TranslationInterview: Milton GlaserRussian Translation

You'll find this essay and 14 others in

Hackers & Painters.

Why Arc Isn't Especially Object-Oriented

There is a kind of mania for object-oriented programming at the moment, but

some of the smartest programmers I know are some of the least excited about it.My own feeling is that object-oriented

programming is a useful technique in some

cases, but it isn't something that has to pervade every program you

write. You should be able to define new types,

but you shouldn't have to express every program as the

definition of new types.I think there are five reasons people like object-oriented

programming, and three and a half of them are bad:

Object-oriented programming is exciting

if you have a statically-typed language without

lexical closures or macros. To some degree, it offers a way around these

limitations. (See Greenspun's Tenth Rule.) Object-oriented programming is popular in big companies,

because it suits the way they write software. At big companies,

software tends to be written by large (and frequently changing)

teams of

mediocre programmers. Object-oriented programming imposes a

discipline on these programmers that prevents any one of them

from doing too much damage. The price is that the resulting

code is bloated with protocols and full of duplication.

This is not too high a price for big companies, because their

software is probably going to be bloated and full of

duplication anyway. Object-oriented

programming generates a lot of what looks like work.

Back in the days of fanfold, there was a type of programmer who

would only put five or ten lines of code on a page, preceded

by twenty lines of elaborately formatted comments.

Object-oriented programming is like crack for these people: it lets

you incorporate all this scaffolding right into your source

code. Something that a Lisp hacker might handle by pushing

a symbol onto a list becomes a whole file of classes and

methods. So it is a good tool if you want to convince yourself,

or someone else, that you are doing a lot of work. If a language is itself an object-oriented program, it can

be extended by users. Well, maybe. Or maybe you can do

even better by offering the sub-concepts

of object-oriented programming a la carte. Overloading,

for example, is not intrinsically tied to classes. We'll see. Object-oriented abstractions map neatly onto the domains

of certain specific kinds of programs, like simulations and CAD

systems.

I personally have never needed object-oriented abstractions.

Common Lisp has an enormously powerful object system and I've

never used it once. I've done a lot of things (e.g. making

hash tables full of closures) that would have required

object-oriented techniques to do in wimpier languages, but

I have never had to use CLOS.Maybe I'm just stupid, or have worked on some limited subset

of applications. There is a danger in designing a language

based on one's own experience of programming. But it seems

more dangerous to put stuff in that you've never needed

because it's thought to be a good idea.Rees Re: OOSpanish Translation

What Made Lisp Different

December 2001 (rev. May 2002)

(This article came about in response to some questions on

the LL1 mailing list. It is now

incorporated in Revenge of the Nerds.)When McCarthy designed Lisp in the late 1950s, it was

a radical departure from existing languages,

the most important of which was Fortran.Lisp embodied nine new ideas:

1. Conditionals. A conditional is an if-then-else

construct. We take these for granted now. They were

invented

by McCarthy in the course of developing Lisp.

(Fortran at that time only had a conditional

goto, closely based on the branch instruction in the

underlying hardware.) McCarthy, who was on the Algol committee, got

conditionals into Algol, whence they spread to most other

languages.2. A function type. In Lisp, functions are first class

objects-- they're a data type just like integers, strings,

etc, and have a literal representation, can be stored in variables,

can be passed as arguments, and so on.3. Recursion. Recursion existed as a mathematical concept

before Lisp of course, but Lisp was the first programming language to support

it. (It's arguably implicit in making functions first class

objects.)4. A new concept of variables. In Lisp, all variables

are effectively pointers. Values are what

have types, not variables, and assigning or binding

variables means copying pointers, not what they point to.5. Garbage-collection.6. Programs composed of expressions. Lisp programs are

trees of expressions, each of which returns a value.

(In some Lisps expressions

can return multiple values.) This is in contrast to Fortran

and most succeeding languages, which distinguish between

expressions and statements.It was natural to have this

distinction in Fortran because (not surprisingly in a language

where the input format was punched cards) the language was

line-oriented. You could not nest statements. And

so while you needed expressions for math to work, there was

no point in making anything else return a value, because

there could not be anything waiting for it.This limitation

went away with the arrival of block-structured languages,

but by then it was too late. The distinction between

expressions and statements was entrenched. It spread from

Fortran into Algol and thence to both their descendants.When a language is made entirely of expressions, you can

compose expressions however you want. You can say either

(using Arc syntax)(if foo (= x 1) (= x 2))or(= x (if foo 1 2))7. A symbol type. Symbols differ from strings in that

you can test equality by comparing a pointer.8. A notation for code using trees of symbols.9. The whole language always available.

There is

no real distinction between read-time, compile-time, and runtime.

You can compile or run code while reading, read or run code

while compiling, and read or compile code at runtime.Running code at read-time lets users reprogram Lisp's syntax;

running code at compile-time is the basis of macros; compiling

at runtime is the basis of Lisp's use as an extension

language in programs like Emacs; and reading at runtime

enables programs to communicate using s-expressions, an

idea recently reinvented as XML.

When Lisp was first invented, all these ideas were far

removed from ordinary programming practice, which was

dictated largely by the hardware available in the late 1950s.Over time, the default language, embodied

in a succession of popular languages, has

gradually evolved toward Lisp. 1-5 are now widespread.

6 is starting to appear in the mainstream.

Python has a form of 7, though there doesn't seem to be

any syntax for it.

8, which (with 9) is what makes Lisp macros

possible, is so far still unique to Lisp,

perhaps because (a) it requires those parens, or something

just as bad, and (b) if you add that final increment of power,

you can no

longer claim to have invented a new language, but only

to have designed a new dialect of Lisp ; -)Though useful to present-day programmers, it's

strange to describe Lisp in terms of its

variation from the random expedients other languages

adopted. That was not, probably, how McCarthy

thought of it. Lisp wasn't designed to fix the mistakes

in Fortran; it came about more as the byproduct of an

attempt to axiomatize computation.Japanese Translation

The Other Road Ahead

September 2001

(This article explains why much of the next generation of software

may be server-based, what that will mean for programmers,

and why this new kind of software is a great opportunity for startups.

It's derived from a talk at BBN Labs.)

In the summer of 1995, my friend Robert Morris and I decided to

start a startup. The PR campaign leading up to Netscape's IPO was

running full blast then, and there was a lot of talk in the press

about online commerce. At the time there might have been thirty

actual stores on the Web, all made by hand. If there were going

to be a lot of online stores, there would need to be software for making

them, so we decided to write some.For the first week or so we intended to make this an ordinary

desktop application. Then one day we had the idea of making the

software run on our Web server, using the browser as an

interface. We tried rewriting the software to work over

the Web, and it was clear that this was the way to go.

If we wrote our software to run on the server, it would be a lot easier

for the users and for us as well.This turned out to be a good plan. Now, as

Yahoo Store, this

software is the most popular online store builder, with

about 14,000 users.When we started Viaweb, hardly anyone understood what we meant when

we said that the software ran on the server. It was not until

Hotmail was launched a year later that people started to get it.

Now everyone knows that this is a valid approach. There is

a name now for what we were: an Application Service Provider,

or ASP.I think that a lot of the next generation of software will be

written on this model. Even Microsoft, who have the most to

lose, seem to see the inevitablity of moving some things off

the desktop. If software moves

off the desktop and onto servers, it will mean a very different

world for developers. This article describes the surprising

things we saw, as some of the first visitors to this new world.

To the extent software does move onto

servers, what I'm describing here is the future.The Next Thing?When we look back on the desktop software era, I think we'll marvel

at the inconveniences people put up with, just as we marvel now at

what early car owners put up with. For the first twenty or thirty

years, you had to be a car expert to own a car. But cars were such

a big win that lots of people who weren't car experts wanted to

have them as well.Computers are in this phase now. When you own a desktop computer,

you end up learning a lot more than you wanted to know about what's

happening inside it. But more than half the households in the US

own one. My mother has a computer that she uses for email and for

keeping accounts. About a year ago she was alarmed to receive a

letter from Apple, offering her a discount on a new version of the

operating system. There's something wrong when a sixty-five year

old woman who wants to use a computer for email and accounts has

to think about installing new operating systems. Ordinary users

shouldn't even know the words "operating system," much less "device

driver" or "patch."There is now another way to deliver software that will save users

from becoming system administrators. Web-based applications are

programs that run on Web servers and use Web pages as the user

interface. For the average user this new kind of software will be

easier, cheaper, more mobile, more reliable, and often more powerful

than desktop software.With Web-based software, most users won't have to think about

anything except the applications they use. All the messy, changing

stuff will be sitting on a server somewhere, maintained by the kind

of people who are good at that kind of thing. And so you won't

ordinarily need a computer, per se, to use software. All you'll

need will be something with a keyboard, a screen, and a Web browser.

Maybe it will have wireless Internet access. Maybe it will also

be your cell phone. Whatever it is, it will be consumer electronics:

something that costs about $200, and that people choose mostly

based on how the case looks. You'll pay more for Internet services

than you do for the hardware, just as you do now with telephones. [1]It will take about a tenth of a second for a click to get to the

server and back, so users of heavily interactive software, like

Photoshop, will still want to have the computations happening on

the desktop. But if you look at the kind of things most people

use computers for, a tenth of a second latency would not be a

problem. My mother doesn't really need a desktop computer, and

there are a lot of people like her.The Win for UsersNear my house there is a car with a bumper sticker that reads "death

before inconvenience." Most people, most of the time, will take

whatever choice requires least work. If Web-based software wins,

it will be because it's more convenient. And it looks as if it

will be, for users and developers both.To use a purely Web-based application, all you need is a browser

connected to the Internet. So you can use a Web-based application

anywhere. When you install software on your desktop computer, you

can only use it on that computer. Worse still, your files are

trapped on that computer. The inconvenience of this model becomes

more and more evident as people get used to networks.The thin end of the wedge here was Web-based email. Millions of

people now realize that you should have access to email messages

no matter where you are. And if you can see your email, why not

your calendar?

If you can discuss a document with your colleagues,

why can't you edit it? Why should any of your data be trapped on

some computer sitting on a faraway desk?The whole idea of "your computer" is going away, and being replaced

with "your data." You should be able to get at your data from any

computer. Or rather, any client, and a client doesn't have to be

a computer.Clients shouldn't store data; they should be like telephones. In

fact they may become telephones, or vice versa. And as clients

get smaller, you have another reason not to keep your data on them:

something you carry around with you can be lost or stolen. Leaving

your PDA in a taxi is like a disk crash, except that your data is

handed to someone else

instead of being vaporized.With purely Web-based software, neither your data nor the applications

are kept on the client. So you don't have to install anything to

use it. And when there's no installation, you don't have to worry

about installation going wrong. There can't be incompatibilities

between the application and your operating system, because the

software doesn't run on your operating system.Because it needs no installation, it will be easy, and common, to

try Web-based software before you "buy" it. You should expect to

be able to test-drive any Web-based application for free, just by

going to the site where it's offered. At Viaweb our whole site

was like a big arrow pointing users to the test drive.After trying the demo, signing up for the service should require

nothing more than filling out a brief form (the briefer the better).

And that should be the last work the user has to do. With Web-based

software, you should get new releases without paying extra, or

doing any work, or possibly even knowing about it.Upgrades won't be the big shocks they are now. Over time applications

will quietly grow more powerful. This will take some effort on

the part of the developers. They will have to design software so

that it can be updated without confusing the users. That's a new

problem, but there are ways to solve it.With Web-based applications, everyone uses the same version, and

bugs can be fixed as soon as they're discovered. So Web-based

software should have far fewer bugs than desktop software. At

Viaweb, I doubt we ever had ten known bugs at any one time. That's

orders of magnitude better than desktop software.Web-based applications can be used by several people at the same

time. This is an obvious win for collaborative applications, but

I bet users will start to want this in most applications once they

realize it's possible. It will often be useful to let two people

edit the same document, for example. Viaweb let multiple users

edit a site simultaneously, more because that was the right way to

write the software than because we expected users to want to, but

it turned out that many did.When you use a Web-based application, your data will be safer.

Disk crashes won't be a thing of the past, but users won't hear

about them anymore. They'll happen within server farms. And

companies offering Web-based applications will actually do backups--

not only because they'll have real system administrators worrying

about such things, but because an ASP that does lose people's data

will be in big, big trouble. When people lose their own data in

a disk crash, they can't get that mad, because they only have

themselves to be mad at. When a company loses their data for them,

they'll get a lot madder.Finally, Web-based software should be less vulnerable to viruses.

If the client doesn't run anything except a browser, there's less

chance of running viruses, and no data locally to damage. And a

program that attacked the servers themselves should find them very

well defended. [2]For users, Web-based software will be less stressful. I think if

you looked inside the average Windows user you'd find a huge and

pretty much untapped desire for software meeting that description.

Unleashed, it could be a powerful force.City of CodeTo developers, the most conspicuous difference between Web-based

and desktop software is that a Web-based application is not a single

piece of code. It will be a collection of programs of different

types rather than a single big binary. And so designing Web-based

software is like desiging a city rather than a building: as well

as buildings you need roads, street signs, utilities, police and

fire departments, and plans for both growth and various kinds of

disasters.At Viaweb, software included fairly big applications that users

talked to directly, programs that those programs used, programs

that ran constantly in the background looking for problems, programs

that tried to restart things if they broke, programs that ran

occasionally to compile statistics or build indexes for searches,

programs we ran explicitly to garbage-collect resources or to move

or restore data, programs that pretended to be users (to measure

performance or expose bugs), programs for diagnosing network

troubles, programs for doing backups, interfaces to outside services,

software that drove an impressive collection of dials displaying

real-time server statistics (a hit with visitors, but indispensable

for us too), modifications (including bug fixes) to open-source

software, and a great many configuration files and settings. Trevor

Blackwell wrote a spectacular program for moving stores to new

servers across the country, without shutting them down, after we

were bought by Yahoo. Programs paged us, sent faxes and email to

users, conducted transactions with credit card processors, and

talked to one another through sockets, pipes, http requests, ssh,

udp packets, shared memory, and files. Some of Viaweb even consisted

of the absence of programs, since one of the keys to Unix security

is not to run unnecessary utilities that people might use to break

into your servers.It did not end with software. We spent a lot of time thinking

about server configurations. We built the servers ourselves, from

components-- partly to save money, and partly to get exactly what

we wanted. We had to think about whether our upstream ISP had fast

enough connections to all the backbones. We serially

dated

RAID suppliers.But hardware is not just something to worry about. When you control

it you can do more for users. With a desktop application, you can

specify certain minimum hardware, but you can't add more. If you

administer the servers, you can in one step enable all your users

to page people, or send faxes, or send commands by phone, or process

credit cards, etc, just by installing the relevant hardware. We

always looked for new ways to add features with hardware, not just

because it pleased users, but also as a way to distinguish ourselves

from competitors who (either because they sold desktop software,

or resold Web-based applications through ISPs) didn't have direct

control over the hardware.Because the software in a Web-based application will be a collection

of programs rather than a single binary, it can be written in any

number of different languages. When you're writing desktop software,

you're practically forced to write the application in the same

language as the underlying operating system-- meaning C and C++.

And so these languages (especially among nontechnical people like

managers and VCs) got to be considered as the languages for "serious"

software development. But that was just an artifact of the way

desktop software had to be delivered. For server-based software

you can use any language you want. [3] Today a lot of the top

hackers are using languages far removed from C and C++: Perl,

Python, and even Lisp.With server-based software, no one can tell you what language to

use, because you control the whole system, right down to the

hardware. Different languages are good for different tasks. You

can use whichever is best for each. And when you have competitors,

"you can" means "you must" (we'll return to this later), because

if you don't take advantage of this possibility, your competitors

will.Most of our competitors used C and C++, and this made their software

visibly inferior because (among other things), they had no way

around the statelessness of CGI scripts. If you were going to

change something, all the changes had to happen on one page, with

an Update button at the bottom. As I've written elsewhere, by

using Lisp, which many people still consider

a research language,

we could make the Viaweb editor behave more like desktop software.ReleasesOne of the most important changes in this new world is the way you

do releases. In the desktop software business, doing a release is

a huge trauma, in which the whole company sweats and strains to

push out a single, giant piece of code. Obvious comparisons suggest

themselves, both to the process and the resulting product.With server-based software, you can make changes almost as you

would in a program you were writing for yourself. You release

software as a series of incremental changes instead of an occasional

big explosion. A typical desktop software company might do one or

two releases a year. At Viaweb we often did three to five releases

a day.When you switch to this new model, you realize how much software

development is affected by the way it is released. Many of the

nastiest problems you see in the desktop software business are due

to catastrophic nature of releases.When you release only one new version a year, you tend to deal with

bugs wholesale. Some time before the release date you assemble a

new version in which half the code has been torn out and replaced,

introducing countless bugs. Then a squad of QA people step in and

start counting them, and the programmers work down the list, fixing

them. They do not generally get to the end of the list, and indeed,

no one is sure where the end is. It's like fishing rubble out of

a pond. You never really know what's happening inside the software.

