# Modeling a Wealth Tax

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%.   
[1]  
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.Note[1]  
In practice, eventually some of this 8% would come in the form of   
dividends, which are taxed as income at issue, so this model actually  
represents the most optimistic case for the founder.  
  
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.Note[1]  
In practice, eventually some of this 8% would come in the form of   
dividends, which are taxed as income at issue, so this model actually  
represents the most optimistic case for the founder.

# Before the Startup

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

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.

# Investor Herd Dynamics

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.

# How to Convince Investors

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

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.

# How to Get Startup Ideas

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.

# The Hardware Renaissance

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.

# Startup = Growth

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 intuitions 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.  
  
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 intuitions 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.

# Black Swan Farming

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.

# How Y Combinator Started

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.  
  
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.  
  
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.  
  
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.  
  
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.  
  
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.  
  
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.  
  
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.  
  
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.  
  
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.  
  
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.  
  
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.  
  
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.  
  
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.  
  
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.  
  
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.

# Frighteningly Ambitious Startup Ideas

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

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

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.

# Why Startup Hubs Work

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.

# Founder Control

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.

# What We Look for in Founders

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.  
  
(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

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

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

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

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

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 Top Idea in Your Mind

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.

# Organic Startup Ideas

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

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.

# What Startups Are Really Like

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:

# The Anatomy of Determination

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 Shakespeare's 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.

# Ramen Profitable

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.

# Maker's Schedule, Manager's Schedule

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:

# Relentlessly Resourceful

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.

# Startups in 13 Sentences

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.  
  
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.  
  
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.

# Why to Start a Startup in a Bad Economy

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.

# A Fundraising Survival Guide

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.

# Why There Aren't More Googles

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.

# You Weren't Meant to Have a Boss

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.

# The Future of Web Startups

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.

# How Not to Die

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.

# Why to Not Not Start a Startup

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.

# The 18 Mistakes That Kill Startups

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.

# A Student's Guide to Startups

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.

# How to Present to Investors

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.

# The Power of the Marginal

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. 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.

# How to Do What You Love

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.

# Web 2.0

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.

# How to Fund a Startup

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.

# Ideas for Startups

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.

# Hiring is Obsolete

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.

# Why Smart People Have Bad Ideas

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.

# Undergraduation

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.

# How to Start a Startup

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 million,  
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.

# Great Hackers

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.

# How to Make Wealth

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.

# A Plan for Spam

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:

# Revenge of the Nerds

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

# 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.  
  
  
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.

# 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.  
  
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.

# What Made Lisp Different

(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.

# The Roots of Lisp

(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.

# Five Questions about Language Design

(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?  
  
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?  
  
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?

# Beating the Averages

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.

# 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.

# Programming Bottom-Up

(This essay is from the introduction to On Lisp.)  
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.  
  
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.