

PAUL GRAHAM

HOW TO GET STARTUP IDEAS

Want to start a startup? Get funded by [YCombinator](#).

November 2012

The 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.

Problems

Why 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]

Well

When 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.

Self

How 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 foreseeable 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.

Noticing

Once 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.

School

That'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.

Competition

Because 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). [\[12\]](#)

Filters

There 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. [\[13\]](#)

Recipes

While 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?

Organic

But 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.

PAUL GRAHAM

STARTUP INVESTING TRENDS

June 2013

(This talk was written for an audience of investors.)

Y Combinator has now funded 564 startups including the current batch, which has 53. The total valuation of the 287 that have valuations (either by raising an equity round, getting acquired, or dying) is about \$11.7 billion, and the 511 prior to the current batch have collectively raised about \$1.7 billion. [\[1\]](#)

As usual those numbers are dominated by a few big winners. The top 10 startups account for 8.6 of that 11.7 billion. But there is a peloton of younger startups behind them. There are about 40 more that have a shot at being really big.

Things got a little out of hand last summer when we had 84 companies in the batch, so we tightened up our filter to decrease the batch size. [2] Several journalists have tried to interpret that as evidence for some macro story they were telling, but the reason had nothing to do with any external trend. The reason was that we discovered we were using an n^2 algorithm, and we needed to buy time to fix it. Fortunately we've come up with several techniques for sharding YC, and the problem now seems to be fixed. With a new more scaleable model and only 53 companies, the current batch feels like a walk in the park. I'd guess we can grow another 2 or 3x before hitting the next bottleneck. [3]

One consequence of funding such a large number of startups is that we see trends early. And since fundraising is one of the main things we help startups with, we're in a good position to notice trends in investing.

I'm going to take a shot at describing where these trends are leading. Let's start with the most basic question: will the future be better or worse than the past? Will investors, in the aggregate, make more money or less?

I think more. There are multiple forces at work, some of which will decrease returns, and some of which will increase them. I can't predict for sure which forces will prevail, but I'll describe them and you can decide for yourself.

There are two big forces driving change in startup funding: it's becoming cheaper to start a startup, and startups are becoming a more normal thing to do.

When I graduated from college in 1986, there were essentially two options: get a job or go to grad school. Now there's a third: start your own company. That's a big change. In principle it was possible to start your own company in 1986 too, but it didn't seem like a real possibility. It seemed possible to start a consulting company, or a niche product company, but it didn't seem possible to start a company that would become big. [4]

That kind of change, from 2 paths to 3, is the sort of big social shift that only happens once every few generations. I think we're still at the beginning of this one. It's hard to predict how big a deal it will be. As big a deal as the Industrial Revolution? Maybe. Probably not. But it will be a big enough deal that it takes almost everyone by surprise, because those big social shifts always do.

One thing we can say for sure is that there will be a lot more

startups. The monolithic, hierarchical companies of the mid 20th century are being [replaced](#) by networks of smaller companies. This process is not just something happening now in Silicon Valley. It started decades ago, and it's happening as far afield as the car industry. It has a long way to run. [\[5\]](#)

The other big driver of change is that startups are becoming cheaper to start. And in fact the two forces are related: the decreasing cost of starting a startup is one of the reasons startups are becoming a more normal thing to do.

The fact that startups need less money means founders will increasingly have the upper hand over investors. You still need just as much of their energy and imagination, but they don't need as much of your money. Because founders have the upper hand, they'll retain an increasingly large share of the stock in, and [control of](#), their companies. Which means investors will get less stock and less control.

Does that mean investors will make less money? Not necessarily, because there will be more good startups. The total amount of desirable startup stock available to investors will probably increase, because the number of desirable startups will probably grow faster than the percentage they sell to investors shrinks.

There's a rule of thumb in the VC business that there are about 15 companies a year that will be really successful. Although a lot of investors unconsciously treat this number as if it were some sort of cosmological constant, I'm certain it isn't. There are probably limits on the rate at which technology can develop, but that's not the limiting factor now. If it were, each successful startup would be founded the month it became possible, and that is not the case. Right now the limiting factor on the number of big hits is the number of sufficiently good founders starting companies, and that number can and will increase. There are still a lot of people who'd make great founders who never end up starting a company. You can see that from how randomly some of the most successful startups got started. So many of the biggest startups almost didn't happen that there must be a lot of equally good startups that actually didn't happen.

There might be 10x or even 50x more good founders out there. As more of them go ahead and start startups, those 15 big hits a year could easily become 50 or even 100. [\[6\]](#)

What about returns, though? Are we heading for a world in which returns will be pinched by increasingly high valuations? I think the top firms will actually make more money than they

have in the past. High returns don't come from investing at low valuations. They come from investing in the companies that do really well. So if there are more of those to be had each year, the best pickers should have more hits.

This means there should be more variability in the VC business. The firms that can recognize and attract the best startups will do even better, because there will be more of them to recognize and attract. Whereas the bad firms will get the leftovers, as they do now, and yet pay a higher price for them.

Nor do I think it will be a problem that founders keep control of their companies for longer. The empirical evidence on that is already clear: investors make more money as founders' bitches than their bosses. Though somewhat humiliating, this is actually good news for investors, because it takes less time to serve founders than to micromanage them.

What about angels? I think there is a lot of opportunity there. It used to suck to be an angel investor. You couldn't get access to the best deals, unless you got lucky like Andy Bechtolsheim, and when you did invest in a startup, VCs might try to strip you of your stock when they arrived later. Now an angel can go to something like Demo Day or AngelList and have access to the same deals VCs do. And the days when VCs could wash angels out of the cap table are long gone.

I think one of the biggest unexploited opportunities in startup investing right now is angel-sized investments made quickly. Few investors understand the cost that raising money from them imposes on startups. When the company consists only of the founders, everything grinds to a halt during fundraising, which can easily take 6 weeks. The current high cost of fundraising means there is room for low-cost investors to undercut the rest. And in this context, low-cost means deciding quickly. If there were a reputable investor who invested \$100k on good terms and promised to decide yes or no within 24 hours, they'd get access to almost all the best deals, because every good startup would approach them first. It would be up to them to pick, because every bad startup would approach them first too, but at least they'd see everything. Whereas if an investor is notorious for taking a long time to make up their mind or negotiating a lot about valuation, founders will save them for last. And in the case of the most promising startups, which tend to have an easy time raising money, last can easily become never.

Will the number of big hits grow linearly with the total number of new startups? Probably not, for two reasons. One is that the

scariness of starting a startup in the old days was a pretty effective filter. Now that the cost of failing is becoming lower, we should expect founders to do it more. That's not a bad thing. It's common in technology for an innovation that decreases the cost of failure to increase the number of failures and yet leave you net ahead.

The other reason the number of big hits won't grow proportionately to the number of startups is that there will start to be an increasing number of idea clashes. Although the finiteness of the number of good ideas is not the reason there are only 15 big hits a year, the number has to be finite, and the more startups there are, the more we'll see multiple companies doing the same thing at the same time. It will be interesting, in a bad way, if idea clashes become a lot more common. [\[Z\]](#)

Mostly because of the increasing number of early failures, the startup business of the future won't simply be the same shape, scaled up. What used to be an obelisk will become a pyramid. It will be a little wider at the top, but a lot wider at the bottom.

What does that mean for investors? One thing it means is that there will be more opportunities for investors at the earliest stage, because that's where the volume of our imaginary solid is growing fastest. Imagine the obelisk of investors that corresponds to the obelisk of startups. As it widens out into a pyramid to match the startup pyramid, all the contents are adhering to the top, leaving a vacuum at the bottom.

That opportunity for investors mostly means an opportunity for new investors, because the degree of risk an existing investor or firm is comfortable taking is one of the hardest things for them to change. Different types of investors are adapted to different degrees of risk, but each has its specific degree of risk deeply imprinted on it, not just in the procedures they follow but in the personalities of the people who work there.

I think the biggest danger for VCs, and also the biggest opportunity, is at the series A stage. Or rather, what used to be the series A stage before series As turned into de facto series B rounds.

Right now, VCs often knowingly invest too much money at the series A stage. They do it because they feel they need to get a big chunk of each series A company to compensate for the opportunity cost of the board seat it consumes. Which means when there is a lot of competition for a deal, the number that moves is the valuation (and thus amount invested) rather than the percentage of the company being sold. Which means,

especially in the case of more promising startups, that series A investors often make companies take more money than they want.