At best you end up with a statistical sort of correctness.With server-based software, most of the change is small and

incremental. That in itself is less likely to introduce bugs. It

also means you know what to test most carefully when you're about

to release software: the last thing you changed. You end up with

a much firmer grip on the code. As a general rule, you do know

what's happening inside it. You don't have the source code memorized,

of course, but when you read the source you do it like a pilot

scanning the instrument panel, not like a detective trying to

unravel some mystery.Desktop software breeds a certain fatalism about bugs. You know

that you're shipping something loaded with bugs, and you've even

set up mechanisms to compensate for it (e.g. patch releases). So

why worry about a few more? Soon you're releasing whole features

you know are broken.

Apple

did this earlier this year. They felt

under pressure to release their new OS, whose release date had

already slipped four times, but some of the software (support for

CDs and DVDs) wasn't ready. The solution? They released the OS

without the unfinished parts, and users will have to install them

later.With Web-based software, you never have to release software before

it works, and you can release it as soon as it does work.The industry veteran may be thinking, it's a fine-sounding idea to

say that you never have to release software before it works, but

what happens when you've promised to deliver a new version of your

software by a certain date? With Web-based software, you wouldn't

make such a promise, because there are no versions. Your software

changes gradually and continuously. Some changes might be bigger

than others, but the idea of versions just doesn't naturally fit

onto Web-based software.If anyone remembers Viaweb this might sound odd, because we were

always announcing new versions. This was done entirely for PR

purposes. The trade press, we learned, thinks in version numbers.

They will give you major coverage for a major release, meaning a

new first digit on the version number, and generally a paragraph

at most for a point release, meaning a new digit after the decimal

point.Some of our competitors were offering desktop software and actually

had version numbers. And for these releases, the mere fact of

which seemed to us evidence of their backwardness, they would get

all kinds of publicity. We didn't want to miss out, so we started

giving version numbers to our software too. When we wanted some

publicity, we'd make a list of all the features we'd added since

the last "release," stick a new version number on the software,

and issue a press release saying that the new version was available

immediately. Amazingly, no one ever called us on it.By the time we were bought, we had done this three times, so we

were on Version 4. Version 4.1 if I remember correctly. After

Viaweb became Yahoo Store, there was no longer such a desperate

need for publicity, so although the software continued to evolve,

the whole idea of version numbers was quietly dropped.BugsThe other major technical advantage of Web-based software is that

you can reproduce most bugs. You have the users' data right there

on your disk. If someone breaks your software, you don't have to

try to guess what's going on, as you would with desktop software:

you should be able to reproduce the error while they're on the

phone with you. You might even know about it already, if you have

code for noticing errors built into your application.Web-based software gets used round the clock, so everything you do

is immediately put through the wringer. Bugs turn up quickly.Software companies are sometimes accused of letting the users debug

their software. And that is just what I'm advocating. For Web-based

software it's actually a good plan, because the bugs are fewer and

transient. When you release software gradually you get far fewer

bugs to start with. And when you can reproduce errors and release

changes instantly, you can find and fix most bugs as soon as they

appear. We never had enough bugs at any one time to bother with

a formal bug-tracking system.You should test changes before you release them, of course, so no

major bugs should get released. Those few that inevitably slip

through will involve borderline cases and will only affect the few

users that encounter them before someone calls in to complain. As

long as you fix bugs right away, the net effect, for the average

user, is far fewer bugs. I doubt the average Viaweb user ever saw

a bug.Fixing fresh bugs is easier than fixing old ones. It's usually

fairly quick to find a bug in code you just wrote. When it turns

up you often know what's wrong before you even look at the source,

because you were already worrying about it subconsciously. Fixing

a bug in something you wrote six months ago (the average case if

you release once a year) is a lot more work. And since you don't

understand the code as well, you're more likely to fix it in an

ugly way, or even introduce more bugs. [4]When you catch bugs early, you also get fewer compound bugs.

Compound bugs are two separate bugs that interact: you trip going

downstairs, and when you reach for the handrail it comes off in

your hand. In software this kind of bug is the hardest to find,

and also tends to have the worst consequences. [5] The traditional

"break everything and then filter out the bugs" approach inherently

yields a lot of compound bugs. And software that's released in a

series of small changes inherently tends not to. The floors are

constantly being swept clean of any loose objects that might later

get stuck in something.It helps if you use a technique called functional programming.

Functional programming means avoiding side-effects. It's something

you're more likely to see in research papers than commercial

software, but for Web-based applications it turns out to be really

useful. It's hard to write entire programs as purely functional

code, but you can write substantial chunks this way. It makes

those parts of your software easier to test, because they have no

state, and that is very convenient in a situation where you are

constantly making and testing small modifications. I wrote much

of Viaweb's editor in this style, and we made our scripting language,

RTML,

a purely functional language.People from the desktop software business will find this hard to

credit, but at Viaweb bugs became almost a game. Since most released

bugs involved borderline cases, the users who encountered them were

likely to be advanced users, pushing the envelope. Advanced users

are more forgiving about bugs, especially since you probably

introduced them in the course of adding some feature they were

asking for. In fact, because bugs were rare and you had to be

doing sophisticated things to see them, advanced users were often

proud to catch one. They would call support in a spirit more of

triumph than anger, as if they had scored points off us.SupportWhen you can reproduce errors, it changes your approach to customer

support. At most software companies, support is offered as a way

to make customers feel better. They're either calling you about

a known bug, or they're just doing something wrong and you have to

figure out what. In either case there's not much you can learn

from them. And so you tend to view support calls as a pain in the

ass that you want to isolate from your developers as much as

possible.This was not how things worked at Viaweb. At Viaweb, support was

free, because we wanted to hear from customers. If someone had a

problem, we wanted to know about it right away so that we could

reproduce the error and release a fix.So at Viaweb the developers were always in close contact with

support. The customer support people were about thirty feet away

from the programmers, and knew that they could always interrupt

anything with a report of a genuine bug. We would leave a board

meeting to fix a serious bug.Our approach to support made everyone happier. The customers were

delighted. Just imagine how it would feel to call a support line

and be treated as someone bringing important news. The customer

support people liked it because it meant they could help the users,

instead of reading scripts to them. And the programmers liked it

because they could reproduce bugs instead of just hearing vague

second-hand reports about them.Our policy of fixing bugs on the fly changed the relationship

between customer support people and hackers. At most software

companies, support people are underpaid human shields, and hackers

are little copies of God the Father, creators of the world. Whatever

the procedure for reporting bugs, it is likely to be one-directional:

support people who hear about bugs fill out some form that eventually

gets passed on (possibly via QA) to programmers, who put it on

their list of things to do. It was very different at Viaweb.

Within a minute of hearing about a bug from a customer, the support

people could be standing next to a programmer hearing him say "Shit,

you're right, it's a bug." It delighted the support people to hear

that "you're right" from the hackers. They used to bring us bugs

with the same expectant air as a cat bringing you a mouse it has

just killed. It also made them more careful in judging the

seriousness of a bug, because now their honor was on the line.After we were bought by Yahoo, the customer support people were

moved far away from the programmers. It was only then that we

realized that they were effectively QA and to some extent marketing

as well. In addition to catching bugs, they were the keepers of

the knowledge of vaguer, buglike things, like features that confused

users. [6] They were also a kind of proxy focus group; we could

ask them which of two new features users wanted more, and they were

always right.MoraleBeing able to release software immediately is a big motivator.

Often as I was walking to work I would think of some change I wanted

to make to the software, and do it that day. This worked for bigger

features as well. Even if something was going to take two weeks

to write (few projects took longer), I knew I could see the effect

in the software as soon as it was done.If I'd had to wait a year for the next release, I would have shelved

most of these ideas, for a while at least. The thing about ideas,

though, is that they lead to more ideas. Have you ever noticed

that when you sit down to write something, half the ideas that end

up in it are ones you thought of while writing it? The same thing

happens with software. Working to implement one idea gives you

more ideas. So shelving an idea costs you not only that delay in

implementing it, but also all the ideas that implementing it would

have led to. In fact, shelving an idea probably even inhibits new

ideas: as you start to think of some new feature, you catch sight

of the shelf and think "but I already have a lot of new things I

want to do for the next release."What big companies do instead of implementing features is plan

them. At Viaweb we sometimes ran into trouble on this account.

Investors and analysts would ask us what we had planned for the

future. The truthful answer would have been, we didn't have any

plans. We had general ideas about things we wanted to improve,

but if we knew how we would have done it already. What were we

going to do in the next six months? Whatever looked like the biggest

win. I don't know if I ever dared give this answer, but that was

the truth. Plans are just another word for ideas on the shelf.

When we thought of good ideas, we implemented them.At Viaweb, as at many software companies, most code had one definite

owner. But when you owned something you really owned it: no one

except the owner of a piece of software had to approve (or even

know about) a release. There was no protection against breakage

except the fear of looking like an idiot to one's peers, and that

was more than enough. I may have given the impression that we just

blithely plowed forward writing code. We did go fast, but we

thought very carefully before we released software onto those

servers. And paying attention is more important to reliability

than moving slowly. Because he pays close attention, a Navy pilot

can land a 40,000 lb. aircraft at 140 miles per hour on a pitching

carrier deck, at night, more safely than the average teenager can

cut a bagel.This way of writing software is a double-edged sword of course.

It works a lot better for a small team of good, trusted programmers

than it would for a big company of mediocre ones, where bad ideas

are caught by committees instead of the people that had them.Brooks in ReverseFortunately, Web-based software does require fewer programmers.

I once worked for a medium-sized desktop software company that had

over 100 people working in engineering as a whole. Only 13 of

these were in product development. All the rest were working on

releases, ports, and so on. With Web-based software, all you need

(at most) are the 13 people, because there are no releases, ports,

and so on.Viaweb was written by just three people. [7] I was always under

pressure to hire more, because we wanted to get bought, and we knew

that buyers would have a hard time paying a high price for a company

with only three programmers. (Solution: we hired more, but created

new projects for them.)When you can write software with fewer programmers, it saves you

more than money. As Fred Brooks pointed out in The Mythical

Man-Month, adding people to a project tends to slow it down. The

number of possible connections between developers grows exponentially

with the size of the group. The larger the group, the more time

they'll spend in meetings negotiating how their software will work

together, and the more bugs they'll get from unforeseen interactions.

Fortunately, this process also works in reverse: as groups get

smaller, software development gets exponentially more efficient.

I can't remember the programmers at Viaweb ever having an actual

meeting. We never had more to say at any one time than we could

say as we were walking to lunch.If there is a downside here, it is that all the programmers have

to be to some degree system administrators as well. When you're

hosting software, someone has to be watching the servers, and in

practice the only people who can do this properly are the ones who

wrote the software. At Viaweb our system had so many components

and changed so frequently that there was no definite border between

software and infrastructure. Arbitrarily declaring such a border

would have constrained our design choices. And so although we were

constantly hoping that one day ("in a couple months") everything

would be stable enough that we could hire someone whose job was

just to worry about the servers, it never happened.I don't think it could be any other way, as long as you're still

actively developing the product. Web-based software is never going

to be something you write, check in, and go home. It's a live

thing, running on your servers right now. A bad bug might not just

crash one user's process; it could crash them all. If a bug in

your code corrupts some data on disk, you have to fix it. And so

on. We found that you don't have to watch the servers every minute

(after the first year or so), but you definitely want to keep an

eye on things you've changed recently. You don't release code late

at night and then go home.Watching UsersWith server-based software, you're in closer touch with your code.

You can also be in closer touch with your users. Intuit is famous

for introducing themselves to customers at retail stores and asking

to follow them home. If you've ever watched someone use your

software for the first time, you know what surprises must have

awaited them.Software should do what users think it will. But you can't have

any idea what users will be thinking, believe me, until you watch

them. And server-based software gives you unprecedented information

about their behavior. You're not limited to small, artificial

focus groups. You can see every click made by every user. You

have to consider carefully what you're going to look at, because

you don't want to violate users' privacy, but even the most general

statistical sampling can be very useful.When you have the users on your server, you don't have to rely on

benchmarks, for example. Benchmarks are simulated users. With

server-based software, you can watch actual users. To decide what

to optimize, just log into a server and see what's consuming all

the CPU. And you know when to stop optimizing too: we eventually

got the Viaweb editor to the point where it was memory-bound rather

than CPU-bound, and since there was nothing we could do to decrease

the size of users' data (well, nothing easy), we knew we might as

well stop there.Efficiency matters for server-based software, because you're paying

for the hardware. The number of users you can support per server

is the divisor of your capital cost, so if you can make your software

very efficient you can undersell competitors and still make a

profit. At Viaweb we got the capital cost per user down to about

$5. It would be less now, probably less than the cost of sending

them the first month's bill. Hardware is free now, if your software

is reasonably efficient.Watching users can guide you in design as well as optimization.

Viaweb had a scripting language called RTML that let advanced users

define their own page styles. We found that RTML became a kind of

suggestion box, because users only used it when the predefined page

styles couldn't do what they wanted. Originally the editor put

button bars across the page, for example, but after a number of

users used RTML to put buttons down the left

side,

we made that an

option (in fact the default) in the predefined page styles.Finally, by watching users you can often tell when they're in

trouble. And since the customer is always right, that's a sign of

something you need to fix. At Viaweb the key to getting users was

the online test drive. It was not just a series of slides built

by marketing people. In our test drive, users actually used the

software. It took about five minutes, and at the end of it they

had built a real, working store.

The test drive was the way we got nearly all our new users. I

think it will be the same for most Web-based applications. If

users can get through a test drive successfully, they'll like the

product. If they get confused or bored, they won't. So anything

we could do to get more people through the test drive would increase

our growth rate.I studied click trails of people taking the test drive and found

that at a certain step they would get confused and click on the

browser's Back button. (If you try writing Web-based applications,

you'll find that the Back button becomes one of your most interesting

philosophical problems.) So I added a message at that point, telling

users that they were nearly finished, and reminding them not to

click on the Back button. Another great thing about Web-based

software is that you get instant feedback from changes: the number

of people completing the test drive rose immediately from 60% to

90%. And since the number of new users was a function of the number

of completed test drives, our revenue growth increased by 50%, just

from that change.MoneyIn the early 1990s I read an article in which someone said that

software was a subscription business. At first this seemed a very

cynical statement. But later I realized that it reflects reality:

software development is an ongoing process. I think it's cleaner

if you openly charge subscription fees, instead of forcing people

to keep buying and installing new versions so that they'll keep

paying you. And fortunately, subscriptions are the natural way to

bill for Web-based applications.Hosting applications is an area where companies will play a role

that is not likely to be filled by freeware. Hosting applications

is a lot of stress, and has real expenses. No one is going to want

to do it for free.For companies, Web-based applications are an ideal source of revenue.

Instead of starting each quarter with a blank slate, you have a

recurring revenue stream. Because your software evolves gradually,

you don't have to worry that a new model will flop; there never

need be a new model, per se, and if you do something to the software

that users hate, you'll know right away. You have no trouble with

uncollectable bills; if someone won't pay you can just turn off

the service. And there is no possibility of piracy.That last "advantage" may turn out to be a problem. Some amount

of piracy is to the advantage of software companies. If some user

really would not have bought your software at any price, you haven't

lost anything if he uses a pirated copy. In fact you gain, because

he is one more user helping to make your software the standard--

or who might buy a copy later, when he graduates from high school.When they can, companies like to do something called price

discrimination, which means charging each customer as much as they

can afford. [8] Software is particularly suitable for price

discrimination, because the marginal cost is close to zero. This

is why some software costs more to run on Suns than on Intel boxes:

a company that uses Suns is not interested in saving money and can

safely be charged more. Piracy is effectively the lowest tier of

price discrimination. I think that software companies understand

this and deliberately turn a blind eye to some kinds of piracy. [9]

With server-based software they are going to have to come up with

some other solution.Web-based software sells well, especially in comparison to desktop

software, because it's easy to buy. You might think that people

decide to buy something, and then buy it, as two separate steps.

That's what I thought before Viaweb, to the extent I thought about

the question at all. In fact the second step can propagate back

into the first: if something is hard to buy, people will change

their mind about whether they wanted it. And vice versa: you'll

sell more of something when it's easy to buy. I buy more books

because Amazon exists. Web-based software is just about the easiest

thing in the world to buy, especially if you have just done an

online demo. Users should not have to do much more than enter a

credit card number. (Make them do more at your peril.)

Sometimes Web-based software is offered through ISPs acting as

resellers. This is a bad idea. You have to be administering the

servers, because you need to be constantly improving both hardware

and software. If you give up direct control of the servers, you

give up most of the advantages of developing Web-based applications.Several of our competitors shot themselves in the foot this way--

usually, I think, because they were overrun by suits who were

excited about this huge potential channel, and didn't realize that

it would ruin the product they hoped to sell through it. Selling

Web-based software through ISPs is like selling sushi through

vending machines.CustomersWho will the customers be? At Viaweb they were initially individuals

and smaller companies, and I think this will be the rule with

Web-based applications. These are the users who are ready to try

new things, partly because they're more flexible, and partly because

they want the lower costs of new technology.Web-based applications will often be the best thing for big companies

too (though they'll be slow to realize it). The best intranet is

the Internet. If a company uses true Web-based applications, the

software will work better, the servers will be better administered,

and employees will have access to the system from anywhere.The argument against this approach usually hinges on security: if

access is easier for employees, it will be for bad guys too. Some

larger merchants were reluctant to use Viaweb because they thought

customers' credit card information would be safer on their own

servers. It was not easy to make this point diplomatically, but

in fact the data was almost certainly safer in our hands than

theirs. Who can hire better people to manage security, a technology

startup whose whole business is running servers, or a clothing

retailer? Not only did we have better people worrying about

security, we worried more about it. If someone broke into the

clothing retailer's servers, it would affect at most one merchant,

could probably be hushed up, and in the worst case might get one

person fired. If someone broke into ours, it could affect thousands

of merchants, would probably end up as news on CNet, and could put

us out of business.If you want to keep your money safe, do you keep it under your

mattress at home, or put it in a bank? This argument applies to

every aspect of server administration: not just security, but

uptime, bandwidth, load management, backups, etc. Our existence

depended on doing these things right. Server problems were the

big no-no for us, like a dangerous toy would be for a toy maker,

or a salmonella outbreak for a food processor.A big company that uses Web-based applications is to that extent

outsourcing IT. Drastic as it sounds, I think this is generally

a good idea. Companies are likely to get better service this way

than they would from in-house system administrators. System

administrators can become cranky and unresponsive because they're

not directly exposed to competitive pressure: a salesman has to

deal with customers, and a developer has to deal with competitors'

software, but a system administrator, like an old bachelor, has

few external forces to keep him in line. [10] At Viaweb we had

external forces in plenty to keep us in line. The people calling

us were customers, not just co-workers. If a server got wedged,

we jumped; just thinking about it gives me a jolt of adrenaline,

years later.So Web-based applications will ordinarily be the right answer for

big companies too. They will be the last to realize it, however,

just as they were with desktop computers. And partly for the same

reason: it will be worth a lot of money to convince big companies

that they need something more expensive.There is always a tendency for rich customers to buy expensive

solutions, even when cheap solutions are better, because the people

offering expensive solutions can spend more to sell them. At Viaweb

we were always up against this. We lost several high-end merchants

to Web consulting firms who convinced them they'd be better off if

they paid half a million dollars for a custom-made online store on

their own server. They were, as a rule, not better off, as more

than one discovered when Christmas shopping season came around and

loads rose on their server. Viaweb was a lot more sophisticated

than what most of these merchants got, but we couldn't afford to

tell them. At $300 a month, we couldn't afford to send a team of

well-dressed and authoritative-sounding people to make presentations

to customers.A large part of what big companies pay extra for is the cost of

selling expensive things to them. (If the Defense Department pays

a thousand dollars for toilet seats, it's partly because it costs

a lot to sell toilet seats for a thousand dollars.) And this is

one reason intranet software will continue to thrive, even though

it is probably a bad idea. It's simply more expensive. There is

nothing you can do about this conundrum, so the best plan is to go

for the smaller customers first. The rest will come in time.Son of ServerRunning software on the server is nothing new. In fact it's the

old model: mainframe applications are all server-based. If

server-based software is such a good idea, why did it lose last

time? Why did desktop computers eclipse mainframes?At first desktop computers didn't look like much of a threat. The

first users were all hackers-- or hobbyists, as they were called

then. They liked microcomputers because they were cheap. For the

first time, you could have your own computer. The phrase "personal

computer" is part of the language now, but when it was first used

it had a deliberately audacious sound, like the phrase "personal

satellite" would today.Why did desktop computers take over? I think it was because they

had better software. And I think the reason microcomputer software

was better was that it could be written by small companies.I don't think many people realize how fragile and tentative startups

are in the earliest stage. Many startups begin almost by accident--

as a couple guys, either with day jobs or in school, writing a

prototype of something that might, if it looks promising, turn into

a company. At this larval stage, any significant obstacle will stop

the startup dead in its tracks. Writing mainframe software required

too much commitment up front. Development machines were expensive,

and because the customers would be big companies, you'd need an

impressive-looking sales force to sell it to them. Starting a

startup to write mainframe software would be a much more serious

undertaking than just hacking something together on your Apple II

in the evenings. And so you didn't get a lot of startups writing

mainframe applications.The arrival of desktop computers inspired a lot of new software,

because writing applications for them seemed an attainable goal to

larval startups. Development was cheap, and the customers would

be individual people that you could reach through computer stores

or even by mail-order.The application that pushed desktop computers out into the mainstream

was VisiCalc, the

first spreadsheet. It was written by two guys

working in an attic, and yet did things no mainframe software could

do. [11] VisiCalc was such an advance, in its time, that people

bought Apple IIs just to run it. And this was the beginning of a

trend: desktop computers won because startups wrote software for

them.It looks as if server-based software will be good this time around,

because startups will write it. Computers are so cheap now that

you can get started, as we did, using a desktop computer as a

server. Inexpensive processors have eaten the workstation market

(you rarely even hear the word now) and are most of the way through

the server market; Yahoo's servers, which deal with loads as high

as any on the Internet, all have the same inexpensive Intel processors

that you have in your desktop machine. And once you've written

the software, all you need to sell it is a Web site. Nearly all

our users came direct to our site through word of mouth and references

in the press. [12]Viaweb was a typical larval startup. We were terrified of starting

a company, and for the first few months comforted ourselves by

treating the whole thing as an experiment that we might call off

at any moment. Fortunately, there were few obstacles except

technical ones. While we were writing the software, our Web server

was the same desktop machine we used for development, connected to

the outside world by a dialup line. Our only expenses in that

phase were food and rent.There is all the more reason for startups to write Web-based software

now, because writing desktop software has become a lot less fun.

If you want to write desktop software now you do it on Microsoft's

terms, calling their APIs and working around their buggy OS. And

if you manage to write something that takes off, you may find that

you were merely doing market research for Microsoft.If a company wants to make a platform that startups will build on,

they have to make it something that hackers themselves will want

to use. That means it has to be inexpensive and well-designed.

The Mac was popular with hackers when it first came out, and a lot

of them wrote software for it. [13] You see this less with Windows,

because hackers don't use it. The kind of people who are good at

writing software tend to be running Linux or FreeBSD now.I don't think we would have started a startup to write desktop

software, because desktop software has to run on Windows, and before

we could write software for Windows we'd have to use it. The Web

let us do an end-run around Windows, and deliver software running

on Unix direct to users through the browser. That is a liberating

prospect, a lot like the arrival of PCs twenty-five years ago.MicrosoftBack when desktop computers arrived, IBM was the giant that everyone

was afraid of. It's hard to imagine now, but I remember the feeling

very well. Now the frightening giant is Microsoft, and I don't

think they are as blind to the threat facing them as IBM was.

After all, Microsoft deliberately built their business in IBM's

blind spot.I mentioned earlier that my mother doesn't really need a desktop

computer. Most users probably don't. That's a problem for Microsoft,

and they know it. If applications run on remote servers, no one

needs Windows. What will Microsoft do? Will they be able to use

their control of the desktop to prevent, or constrain, this new

generation of software?My guess is that Microsoft will develop some kind of server/desktop

hybrid, where the operating system works together with servers they

control. At a minimum, files will be centrally available for users

who want that. I don't expect Microsoft to go all the way to the

extreme of doing the computations on the server, with only a browser

for a client, if they can avoid it. If you only need a browser for

a client, you don't need Microsoft on the client, and if Microsoft

doesn't control the client, they can't push users towards their

server-based applications.I think Microsoft will have a hard time keeping the genie in the

bottle. There will be too many different types of clients for them

to control them all. And if Microsoft's applications only work

with some clients, competitors will be able to trump them by offering

applications that work from any client. [14]In a world of Web-based applications, there is no automatic place

for Microsoft. They may succeed in making themselves a place, but

I don't think they'll dominate this new world as they did the world

of desktop applications.It's not so much that a competitor will trip them up as that they

will trip over themselves. With the rise of Web-based software,

they will be facing not just technical problems but their own

wishful thinking. What they need to do is cannibalize their existing

business, and I can't see them facing that. The same single-mindedness

that has brought them this far will now be working against them.

IBM was in exactly the same situation, and they could not master

it. IBM made a late and half-hearted entry into the microcomputer

business because they were ambivalent about threatening their cash

cow, mainframe computing. Microsoft will likewise be hampered by

wanting to save the desktop. A cash cow can be a damned heavy

monkey on your back.I'm not saying that no one will dominate server-based applications.

Someone probably will eventually. But I think that there will be

a good long period of cheerful chaos, just as there was in the

early days of microcomputers. That was a good time for startups.

Lots of small companies flourished, and did it by making cool

things.Startups but More SoThe classic startup is fast and informal, with few people and little

money. Those few people work very hard, and technology magnifies

the effect of the decisions they make. If they win, they win big.In a startup writing Web-based applications, everything you associate

with startups is taken to an extreme. You can write and launch a

product with even fewer people and even less money. You have to

be even faster, and you can get away with being more informal.

You can literally launch your product as three guys sitting in the

living room of an apartment, and a server collocated at an ISP.

We did.Over time the teams have gotten smaller, faster, and more informal.

In 1960, software development meant a roomful of men with horn

rimmed glasses and narrow black neckties, industriously writing

ten lines of code a day on IBM coding forms. In 1980, it was a

team of eight to ten people wearing jeans to the office and typing

into vt100s. Now it's a couple of guys sitting in a living room

with laptops. (And jeans turn out not to be the last word in

informality.)Startups are stressful, and this, unfortunately, is also taken to

an extreme with Web-based applications.

Many software companies, especially at the beginning, have periods

where the developers slept under their desks and so on. The alarming

thing about Web-based software is that there is nothing to prevent

this becoming the default. The stories about sleeping under desks

usually end: then at last we shipped it and we all went home and

slept for a week. Web-based software never ships. You can work

16-hour days for as long as you want to. And because you can, and

your competitors can, you tend to be forced to. You can, so you

must. It's Parkinson's Law running in reverse.The worst thing is not the hours but the responsibility. Programmers

and system administrators traditionally each have their own separate

worries. Programmers have to worry about bugs, and system

administrators have to worry about infrastructure. Programmers

may spend a long day up to their elbows in source code, but at some

point they get to go home and forget about it. System administrators

never quite leave the job behind, but when they do get paged at

4:00 AM, they don't usually have to do anything very complicated.

With Web-based applications, these two kinds of stress get combined.

The programmers become system administrators, but without the

sharply defined limits that ordinarily make the job bearable.At Viaweb we spent the first six months just writing software. We

worked the usual long hours of an early startup. In a desktop

software company, this would have been the part where we were

working hard, but it felt like a vacation compared to the next

phase, when we took users onto our server. The second biggest

benefit of selling Viaweb to Yahoo (after the money) was to be able

to dump ultimate responsibility for the whole thing onto the

shoulders of a big company.Desktop software forces users to become system administrators.

Web-based software forces programmers to. There is less stress in

total, but more for the programmers. That's not necessarily bad

news. If you're a startup competing with a big company, it's good

news. [15] Web-based applications offer a straightforward way to

outwork your competitors. No startup asks for more.Just Good EnoughOne thing that might deter you from writing Web-based applications

is the lameness of Web pages as a UI. That is a problem, I admit.

There were a few things we would have really liked to add to

HTML and HTTP. What matters, though, is that Web pages are just

good enough.There is a parallel here with the first microcomputers. The

processors in those machines weren't actually intended to be the

CPUs of computers. They were designed to be used in things like

traffic lights. But guys like Ed Roberts, who designed the

Altair,

realized that they were just good enough. You could combine one

of these chips with some memory (256 bytes in the first Altair),

and front panel switches, and you'd have a working computer. Being

able to have your own computer was so exciting that there were

plenty of people who wanted to buy them, however limited.Web pages weren't designed to be a UI for applications, but they're

just good enough. And for a significant number of users, software

that you can use from any browser will be enough of a win in itself

to outweigh any awkwardness in the UI. Maybe you can't write the

best-looking spreadsheet using HTML, but you can write a spreadsheet

that several people can use simultaneously from different locations

without special client software, or that can incorporate live data

feeds, or that can page you when certain conditions are triggered.

More importantly, you can write new kinds of applications that

don't even have names yet. VisiCalc was not merely a microcomputer

version of a mainframe application, after all-- it was a new type

of application.Of course, server-based applications don't have to be Web-based.

You could have some other kind of client. But I'm pretty sure

that's a bad idea. It would be very convenient if you could assume

that everyone would install your client-- so convenient that you

could easily convince yourself that they all would-- but if they

don't, you're hosed. Because Web-based software assumes nothing

about the client, it will work anywhere the Web works. That's a

big advantage already, and the advantage will grow as new Web

devices proliferate. Users will like you because your software

just works, and your life will be easier because you won't have to

tweak it for every new client. [16]I feel like I've watched the evolution of the Web as closely as

anyone, and I can't predict what's going to happen with clients.

Convergence is probably coming, but where? I can't pick a winner.

One thing I can predict is conflict between AOL and Microsoft.

Whatever Microsoft's .NET turns out to be, it will probably involve

connecting the desktop to servers. Unless AOL fights back, they

will either be pushed aside or turned into a pipe between Microsoft

client and server software. If Microsoft and AOL get into a client

war, the only thing sure to work on both will be browsing the Web,

meaning Web-based applications will be the only kind that work

everywhere.How will it all play out? I don't know. And you don't have to

know if you bet on Web-based applications. No one can break that

without breaking browsing. The Web may not be the only way to

deliver software, but it's one that works now and will continue to

work for a long time. Web-based applications are cheap to develop,

and easy for even the smallest startup to deliver. They're a lot

of work, and of a particularly stressful kind, but that only makes

the odds better for startups.Why Not?E. B. White was amused to learn from a farmer friend that many

electrified fences don't have any current running through them.

The cows apparently learn to stay away from them, and after that

you don't need the current. "Rise up, cows!" he wrote, "Take your

liberty while despots snore!"If you're a hacker who has thought of one day starting a startup,

there are probably two things keeping you from doing it. One is

that you don't know anything about business. The other is that

you're afraid of competition. Neither of these fences have any

current in them.There are only two things you have to know about business: build

something users love, and make more than you spend. If you get

these two right, you'll be ahead of most startups. You can figure

out the rest as you go.You may not at first make more than you spend, but as long as the

gap is closing fast enough you'll be ok. If you start out underfunded,

it will at least encourage a habit of frugality. The less you

spend, the easier it is to make more than you spend. Fortunately,

it can be very cheap to launch a Web-based application. We launched

on under $10,000, and it would be even cheaper today. We had to

spend thousands on a server, and thousands more to get SSL. (The

only company selling SSL software at the time was Netscape.) Now

you can rent a much more powerful server, with SSL included, for

less than we paid for bandwidth alone. You could launch a Web-based

application now for less than the cost of a fancy office chair.As for building something users love, here are some general tips.

Start by making something clean and simple that you would want to

use yourself. Get a version 1.0 out fast, then continue to improve

the software, listening closely to the users as you do. The customer

is always right, but different customers are right about different

things; the least sophisticated users show you what you need to

simplify and clarify, and the most sophisticated tell you what

features you need to add. The best thing software can be is easy,

but the way to do this is to get the defaults right, not to limit

users' choices. Don't get complacent if your competitors' software

is lame; the standard to compare your software to is what it could

be, not what your current competitors happen to have. Use your

software yourself, all the time. Viaweb was supposed to be an

online store builder, but we used it to make our own site too.

Don't listen to marketing people or designers or product managers

just because of their job titles. If they have good ideas, use

them, but it's up to you to decide; software has to be designed by

hackers who understand design, not designers who know a little

about software. If you can't design software as well as implement

it, don't start a startup.Now let's talk about competition. What you're afraid of is not

presumably groups of hackers like you, but actual companies, with

offices and business plans and salesmen and so on, right? Well,

they are more afraid of you than you are of them, and they're right.

It's a lot easier for a couple of hackers to figure out how to rent

office space or hire sales people than it is for a company of any

size to get software written. I've been on both sides, and I know.