Some VCs lie and claim the company really needs that much. Others are more candid, and admit their financial models require them to own a certain percentage of each company. But we all know the amounts being raised in series A rounds are not determined by asking what would be best for the companies. They're determined by VCs starting from the amount of the company they want to own, and the market setting the valuation and thus the amount invested.

Like a lot of bad things, this didn't happen intentionally. The VC business backed into it as their initial assumptions gradually became obsolete. The traditions and financial models of the VC business were established when founders needed investors more. In those days it was natural for founders to sell VCs a big chunk of their company in the series A round. Now founders would prefer to sell less, and VCs are digging in their heels because they're not sure if they can make money buying less than 20% of each series A company.

The reason I describe this as a danger is that series A investors are increasingly at odds with the startups they supposedly serve, and that tends to come back to bite you eventually. The reason I describe it as an opportunity is that there is now a lot of potential energy built up, as the market has moved away from VCs' traditional business model. Which means the first VC to break ranks and start to do series A rounds for as much equity as founders want to sell (and with no "option pool" that comes only from the founders' shares) stands to reap huge benefits.

What will happen to the VC business when that happens? Hell if I know. But I bet that particular firm will end up ahead. If one top-tier VC firm started to do series A rounds that started from the amount the company needed to raise and let the percentage acquired vary with the market, instead of the other way around, they'd instantly get almost all the best startups. And that's where the money is.

You can't fight market forces forever. Over the last decade we've seen the percentage of the company sold in series A rounds creep inexorably downward. 40% used to be common. Now VCs are fighting to hold the line at 20%. But I am daily waiting for the line to collapse. It's going to happen. You may as well anticipate it, and look bold.

Who knows, maybe VCs will make more money by doing the right thing. It wouldn't be the first time that happened. Venture capital is a business where occasional big successes generate hundredfold returns. How much confidence can you really have in financial models for something like that anyway? The big successes only have to get a tiny bit less occasional to compensate for a 2x decrease in the stock sold in series A rounds.

If you want to find new opportunities for investing, look for things founders complain about. Founders are your customers, and the things they complain about are unsatisfied demand. I've given two examples of things founders complain about most—investors who take too long to make up their minds, and excessive dilution in series A rounds—so those are good places to look now. But the more general recipe is: do something founders want.

Notes

[1] I realize revenue and not fundraising is the proper test of success for a startup. The reason we quote statistics about fundraising is because those are the numbers we have. We couldn't talk meaningfully about revenues without including the numbers from the most successful startups, and we don't have those. We often discuss revenue growth with the earlier stage startups, because that's how we gauge their progress, but when companies reach a certain size it gets presumptuous for a seed investor to do that.

In any case, companies' market caps do eventually become a function of revenues, and post-money valuations of funding rounds are at least guesses by pros about where those market caps will end up.

The reason only 287 have valuations is that the rest have mostly raised money on convertible notes, and although convertible notes often have valuation caps, a valuation cap is merely an upper bound on a valuation.

[2] We didn't try to accept a particular number. We have no way of doing that even if we wanted to. We just tried to be significantly pickier.

[3] Though you never know with bottlenecks, I'm guessing the

next one will be coordinating efforts among partners.

[4] I realize starting a company doesn't have to mean starting a [startup](#). There will be lots of people starting normal companies too. But that's not relevant to an audience of investors.

Geoff Ralston reports that in Silicon Valley it seemed thinkable to start a startup in the mid 1980s. It would have started there. But I know it didn't to undergraduates on the East Coast.

[5] This trend is one of the main causes of the increase in economic inequality in the US since the mid twentieth century. The person who would in 1950 have been the general manager of the x division of Megacorp is now the founder of the x company, and owns significant equity in it.

[6] If Congress passes the [founder visa](#) in a non-broken form, that alone could in principle get us up to 20x, since 95% of the world's population lives outside the US.

[7] If idea clashes got bad enough, it could change what it means to be a startup. We currently advise startups mostly to ignore competitors. We tell them startups are competitive like running, not like soccer; you don't have to go and steal the ball away from the other team. But if idea clashes became common enough, maybe you'd start to have to. That would be unfortunate.

Thanks to Sam Altman, Paul Buchheit, Dalton Caldwell, Patrick Collison, Jessica Livingston, Andrew Mason, Geoff Ralston, and Garry Tan for reading drafts of this.

PAUL GRAHAM

DO THINGS THAT DON'T SCALE

Want to start a startup? Get funded by [Y Combinator](#).

July 2013

One 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.

Recruit

The 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.

Fragile

Airbnb 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. [\[5\]](#)

Delight

You 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.

Experience

I 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. [\[Z\]](#)

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.

Fire

Sometimes 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.

Meraki

For [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?

Consult

Sometimes 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 foothold 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.

Manual

There'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.

Big

I 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.

Vector

The 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 CONVINCE INVESTORS

Want to start a startup? Get funded by [Y Combinator](#).

August 2013

When 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.

Formidable

The 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.

Truth

The 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.

Market

Notice 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?

Rejection

If 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. [\[12\]](#)

Different

Not 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:

1. Make something worth investing in.
2. Understand why it's worth investing in.
3. 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.

PAUL GRAHAM

INVESTOR HERD DYNAMICS

Want to start a startup? Get funded by [Y Combinator](#).

August 2013

The 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.

BEFORE THE STARTUP

Want to start a startup? Get funded by [Y Combinator](#).

October 2014

(This essay is derived from a guest lecture in Sam Altman's [startup class](#) at Stanford. It's intended for college students, but much of it is applicable to potential founders at other ages.)

One of the advantages of having kids is that when you have to give advice, you can ask yourself "what would I tell my own kids?" My kids are little, but I can imagine what I'd tell them about startups if they were in college, and that's what I'm going to tell you.

Startups are very counterintuitive. I'm not sure why. Maybe it's just because knowledge about them hasn't permeated our culture yet. But whatever the reason, starting a startup is a task where you can't always trust your instincts.

It's like skiing in that way. When you first try skiing and you want to slow down, your instinct is to lean back. But if you lean back on skis you fly down the hill out of control. So part of learning to ski is learning to suppress that impulse. Eventually you get new habits, but at first it takes a conscious effort. At

first there's a list of things you're trying to remember as you start down the hill.

Startups are as unnatural as skiing, so there's a similar list for startups. Here I'm going to give you the first part of it — the things to remember if you want to prepare yourself to start a startup.

Counterintuitive

The 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'être*. 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.

Expertise

The 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.

Game

We 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-Consuming

That 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.

Try

Should 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.

Ideas

So 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.

PAUL GRAHAM

MEAN PEOPLE FAIL

November 2014

It struck me recently how few of the most successful people I know are mean. There are exceptions, but remarkably few.

Meanness isn't rare. In fact, one of the things the internet has shown us is how mean people can be. A few decades ago, only

famous people and professional writers got to publish their opinions. Now everyone can, and we can all see the long tail of meanness that had previously been hidden.

And yet while there are clearly a lot of mean people out there, there are next to none among the most successful people I know. What's going on here? Are meanness and success inversely correlated?

Part of what's going on, of course, is selection bias. I only know people who work in certain fields: startup founders, programmers, professors. I'm willing to believe that successful people in other fields are mean. Maybe successful hedge fund managers are mean; I don't know enough to say. It seems quite likely that most successful drug lords are mean. But there are at least big chunks of the world that mean people don't rule, and that territory seems to be growing.

My wife and Y Combinator cofounder Jessica is one of those rare people who have x-ray vision for character. Being married to her is like standing next to an airport baggage scanner. She came to the startup world from investment banking, and she has always been struck both by how consistently successful startup founders turn out to be good people, and how consistently bad people fail as startup founders.

Why? I think there are several reasons. One is that being mean makes you stupid. That's why I hate fights. You never do your best work in a fight, because fights are not sufficiently general. Winning is always a function of the situation and the people involved. You don't win fights by thinking of big ideas but by thinking of tricks that work in one particular case. And yet fighting is just as much work as thinking about real problems. Which is particularly painful to someone who cares how their brain is used: your brain goes fast but you get nowhere, like a car spinning its wheels.

Startups don't win by attacking. They win by transcending. There are exceptions of course, but usually the way to win is to race ahead, not to stop and fight.

Another reason mean founders lose is that they can't get the best people to work for them. They can hire people who will put up with them because they need a job. But the best people have other options. A mean person can't convince the best people to work for him unless he is super convincing. And while having the best people helps any organization, it's critical for startups.

There is also a complementary force at work: if you want to build great things, it helps to be driven by a spirit of benevolence. The startup founders who end up richest are not the ones driven by money. The ones driven by money take the big acquisition offer that nearly every successful startup gets en route. [1] The ones who keep going are driven by something else. They may not say so explicitly, but they're usually trying to improve the world. Which means people with a desire to improve the world have a natural advantage. [2]

The exciting thing is that startups are not just one random type of work in which meanness and success are inversely correlated. This kind of work is the future.

For most of history success meant control of scarce resources. One got that by fighting, whether literally in the case of pastoral nomads driving hunter-gatherers into marginal lands, or metaphorically in the case of Gilded Age financiers contending with one another to assemble railroad monopolies. For most of history, success meant success at zero-sum games. And in most of them meanness was not a handicap but probably an advantage.

That is changing. Increasingly the games that matter are not zero-sum. Increasingly you win not by fighting to get control of a scarce resource, but by having new ideas and building new things. [3]

There have long been games where you won by having new ideas. In the third century BC, Archimedes won by doing that. At least until an invading Roman army killed him. Which illustrates why this change is happening: for new ideas to matter, you need a certain degree of civil order. And not just not being at war. You also need to prevent the sort of economic violence that nineteenth century magnates practiced against one another and communist countries practiced against their citizens. People need to feel that what they create can't be stolen. [4]

That has always been the case for thinkers, which is why this trend began with them. When you think of successful people from history who weren't ruthless, you get mathematicians and writers and artists. The exciting thing is that their m.o. seems to be spreading. The games played by intellectuals are leaking into the real world, and this is reversing the historical polarity of the relationship between meanness and success.

So I'm really glad I stopped to think about this. Jessica and I have always worked hard to teach our kids not to be mean. We

tolerate noise and mess and junk food, but not meanness. And now I have both an additional reason to crack down on it, and an additional argument to use when I do: that being mean makes you fail.

Notes

[1] I'm not saying all founders who take big acquisition offers are driven only by money, but rather that those who don't aren't. Plus one can have benevolent motives for being driven by money — for example, to take care of one's family, or to be free to work on projects that improve the world.

[2] It's unlikely that every successful startup improves the world. But their founders, like parents, truly believe they do. Successful founders are in love with their companies. And while this sort of love is as blind as the love people have for one another, it is genuine.

[3] [Peter Thiel](#) would point out that successful founders still get rich from controlling monopolies, just monopolies they create rather than ones they capture. And while this is largely true, it means a big change in the sort of person who wins.

[4] To be fair, the Romans didn't mean to kill Archimedes. The Roman commander specifically ordered that he be spared. But he got killed in the chaos anyway.

In sufficiently disordered times, even thinking requires control of scarce resources, because living at all is a scarce resource.

Thanks to Sam Altman, Ron Conway, Daniel Gackle, Jessica Livingston, Robert Morris, Geoff Ralston, and Fred Wilson for reading drafts of this.

PAUL GRAHAM

HOW Y COMBINATOR STARTED

March 2012

Y Combinator's 7th birthday was March 11. As usual we were so busy we didn't notice till a few days after. I don't think we've ever managed to remember our birthday on our birthday.

On March 11 2005, Jessica and I were walking home from dinner in Harvard Square. Jessica was working at an investment bank at the time, but she didn't like it much, so she had interviewed for a job as director of marketing at a Boston VC fund. The VC fund was doing what now seems a comically familiar thing for a VC fund to do: taking a long time to make up their mind. Meanwhile I had been telling Jessica all the things they should change about the VC business — essentially the ideas now underlying Y Combinator: investors should be making more, smaller investments, they should be funding hackers instead of suits, they should be willing to fund younger founders, etc.

At the time I had been thinking about doing some angel investing. I had just given a talk to the undergraduate computer club at Harvard about [how to start a startup](#), and it hit me afterward that although I had always meant to do angel investing, 7 years had now passed since I got enough money to do it, and I still hadn't started. I had also been thinking about ways to work with Robert Morris and Trevor Blackwell again. A few hours before I had sent them an email trying to figure out what we could do together.

Between Harvard Square and my house the idea gelled. We'd start our own investment firm and Jessica could work for that instead. As we turned onto Walker Street we decided to do it. I agreed to put \$100k into the new fund and Jessica agreed to quit her job to work for it. Over the next couple days I recruited Robert and Trevor, who put in another \$50k each. So YC started with \$200k.

Jessica was so happy to be able to quit her job and start her own company that I took her [picture](#) when we got home.

The company wasn't called Y Combinator yet. At first we called it Cambridge Seed. But that name never saw the light of day, because by the time we announced it a few days later, we'd changed the name to Y Combinator. We realized early on that what we were doing could be national in scope and we didn't want a name that tied us to one place.

Initially we only had part of the idea. We were going to do seed funding with standardized terms. Before YC, seed funding was very haphazard. You'd get that first \$10k from your friend's rich uncle. The deal terms were often a disaster; often neither the investor nor the founders nor the lawyer knew what the documents should look like. Facebook's early history as a Florida LLC shows how random things could be in those days. We were going to be something there had not been before: a standard source of seed funding.

We modelled YC on the seed funding we ourselves had taken when we started Viaweb. We started Viaweb with \$10k we got from our friend [Julian Weber](#), the husband of Idelle Weber, whose painting class I took as a grad student at Harvard. Julian knew about business, but you would not describe him as a suit. Among other things he'd been president of the *National Lampoon*. He was also a lawyer, and got all our paperwork set up properly. In return for \$10k, getting us set up as a company, teaching us what business was about, and remaining calm in times of crisis, Julian got 10% of Viaweb. I remember thinking once what a good deal Julian got. And then a second later I realized that without Julian, Viaweb would never have made it. So even though it was a good deal for him, it was a good deal for us too. That's why I knew there was room for something like Y Combinator.

Initially we didn't have what turned out to be the most important idea: funding startups synchronously, instead of asynchronously as it had always been done before. Or rather we had the idea, but we didn't realize its significance. We decided very early that the first thing we'd do would be to fund a bunch of startups over the coming summer. But we didn't realize initially that this would be the way we'd do all our investing. The reason we began by funding a bunch of startups at once was not that we thought it would be a better way to fund startups, but simply because we wanted to learn how to be angel investors, and a summer program for undergrads seemed the fastest way to do it. No one takes summer jobs that seriously. The opportunity cost for a bunch of undergrads to spend a summer working on startups was low enough that we wouldn't feel guilty encouraging them to do it.

We knew students would already be making plans for the summer, so we did what we're always telling startups to do: we launched fast. Here are the initial [announcement](#) and [description](#) of what was at the time called the Summer Founders Program.

We got lucky in that the length and structure of a summer program turns out to be perfect for what we do. The structure of the YC cycle is still almost identical to what it was that first summer.

We also got lucky in who the first batch of founders were. We never expected to make any money from that first batch. We thought of the money we were investing as a combination of an educational expense and a charitable donation. But the founders in the first batch turned out to be surprisingly good. And great people too. We're still friends with a lot of them today.

It's hard for people to realize now how inconsequential YC seemed at the time. I can't blame people who didn't take us seriously, because we ourselves didn't take that first summer program seriously in the very beginning. But as the summer progressed we were increasingly impressed by how well the startups were doing. Other people started to be impressed too.

Jessica and I invented a term, "the Y Combinator effect," to describe the moment when the realization hit someone that YC was not totally lame. When people came to YC to speak at the dinners that first summer, they came in the spirit of someone coming to address a Boy Scout troop. By the time they left the building they were all saying some variant of "Wow, these companies might actually succeed."

Now YC is well enough known that people are no longer surprised when the companies we fund are legit, but it took a while for reputation to catch up with reality. That's one of the reasons we especially like funding ideas that might be dismissed as "toys" — because YC itself was dismissed as one initially.

When we saw how well it worked to fund companies synchronously, we decided we'd keep doing that. We'd fund two batches of startups a year.

We funded the second batch in Silicon Valley. That was a last minute decision. In retrospect I think what pushed me over the edge was going to Foo Camp that fall. The density of startup people in the Bay Area was so much greater than in Boston, and the weather was so nice. I remembered that from living there in the 90s. Plus I didn't want someone else to copy us and describe it as the Y Combinator of Silicon Valley. I wanted YC to be the Y Combinator of Silicon Valley. So doing the winter batch in California seemed like one of those rare cases where the self-indulgent choice and the ambitious one were the same.