When Viaweb was bought by Yahoo, I suddenly found myself working

for a big company, and it was like trying to run through waist-deep

water.I don't mean to disparage Yahoo. They had some good hackers, and

the top management were real butt-kickers. For a big company, they

were exceptional. But they were still only about a tenth as

productive as a small startup. No big company can do much better

than that. What's scary about Microsoft is that a company so

big can develop software at all. They're like a mountain that

can walk.Don't be intimidated. You can do as much that Microsoft can't as

they can do that you can't. And no one can stop you. You don't

have to ask anyone's permission to develop Web-based applications.

You don't have to do licensing deals, or get shelf space in retail

stores, or grovel to have your application bundled with the OS.

You can deliver software right to the browser, and no one can get

between you and potential users without preventing them from browsing

the Web.You may not believe it, but I promise you, Microsoft is scared of

you. The complacent middle managers may not be, but Bill is,

because he was you once, back in 1975, the last time a new way of

delivering software appeared.Notes[1] Realizing that much of the money is in the services, companies

building lightweight clients have usually tried to combine the

hardware with an

online service.

This approach has not worked

well, partly because you need two different kinds of companies to

build consumer electronics and to run an online service, and partly

because users hate the idea. Giving away the razor and making

money on the blades may work for Gillette, but a razor is much

smaller commitment than a Web terminal. Cell phone handset makers

are satisfied to sell hardware without trying to capture the service

revenue as well. That should probably be the model for Internet

clients too. If someone just sold a nice-looking little box with

a Web browser that you could use to connect through any ISP, every

technophobe in the country would buy one.[2] Security always depends more on not screwing up than any design

decision, but the nature of server-based software will make developers

pay more attention to not screwing up. Compromising a server could

cause such damage that ASPs (that want to stay in business) are

likely to be careful about security.[3] In 1995, when we started Viaweb, Java applets were supposed to

be the technology everyone was going to use to develop server-based

applications. Applets seemed to us an old-fashioned idea. Download

programs to run on the client? Simpler just to go all the way and

run the programs on the server. We wasted little time

on applets, but countless other startups must have been lured into

this tar pit. Few can have escaped alive, or Microsoft could not

have gotten away with dropping Java in the most recent version of

Explorer.[4] This point is due to Trevor Blackwell, who adds "the cost of

writing software goes up more than linearly with its size. Perhaps

this is mainly due to fixing old bugs, and the cost can be more

linear if all bugs are found quickly."[5] The hardest kind of bug to find may be a variant of compound

bug where one bug happens to compensate for another. When you fix

one bug, the other becomes visible. But it will seem as if the

fix is at fault, since that was the last thing you changed.[6] Within Viaweb we once had a contest to describe the worst thing

about our software. Two customer support people tied for first

prize with entries I still shiver to recall. We fixed both problems

immediately.[7] Robert Morris wrote the ordering system, which shoppers used

to place orders. Trevor Blackwell wrote the image generator and

the manager, which merchants used to retrieve orders, view statistics,

and configure domain names etc. I wrote the editor, which merchants

used to build their sites. The ordering system and image generator

were written in C and C++, the manager mostly in Perl, and the editor

in Lisp.[8] Price discrimination is so pervasive (how often have you heard

a retailer claim that their buying power meant lower prices for

you?) that I was surprised to find it was outlawed in the U.S. by

the Robinson-Patman Act of 1936. This law does not appear to be

vigorously enforced.[9] In No Logo, Naomi Klein says that clothing brands favored by

"urban youth" do not try too hard to prevent shoplifting because

in their target market the shoplifters are also the fashion leaders.[10] Companies often wonder what to outsource and what not to.

One possible answer: outsource any job that's not directly exposed

to competitive pressure, because outsourcing it will thereby expose

it to competitive pressure.[11] The two guys were Dan Bricklin and Bob Frankston. Dan wrote

a prototype in Basic in a couple days, then over the course of the

next year they worked together (mostly at night) to make a more

powerful version written in 6502 machine language. Dan was at

Harvard Business School at the time and Bob nominally had a day

job writing software. "There was no great risk in doing a business,"

Bob wrote, "If it failed it failed. No big deal."[12] It's not quite as easy as I make it sound. It took a painfully

long time for word of mouth to get going, and we did not start to

get a lot of press coverage until we hired a

PR firm

(admittedly

the best in the business) for $16,000 per month. However, it was

true that the only significant channel was our own Web site.[13] If the Mac was so great, why did it lose? Cost, again.

Microsoft concentrated on the software business, and unleashed a

swarm of cheap component suppliers on Apple hardware. It did not

help, either, that suits took over during a critical period.[14] One thing that would help Web-based applications, and help

keep the next generation of software from being overshadowed by

Microsoft, would be a good open-source browser. Mozilla is

open-source but seems to have suffered from having been corporate

software for so long. A small, fast browser that was actively

maintained would be a great thing in itself, and would probably

also encourage companies to build little Web appliances.Among other things, a proper open-source browser would cause HTTP

and HTML to continue to evolve (as e.g. Perl has). It would help

Web-based applications greatly to be able to distinguish between

selecting a link and following it; all you'd need to do this would

be a trivial enhancement of HTTP, to allow multiple urls in a

request. Cascading menus would also be good.If you want to change the world, write a new Mosaic. Think it's

too late? In 1998 a lot of people thought it was too late to launch

a new search engine, but Google proved them wrong. There is always

room for something new if the current options suck enough. Make

sure it works on all the free OSes first-- new things start with

their users.[15] Trevor Blackwell, who probably knows more about this from

personal experience than anyone, writes:"I would go farther in saying that because server-based software

is so hard on the programmers, it causes a fundamental economic

shift away from large companies. It requires the kind of intensity

and dedication from programmers that they will only be willing to

provide when it's their own company. Software companies can hire

skilled people to work in a not-too-demanding environment, and can

hire unskilled people to endure hardships, but they can't hire

highly skilled people to bust their asses. Since capital is no

longer needed, big companies have little to bring to the table."[16] In the original version of this essay, I advised avoiding

Javascript. That was a good plan in 2001, but Javascript now works.

Thanks to Sarah Harlin, Trevor Blackwell, Robert Morris, Eric Raymond, Ken Anderson,

and Dan Giffin for reading drafts of this paper; to Dan Bricklin and

Bob Frankston for information about VisiCalc; and again to Ken Anderson

for inviting me to speak at BBN.

You'll find this essay and 14 others in

Hackers & Painters.

Some Technical DetailsJapanese TranslationMicrosoft finally agreesGates Email

The Roots of Lisp

May 2001

(I wrote this article to help myself understand exactly

what McCarthy discovered. You don't need to know this stuff

to program in Lisp, but it should be helpful to

anyone who wants to

understand the essence of Lisp both in the sense of its

origins and its semantic core. The fact that it has such a core

is one of Lisp's distinguishing features, and the reason why,

unlike other languages, Lisp has dialects.)In 1960, John

McCarthy published a remarkable paper in

which he did for programming something like what Euclid did for

geometry. He showed how, given a handful of simple

operators and a notation for functions, you can

build a whole programming language.

He called this language Lisp, for "List Processing,"

because one of his key ideas was to use a simple

data structure called a list for both

code and data.It's worth understanding what McCarthy discovered, not

just as a landmark in the history of computers, but as

a model for what programming is tending to become in

our own time. It seems to me that there have been

two really clean, consistent models of programming so

far: the C model and the Lisp model.

These two seem points of high ground, with swampy lowlands

between them. As computers have grown more powerful,

the new languages being developed have been moving

steadily toward the Lisp model. A popular recipe

for new programming languages in the past 20 years

has been to take the C model of computing and add to

it, piecemeal, parts taken from the Lisp model,

like runtime typing and garbage collection.In this article I'm going to try to explain in the

simplest possible terms what McCarthy discovered.

The point is not just to learn about an interesting

theoretical result someone figured out forty years ago,

but to show where languages are heading.

The unusual thing about Lisp in fact, the defining

quality of Lisp is that it can be written in

itself. To understand what McCarthy meant by this,

we're going to retrace his steps, with his mathematical

notation translated into running Common Lisp code.Complete Article (Postscript)What Made Lisp DifferentThe CodeChinese TranslationJapanese TranslationPortuguese TranslationKorean Translation

Five Questions about Language Design

May 2001

(These are some notes I made

for a panel discussion on programming language design

at MIT on May 10, 2001.)1. Programming Languages Are for People.Programming languages

are how people talk to computers. The computer would be just as

happy speaking any language that was unambiguous. The reason we

have high level languages is because people can't deal with

machine language. The point of programming

languages is to prevent our poor frail human brains from being

overwhelmed by a mass of detail.Architects know that some kinds of design problems are more personal

than others. One of the cleanest, most abstract design problems

is designing bridges. There your job is largely a matter of spanning

a given distance with the least material. The other end of the

spectrum is designing chairs. Chair designers have to spend their

time thinking about human butts.Software varies in the same way. Designing algorithms for routing

data through a network is a nice, abstract problem, like designing

bridges. Whereas designing programming languages is like designing

chairs: it's all about dealing with human weaknesses.Most of us hate to acknowledge this. Designing systems of great

mathematical elegance sounds a lot more appealing to most of us

than pandering to human weaknesses. And there is a role for mathematical

elegance: some kinds of elegance make programs easier to understand.

But elegance is not an end in itself.And when I say languages have to be designed to suit human weaknesses,

I don't mean that languages have to be designed for bad programmers.

In fact I think you ought to design for the

best programmers, but

even the best programmers have limitations. I don't think anyone

would like programming in a language where all the variables were

the letter x with integer subscripts.2. Design for Yourself and Your Friends.If you look at the history of programming languages, a lot of the best

ones were languages designed for their own authors to use, and a

lot of the worst ones were designed for other people to use.When languages are designed for other people, it's always a specific

group of other people: people not as smart as the language designer.

So you get a language that talks down to you. Cobol is the most

extreme case, but a lot of languages are pervaded by this spirit.It has nothing to do with how abstract the language is. C is pretty

low-level, but it was designed for its authors to use, and that's

why hackers like it.The argument for designing languages for bad programmers is that

there are more bad programmers than good programmers. That may be

so. But those few good programmers write a disproportionately

large percentage of the software.I'm interested in the question, how do you design a language that

the very best hackers will like? I happen to think this is

identical to the question, how do you design a good programming

language?, but even if it isn't, it is at least an interesting

question.3. Give the Programmer as Much Control as Possible.Many languages

(especially the ones designed for other people) have the attitude

of a governess: they try to prevent you from

doing things that they think aren't good for you. I like the

opposite approach: give the programmer as much

control as you can.When I first learned Lisp, what I liked most about it was

that it considered me an equal partner. In the other languages

I had learned up till then, there was the language and there was my

program, written in the language, and the two were very separate.

But in Lisp the functions and macros I wrote were just like those

that made up the language itself. I could rewrite the language

if I wanted. It had the same appeal as open-source software.4. Aim for Brevity.Brevity is underestimated and even scorned.

But if you look into the hearts of hackers, you'll see that they

really love it. How many times have you heard hackers speak fondly

of how in, say, APL, they could do amazing things with just a couple

lines of code? I think anything that really smart people really

love is worth paying attention to.I think almost anything

you can do to make programs shorter is good. There should be lots

of library functions; anything that can be implicit should be;

the syntax should be terse to a fault; even the names of things

should be short.And it's not only programs that should be short. The manual should

be thin as well. A good part of manuals is taken up with clarifications

and reservations and warnings and special cases. If you force

yourself to shorten the manual, in the best case you do it by fixing

the things in the language that required so much explanation.5. Admit What Hacking Is.A lot of people wish that hacking was

mathematics, or at least something like a natural science. I think

hacking is more like architecture. Architecture is

related to physics, in the sense that architects have to design

buildings that don't fall down, but the actual goal of architects

is to make great buildings, not to make discoveries about statics.What hackers like to do is make great programs.

And I think, at least in our own minds, we have to remember that it's

an admirable thing to write great programs, even when this work

doesn't translate easily into the conventional intellectual

currency of research papers. Intellectually, it is just as

worthwhile to design a language programmers will love as it is to design a

horrible one that embodies some idea you can publish a paper

about.1. How to Organize Big Libraries?Libraries are becoming an

increasingly important component of programming languages. They're

also getting bigger, and this can be dangerous. If it takes longer

to find the library function that will do what you want than it

would take to write it yourself, then all that code is doing nothing

but make your manual thick. (The Symbolics manuals were a case in

point.) So I think we will have to work on ways to organize

libraries. The ideal would be to design them so that the programmer

could guess what library call would do the right thing.2. Are People Really Scared of Prefix Syntax?This is an open

problem in the sense that I have wondered about it for years and

still don't know the answer. Prefix syntax seems perfectly natural

to me, except possibly for math. But it could be that a lot of

Lisp's unpopularity is simply due to having an unfamiliar syntax.

Whether to do anything about it, if it is true, is another question.

3. What Do You Need for Server-Based Software?

I think a lot of the most exciting new applications that get written

in the next twenty years will be Web-based applications, meaning

programs that sit on the server and talk to you through a Web

browser. And to write these kinds of programs we may need some

new things.One thing we'll need is support for the new way that server-based

apps get released. Instead of having one or two big releases a

year, like desktop software, server-based apps get released as a

series of small changes. You may have as many as five or ten

releases a day. And as a rule everyone will always use the latest

version.You know how you can design programs to be debuggable?

Well, server-based software likewise has to be designed to be

changeable. You have to be able to change it easily, or at least

to know what is a small change and what is a momentous one.Another thing that might turn out to be useful for server based

software, surprisingly, is continuations. In Web-based software

you can use something like continuation-passing style to get the

effect of subroutines in the inherently

stateless world of a Web

session. Maybe it would be worthwhile having actual continuations,

if it was not too expensive.4. What New Abstractions Are Left to Discover?I'm not sure how

reasonable a hope this is, but one thing I would really love to

do, personally, is discover a new abstraction-- something that would

make as much of a difference as having first class functions or

recursion or even keyword parameters. This may be an impossible

dream. These things don't get discovered that often. But I am always

looking.1. You Can Use Whatever Language You Want.Writing application

programs used to mean writing desktop software. And in desktop

software there is a big bias toward writing the application in the

same language as the operating system. And so ten years ago,

writing software pretty much meant writing software in C.

Eventually a tradition evolved:

application programs must not be written in unusual languages.

And this tradition had so long to develop that nontechnical people

like managers and venture capitalists also learned it.Server-based software blows away this whole model. With server-based

software you can use any language you want. Almost nobody understands

this yet (especially not managers and venture capitalists).

A few hackers understand it, and that's why we even hear

about new, indy languages like Perl and Python. We're not hearing

about Perl and Python because people are using them to write Windows

apps.What this means for us, as people interested in designing programming

languages, is that there is now potentially an actual audience for

our work.2. Speed Comes from Profilers.Language designers, or at least

language implementors, like to write compilers that generate fast

code. But I don't think this is what makes languages fast for users.

Knuth pointed out long ago that speed only matters in a few critical

bottlenecks. And anyone who's tried it knows that you can't guess

where these bottlenecks are. Profilers are the answer.Language designers are solving the wrong problem. Users don't need

benchmarks to run fast. What they need is a language that can show

them what parts of their own programs need to be rewritten. That's

where speed comes from in practice. So maybe it would be a net

win if language implementors took half the time they would

have spent doing compiler optimizations and spent it writing a

good profiler instead.3. You Need an Application to Drive the Design of a Language.This may not be an absolute rule, but it seems like the best languages

all evolved together with some application they were being used to

write. C was written by people who needed it for systems programming.

Lisp was developed partly to do symbolic differentiation, and

McCarthy was so eager to get started that he was writing differentiation

programs even in the first paper on Lisp, in 1960.It's especially good if your application solves some new problem.

That will tend to drive your language to have new features that

programmers need. I personally am interested in writing

a language that will be good for writing server-based applications.[During the panel, Guy Steele also made this point, with the

additional suggestion that the application should not consist of

writing the compiler for your language, unless your language

happens to be intended for writing compilers.]4. A Language Has to Be Good for Writing Throwaway Programs.You know what a throwaway program is: something you write quickly for

some limited task. I think if you looked around you'd find that

a lot of big, serious programs started as throwaway programs. I

would not be surprised if most programs started as throwaway

programs. And so if you want to make a language that's good for

writing software in general, it has to be good for writing throwaway

programs, because that is the larval stage of most software.5. Syntax Is Connected to Semantics.It's traditional to think of

syntax and semantics as being completely separate. This will

sound shocking, but it may be that they aren't.