If we'd had enough time to do what we wanted, Y Combinator would have been in Berkeley. That was our favorite part of the Bay Area. But we didn't have time to get a building in Berkeley. We didn't have time to get our own building anywhere. The only way to get enough space in time was to convince Trevor to let us take over part of his (as it then seemed) giant building in Mountain View. Yet again we lucked out, because Mountain View turned out to be the ideal place to put something like YC. But even then we barely made it. The first dinner in California, we had to warn all the founders not to touch the walls, because the paint was still wet.

PAUL GRAHAM

WHAT WE LOOK FOR IN FOUNDERS

Want to start a startup? Get funded by [Y Combinator](#).

October 2010

(I wrote this for Forbes, who asked me to write something about the qualities we look for in founders. In print they had to cut the last item because they didn't have room.)

1. Determination

This 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. Flexibility

You 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. Imagination

Intelligence 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. Naughtiness

Though 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. Friendship

Empirically 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.

PAUL GRAHAM

THE HARDEST LESSONS FOR STARTUPS TO LEARN

April 2006

(This essay is derived from a talk at the 2006 [Startup School](#).)

The startups we've funded so far are pretty quick, but they seem quicker to learn some lessons than others. I think it's because some things about startups are kind of counterintuitive.

We've now [invested](#) in enough companies that I've learned a trick for determining which points are the counterintuitive ones: they're the ones I have to keep repeating.

So I'm going to number these points, and maybe with future startups I'll be able to pull off a form of Huffman coding. I'll make them all read this, and then instead of nagging them in

detail, I'll just be able to say: *number four!*

1. Release Early.

The thing I probably repeat most is this recipe for a startup: get a version 1 out fast, then improve it based on users' reactions.

By "release early" I don't mean you should release something full of bugs, but that you should release something minimal. Users hate bugs, but they don't seem to mind a minimal version 1, if there's more coming soon.

There are several reasons it pays to get version 1 done fast. One is that this is simply the right way to write software, whether for a startup or not. I've been repeating that since 1993, and I haven't seen much since to contradict it. I've seen a lot of startups die because they were too slow to release stuff, and none because they were too quick. [1]

One of the things that will surprise you if you build something popular is that you won't know your users. [Reddit](#) now has almost half a million unique visitors a month. Who are all those people? They have no idea. No web startup does. And since you don't know your users, it's dangerous to guess what they'll like. Better to release something and let them tell you.

[Wufoo](#) took this to heart and released their form-builder before the underlying database. You can't even drive the thing yet, but 83,000 people came to sit in the driver's seat and hold the steering wheel. And Wufoo got valuable feedback from it: Linux users complained they used too much Flash, so they rewrote their software not to. If they'd waited to release everything at once, they wouldn't have discovered this problem till it was more deeply wired in.

Even if you had no users, it would still be important to release quickly, because for a startup the initial release acts as a shakedown cruise. If anything major is broken-- if the idea's no good, for example, or the founders hate one another-- the stress of getting that first version out will expose it. And if you have such problems you want to find them early.

Perhaps the most important reason to release early, though, is that it makes you work harder. When you're working on something that isn't released, problems are intriguing. In something that's out there, problems are alarming. There is a lot more urgency once you release. And I think that's precisely why people put it off. They know they'll have to work a lot harder once they do. [2]

2. Keep Pumping Out Features.

Of course, "release early" has a second component, without which it would be bad advice. If you're going to start with something that doesn't do much, you better improve it fast.

What I find myself repeating is "pump out features." And this rule isn't just for the initial stages. This is something all startups should do for as long as they want to be considered startups.

I don't mean, of course, that you should make your application ever more complex. By "feature" I mean one unit of hacking-- one quantum of making users' lives better.

As with exercise, improvements beget improvements. If you run every day, you'll probably feel like running tomorrow. But if you skip running for a couple weeks, it will be an effort to drag yourself out. So it is with hacking: the more ideas you implement, the more ideas you'll have. You should make your system better at least in some small way every day or two.

This is not just a good way to get development done; it is also a form of marketing. Users love a site that's constantly improving. In fact, users expect a site to improve. Imagine if you visited a site that seemed very good, and then returned two months later and not one thing had changed. Wouldn't it start to seem lame? [\[3\]](#)

They'll like you even better when you improve in response to their comments, because customers are used to companies ignoring them. If you're the rare exception-- a company that actually listens-- you'll generate fanatical loyalty. You won't need to advertise, because your users will do it for you.

This seems obvious too, so why do I have to keep repeating it? I think the problem here is that people get used to how things are. Once a product gets past the stage where it has glaring flaws, you start to get used to it, and gradually whatever features it happens to have become its identity. For example, I doubt many people at Yahoo (or Google for that matter) realized how much better web mail could be till Paul Buchheit showed them.

I think the solution is to assume that anything you've made is far short of what it could be. Force yourself, as a sort of intellectual exercise, to keep thinking of improvements. Ok, sure, what you have is perfect. But if you had to change something, what would it be?

If your product seems finished, there are two possible explanations: (a) it is finished, or (b) you lack imagination. Experience suggests (b) is a thousand times more likely.

3. Make Users Happy.

Improving constantly is an instance of a more general rule: make users happy. One thing all startups have in common is that they can't force anyone to do anything. They can't force anyone to use their software, and they can't force anyone to do deals with them. A startup has to sing for its supper. That's why the successful ones make great things. They have to, or die.

When you're running a startup you feel like a little bit of debris blown about by powerful winds. The most powerful wind is users. They can either catch you and loft you up into the sky, as they did with Google, or leave you flat on the pavement, as they do with most startups. Users are a fickle wind, but more powerful than any other. If they take you up, no competitor can keep you down.

As a little piece of debris, the rational thing for you to do is not to lie flat, but to curl yourself into a shape the wind will catch.

I like the wind metaphor because it reminds you how impersonal the stream of traffic is. The vast majority of people who visit your site will be casual visitors. It's them you have to design your site for. The people who really care will find what they want by themselves.

The median visitor will arrive with their finger poised on the Back button. Think about your own experience: most links you follow lead to something lame. Anyone who has used the web for more than a couple weeks has been *trained* to click on Back after following a link. So your site has to say "Wait! Don't click on Back. This site isn't lame. Look at this, for example."

There are two things you have to do to make people pause. The most important is to explain, as concisely as possible, what the hell your site is about. How often have you visited a site that seemed to assume you already knew what they did? For example, the corporate site that says the company makes enterprise content management solutions for business that enable organizations to unify people, content and processes to minimize business risk, accelerate time-to-value and sustain lower total cost of ownership.

An established company may get away with such an opaque description, but no startup can. A startup should be able to

explain in one or two sentences exactly what it does. [4] And not just to users. You need this for everyone: investors, acquirers, partners, reporters, potential employees, and even current employees. You probably shouldn't even start a company to do something that can't be described compellingly in one or two sentences.

The other thing I repeat is to give people everything you've got, right away. If you have something impressive, try to put it on the front page, because that's the only one most visitors will see. Though indeed there's a paradox here: the more you push the good stuff toward the front, the more likely visitors are to explore further. [5]

In the best case these two suggestions get combined: you tell visitors what your site is about by *showing* them. One of the standard pieces of advice in fiction writing is "show, don't tell." Don't say that a character's angry; have him grind his teeth, or break his pencil in half. Nothing will explain what your site does so well as using it.

The industry term here is "conversion." The job of your site is to convert casual visitors into users-- whatever your definition of a user is. You can measure this in your growth rate. Either your site is catching on, or it isn't, and you must know which. If you have decent growth, you'll win in the end, no matter how obscure you are now. And if you don't, you need to fix something.

4. Fear the Right Things.

Another thing I find myself saying a lot is "don't worry." Actually, it's more often "don't worry about this; worry about that instead." Startups are right to be paranoid, but they sometimes fear the wrong things.

Most visible disasters are not so alarming as they seem. Disasters are normal in a startup: a founder quits, you discover a patent that covers what you're doing, your servers keep crashing, you run into an insoluble technical problem, you have to change your name, a deal falls through-- these are all par for the course. They won't kill you unless you let them.

Nor will most competitors. A lot of startups worry "what if Google builds something like us?" Actually big companies are not the ones you have to worry about-- not even Google. The people at Google are smart, but no smarter than you; they're not as motivated, because Google is not going to go out of business if this one product fails; and even at Google they have

a lot of bureaucracy to slow them down.

What you should fear, as a startup, is not the established players, but other startups you don't know exist yet. They're way more dangerous than Google because, like you, they're cornered animals.

Looking just at existing competitors can give you a false sense of security. You should compete against what someone else *could* be doing, not just what you can see people doing. A corollary is that you shouldn't relax just because you have no visible competitors yet. No matter what your idea, there's someone else out there working on the same thing.

That's the downside of it being easier to start a startup: more people are doing it. But I disagree with Caterina Fake when she says that makes this a bad time to start a startup. More people are starting startups, but not as many more as could. Most college graduates still think they have to get a job. The average person can't ignore something that's been beaten into their head since they were three just because serving web pages recently got a lot cheaper.

And in any case, competitors are not the biggest threat. Way more startups hose themselves than get crushed by competitors. There are a lot of ways to do it, but the three main ones are internal disputes, inertia, and ignoring users. Each is, by itself, enough to kill you. But if I had to pick the worst, it would be ignoring users. If you want a recipe for a startup that's going to die, here it is: a couple of founders who have some great idea they know everyone is going to love, and that's what they're going to build, no matter what.

Almost everyone's initial plan is broken. If companies stuck to their initial plans, Microsoft would be selling programming languages, and Apple would be selling printed circuit boards. In both cases their customers told them what their business should be-- and they were smart enough to listen.

As Richard Feynman said, the imagination of nature is greater than the imagination of man. You'll find more interesting things by looking at the world than you could ever produce just by thinking. This principle is very powerful. It's why the best abstract painting still falls short of Leonardo, for example. And it applies to startups too. No idea for a product could ever be so clever as the ones you can discover by smashing a beam of prototypes into a beam of users.

5. Commitment Is a Self-Fulfilling Prophecy.

I now have enough experience with startups to be able to say what the most important quality is in a startup founder, and it's not what you might think. The most important quality in a startup founder is determination. Not intelligence-- determination.

This is a little depressing. I'd like to believe Viaweb succeeded because we were smart, not merely determined. A lot of people in the startup world want to believe that. Not just founders, but investors too. They like the idea of inhabiting a world ruled by intelligence. And you can tell they really believe this, because it affects their investment decisions.

Time after time VCs invest in startups founded by eminent professors. This may work in biotech, where a lot of startups simply commercialize existing research, but in software you want to invest in students, not professors. Microsoft, Yahoo, and Google were all founded by people who dropped out of school to do it. What students lack in experience they more than make up in dedication.

Of course, if you want to get rich, it's not enough merely to be determined. You have to be smart too, right? I'd like to think so, but I've had an experience that convinced me otherwise: I spent several years living in New York.

You can lose quite a lot in the brains department and it won't kill you. But lose even a little bit in the commitment department, and that will kill you very rapidly.

Running a startup is like walking on your hands: it's possible, but it requires extraordinary effort. If an ordinary employee were asked to do the things a startup founder has to, he'd be very indignant. Imagine if you were hired at some big company, and in addition to writing software ten times faster than you'd ever had to before, they expected you to answer support calls, administer the servers, design the web site, cold-call customers, find the company office space, and go out and get everyone lunch.

And to do all this not in the calm, womb-like atmosphere of a big company, but against a backdrop of constant disasters. That's the part that really demands determination. In a startup, there's always some disaster happening. So if you're the least bit inclined to find an excuse to quit, there's always one right there.

But if you lack commitment, chances are it will have been

hurting you long before you actually quit. Everyone who deals with startups knows how important commitment is, so if they sense you're ambivalent, they won't give you much attention. If you lack commitment, you'll just find that for some mysterious reason good things happen to your competitors but not to you. If you lack commitment, it will seem to you that you're unlucky.

Whereas if you're determined to stick around, people will pay attention to you, because odds are they'll have to deal with you later. You're a local, not just a tourist, so everyone has to come to terms with you.

At Y Combinator we sometimes mistakenly fund teams who have the attitude that they're going to give this startup thing a shot for three months, and if something great happens, they'll stick with it-- "something great" meaning either that someone wants to buy them or invest millions of dollars in them. But if this is your attitude, "something great" is very unlikely to happen to you, because both acquirers and investors judge you by your level of commitment.

If an acquirer thinks you're going to stick around no matter what, they'll be more likely to buy you, because if they don't and you stick around, you'll probably grow, your price will go up, and they'll be left wishing they'd bought you earlier. Ditto for investors. What really motivates investors, even big VCs, is not the hope of good returns, but the fear of missing out. [6] So if you make it clear you're going to succeed no matter what, and the only reason you need them is to make it happen a little faster, you're much more likely to get money.

You can't fake this. The only way to convince everyone that you're ready to fight to the death is actually to be ready to.

You have to be the right kind of determined, though. I carefully chose the word determined rather than stubborn, because stubbornness is a disastrous quality in a startup. You have to be determined, but flexible, like a running back. A successful running back doesn't just put his head down and try to run through people. He improvises: if someone appears in front of him, he runs around them; if someone tries to grab him, he spins out of their grip; he'll even run in the wrong direction briefly if that will help. The one thing he'll never do is stand still. [7]

6. There Is Always Room.

I was talking recently to a startup founder about whether it might be good to add a social component to their software. He

said he didn't think so, because the whole social thing was tapped out. Really? So in a hundred years the only social networking sites will be the Facebook, MySpace, Flickr, and Del.icio.us? Not likely.

There is always room for new stuff. At every point in history, even the darkest bits of the dark ages, people were discovering things that made everyone say "why didn't anyone think of that before?" We know this continued to be true up till 2004, when the Facebook was founded-- though strictly speaking someone else did think of that.

The reason we don't see the opportunities all around us is that we adjust to however things are, and assume that's how things have to be. For example, it would seem crazy to most people to try to make a better search engine than Google. Surely that field, at least, is tapped out. Really? In a hundred years-- or even twenty-- are people still going to search for information using something like the current Google? Even Google probably doesn't think that.

In particular, I don't think there's any limit to the number of startups. Sometimes you hear people saying "All these guys starting startups now are going to be disappointed. How many little startups are Google and Yahoo going to buy, after all?" That sounds cleverly skeptical, but I can prove it's mistaken. No one proposes that there's some limit to the number of people who can be employed in an economy consisting of big, slow-moving companies with a couple thousand people each. Why should there be any limit to the number who could be employed by small, fast-moving companies with ten each? It seems to me the only limit would be the number of people who want to work that hard.

The limit on the number of startups is not the number that can get acquired by Google and Yahoo-- though it seems even that should be unlimited, if the startups were actually worth buying-- but the amount of wealth that can be created. And I don't think there's any limit on that, except cosmological ones.

So for all practical purposes, there is no limit to the number of startups. Startups make wealth, which means they make things people want, and if there's a limit on the number of things people want, we are nowhere near it. I still don't even have a flying car.

7. Don't Get Your Hopes Up.

This is another one I've been repeating since long before Y

Combinator. It was practically the corporate motto at Viaweb.

Startup founders are naturally optimistic. They wouldn't do it otherwise. But you should treat your optimism the way you'd treat the core of a nuclear reactor: as a source of power that's also very dangerous. You have to build a shield around it, or it will fry you.

The shielding of a reactor is not uniform; the reactor would be useless if it were. It's pierced in a few places to let pipes in. An optimism shield has to be pierced too. I think the place to draw the line is between what you expect of yourself, and what you expect of other people. It's ok to be optimistic about what you can do, but assume the worst about machines and other people.

This is particularly necessary in a startup, because you tend to be pushing the limits of whatever you're doing. So things don't happen in the smooth, predictable way they do in the rest of the world. Things change suddenly, and usually for the worse.

Shielding your optimism is nowhere more important than with deals. If your startup is doing a deal, just assume it's not going to happen. The VCs who say they're going to invest in you aren't. The company that says they're going to buy you isn't. The big customer who wants to use your system in their whole company won't. Then if things work out you can be pleasantly surprised.

The reason I warn startups not to get their hopes up is not to save them from being *disappointed* when things fall through. It's for a more practical reason: to prevent them from leaning their company against something that's going to fall over, taking them with it.

For example, if someone says they want to invest in you, there's a natural tendency to stop looking for other investors. That's why people proposing deals seem so positive: they *want* you to stop looking. And you want to stop too, because doing deals is a pain. Raising money, in particular, is a huge time sink. So you have to consciously force yourself to keep looking.

Even if you ultimately do the first deal, it will be to your advantage to have kept looking, because you'll get better terms. Deals are dynamic; unless you're negotiating with someone unusually honest, there's not a single point where you shake hands and the deal's done. There are usually a lot of subsidiary questions to be cleared up after the handshake, and if the other side senses weakness-- if they sense you need this deal-- they will be very tempted to screw you in the details.