I think that what you want in your language may be related

to how you express it.I was talking recently to Robert Morris, and he pointed out that

operator overloading is a bigger win in languages with infix

syntax. In a language with prefix syntax, any function you define

is effectively an operator. If you want to define a plus for a

new type of number you've made up, you can just define a new function

to add them. If you do that in a language with infix syntax,

there's a big difference in appearance between the use of an

overloaded operator and a function call.1. New Programming Languages.Back in the 1970s

it was fashionable to design new programming languages. Recently

it hasn't been. But I think server-based software will make new

languages fashionable again. With server-based software, you can

use any language you want, so if someone does design a language that

actually seems better than others that are available, there will be

people who take a risk and use it.2. Time-Sharing.Richard Kelsey gave this as an idea whose time

has come again in the last panel, and I completely agree with him.

My guess (and Microsoft's guess, it seems) is that much computing

will move from the desktop onto remote servers. In other words,

time-sharing is back. And I think there will need to be support

for it at the language level. For example, I know that Richard

and Jonathan Rees have done a lot of work implementing process

scheduling within Scheme 48.3. Efficiency.Recently it was starting to seem that computers

were finally fast enough. More and more we were starting to hear

about byte code, which implies to me at least that we feel we have

cycles to spare. But I don't think we will, with server-based

software. Someone is going to have to pay for the servers that

the software runs on, and the number of users they can support per

machine will be the divisor of their capital cost.So I think efficiency will matter, at least in computational

bottlenecks. It will be especially important to do i/o fast,

because server-based applications do a lot of i/o.It may turn out that byte code is not a win, in the end. Sun and

Microsoft seem to be facing off in a kind of a battle of the byte

codes at the moment. But they're doing it because byte code is a

convenient place to insert themselves into the process, not because

byte code is in itself a good idea. It may turn out that this

whole battleground gets bypassed. That would be kind of amusing.1. Clients.This is just a guess, but my guess is that

the winning model for most applications will be purely server-based.

Designing software that works on the assumption that everyone will

have your client is like designing a society on the assumption that

everyone will just be honest. It would certainly be convenient, but

you have to assume it will never happen.I think there will be a proliferation of devices that have some

kind of Web access, and all you'll be able to assume about them is

that they can support simple html and forms. Will you have a

browser on your cell phone? Will there be a phone in your palm

pilot? Will your blackberry get a bigger screen? Will you be able

to browse the Web on your gameboy? Your watch? I don't know.

And I don't have to know if I bet on

everything just being on the server. It's

just so much more robust to have all the

brains on the server.2. Object-Oriented Programming.I realize this is a

controversial one, but I don't think object-oriented programming

is such a big deal. I think it is a fine model for certain kinds

of applications that need that specific kind of data structure,

like window systems, simulations, and cad programs. But I don't

see why it ought to be the model for all programming.I think part of the reason people in big companies like object-oriented

programming is because it yields a lot of what looks like work.

Something that might naturally be represented as, say, a list of

integers, can now be represented as a class with all kinds of

scaffolding and hustle and bustle.Another attraction of

object-oriented programming is that methods give you some of the

effect of first class functions. But this is old news to Lisp

programmers. When you have actual first class functions, you can

just use them in whatever way is appropriate to the task at hand,

instead of forcing everything into a mold of classes and methods.What this means for language design, I think, is that you shouldn't

build object-oriented programming in too deeply. Maybe the

answer is to offer more general, underlying stuff, and let people design

whatever object systems they want as libraries.3. Design by Committee.Having your language designed by a committee is a big pitfall,

and not just for the reasons everyone knows about. Everyone

knows that committees tend to yield lumpy, inconsistent designs.

But I think a greater danger is that they won't take risks.

When one person is in charge he can take risks

that a committee would never agree on.Is it necessary to take risks to design a good language though?

Many people might suspect

that language design is something where you should stick fairly

close to the conventional wisdom. I bet this isn't true.

In everything else people do, reward is proportionate to risk.

Why should language design be any different?Japanese Translation

Being Popular

May 2001(This article was written as a kind of business plan for a

new language.

So it is missing (because it takes for granted) the most important

feature of a good programming language: very powerful abstractions.)A friend of mine once told an eminent operating systems

expert that he wanted to design a really good

programming language. The expert told him that it would be a

waste of time, that programming languages don't become popular

or unpopular based on their merits, and so no matter how

good his language was, no one would use it. At least, that

was what had happened to the language he had designed.What does make a language popular? Do popular

languages deserve their popularity? Is it worth trying to

define a good programming language? How would you do it?I think the answers to these questions can be found by looking

at hackers, and learning what they want. Programming

languages are for hackers, and a programming language

is good as a programming language (rather than, say, an

exercise in denotational semantics or compiler design)

if and only if hackers like it.1 The Mechanics of PopularityIt's true, certainly, that most people don't choose programming

languages simply based on their merits. Most programmers are told

what language to use by someone else. And yet I think the effect

of such external factors on the popularity of programming languages

is not as great as it's sometimes thought to be. I think a bigger

problem is that a hacker's idea of a good programming language is

not the same as most language designers'.Between the two, the hacker's opinion is the one that matters.

Programming languages are not theorems. They're tools, designed

for people, and they have to be designed to suit human strengths

and weaknesses as much as shoes have to be designed for human feet.

If a shoe pinches when you put it on, it's a bad shoe, however

elegant it may be as a piece of sculpture.It may be that the majority of programmers can't tell a good language

from a bad one. But that's no different with any other tool. It

doesn't mean that it's a waste of time to try designing a good

language. Expert hackers

can tell a good language when they see

one, and they'll use it. Expert hackers are a tiny minority,

admittedly, but that tiny minority write all the good software,

and their influence is such that the rest of the programmers will

tend to use whatever language they use. Often, indeed, it is not

merely influence but command: often the expert hackers are the very

people who, as their bosses or faculty advisors, tell the other

programmers what language to use.The opinion of expert hackers is not the only force that determines

the relative popularity of programming languages — legacy software

(Cobol) and hype (Ada, Java) also play a role — but I think it is

the most powerful force over the long term. Given an initial critical

mass and enough time, a programming language probably becomes about

as popular as it deserves to be. And popularity further separates

good languages from bad ones, because feedback from real live users

always leads to improvements. Look at how much any popular language

has changed during its life. Perl and Fortran are extreme cases,

but even Lisp has changed a lot. Lisp 1.5 didn't have macros, for

example; these evolved later, after hackers at MIT had spent a

couple years using Lisp to write real programs. [1]So whether or not a language has to be good to be popular, I think

a language has to be popular to be good. And it has to stay popular

to stay good. The state of the art in programming languages doesn't

stand still. And yet the Lisps we have today are still pretty much

what they had at MIT in the mid-1980s, because that's the last time

Lisp had a sufficiently large and demanding user base.Of course, hackers have to know about a language before they can

use it. How are they to hear? From other hackers. But there has to

be some initial group of hackers using the language for others even

to hear about it. I wonder how large this group has to be; how many

users make a critical mass? Off the top of my head, I'd say twenty.

If a language had twenty separate users, meaning twenty users who

decided on their own to use it, I'd consider it to be real.Getting there can't be easy. I would not be surprised if it is

harder to get from zero to twenty than from twenty to a thousand.

The best way to get those initial twenty users is probably to use

a trojan horse: to give people an application they want, which

happens to be written in the new language.2 External FactorsLet's start by acknowledging one external factor that does affect

the popularity of a programming language. To become popular, a

programming language has to be the scripting language of a popular

system. Fortran and Cobol were the scripting languages of early

IBM mainframes. C was the scripting language of Unix, and so, later,

was Perl. Tcl is the scripting language of Tk. Java and Javascript

are intended to be the scripting languages of web browsers.Lisp is not a massively popular language because it is not the

scripting language of a massively popular system. What popularity

it retains dates back to the 1960s and 1970s, when it was the

scripting language of MIT. A lot of the great programmers of the

day were associated with MIT at some point. And in the early 1970s,

before C, MIT's dialect of Lisp, called MacLisp, was one of the

only programming languages a serious hacker would want to use.Today Lisp is the scripting language of two moderately popular

systems, Emacs and Autocad, and for that reason I suspect that most

of the Lisp programming done today is done in Emacs Lisp or AutoLisp.Programming languages don't exist in isolation. To hack is a

transitive verb — hackers are usually hacking something — and in

practice languages are judged relative to whatever they're used to

hack. So if you want to design a popular language, you either have

to supply more than a language, or you have to design your language

to replace the scripting language of some existing system.Common Lisp is unpopular partly because it's an orphan. It did

originally come with a system to hack: the Lisp Machine. But Lisp

Machines (along with parallel computers) were steamrollered by the

increasing power of general purpose processors in the 1980s. Common

Lisp might have remained popular if it had been a good scripting

language for Unix. It is, alas, an atrociously bad one.One way to describe this situation is to say that a language isn't

judged on its own merits. Another view is that a programming language

really isn't a programming language unless it's also the scripting

language of something. This only seems unfair if it comes as a

surprise. I think it's no more unfair than expecting a programming

language to have, say, an implementation. It's just part of what

a programming language is.A programming language does need a good implementation, of course,

and this must be free. Companies will pay for software, but individual

hackers won't, and it's the hackers you need to attract.A language also needs to have a book about it. The book should be

thin, well-written, and full of good examples. K&R is the ideal

here. At the moment I'd almost say that a language has to have a

book published by O'Reilly. That's becoming the test of mattering

to hackers.There should be online documentation as well. In fact, the book

can start as online documentation. But I don't think that physical

books are outmoded yet. Their format is convenient, and the de

facto censorship imposed by publishers is a useful if imperfect

filter. Bookstores are one of the most important places for learning

about new languages.3 BrevityGiven that you can supply the three things any language needs — a

free implementation, a book, and something to hack — how do you

make a language that hackers will like?One thing hackers like is brevity. Hackers are lazy, in the same

way that mathematicians and modernist architects are lazy: they

hate anything extraneous. It would not be far from the truth to

say that a hacker about to write a program decides what language

to use, at least subconsciously, based on the total number of

characters he'll have to type. If this isn't precisely how hackers

think, a language designer would do well to act as if it were.It is a mistake to try to baby the user with long-winded expressions

that are meant to resemble English. Cobol is notorious for this

flaw. A hacker would consider being asked to writeadd x to y giving zinstead ofz = x+yas something between an insult to his intelligence and a sin against

God.It has sometimes been said that Lisp should use first and rest

instead of car and cdr, because it would make programs easier to

read. Maybe for the first couple hours. But a hacker can learn

quickly enough that car means the first element of a list and cdr

means the rest. Using first and rest means 50% more typing. And

they are also different lengths, meaning that the arguments won't

line up when they're called, as car and cdr often are, in successive

lines. I've found that it matters a lot how code lines up on the

page. I can barely read Lisp code when it is set in a variable-width

font, and friends say this is true for other languages too.Brevity is one place where strongly typed languages lose. All other

things being equal, no one wants to begin a program with a bunch

of declarations. Anything that can be implicit, should be.The individual tokens should be short as well. Perl and Common Lisp

occupy opposite poles on this question. Perl programs can be almost

cryptically dense, while the names of built-in Common Lisp operators

are comically long. The designers of Common Lisp probably expected

users to have text editors that would type these long names for

them. But the cost of a long name is not just the cost of typing

it. There is also the cost of reading it, and the cost of the space

it takes up on your screen.4 HackabilityThere is one thing more important than brevity to a hacker: being

able to do what you want. In the history of programming languages

a surprising amount of effort has gone into preventing programmers

from doing things considered to be improper. This is a dangerously

presumptuous plan. How can the language designer know what the

programmer is going to need to do? I think language designers would

do better to consider their target user to be a genius who will

need to do things they never anticipated, rather than a bumbler

who needs to be protected from himself. The bumbler will shoot

himself in the foot anyway. You may save him from referring to

variables in another package, but you can't save him from writing

a badly designed program to solve the wrong problem, and taking

forever to do it.Good programmers often want to do dangerous and unsavory things.

By unsavory I mean things that go behind whatever semantic facade

the language is trying to present: getting hold of the internal

representation of some high-level abstraction, for example. Hackers

like to hack, and hacking means getting inside things and second

guessing the original designer.Let yourself be second guessed. When you make any tool, people use

it in ways you didn't intend, and this is especially true of a

highly articulated tool like a programming language. Many a hacker

will want to tweak your semantic model in a way that you never

imagined. I say, let them; give the programmer access to as much

internal stuff as you can without endangering runtime systems like

the garbage collector.In Common Lisp I have often wanted to iterate through the fields

of a struct — to comb out references to a deleted object, for example,

or find fields that are uninitialized. I know the structs are just

vectors underneath. And yet I can't write a general purpose function

that I can call on any struct. I can only access the fields by

name, because that's what a struct is supposed to mean.A hacker may only want to subvert the intended model of things once

or twice in a big program. But what a difference it makes to be

able to. And it may be more than a question of just solving a

problem. There is a kind of pleasure here too. Hackers share the

surgeon's secret pleasure in poking about in gross innards, the

teenager's secret pleasure in popping zits. [2] For boys, at least,

certain kinds of horrors are fascinating. Maxim magazine publishes

an annual volume of photographs, containing a mix of pin-ups and

grisly accidents. They know their audience.Historically, Lisp has been good at letting hackers have their way.

The political correctness of Common Lisp is an aberration. Early

Lisps let you get your hands on everything. A good deal of that

spirit is, fortunately, preserved in macros. What a wonderful thing,

to be able to make arbitrary transformations on the source code.Classic macros are a real hacker's tool — simple, powerful, and

dangerous. It's so easy to understand what they do: you call a

function on the macro's arguments, and whatever it returns gets

inserted in place of the macro call. Hygienic macros embody the

opposite principle. They try to protect you from understanding what

they're doing. I have never heard hygienic macros explained in one

sentence. And they are a classic example of the dangers of deciding

what programmers are allowed to want. Hygienic macros are intended

to protect me from variable capture, among other things, but variable

capture is exactly what I want in some macros.A really good language should be both clean and dirty: cleanly

designed, with a small core of well understood and highly orthogonal

operators, but dirty in the sense that it lets hackers have their

way with it. C is like this. So were the early Lisps. A real hacker's

language will always have a slightly raffish character.A good programming language should have features that make the kind

of people who use the phrase "software engineering" shake their

heads disapprovingly. At the other end of the continuum are languages

like Ada and Pascal, models of propriety that are good for teaching

and not much else.5 Throwaway ProgramsTo be attractive to hackers, a language must be good for writing

the kinds of programs they want to write. And that means, perhaps

surprisingly, that it has to be good for writing throwaway programs.A throwaway program is a program you write quickly for some limited

task: a program to automate some system administration task, or

generate test data for a simulation, or convert data from one format

to another. The surprising thing about throwaway programs is that,

like the "temporary" buildings built at so many American universities

during World War II, they often don't get thrown away. Many evolve

into real programs, with real features and real users.I have a hunch that the best big programs begin life this way,

rather than being designed big from the start, like the Hoover Dam.

It's terrifying to build something big from scratch. When people

take on a project that's too big, they become overwhelmed. The

project either gets bogged down, or the result is sterile and

wooden: a shopping mall rather than a real downtown, Brasilia rather

than Rome, Ada rather than C.Another way to get a big program is to start with a throwaway

program and keep improving it. This approach is less daunting, and

the design of the program benefits from evolution. I think, if one

looked, that this would turn out to be the way most big programs

were developed. And those that did evolve this way are probably

still written in whatever language they were first written in,

because it's rare for a program to be ported, except for political

reasons. And so, paradoxically, if you want to make a language that

is used for big systems, you have to make it good for writing

throwaway programs, because that's where big systems come from.Perl is a striking example of this idea. It was not only designed

for writing throwaway programs, but was pretty much a throwaway

program itself. Perl began life as a collection of utilities for

generating reports, and only evolved into a programming language

as the throwaway programs people wrote in it grew larger. It was

not until Perl 5 (if then) that the language was suitable for

writing serious programs, and yet it was already massively popular.What makes a language good for throwaway programs? To start with,

it must be readily available. A throwaway program is something that

you expect to write in an hour. So the language probably must

already be installed on the computer you're using. It can't be

something you have to install before you use it. It has to be there.

C was there because it came with the operating system. Perl was

there because it was originally a tool for system administrators,

and yours had already installed it.Being available means more than being installed, though. An

interactive language, with a command-line interface, is more

available than one that you have to compile and run separately. A

popular programming language should be interactive, and start up

fast.Another thing you want in a throwaway program is brevity. Brevity

is always attractive to hackers, and never more so than in a program

they expect to turn out in an hour.6 LibrariesOf course the ultimate in brevity is to have the program already

written for you, and merely to call it. And this brings us to what

I think will be an increasingly important feature of programming

languages: library functions. Perl wins because it has large

libraries for manipulating strings. This class of library functions

are especially important for throwaway programs, which are often

originally written for converting or extracting data. Many Perl

programs probably begin as just a couple library calls stuck

together.I think a lot of the advances that happen in programming languages

in the next fifty years will have to do with library functions. I

think future programming languages will have libraries that are as

carefully designed as the core language. Programming language design

will not be about whether to make your language strongly or weakly

typed, or object oriented, or functional, or whatever, but about

how to design great libraries. The kind of language designers who

like to think about how to design type systems may shudder at this.