VCs and corp dev guys are professional negotiators. They're trained to take advantage of weakness. [8] So while they're often nice guys, they just can't help it. And as pros they do this more than you. So don't even try to bluff them. The only way a startup can have any leverage in a deal is genuinely not to need it. And if you don't believe in a deal, you'll be less likely to depend on it.

So I want to plant a hypnotic suggestion in your heads: when you hear someone say the words "we want to invest in you" or "we want to acquire you," I want the following phrase to appear automatically in your head: *don't get your hopes up*. Just continue running your company as if this deal didn't exist. Nothing is more likely to make it close.

The way to succeed in a startup is to focus on the goal of getting lots of users, and keep walking swiftly toward it while investors and acquirers scurry alongside trying to wave money in your face.

Speed, not Money

The way I've described it, starting a startup sounds pretty stressful. It is. When I talk to the founders of the companies we've funded, they all say the same thing: I knew it would be hard, but I didn't realize it would be this hard.

So why do it? It would be worth enduring a lot of pain and stress to do something grand or heroic, but just to make money? Is making money really that important?

No, not really. It seems ridiculous to me when people take business too seriously. I regard making money as a boring errand to be got out of the way as soon as possible. There is nothing grand or heroic about starting a startup per se.

So why do I spend so much time thinking about startups? I'll tell you why. Economically, a startup is best seen not as a way to get rich, but as a way to work faster. You have to make a living, and a startup is a way to get that done quickly, instead of letting it drag on through your whole life. [9]

We take it for granted most of the time, but human life is fairly miraculous. It is also palpably short. You're given this marvellous thing, and then poof, it's taken away. You can see why people invent gods to explain it. But even to people who don't believe in gods, life commands respect. There are times in most of our lives when the days go by in a blur, and almost

everyone has a sense, when this happens, of wasting something precious. As Ben Franklin said, if you love life, don't waste time, because time is what life is made of.

So no, there's nothing particularly grand about making money. That's not what makes startups worth the trouble. What's important about startups is the speed. By compressing the dull but necessary task of making a living into the smallest possible time, you show respect for life, and there is something grand about that.

Notes

[1] Startups can die from releasing something full of bugs, and not fixing them fast enough, but I don't know of any that died from releasing something stable but minimal very early, then promptly improving it.

[2] I know this is why I haven't released Arc. The moment I do, I'll have people nagging me for features.

[3] A web site is different from a book or movie or desktop application in this respect. Users judge a site not as a single snapshot, but as an animation with multiple frames. Of the two, I'd say the rate of improvement is more important to users than where you currently are.

[4] It should not always tell this to users, however. For example, MySpace is basically a replacement mall for mallrats. But it was wiser for them, initially, to pretend that the site was about bands.

[5] Similarly, don't make users register to try your site. Maybe what you have is so valuable that visitors should gladly register to get at it. But they've been trained to expect the opposite. Most of the things they've tried on the web have sucked-- and probably especially those that made them register.

[6] VCs have rational reasons for behaving this way. They don't make their money (if they make money) off their median investments. In a typical fund, half the companies fail, most of the rest generate mediocre returns, and one or two "make the fund" by succeeding spectacularly. So if they miss just a few of the most promising opportunities, it could hose the whole fund.

[7] The attitude of a running back doesn't translate to soccer. Though it looks great when a forward dribbles past multiple defenders, a player who persists in trying such things will do worse in the long term than one who passes.

[8] The reason Y Combinator never negotiates valuations is that we're not professional negotiators, and don't want to turn into them.

[9] There are two ways to do [work you love](#): (a) to make money, then work on what you love, or (b) to get a job where you get paid to work on stuff you love. In practice the first phases of both consist mostly of unedifying schleps, and in (b) the second phase is less secure.

Thanks to Sam Altman, Trevor Blackwell, Beau Hartshorne, Jessica Livingston, and Robert Morris for reading drafts of this.

PAUL GRAHAM

HOW TO BE SILICON VALLEY

May 2006

(This essay is derived from a keynote at Xtech.)

Could you reproduce Silicon Valley elsewhere, or is there something unique about it?

It wouldn't be surprising if it were hard to reproduce in other countries, because you couldn't reproduce it in most of the US either. What does it take to make a silicon valley even here?

What it takes is the right people. If you could get the right ten thousand people to move from Silicon Valley to Buffalo, Buffalo would become Silicon Valley. [\[1\]](#)

That's a striking departure from the past. Up till a couple decades ago, geography was destiny for cities. All great cities were located on waterways, because cities made money by trade, and water was the only economical way to ship.

Now you could make a great city anywhere, if you could get the right people to move there. So the question of how to make a silicon valley becomes: who are the right people, and how do you get them to move?

Two Types

I think you only need two kinds of people to create a technology hub: rich people and nerds. They're the limiting reagents in the reaction that produces startups, because they're the only ones present when startups get started. Everyone else will move.

Observation bears this out: within the US, towns have become startup hubs if and only if they have both rich people and nerds. Few startups happen in Miami, for example, because although it's full of rich people, it has few nerds. It's not the kind of place nerds like.

Whereas Pittsburgh has the opposite problem: plenty of nerds, but no rich people. The top US Computer Science departments are said to be MIT, Stanford, Berkeley, and Carnegie-Mellon. MIT yielded Route 128. Stanford and Berkeley yielded Silicon Valley. But Carnegie-Mellon? The record skips at that point. Lower down the list, the University of Washington yielded a high-tech community in Seattle, and the University of Texas at Austin yielded one in Austin. But what happened in Pittsburgh? And in Ithaca, home of Cornell, which is also high on the list?

I grew up in Pittsburgh and went to college at Cornell, so I can answer for both. The weather is terrible, particularly in winter, and there's no interesting old city to make up for it, as there is in Boston. Rich people don't want to live in Pittsburgh or Ithaca. So while there are plenty of hackers who could start startups, there's no one to invest in them.

Not Bureaucrats

Do you really need the rich people? Wouldn't it work to have the government invest in the nerds? No, it would not. Startup investors are a distinct type of rich people. They tend to have a lot of experience themselves in the technology business. This (a) helps them pick the right startups, and (b) means they can supply advice and connections as well as money. And the fact that they have a personal stake in the outcome makes them really pay attention.

Bureaucrats by their nature are the exact opposite sort of people from startup investors. The idea of them making startup investments is comic. It would be like mathematicians running *Vogue*-- or perhaps more accurately, *Vogue* editors running a math journal. [\[2\]](#)

Though indeed, most things bureaucrats do, they do badly. We just don't notice usually, because they only have to compete against other bureaucrats. But as startup investors they'd have to compete against pros with a great deal more experience and

motivation.

Even corporations that have in-house VC groups generally forbid them to make their own investment decisions. Most are only allowed to invest in deals where some reputable private VC firm is willing to act as lead investor.

Not Buildings

If you go to see Silicon Valley, what you'll see are buildings. But it's the people that make it Silicon Valley, not the buildings. I read occasionally about attempts to set up "[technology parks](#)" in other places, as if the active ingredient of Silicon Valley were the office space. An article about Sophia Antipolis bragged that companies there included Cisco, Compaq, IBM, NCR, and Nortel. Don't the French realize these aren't startups?

Building office buildings for technology companies won't get you a silicon valley, because the key stage in the life of a startup happens before they want that kind of space. The key stage is when they're three guys operating out of an apartment. Wherever the startup is when it gets funded, it will stay. The defining quality of Silicon Valley is not that Intel or Apple or Google have offices there, but that they were *started* there.

So if you want to reproduce Silicon Valley, what you need to reproduce is those two or three founders sitting around a kitchen table deciding to start a company. And to reproduce that you need those people.

Universities

The exciting thing is, *all* you need are the people. If you could attract a critical mass of nerds and investors to live somewhere, you could reproduce Silicon Valley. And both groups are highly mobile. They'll go where life is good. So what makes a place good to them?

What nerds like is other nerds. Smart people will go wherever other smart people are. And in particular, to great universities. In theory there could be other ways to attract them, but so far universities seem to be indispensable. Within the US, there are no technology hubs without first-rate universities-- or at least, first-rate computer science departments.

So if you want to make a silicon valley, you not only need a university, but one of the top handful in the world. It has to be good enough to act as a magnet, drawing the best people from thousands of miles away. And that means it has to stand up to

existing magnets like MIT and Stanford.

This sounds hard. Actually it might be easy. My professor friends, when they're deciding where they'd like to work, consider one thing above all: the quality of the other faculty. What attracts professors is good colleagues. So if you managed to recruit, en masse, a significant number of the best young researchers, you could create a first-rate university from nothing overnight. And you could do that for surprisingly little. If you paid 200 people hiring bonuses of \$3 million apiece, you could put together a faculty that would bear comparison with any in the world. And from that point the chain reaction would be self-sustaining. So whatever it costs to establish a mediocre university, for an additional half billion or so you could have a great one. [3]

Personality

However, merely creating a new university would not be enough to start a silicon valley. The university is just the seed. It has to be planted in the right soil, or it won't germinate. Plant it in the wrong place, and you just create Carnegie-Mellon.

To spawn startups, your university has to be in a town that has attractions other than the university. It has to be a place where investors want to live, and students want to stay after they graduate.

The two like much the same things, because most startup investors are nerds themselves. So what do nerds look for in a town? Their tastes aren't completely different from other people's, because a lot of the towns they like most in the US are also big tourist destinations: San Francisco, Boston, Seattle. But their tastes can't be quite mainstream either, because they dislike other big tourist destinations, like New York, Los Angeles, and Las Vegas.

There has been a lot written lately about the "creative class." The thesis seems to be that as wealth derives increasingly from ideas, cities will prosper only if they attract those who have them. That is certainly true; in fact it was the basis of Amsterdam's prosperity 400 years ago.

A lot of nerd tastes they share with the creative class in general. For example, they like well-preserved old neighborhoods instead of cookie-cutter suburbs, and locally-owned shops and restaurants instead of national chains. Like the rest of the creative class, they want to live somewhere with personality.

What exactly is personality? I think it's the feeling that each building is the work of a distinct group of people. A town with personality is one that doesn't feel mass-produced. So if you want to make a startup hub-- or any town to attract the "creative class"-- you probably have to ban large development projects. When a large tract has been developed by a single organization, you can always tell. [\[4\]](#)

Most towns with personality are old, but they don't have to be. Old towns have two advantages: they're denser, because they were laid out before cars, and they're more varied, because they were built one building at a time. You could have both now. Just have building codes that ensure density, and ban large scale developments.

A corollary is that you have to keep out the biggest developer of all: the government. A government that asks "How can we build a silicon valley?" has probably ensured failure by the way they framed the question. You don't build a silicon valley; you let one grow.

Nerds

If you want to attract nerds, you need more than a town with personality. You need a town with the right personality. Nerds are a distinct subset of the creative class, with different tastes from the rest. You can see this most clearly in New York, which attracts a lot of creative people, but few nerds. [\[5\]](#)

What nerds like is the kind of town where people walk around smiling. This excludes LA, where no one walks at all, and also New York, where people walk, but not smiling. When I was in grad school in Boston, a friend came to visit from New York. On the subway back from the airport she asked "Why is everyone smiling?" I looked and they weren't smiling. They just looked like they were compared to the facial expressions she was used to.

If you've lived in New York, you know where these facial expressions come from. It's the kind of place where your mind may be excited, but your body knows it's having a bad time. People don't so much enjoy living there as endure it for the sake of the excitement. And if you like certain kinds of excitement, New York is incomparable. It's a hub of glamour, a magnet for all the shorter half-life isotopes of style and fame.

Nerds don't care about glamour, so to them the appeal of New York is a mystery. People who like New York will pay a fortune for a small, dark, noisy apartment in order to live in a town

where the cool people are really cool. A nerd looks at that deal and sees only: pay a fortune for a small, dark, noisy apartment.

Nerds *will* pay a premium to live in a town where the smart people are really smart, but you don't have to pay as much for that. It's supply and demand: glamour is popular, so you have to pay a lot for it.

Most nerds like quieter pleasures. They like cafes instead of clubs; used bookshops instead of fashionable clothing shops; hiking instead of dancing; sunlight instead of tall buildings. A nerd's idea of paradise is Berkeley or Boulder.

Youth

It's the young nerds who start startups, so it's those specifically the city has to appeal to. The startup hubs in the US are all young-feeling towns. This doesn't mean they have to be new. Cambridge has the oldest town plan in America, but it feels young because it's full of students.

What you can't have, if you want to create a silicon valley, is a large, existing population of stodgy people. It would be a waste of time to try to reverse the fortunes of a declining industrial town like Detroit or Philadelphia by trying to encourage startups. Those places have too much momentum in the wrong direction. You're better off starting with a blank slate in the form of a small town. Or better still, if there's a town young people already flock to, that one.

The Bay Area was a magnet for the young and optimistic for decades before it was associated with technology. It was a place people went in search of something new. And so it became synonymous with California nuttiness. There's still a lot of that there. If you wanted to start a new fad-- a new way to focus one's "energy," for example, or a new category of things not to eat-- the Bay Area would be the place to do it. But a place that tolerates oddness in the search for the new is exactly what you want in a startup hub, because economically that's what startups are. Most good startup ideas seem a little crazy; if they were obviously good ideas, someone would have done them already.

(How many people are going to want computers in their *houses*? What, *another* search engine?)

That's the connection between technology and liberalism. Without exception the high-tech cities in the US are also the most liberal. But it's not because liberals are smarter that this is

so. It's because liberal cities tolerate odd ideas, and smart people by definition have odd ideas.

Conversely, a town that gets praised for being "solid" or representing "traditional values" may be a fine place to live, but it's never going to succeed as a startup hub. The 2004 presidential election, though a disaster in other respects, conveniently supplied us with a county-by-county [map](#) of such places. [6]

To attract the young, a town must have an intact center. In most American cities the center has been abandoned, and the growth, if any, is in the suburbs. Most American cities have been turned inside out. But none of the startup hubs has: not San Francisco, or Boston, or Seattle. They all have intact centers. [7] My guess is that no city with a dead center could be turned into a startup hub. Young people don't want to live in the suburbs.

Within the US, the two cities I think could most easily be turned into new silicon valleys are Boulder and Portland. Both have the kind of effervescent feel that attracts the young. They're each only a great university short of becoming a silicon valley, if they wanted to.

Time

A great university near an attractive town. Is that all it takes? That was all it took to make the original Silicon Valley. Silicon Valley traces its origins to William Shockley, one of the inventors of the transistor. He did the research that won him the Nobel Prize at Bell Labs, but when he started his own company in 1956 he moved to Palo Alto to do it. At the time that was an odd thing to do. Why did he? Because he had grown up there and remembered how nice it was. Now Palo Alto is suburbia, but then it was a charming college town-- a charming college town with perfect weather and San Francisco only an hour away.

The companies that rule Silicon Valley now are all descended in various ways from Shockley Semiconductor. Shockley was a difficult man, and in 1957 his top people-- "the traitorous eight"-- left to start a new company, Fairchild Semiconductor. Among them were Gordon Moore and Robert Noyce, who went on to found Intel, and Eugene Kleiner, who founded the VC firm Kleiner Perkins. Forty-two years later, Kleiner Perkins funded Google, and the partner responsible for the deal was John Doerr, who came to Silicon Valley in 1974 to work for Intel.

So although a lot of the newest companies in Silicon Valley don't

make anything out of silicon, there always seem to be multiple links back to Shockley. There's a lesson here: startups beget startups. People who work for startups start their own. People who get rich from startups fund new ones. I suspect this kind of organic growth is the only way to produce a startup hub, because it's the only way to grow the expertise you need.

That has two important implications. The first is that you need time to grow a silicon valley. The university you could create in a couple years, but the startup community around it has to grow organically. The cycle time is limited by the time it takes a company to succeed, which probably averages about five years.

The other implication of the organic growth hypothesis is that you can't be somewhat of a startup hub. You either have a self-sustaining chain reaction, or not. Observation confirms this too: cities either have a startup scene, or they don't. There is no middle ground. Chicago has the third largest metropolitan area in America. As a source of startups it's negligible compared to Seattle, number 15.

The good news is that the initial seed can be quite small. Shockley Semiconductor, though itself not very successful, was big enough. It brought a critical mass of experts in an important new technology together in a place they liked enough to stay.

Competing

Of course, a would-be silicon valley faces an obstacle the original one didn't: it has to compete with Silicon Valley. Can that be done? Probably.

One of Silicon Valley's biggest advantages is its venture capital firms. This was not a factor in Shockley's day, because VC funds didn't exist. In fact, Shockley Semiconductor and Fairchild Semiconductor were not startups at all in our sense. They were subsidiaries-- of Beckman Instruments and Fairchild Camera and Instrument respectively. Those companies were apparently willing to establish subsidiaries wherever the experts wanted to live.