It's almost like writing applications! Too bad. Languages are for

programmers, and libraries are what programmers need.It's hard to design good libraries. It's not simply a matter of

writing a lot of code. Once the libraries get too big, it can

sometimes take longer to find the function you need than to write

the code yourself. Libraries need to be designed using a small set

of orthogonal operators, just like the core language. It ought to

be possible for the programmer to guess what library call will do

what he needs.Libraries are one place Common Lisp falls short. There are only

rudimentary libraries for manipulating strings, and almost none

for talking to the operating system. For historical reasons, Common

Lisp tries to pretend that the OS doesn't exist. And because you

can't talk to the OS, you're unlikely to be able to write a serious

program using only the built-in operators in Common Lisp. You have

to use some implementation-specific hacks as well, and in practice

these tend not to give you everything you want. Hackers would think

a lot more highly of Lisp if Common Lisp had powerful string

libraries and good OS support.7 SyntaxCould a language with Lisp's syntax, or more precisely, lack of

syntax, ever become popular? I don't know the answer to this

question. I do think that syntax is not the main reason Lisp isn't

currently popular. Common Lisp has worse problems than unfamiliar

syntax. I know several programmers who are comfortable with prefix

syntax and yet use Perl by default, because it has powerful string

libraries and can talk to the os.There are two possible problems with prefix notation: that it is

unfamiliar to programmers, and that it is not dense enough. The

conventional wisdom in the Lisp world is that the first problem is

the real one. I'm not so sure. Yes, prefix notation makes ordinary

programmers panic. But I don't think ordinary programmers' opinions

matter. Languages become popular or unpopular based on what expert

hackers think of them, and I think expert hackers might be able to

deal with prefix notation. Perl syntax can be pretty incomprehensible,

but that has not stood in the way of Perl's popularity. If anything

it may have helped foster a Perl cult.A more serious problem is the diffuseness of prefix notation. For

expert hackers, that really is a problem. No one wants to write

(aref a x y) when they could write a[x,y].In this particular case there is a way to finesse our way out of

the problem. If we treat data structures as if they were functions

on indexes, we could write (a x y) instead, which is even shorter

than the Perl form. Similar tricks may shorten other types of

expressions.We can get rid of (or make optional) a lot of parentheses by making

indentation significant. That's how programmers read code anyway:

when indentation says one thing and delimiters say another, we go

by the indentation. Treating indentation as significant would

eliminate this common source of bugs as well as making programs

shorter.Sometimes infix syntax is easier to read. This is especially true

for math expressions. I've used Lisp my whole programming life and

I still don't find prefix math expressions natural. And yet it is

convenient, especially when you're generating code, to have operators

that take any number of arguments. So if we do have infix syntax,

it should probably be implemented as some kind of read-macro.I don't think we should be religiously opposed to introducing syntax

into Lisp, as long as it translates in a well-understood way into

underlying s-expressions. There is already a good deal of syntax

in Lisp. It's not necessarily bad to introduce more, as long as no

one is forced to use it. In Common Lisp, some delimiters are reserved

for the language, suggesting that at least some of the designers

intended to have more syntax in the future.One of the most egregiously unlispy pieces of syntax in Common Lisp

occurs in format strings; format is a language in its own right,

and that language is not Lisp. If there were a plan for introducing

more syntax into Lisp, format specifiers might be able to be included

in it. It would be a good thing if macros could generate format

specifiers the way they generate any other kind of code.An eminent Lisp hacker told me that his copy of CLTL falls open to

the section format. Mine too. This probably indicates room for

improvement. It may also mean that programs do a lot of I/O.8 EfficiencyA good language, as everyone knows, should generate fast code. But

in practice I don't think fast code comes primarily from things

you do in the design of the language. As Knuth pointed out long

ago, speed only matters in certain critical bottlenecks. And as

many programmers have observed since, one is very often mistaken

about where these bottlenecks are.So, in practice, the way to get fast code is to have a very good

profiler, rather than by, say, making the language strongly typed.

You don't need to know the type of every argument in every call in

the program. You do need to be able to declare the types of arguments

in the bottlenecks. And even more, you need to be able to find out

where the bottlenecks are.One complaint people have had with Lisp is that it's hard to tell

what's expensive. This might be true. It might also be inevitable,

if you want to have a very abstract language. And in any case I

think good profiling would go a long way toward fixing the problem:

you'd soon learn what was expensive.Part of the problem here is social. Language designers like to

write fast compilers. That's how they measure their skill. They

think of the profiler as an add-on, at best. But in practice a good

profiler may do more to improve the speed of actual programs written

in the language than a compiler that generates fast code. Here,

again, language designers are somewhat out of touch with their

users. They do a really good job of solving slightly the wrong

problem.It might be a good idea to have an active profiler — to push

performance data to the programmer instead of waiting for him to

come asking for it. For example, the editor could display bottlenecks

in red when the programmer edits the source code. Another approach

would be to somehow represent what's happening in running programs.

This would be an especially big win in server-based applications,

where you have lots of running programs to look at. An active

profiler could show graphically what's happening in memory as a

program's running, or even make sounds that tell what's happening.Sound is a good cue to problems. In one place I worked, we had a

big board of dials showing what was happening to our web servers.

The hands were moved by little servomotors that made a slight noise

when they turned. I couldn't see the board from my desk, but I

found that I could tell immediately, by the sound, when there was

a problem with a server.It might even be possible to write a profiler that would automatically

detect inefficient algorithms. I would not be surprised if certain

patterns of memory access turned out to be sure signs of bad

algorithms. If there were a little guy running around inside the

computer executing our programs, he would probably have as long

and plaintive a tale to tell about his job as a federal government

employee. I often have a feeling that I'm sending the processor on

a lot of wild goose chases, but I've never had a good way to look

at what it's doing.A number of Lisps now compile into byte code, which is then executed

by an interpreter. This is usually done to make the implementation

easier to port, but it could be a useful language feature. It might

be a good idea to make the byte code an official part of the

language, and to allow programmers to use inline byte code in

bottlenecks. Then such optimizations would be portable too.The nature of speed, as perceived by the end-user, may be changing.

With the rise of server-based applications, more and more programs

may turn out to be i/o-bound. It will be worth making i/o fast.

The language can help with straightforward measures like simple,

fast, formatted output functions, and also with deep structural

changes like caching and persistent objects.Users are interested in response time. But another kind of efficiency

will be increasingly important: the number of simultaneous users

you can support per processor. Many of the interesting applications

written in the near future will be server-based, and the number of

users per server is the critical question for anyone hosting such

applications. In the capital cost of a business offering a server-based

application, this is the divisor.For years, efficiency hasn't mattered much in most end-user

applications. Developers have been able to assume that each user

would have an increasingly powerful processor sitting on their

desk. And by Parkinson's Law, software has expanded to use the

resources available. That will change with server-based applications.

In that world, the hardware and software will be supplied together.

For companies that offer server-based applications, it will make

a very big difference to the bottom line how many users they can

support per server.In some applications, the processor will be the limiting factor,

and execution speed will be the most important thing to optimize.

But often memory will be the limit; the number of simultaneous

users will be determined by the amount of memory you need for each

user's data. The language can help here too. Good support for

threads will enable all the users to share a single heap. It may

also help to have persistent objects and/or language level support

for lazy loading.9 TimeThe last ingredient a popular language needs is time. No one wants

to write programs in a language that might go away, as so many

programming languages do. So most hackers will tend to wait until

a language has been around for a couple years before even considering

using it.Inventors of wonderful new things are often surprised to discover

this, but you need time to get any message through to people. A

friend of mine rarely does anything the first time someone asks

him. He knows that people sometimes ask for things that they turn

out not to want. To avoid wasting his time, he waits till the third

or fourth time he's asked to do something; by then, whoever's asking

him may be fairly annoyed, but at least they probably really do

want whatever they're asking for.Most people have learned to do a similar sort of filtering on new

things they hear about. They don't even start paying attention

until they've heard about something ten times. They're perfectly

justified: the majority of hot new whatevers do turn out to be a

waste of time, and eventually go away. By delaying learning VRML,

I avoided having to learn it at all.So anyone who invents something new has to expect to keep repeating

their message for years before people will start to get it. We

wrote what was, as far as I know, the first web-server based

application, and it took us years to get it through to people that

it didn't have to be downloaded. It wasn't that they were stupid.

They just had us tuned out.The good news is, simple repetition solves the problem. All you

have to do is keep telling your story, and eventually people will

start to hear. It's not when people notice you're there that they

pay attention; it's when they notice you're still there.It's just as well that it usually takes a while to gain momentum.

Most technologies evolve a good deal even after they're first

launched — programming languages especially. Nothing could be better,

for a new techology, than a few years of being used only by a small

number of early adopters. Early adopters are sophisticated and

demanding, and quickly flush out whatever flaws remain in your

technology. When you only have a few users you can be in close

contact with all of them. And early adopters are forgiving when

you improve your system, even if this causes some breakage.There are two ways new technology gets introduced: the organic

growth method, and the big bang method. The organic growth method

is exemplified by the classic seat-of-the-pants underfunded garage

startup. A couple guys, working in obscurity, develop some new

technology. They launch it with no marketing and initially have

only a few (fanatically devoted) users. They continue to improve

the technology, and meanwhile their user base grows by word of

mouth. Before they know it, they're big.The other approach, the big bang method, is exemplified by the

VC-backed, heavily marketed startup. They rush to develop a product,

launch it with great publicity, and immediately (they hope) have

a large user base.Generally, the garage guys envy the big bang guys. The big bang

guys are smooth and confident and respected by the VCs. They can

afford the best of everything, and the PR campaign surrounding the

launch has the side effect of making them celebrities. The organic

growth guys, sitting in their garage, feel poor and unloved. And

yet I think they are often mistaken to feel sorry for themselves.

Organic growth seems to yield better technology and richer founders

than the big bang method. If you look at the dominant technologies

today, you'll find that most of them grew organically.This pattern doesn't only apply to companies. You see it in sponsored

research too. Multics and Common Lisp were big-bang projects, and

Unix and MacLisp were organic growth projects.10 Redesign"The best writing is rewriting," wrote E. B. White. Every good

writer knows this, and it's true for software too. The most important

part of design is redesign. Programming languages, especially,

don't get redesigned enough.To write good software you must simultaneously keep two opposing

ideas in your head. You need the young hacker's naive faith in

his abilities, and at the same time the veteran's skepticism. You

have to be able to think

how hard can it be? with one half of

your brain while thinking

it will never work with the other.The trick is to realize that there's no real contradiction here.

You want to be optimistic and skeptical about two different things.

You have to be optimistic about the possibility of solving the

problem, but skeptical about the value of whatever solution you've

got so far.People who do good work often think that whatever they're working

on is no good. Others see what they've done and are full of wonder,

but the creator is full of worry. This pattern is no coincidence:

it is the worry that made the work good.If you can keep hope and worry balanced, they will drive a project

forward the same way your two legs drive a bicycle forward. In the

first phase of the two-cycle innovation engine, you work furiously

on some problem, inspired by your confidence that you'll be able

to solve it. In the second phase, you look at what you've done in

the cold light of morning, and see all its flaws very clearly. But

as long as your critical spirit doesn't outweigh your hope, you'll

be able to look at your admittedly incomplete system, and think,

how hard can it be to get the rest of the way?, thereby continuing

the cycle.It's tricky to keep the two forces balanced. In young hackers,

optimism predominates. They produce something, are convinced it's

great, and never improve it. In old hackers, skepticism predominates,

and they won't even dare to take on ambitious projects.Anything you can do to keep the redesign cycle going is good. Prose

can be rewritten over and over until you're happy with it. But

software, as a rule, doesn't get redesigned enough. Prose has

readers, but software has users. If a writer rewrites an essay,

people who read the old version are unlikely to complain that their

thoughts have been broken by some newly introduced incompatibility.Users are a double-edged sword. They can help you improve your

language, but they can also deter you from improving it. So choose

your users carefully, and be slow to grow their number. Having

users is like optimization: the wise course is to delay it. Also,

as a general rule, you can at any given time get away with changing

more than you think. Introducing change is like pulling off a

bandage: the pain is a memory almost as soon as you feel it.Everyone knows that it's not a good idea to have a language designed

by a committee. Committees yield bad design. But I think the worst

danger of committees is that they interfere with redesign. It is

so much work to introduce changes that no one wants to bother.

Whatever a committee decides tends to stay that way, even if most

of the members don't like it.Even a committee of two gets in the way of redesign. This happens

particularly in the interfaces between pieces of software written

by two different people. To change the interface both have to agree

to change it at once. And so interfaces tend not to change at all,

which is a problem because they tend to be one of the most ad hoc

parts of any system.One solution here might be to design systems so that interfaces

are horizontal instead of vertical — so that modules are always

vertically stacked strata of abstraction. Then the interface will

tend to be owned by one of them. The lower of two levels will either

be a language in which the upper is written, in which case the

lower level will own the interface, or it will be a slave, in which

case the interface can be dictated by the upper level.11 LispWhat all this implies is that there is hope for a new Lisp. There

is hope for any language that gives hackers what they want, including

Lisp. I think we may have made a mistake in thinking that hackers

are turned off by Lisp's strangeness. This comforting illusion may

have prevented us from seeing the real problem with Lisp, or at

least Common Lisp, which is that it sucks for doing what hackers

want to do. A hacker's language needs powerful libraries and

something to hack. Common Lisp has neither. A hacker's language is

terse and hackable. Common Lisp is not.The good news is, it's not Lisp that sucks, but Common Lisp. If we

can develop a new Lisp that is a real hacker's language, I think

hackers will use it. They will use whatever language does the job.

All we have to do is make sure this new Lisp does some important

job better than other languages.History offers some encouragement. Over time, successive new

programming languages have taken more and more features from Lisp.

There is no longer much left to copy before the language you've

made is Lisp. The latest hot language, Python, is a watered-down

Lisp with infix syntax and no macros. A new Lisp would be a natural

step in this progression.I sometimes think that it would be a good marketing trick to call

it an improved version of Python. That sounds hipper than Lisp. To

many people, Lisp is a slow AI language with a lot of parentheses.

Fritz Kunze's official biography carefully avoids mentioning the

L-word. But my guess is that we shouldn't be afraid to call the

new Lisp Lisp. Lisp still has a lot of latent respect among the

very best hackers — the ones who took 6.001 and understood it, for

example. And those are the users you need to win.In "How to Become a Hacker," Eric Raymond describes Lisp as something

like Latin or Greek — a language you should learn as an intellectual

exercise, even though you won't actually use it:

Lisp is worth learning for the profound enlightenment experience

you will have when you finally get it; that experience will make

you a better programmer for the rest of your days, even if you

never actually use Lisp itself a lot.

If I didn't know Lisp, reading this would set me asking questions.

A language that would make me a better programmer, if it means

anything at all, means a language that would be better for programming.

And that is in fact the implication of what Eric is saying.As long as that idea is still floating around, I think hackers will

be receptive enough to a new Lisp, even if it is called Lisp. But

this Lisp must be a hacker's language, like the classic Lisps of

the 1970s. It must be terse, simple, and hackable. And it must have

powerful libraries for doing what hackers want to do now.In the matter of libraries I think there is room to beat languages

like Perl and Python at their own game. A lot of the new applications

that will need to be written in the coming years will be

server-based

applications. There's no reason a new Lisp shouldn't have string

libraries as good as Perl, and if this new Lisp also had powerful

libraries for server-based applications, it could be very popular.

Real hackers won't turn up their noses at a new tool that will let

them solve hard problems with a few library calls. Remember, hackers

are lazy.It could be an even bigger win to have core language support for

server-based applications. For example, explicit support for programs

with multiple users, or data ownership at the level of type tags.Server-based applications also give us the answer to the question

of what this new Lisp will be used to hack. It would not hurt to

make Lisp better as a scripting language for Unix. (It would be

hard to make it worse.) But I think there are areas where existing

languages would be easier to beat. I think it might be better to

follow the model of Tcl, and supply the Lisp together with a complete

system for supporting server-based applications. Lisp is a natural

fit for server-based applications. Lexical closures provide a way

to get the effect of subroutines when the ui is just a series of

web pages. S-expressions map nicely onto html, and macros are good

at generating it. There need to be better tools for writing

server-based applications, and there needs to be a new Lisp, and

the two would work very well together.12 The Dream LanguageBy way of summary, let's try describing the hacker's dream language.

The dream language is

beautiful, clean, and terse. It has an

interactive toplevel that starts up fast. You can write programs

to solve common problems with very little code. Nearly all the

code in any program you write is code that's specific to your

application. Everything else has been done for you.The syntax of the language is brief to a fault. You never have to

type an unnecessary character, or even to use the shift key much.Using big abstractions you can write the first version of a program

very quickly. Later, when you want to optimize, there's a really

good profiler that tells you where to focus your attention. You

can make inner loops blindingly fast, even writing inline byte code

if you need to.There are lots of good examples to learn from, and the language is

intuitive enough that you can learn how to use it from examples in

a couple minutes. You don't need to look in the manual much. The

manual is thin, and has few warnings and qualifications.The language has a small core, and powerful, highly orthogonal

libraries that are as carefully designed as the core language. The

libraries all work well together; everything in the language fits

together like the parts in a fine camera. Nothing is deprecated,

or retained for compatibility. The source code of all the libraries

is readily available. It's easy to talk to the operating system

and to applications written in other languages.The language is built in layers. The higher-level abstractions are

built in a very transparent way out of lower-level abstractions,

which you can get hold of if you want.Nothing is hidden from you that doesn't absolutely have to be. The

language offers abstractions only as a way of saving you work,

rather than as a way of telling you what to do. In fact, the language

encourages you to be an equal participant in its design. You can

change everything about it, including even its syntax, and anything

you write has, as much as possible, the same status as what comes

predefined.Notes[1] Macros very close to the modern idea were proposed by Timothy

Hart in 1964, two years after Lisp 1.5 was released. What was

missing, initially, were ways to avoid variable capture and multiple

evaluation; Hart's examples are subject to both.[2] In When the Air Hits Your Brain, neurosurgeon Frank Vertosick

recounts a conversation in which his chief resident, Gary, talks

about the difference between surgeons and internists ("fleas"):

Gary and I ordered a large pizza and found an open booth. The

chief lit a cigarette. "Look at those goddamn fleas, jabbering

about some disease they'll see once in their lifetimes. That's

the trouble with fleas, they only like the bizarre stuff. They

hate their bread and butter cases. That's the difference between

us and the fucking fleas. See, we love big juicy lumbar disc

herniations, but they hate hypertension...."

It's hard to think of a lumbar disc herniation as juicy (except

literally). And yet I think I know what they mean. I've often had

a juicy bug to track down. Someone who's not a programmer would

find it hard to imagine that there could be pleasure in a bug.

Surely it's better if everything just works. In one way, it is.

And yet there is undeniably a grim satisfaction in hunting down

certain sorts of bugs.Postscript VersionArcFive Questions about Language DesignHow to Become a HackerJapanese Translation

Java's Cover

April 2001This essay developed out of conversations I've had with

several other programmers about why Java smelled suspicious. It's not

a critique of Java! It is a case study of hacker's radar.Over time, hackers develop a nose for good (and bad) technology.

I thought it might be interesting to try and write down what

made Java seem suspect to me.Some people who've read this think it's an interesting attempt to write about

something that hasn't been written about before. Others say I

will get in trouble for appearing to be writing about

things I don't understand. So, just in

case it does any good, let me clarify that I'm not writing here

about Java (which I have never used) but about hacker's radar

(which I have thought about a lot).The aphorism "you can't tell a book by its cover" originated in

the times when books were sold in plain cardboard covers, to be

bound by each purchaser according to his own taste. In those days,

you couldn't tell a book by its cover. But publishing has advanced

since then: present-day publishers work hard to make the cover

something you can tell a book by.I spend a lot of time in bookshops and I feel as if I have by now

learned to understand everything publishers mean to tell me about

a book, and perhaps a bit more. The time I haven't spent in

bookshops I've spent mostly in front of computers, and I feel as

if I've learned, to some degree, to judge technology by its cover

as well. It may be just luck, but I've saved myself from a few

technologies that turned out to be real stinkers.So far, Java seems like a stinker to me. I've never written a Java

program, never more than glanced over reference books about it,

but I have a hunch that it won't be a very successful language.

I may turn out to be mistaken; making predictions about technology

is a dangerous business. But for what it's worth, as a sort of

time capsule, here's why I don't like the look of Java:

1. It has been so energetically hyped. Real standards don't have

to be promoted. No one had to promote C, or Unix, or HTML. A real

standard tends to be already established by the time most people

hear about it. On the hacker radar screen, Perl is as big as Java,

or bigger, just on the strength of its own merits.2. It's aimed low. In the original Java white paper, Gosling

explicitly says Java was designed not to be too difficult for

programmers used to C. It was designed to be another C++: C plus

a few ideas taken from more advanced languages. Like the creators

of sitcoms or junk food or package tours, Java's designers were

consciously designing a product for people not as smart as them.

Historically, languages designed for other people to use have been

bad: Cobol, PL/I, Pascal, Ada, C++. The good languages have been

those that were designed for their own creators: C, Perl, Smalltalk,

Lisp.3. It has ulterior motives. Someone once said that the world would

be a better place if people only wrote books because they had

something to say, rather than because they wanted to write a book.

Likewise, the reason we hear about Java all the time is not because

it has something to say about programming languages. We hear about

Java as part of a plan by Sun to undermine Microsoft.4. No one loves it. C, Perl, Python, Smalltalk, and Lisp programmers

love their languages. I've never heard anyone say that they loved

Java.5. People are forced to use it. A lot of the people I know using

Java are using it because they feel they have to. Either it's

something they felt they had to do to get funded, or something they

thought customers would want, or something they were told to do by

management. These are smart people; if the technology was good,

they'd have used it voluntarily.6. It has too many cooks. The best programming languages have been

developed by small groups. Java seems to be run by a committee.

If it turns out to be a good language, it will be the first time

in history that a committee has designed a good language.7. It's bureaucratic. From what little I know about Java, there

seem to be a lot of protocols for doing things. Really good

languages aren't like that. They let you do what you want and get

out of the way.8. It's pseudo-hip. Sun now pretends that Java is a grassroots,

open-source language effort like Perl or Python. This one just

happens to be controlled by a giant company. So the language is

likely to have the same drab clunkiness as anything else that comes

out of a big company.9. It's designed for large organizations. Large organizations have

different aims from hackers. They want languages that are (believed

to be) suitable for use by large teams of mediocre programmers--

languages with features that, like the speed limiters in U-Haul

trucks, prevent fools from doing too much damage. Hackers don't

like a language that talks down to them. Hackers just want power.

Historically, languages designed for large organizations (PL/I,

Ada) have lost, while hacker languages (C, Perl) have won. The

reason: today's teenage hacker is tomorrow's CTO.10. The wrong people like it. The programmers I admire most are

not, on the whole, captivated by Java. Who does like Java? Suits,

who don't know one language from another, but know that they keep

hearing about Java in the press; programmers at big companies, who

are amazed to find that there is something even better than C++;

and plug-and-chug undergrads, who are ready to like anything that

might get them a job (will this be on the test?). These people's

opinions change with every wind.11. Its daddy is in a pinch. Sun's business model is being undermined

on two fronts. Cheap Intel processors, of the same type used in

desktop machines, are now more than fast enough for servers. And

FreeBSD seems to be at least as good an OS for servers as Solaris.

Sun's advertising implies that you need Sun servers for industrial

strength applications. If this were true, Yahoo would be first in

line to buy Suns; but when I worked there, the servers were all

Intel boxes running FreeBSD. This bodes ill for Sun's future. If

Sun runs into trouble, they could drag Java down with them.12. The DoD likes it. The Defense Department is encouraging

developers to use Java. This seems to me the most damning sign of

all. The Defense Department does a fine (though expensive) job of

defending the country, but they love plans and procedures and

protocols. Their culture is the opposite of hacker culture; on

questions of software they will tend to bet wrong. The last time

the DoD really liked a programming language, it was Ada.

Bear in mind, this is not a critique of Java, but a critique of

its cover. I don't know Java well enough to like it or dislike

it. This is just an explanation of why I don't find that I'm eager

to learn it.It may seem cavalier to dismiss a language before you've even tried

writing programs in it. But this is something all programmers have

to do. There are too many technologies out there to learn them

all. You have to learn to judge by outward signs which will be

worth your time. I have likewise cavalierly dismissed Cobol, Ada,

Visual Basic, the IBM AS400, VRML, ISO 9000, the SET protocol, VMS,

Novell Netware, and CORBA, among others. They just smelled wrong.It could be that in Java's case I'm mistaken. It could be that a

language promoted by one big company to undermine another, designed

by a committee for a "mainstream" audience, hyped to the skies,

and beloved of the DoD, happens nonetheless to be a clean, beautiful,

powerful language that I would love programming in. It could be,

but it seems very unlikely.Trevor Re: Java's CoverBerners-Lee Re: JavaBeing PopularSun Internal Memo2005: BusinessWeek AgreesJapanese Translation

Beating the Averages

Want to start a startup? Get funded by

Y Combinator.

April 2001, rev. April 2003(This article is derived from a talk given at the 2001 Franz

Developer Symposium.)

In the summer of 1995, my friend Robert Morris and I

started a startup called

Viaweb.

Our plan was to write

software that would let end users build online stores.

What was novel about this software, at the time, was

that it ran on our server, using ordinary Web pages

as the interface.A lot of people could have been having this idea at the

same time, of course, but as far as I know, Viaweb was

the first Web-based application. It seemed such

a novel idea to us that we named the company after it:

Viaweb, because our software worked via the Web,

instead of running on your desktop computer.Another unusual thing about this software was that it

was written primarily in a programming language called

Lisp. It was one of the first big end-user

applications to be written in Lisp, which up till then

had been used mostly in universities and research labs. [1]The Secret WeaponEric Raymond has written an essay called "How to Become a Hacker,"

and in it, among other things, he tells would-be hackers what

languages they should learn. He suggests starting with Python and

Java, because they are easy to learn. The serious hacker will also

want to learn C, in order to hack Unix, and Perl for system

administration and cgi scripts. Finally, the truly serious hacker

should consider learning Lisp:

Lisp is worth learning for the profound enlightenment experience

you will have when you finally get it; that experience will make

you a better programmer for the rest of your days, even if you

never actually use Lisp itself a lot.

This is the same argument you tend to hear for learning Latin. It

won't get you a job, except perhaps as a classics professor, but

it will improve your mind, and make you a better writer in languages

you do want to use, like English.But wait a minute. This metaphor doesn't stretch that far. The

reason Latin won't get you a job is that no one speaks it. If you

write in Latin, no one can understand you. But Lisp is a computer

language, and computers speak whatever language you, the programmer,

tell them to.So if Lisp makes you a better programmer, like he says, why wouldn't

you want to use it? If a painter were offered a brush that would

make him a better painter, it seems to me that he would want to

use it in all his paintings, wouldn't he? I'm not trying to make

fun of Eric Raymond here. On the whole, his advice is good. What

he says about Lisp is pretty much the conventional wisdom. But

there is a contradiction in the conventional wisdom: Lisp will

make you a better programmer, and yet you won't use it.Why not? Programming languages are just tools, after all. If Lisp

really does yield better programs, you should use it. And if it

doesn't, then who needs it?This is not just a theoretical question. Software is a very

competitive business, prone to natural monopolies. A company that

gets software written faster and better will, all other things

being equal, put its competitors out of business. And when you're

starting a startup, you feel this very keenly. Startups tend to

be an all or nothing proposition. You either get rich, or you get

nothing. In a startup, if you bet on the wrong technology, your

competitors will crush you.Robert and I both knew Lisp well, and we couldn't see any reason

not to trust our instincts and go with Lisp. We knew that everyone

else was writing their software in C++ or Perl. But we also knew

that that didn't mean anything. If you chose technology that way,

you'd be running Windows. When you choose technology, you have to

ignore what other people are doing, and consider only what will

work the best.This is especially true in a startup. In a big company, you can

do what all the other big companies are doing. But a startup can't

do what all the other startups do. I don't think a lot of people

realize this, even in startups.The average big company grows at about ten percent a year. So if

you're running a big company and you do everything the way the

average big company does it, you can expect to do as well as the

average big company-- that is, to grow about ten percent a year.The same thing will happen if you're running a startup, of course.

If you do everything the way the average startup does it, you should

expect average performance. The problem here is, average performance

means that you'll go out of business. The survival rate for startups

is way less than fifty percent. So if you're running a startup,

you had better be doing something odd. If not, you're in trouble.Back in 1995, we knew something that I don't think our competitors

understood, and few understand even now: when you're writing

software that only has to run on your own servers, you can use

any language you want. When you're writing desktop software,

there's a strong bias toward writing applications in the same

language as the operating system. Ten years ago, writing applications

meant writing applications in C. But with Web-based software,

especially when you have the source code of both the language and

the operating system, you can use whatever language you want.This new freedom is a double-edged sword, however. Now that you

can use any language, you have to think about which one to use.

Companies that try to pretend nothing has changed risk finding that

their competitors do not.If you can use any language, which do you use? We chose Lisp.

For one thing, it was obvious that rapid development would be

important in this market. We were all starting from scratch, so

a company that could get new features done before its competitors

would have a big advantage. We knew Lisp was a really good language

for writing software quickly, and server-based applications magnify

the effect of rapid development, because you can release software

the minute it's done.If other companies didn't want to use Lisp, so much the better.

It might give us a technological edge, and we needed all the help

we could get. When we started Viaweb, we had no experience in

business. We didn't know anything about marketing, or hiring

people, or raising money, or getting customers. Neither of us had

ever even had what you would call a real job. The only thing we

were good at was writing software. We hoped that would save us.

Any advantage we could get in the software department, we would

take.So you could say that using Lisp was an experiment. Our hypothesis

was that if we wrote our software in Lisp, we'd be able to get

features done faster than our competitors, and also to do things

in our software that they couldn't do. And because Lisp was so

high-level, we wouldn't need a big development team, so our costs

would be lower. If this were so, we could offer a better product

for less money, and still make a profit. We would end up getting

all the users, and our competitors would get none, and eventually

go out of business. That was what we hoped would happen, anyway.What were the results of this experiment? Somewhat surprisingly,

it worked. We eventually had many competitors, on the order of

twenty to thirty of them, but none of their software could compete

with ours. We had a wysiwyg online store builder that ran on the

server and yet felt like a desktop application. Our competitors

had cgi scripts. And we were always far ahead of them in features.

Sometimes, in desperation, competitors would try to introduce

features that we didn't have. But with Lisp our development cycle

was so fast that we could sometimes duplicate a new feature within

a day or two of a competitor announcing it in a press release. By

the time journalists covering the press release got round to calling

us, we would have the new feature too.It must have seemed to our competitors that we had some kind of

secret weapon-- that we were decoding their Enigma traffic or

something. In fact we did have a secret weapon, but it was simpler

than they realized. No one was leaking news of their features to

us. We were just able to develop software faster than anyone

thought possible.When I was about nine I happened to get hold of a copy of The Day

of the Jackal, by Frederick Forsyth. The main character is an

assassin who is hired to kill the president of France. The assassin

has to get past the police to get up to an apartment that overlooks

the president's route. He walks right by them, dressed up as an

old man on crutches, and they never suspect him.Our secret weapon was similar. We wrote our software in a weird

AI language, with a bizarre syntax full of parentheses. For years

it had annoyed me to hear Lisp described that way. But now it

worked to our advantage. In business, there is nothing more valuable

than a technical advantage your competitors don't understand. In

business, as in war, surprise is worth as much as force.And so, I'm a little embarrassed to say, I never said anything

publicly about Lisp while we were working on Viaweb. We never

mentioned it to the press, and if you searched for Lisp on our Web

site, all you'd find were the titles of two books in my bio. This

was no accident. A startup should give its competitors as little

information as possible. If they didn't know what language our

software was written in, or didn't care, I wanted to keep it that

way.[2]The people who understood our technology best were the customers.

They didn't care what language Viaweb was written in either, but

they noticed that it worked really well. It let them build great

looking online stores literally in minutes. And so, by word of

mouth mostly, we got more and more users. By the end of 1996 we

had about 70 stores online. At the end of 1997 we had 500. Six

months later, when Yahoo bought us, we had 1070 users. Today, as

Yahoo Store, this software continues to dominate its market. It's

one of the more profitable pieces of Yahoo, and the stores built

with it are the foundation of Yahoo Shopping. I left Yahoo in

1999, so I don't know exactly how many users they have now, but

the last I heard there were about 20,000.

The Blub ParadoxWhat's so great about Lisp? And if Lisp is so great, why doesn't

everyone use it? These sound like rhetorical questions, but actually

they have straightforward answers. Lisp is so great not because

of some magic quality visible only to devotees, but because it is

simply the most powerful language available. And the reason everyone

doesn't use it is that programming languages are not merely

technologies, but habits of mind as well, and nothing changes

slower. Of course, both these answers need explaining.I'll begin with a shockingly controversial statement: programming

languages vary in power.Few would dispute, at least, that high level languages are more

powerful than machine language. Most programmers today would agree

that you do not, ordinarily, want to program in machine language.

Instead, you should program in a high-level language, and have a

compiler translate it into machine language for you. This idea is

even built into the hardware now: since the 1980s, instruction sets

have been designed for compilers rather than human programmers.Everyone knows it's a mistake to write your whole program by hand

in machine language. What's less often understood is that there

is a more general principle here: that if you have a choice of

several languages, it is, all other things being equal, a mistake

to program in anything but the most powerful one. [3]There are many exceptions to this rule. If you're writing a program

that has to work very closely with a program written in a certain

language, it might be a good idea to write the new program in the

same language. If you're writing a program that only has to do

something very simple, like number crunching or bit manipulation,

you may as well use a less abstract language, especially since it

may be slightly faster. And if you're writing a short, throwaway

program, you may be better off just using whatever language has

the best library functions for the task. But in general, for

application software, you want to be using the most powerful

(reasonably efficient) language you can get, and using anything

else is a mistake, of exactly the same kind, though possibly in a

lesser degree, as programming in machine language.You can see that machine language is very low level. But, at least

as a kind of social convention, high-level languages are often all

treated as equivalent. They're not. Technically the term "high-level

language" doesn't mean anything very definite. There's no dividing

line with machine languages on one side and all the high-level

languages on the other. Languages fall along a continuum [4] of

abstractness, from the most powerful all the way down to machine

languages, which themselves vary in power.Consider Cobol. Cobol is a high-level language, in the sense that

it gets compiled into machine language. Would anyone seriously

argue that Cobol is equivalent in power to, say, Python? It's

probably closer to machine language than Python.Or how about Perl 4? Between Perl 4 and Perl 5, lexical closures

got added to the language. Most Perl hackers would agree that Perl

5 is more powerful than Perl 4. But once you've admitted that,

you've admitted that one high level language can be more powerful

than another. And it follows inexorably that, except in special

cases, you ought to use the most powerful you can get.This idea is rarely followed to its conclusion, though. After a

certain age, programmers rarely switch languages voluntarily.

Whatever language people happen to be used to, they tend to consider

just good enough.Programmers get very attached to their favorite languages, and I

don't want to hurt anyone's feelings, so to explain this point I'm

going to use a hypothetical language called Blub. Blub falls right

in the middle of the abstractness continuum. It is not the most

powerful language, but it is more powerful than Cobol or machine

language.And in fact, our hypothetical Blub programmer wouldn't use either

of them. Of course he wouldn't program in machine language. That's

what compilers are for. And as for Cobol, he doesn't know how

anyone can get anything done with it. It doesn't even have x (Blub

feature of your choice).As long as our hypothetical Blub programmer is looking down the

power continuum, he knows he's looking down. Languages less powerful

than Blub are obviously less powerful, because they're missing some

feature he's used to. But when our hypothetical Blub programmer

looks in the other direction, up the power continuum, he doesn't

realize he's looking up. What he sees are merely weird languages.

He probably considers them about equivalent in power to Blub, but

with all this other hairy stuff thrown in as well. Blub is good

enough for him, because he thinks in Blub.When we switch to the point of view of a programmer using any of

the languages higher up the power continuum, however, we find that

he in turn looks down upon Blub. How can you get anything done in

Blub? It doesn't even have y.By induction, the only programmers in a position to see all the

differences in power between the various languages are those who

understand the most powerful one. (This is probably what Eric

Raymond meant about Lisp making you a better programmer.) You can't

trust the opinions of the others, because of the Blub paradox:

they're satisfied with whatever language they happen to use, because

it dictates the way they think about programs.I know this from my own experience, as a high school kid writing

programs in Basic. That language didn't even support recursion.

It's hard to imagine writing programs without using recursion, but

I didn't miss it at the time. I thought in Basic. And I was a

whiz at it. Master of all I surveyed.The five languages that Eric Raymond recommends to hackers fall at

various points on the power continuum. Where they fall relative

to one another is a sensitive topic. What I will say is that I

think Lisp is at the top. And to support this claim I'll tell you

about one of the things I find missing when I look at the other

four languages. How can you get anything done in them, I think,

without macros? [5]Many languages have something called a macro. But Lisp macros are

unique. And believe it or not, what they do is related to the

parentheses. The designers of Lisp didn't put all those parentheses

in the language just to be different. To the Blub programmer, Lisp

code looks weird. But those parentheses are there for a reason.

They are the outward evidence of a fundamental difference between

Lisp and other languages.Lisp code is made out of Lisp data objects. And not in the trivial

sense that the source files contain characters, and strings are

one of the data types supported by the language. Lisp code, after

it's read by the parser, is made of data structures that you can

traverse.If you understand how compilers work, what's really going on is

not so much that Lisp has a strange syntax as that Lisp has no

syntax. You write programs in the parse trees that get generated

within the compiler when other languages are parsed. But these

parse trees are fully accessible to your programs. You can write

programs that manipulate them. In Lisp, these programs are called

macros. They are programs that write programs.Programs that write programs? When would you ever want to do that?

Not very often, if you think in Cobol. All the time, if you think

in Lisp. It would be convenient here if I could give an example

of a powerful macro, and say there! how about that? But if I did,

it would just look like gibberish to someone who didn't know Lisp;

there isn't room here to explain everything you'd need to know to

understand what it meant. In

Ansi Common Lisp I tried to move

things along as fast as I could, and even so I didn't get to macros

until page 160.But I think I can give a kind of argument that might be convincing.

The source code of the Viaweb editor was probably about 20-25%

macros. Macros are harder to write than ordinary Lisp functions,

and it's considered to be bad style to use them when they're not

necessary. So every macro in that code is there because it has to

be. What that means is that at least 20-25% of the code in this

program is doing things that you can't easily do in any other

language. However skeptical the Blub programmer might be about my

claims for the mysterious powers of Lisp, this ought to make him

curious. We weren't writing this code for our own amusement. We

were a tiny startup, programming as hard as we could in order to

put technical barriers between us and our competitors.A suspicious person might begin to wonder if there was some

correlation here. A big chunk of our code was doing things that

are very hard to do in other languages. The resulting software

did things our competitors' software couldn't do. Maybe there was

some kind of connection. I encourage you to follow that thread.

There may be more to that old man hobbling along on his crutches

than meets the eye.Aikido for StartupsBut I don't expect to convince anyone

(over 25)

to go out and learn

Lisp. The purpose of this article is not to change anyone's mind,

but to reassure people already interested in using Lisp-- people

who know that Lisp is a powerful language, but worry because it

isn't widely used. In a competitive situation, that's an advantage.

Lisp's power is multiplied by the fact that your competitors don't

get it.If you think of using Lisp in a startup, you shouldn't worry that

it isn't widely understood. You should hope that it stays that

way. And it's likely to. It's the nature of programming languages

to make most people satisfied with whatever they currently use.

Computer hardware changes so much faster than personal habits that

programming practice is usually ten to twenty years behind the

processor. At places like MIT they were writing programs in

high-level languages in the early 1960s, but many companies continued

to write code in machine language well into the 1980s. I bet a

lot of people continued to write machine language until the processor,

like a bartender eager to close up and go home, finally kicked them

out by switching to a risc instruction set.Ordinarily technology changes fast. But programming languages are

different: programming languages are not just technology, but what

programmers think in. They're half technology and half religion.[6]

And so the median language, meaning whatever language the median

programmer uses, moves as slow as an iceberg. Garbage collection,

introduced by Lisp in about 1960, is now widely considered to be

a good thing. Runtime typing, ditto, is growing in popularity.

Lexical closures, introduced by Lisp in the early 1970s, are now,

just barely, on the radar screen. Macros, introduced by Lisp in the

mid 1960s, are still terra incognita.Obviously, the median language has enormous momentum. I'm not

proposing that you can fight this powerful force. What I'm proposing

is exactly the opposite: that, like a practitioner of Aikido, you

can use it against your opponents.If you work for a big company, this may not be easy. You will have

a hard time convincing the pointy-haired boss to let you build

things in Lisp, when he has just read in the paper that some other

language is poised, like Ada was twenty years ago, to take over

the world. But if you work for a startup that doesn't have

pointy-haired bosses yet, you can, like we did, turn the Blub

paradox to your advantage: you can use technology that your

competitors, glued immovably to the median language, will never be

able to match.If you ever do find yourself working for a startup, here's a handy

tip for evaluating competitors. Read their job listings. Everything

else on their site may be stock photos or the prose equivalent,

but the job listings have to be specific about what they want, or

they'll get the wrong candidates.During the years we worked on Viaweb I read a lot of job descriptions.

A new competitor seemed to emerge out of the woodwork every month

or so. The first thing I would do, after checking to see if they

had a live online demo, was look at their job listings. After a

couple years of this I could tell which companies to worry about

and which not to. The more of an IT flavor the job descriptions

had, the less dangerous the company was. The safest kind were the

ones that wanted Oracle experience. You never had to worry about

those. You were also safe if they said they wanted C++ or Java

developers. If they wanted Perl or Python programmers, that would

be a bit frightening-- that's starting to sound like a company

where the technical side, at least, is run by real hackers. If I

had ever seen a job posting looking for Lisp hackers, I would have

been really worried.

Notes[1] Viaweb at first had two parts: the editor, written in Lisp,

which people used to build their sites, and the ordering system,

written in C, which handled orders. The first version was mostly

Lisp, because the ordering system was small. Later we added two

more modules, an image generator written in C, and a back-office

manager written mostly in Perl.In January 2003, Yahoo released a new version of the editor

written in C++ and Perl. It's hard to say whether the program is no

longer written in Lisp, though, because to translate this program

into C++ they literally had to write a Lisp interpreter: the source

files of all the page-generating templates are still, as far as I

know, Lisp code. (See Greenspun's Tenth Rule.)[2] Robert Morris says that I didn't need to be secretive, because

even if our competitors had known we were using Lisp, they wouldn't

have understood why: "If they were that smart they'd already be

programming in Lisp."[3] All languages are equally powerful in the sense of being Turing

equivalent, but that's not the sense of the word programmers care

about. (No one wants to program a Turing machine.) The kind of

power programmers care about may not be formally definable, but

one way to explain it would be to say that it refers to features

you could only get in the less powerful language by writing an

interpreter for the more powerful language in it. If language A

has an operator for removing spaces from strings and language B

doesn't, that probably doesn't make A more powerful, because you

can probably write a subroutine to do it in B. But if A supports,

say, recursion, and B doesn't, that's not likely to be something

you can fix by writing library functions.[4] Note to nerds: or possibly a lattice, narrowing toward the top;

it's not the shape that matters here but the idea that there is at

least a partial order.[5] It is a bit misleading to treat macros as a separate feature.

In practice their usefulness is greatly enhanced by other Lisp

features like lexical closures and rest parameters.[6] As a result, comparisons of programming languages either take

the form of religious wars or undergraduate textbooks so determinedly

neutral that they're really works of anthropology. People who

value their peace, or want tenure, avoid the topic. But the question

is only half a religious one; there is something there worth

studying, especially if you want to design new languages.More Technical DetailsJapanese TranslationTurkish TranslationUzbek TranslationOrbitz Uses Lisp TooHow To Become A HackerA Scheme StoryItalian Translation

You'll find this essay and 14 others in

Hackers & Painters.

Lisp for Web-Based Applications

After a link to

Beating the Averages was posted on slashdot,

some readers wanted to hear in more detail

about the specific technical advantages we got from using

Lisp in Viaweb. For those who are interested,

here are some excerpts from a talk I gave in April 2001 at

BBN Labs in Cambridge, MA.BBN Talk Excerpts (ASCII)

Programming Bottom-Up

1993

(This essay is from the introduction to On Lisp. The red text

explains the origins of Arc's name.)It's a long-standing principle of programming style that the functional

elements of a program should not be too large. If some component of a

program grows beyond the stage where it's readily comprehensible,

it becomes a mass of complexity which conceals errors as easily

as a big city conceals fugitives. Such software will be

hard to read, hard to test, and hard to debug.In accordance with this principle, a large program must be divided

into pieces, and the larger the program, the more it must be divided.

How do you divide a program? The traditional approach is

called top-down design: you say "the purpose of the

program is to do these seven things, so I divide it into seven major

subroutines. The first subroutine has to do these four things, so

it in turn will have four of its own subroutines," and so on.

This process continues until the whole program has the right level

of granularity-- each part large enough to do something substantial,

but small enough to be understood as a single unit.Experienced Lisp programmers divide up their programs differently.

As well as top-down design, they follow a principle which

could be called bottom-up design-- changing the language

to suit the problem.

In Lisp, you don't just write your program down toward the language,

you also build the language up toward your program. As you're

writing a program you may think "I wish Lisp had such-and-such an

operator." So you go and write it. Afterward

you realize that using the new operator would simplify the design

of another part of the program, and so on.

Language and program evolve together.

Like the border between two warring states,

the boundary between language and program is drawn and redrawn,

until eventually it comes to rest along the mountains and rivers,

the natural frontiers of your problem.

In the end your program will look as if the language had been

designed for it.

And when language and

program fit one another well, you end up with code which is

clear, small, and efficient.

It's worth emphasizing that bottom-up design doesn't mean

just writing the same program in a different order. When you

work bottom-up, you usually end up with a different program.

Instead of a single, monolithic program,

you will get a larger language with more abstract operators,

and a smaller program written in it. Instead of a lintel,

you'll get an arch.

In typical code, once you abstract out the parts which are

merely bookkeeping, what's left is much shorter;

the higher you build up the language, the less distance you

will have to travel from the top down to it.

This brings several advantages:

By making the language do more of the work, bottom-up design

yields programs which are smaller and more agile. A shorter

program doesn't have to be divided into so many components, and

fewer components means programs which are easier to read or

modify. Fewer components also means fewer connections between

components, and thus less chance for errors there. As

industrial designers strive to reduce the number of moving parts

in a machine, experienced Lisp programmers use bottom-up design

to reduce the size and complexity of their programs. Bottom-up design promotes code re-use.

When you write two

or more programs, many of the utilities you wrote for the first

program will also be useful in the succeeding ones. Once you've

acquired a large substrate of utilities, writing a new program can

take only a fraction of the effort it would require if you had to

start with raw Lisp. Bottom-up design makes programs easier to read.

An instance of this type

of abstraction asks the reader to understand a general-purpose operator;

an instance of functional abstraction asks the reader to understand

a special-purpose subroutine. [1] Because it causes you always to be on the lookout for patterns

in your code, working bottom-up helps to clarify your ideas about

the design of your program. If two distant components of a program

are similar in form, you'll be led to notice the similarity and

perhaps to redesign the program in a simpler way.

Bottom-up design is possible to a certain degree in languages

other than Lisp. Whenever you see library functions,

bottom-up design is happening. However, Lisp gives you much broader

powers in this department, and augmenting the language plays a

proportionately larger role in Lisp style-- so much so that

Lisp is not just a different language, but a whole different way

of programming.It's true that this style of development is better suited to

programs which can be written by small groups. However, at the

same time, it extends the limits of what can be done by a small

group. In The Mythical Man-Month,

Frederick Brooks

proposed that the productivity of a group of programmers

does not grow linearly with its size. As the size of the

group increases, the productivity of individual programmers

goes down. The experience of Lisp programming

suggests a more cheerful way

to phrase this law: as the size of the group decreases, the

productivity of individual programmers goes up.

A small group wins, relatively speaking, simply because it's

smaller. When a small group also takes advantage of the

techniques that Lisp makes possible, it can

win outright.New: Download On Lisp for Free.

[1] "But no one can read

the program without understanding all your new utilities."

To see why such statements are usually mistaken,

see Section 4.8.

This Year We Can End the Death Penalty in California

November 2016If you're a California voter, there is an important proposition

on your ballot this year: Proposition 62, which bans the death

penalty.When I was younger I used to think the debate about the death

penalty was about when it's ok to take a human life. Is it ok

to kill a killer?But that is not the issue here.The real world does not work like the version I was shown on TV growing up. The police

often arrest the wrong person.

Defendants' lawyers are often incompetent. And prosecutors

are often motivated more by publicity than justice.In the real world,

about 4% of people sentenced to death

are innocent.

So this is not about whether it's ok to kill killers. This

is about whether it's ok to kill innocent people.A child could answer that one for you.This year, in California, you have a chance to end this, by

voting yes on Proposition 62. But beware, because there is another

proposition, Proposition 66, whose goal is to make it

easier to execute people. So yes on 62, no on 66.It's time.

RSS

Aaron Swartz created a scraped

feed

of the essays page.