Venture investors, however, prefer to fund startups within an hour's drive. For one, they're more likely to notice startups nearby. But when they do notice startups in other towns they prefer them to move. They don't want to have to travel to attend board meetings, and in any case the odds of succeeding are higher in a startup hub.

The centralizing effect of venture firms is a double one: they

cause startups to form around them, and those draw in more startups through acquisitions. And although the first may be weakening because it's now so cheap to start some startups, the second seems as strong as ever. Three of the most admired "Web 2.0" companies were started outside the usual startup hubs, but two of them have already been reeled in through acquisitions.

Such centralizing forces make it harder for new silicon valleys to get started. But by no means impossible. Ultimately power rests with the founders. A startup with the best people will beat one with funding from famous VCs, and a startup that was sufficiently successful would never have to move. So a town that could exert enough pull over the right people could resist and perhaps even surpass Silicon Valley.

For all its power, Silicon Valley has a great weakness: the paradise Shockley found in 1956 is now one giant parking lot. San Francisco and Berkeley are great, but they're forty miles away. Silicon Valley proper is soul-crushing suburban [sprawl](#). It has fabulous weather, which makes it significantly better than the soul-crushing sprawl of most other American cities. But a competitor that managed to avoid sprawl would have real leverage. All a city needs is to be the kind of place the next traitorous eight look at and say "I want to stay here," and that would be enough to get the chain reaction started.

Notes

[1] It's interesting to consider how low this number could be made. I suspect five hundred would be enough, even if they could bring no assets with them. Probably just thirty, if I could pick them, would be enough to turn Buffalo into a significant startup hub.

[2] Bureaucrats manage to allocate research funding moderately well, but only because (like an in-house VC fund) they outsource most of the work of selection. A professor at a famous university who is highly regarded by his peers will get funding, pretty much regardless of the proposal. That wouldn't work for startups, whose founders aren't sponsored by organizations, and are often unknowns.

[3] You'd have to do it all at once, or at least a whole department at a time, because people would be more likely to

come if they knew their friends were. And you should probably start from scratch, rather than trying to upgrade an existing university, or much energy would be lost in friction.

[4] Hypothesis: Any plan in which multiple independent buildings are gutted or demolished to be "redeveloped" as a single project is a net loss of personality for the city, with the exception of the conversion of buildings not previously public, like warehouses.

[5] A few startups get started in New York, but less than a tenth as many per capita as in Boston, and mostly in less nerdy fields like finance and media.

[6] Some blue counties are false positives (reflecting the remaining power of Democratic party machines), but there are no false negatives. You can safely write off all the red counties.

[7] Some "urban renewal" experts took a shot at destroying Boston's in the 1960s, leaving the area around city hall a bleak [wasteland](#), but most neighborhoods successfully resisted them.

Thanks to Chris Anderson, Trevor Blackwell, Marc Hedlund, Jessica Livingston, Robert Morris, Greg Mcadoo, Fred Wilson, and Stephen Wolfram for reading drafts of this, and to Ed Dumbill for inviting me to speak.

(The second part of this talk became [Why Startups Condense in America](#).)

PAUL GRAHAM

A STUDENT'S GUIDE TO STARTUPS

Want to start a startup? Get funded by [Y Combinator](#).

October 2006

(This essay is derived from a talk at MIT.)

Till recently graduating seniors had two choices: get a job or go to grad school. I think there will increasingly be a third option: to start your own startup. But how common will that be?

I'm sure the default will always be to get a job, but starting a startup could well become as popular as grad school. In the late 90s my professor friends used to complain that they couldn't get grad students, because all the undergrads were going to

work for startups. I wouldn't be surprised if that situation returns, but with one difference: this time they'll be starting their own instead of going to work for other people's.

The most ambitious students will at this point be asking: Why wait till you graduate? Why not start a startup while you're in college? In fact, why go to college at all? Why not start a startup instead?

A year and a half ago I gave a [talk](#) where I said that the average age of the founders of Yahoo, Google, and Microsoft was 24, and that if grad students could start startups, why not undergrads? I'm glad I phrased that as a question, because now I can pretend it wasn't merely a rhetorical one. At the time I couldn't imagine why there should be any lower limit for the age of startup founders. Graduation is a bureaucratic change, not a biological one. And certainly there are undergrads as competent technically as most grad students. So why shouldn't undergrads be able to start startups as well as grad students?

I now realize that something does change at graduation: you lose a huge excuse for failing. Regardless of how complex your life is, you'll find that everyone else, including your family and friends, will discard all the low bits and regard you as having a single occupation at any given time. If you're in college and have a summer job writing software, you still read as a student. Whereas if you graduate and get a job programming, you'll be instantly regarded by everyone as a programmer.

The problem with starting a startup while you're still in school is that there's a built-in escape hatch. If you start a startup in the summer between your junior and senior year, it reads to everyone as a summer job. So if it goes nowhere, big deal; you return to school in the fall with all the other seniors; no one regards you as a failure, because your occupation is student, and you didn't fail at that. Whereas if you start a startup just one year later, after you graduate, as long as you're not accepted to grad school in the fall the startup reads to everyone as your occupation. You're now a startup founder, so you have to do well at that.

For nearly everyone, the opinion of one's peers is the most powerful motivator of all—more powerful even than the nominal goal of most startup founders, getting rich. [\[1\]](#) About a month into each funding cycle we have an event called Prototype Day where each startup presents to the others what they've got so far. You might think they wouldn't need any more motivation. They're working on their cool new idea; they have funding for the immediate future; and they're playing a game with only two outcomes: wealth or failure. You'd think that would be motivation enough. And yet the prospect of a demo pushes most of them into a rush of activity.

Even if you start a startup explicitly to get rich, the money you might get seems pretty theoretical most of the time. What drives you day to day is not wanting to look bad.

You probably can't change that. Even if you could, I don't think you'd want to; someone who really, truly doesn't care what his peers think of him is probably a psychopath. So the best you can do is consider this force like a wind, and set up your boat accordingly. If you know your peers are going to push you in some direction, choose good peers, and position yourself so they push you in a direction you like.

Graduation changes the prevailing winds, and those make a difference. Starting a startup is so hard that it's a close call even for the ones that succeed. However high a startup may be flying now, it probably has a few leaves stuck in the landing gear from those trees it barely cleared at the end of the runway. In such a close game, the smallest increase in the forces against you can be enough to flick you over the edge into failure.

When we first started [Y Combinator](#) we encouraged people to start startups while they were still in college. That's partly because Y Combinator began as a kind of summer program. We've kept the program shape—all of us having dinner together once a week turns out to be a good idea—but we've decided now that the party line should be to tell people to wait till they graduate.

Does that mean you can't start a startup in college? Not at all. Sam Altman, the co-founder of [Loopt](#), had just finished his sophomore year when we funded them, and Loopt is probably the most promising of all the startups we've funded so far. But Sam Altman is a very unusual guy. Within about three minutes of meeting him, I remember thinking "Ah, so this is what Bill Gates must have been like when he was 19."

If it can work to start a startup during college, why do we tell people not to? For the same reason that the probably apocryphal violinist, whenever he was asked to judge someone's playing, would always say they didn't have enough talent to make it as a pro. Succeeding as a musician takes determination as well as talent, so this answer works out to be the right advice for everyone. The ones who are uncertain believe it and give up, and the ones who are sufficiently determined think "screw that, I'll succeed anyway."

So our official policy now is only to fund undergrads we can't talk out of it. And frankly, if you're not certain, you *should* wait. It's not as if all the opportunities to start companies are going to be gone if you don't do it now. Maybe the window will close on some idea you're working on, but that won't be the last idea you'll have. For every idea that times out, new ones become feasible. Historically the opportunities to start startups have only increased with time.

In that case, you might ask, why not wait longer? Why not go work for a while, or go to grad school, and then start a startup? And indeed, that might be a good idea. If I had to pick the sweet spot for startup founders, based on who we're most excited to see applications from, I'd say it's probably the mid-

twenties. Why? What advantages does someone in their mid-twenties have over someone who's 21? And why isn't it older? What can 25 year olds do that 32 year olds can't? Those turn out to be questions worth examining.

Plus

If 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 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.

Minus

So 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.

Now

So suppose you think you might start a startup at some point, either when you graduate or a few years after. What should you do now? For both jobs and grad school, there are ways to prepare while you're in college. If you want to get a job when you graduate, you should get summer jobs at places you'd like to work. If you want to go to grad school, it will help to work on research projects as an undergrad. What's the equivalent for startups? How do you keep your options maximally open?

One thing you can do while you're still in school is to learn how startups work. Unfortunately that's not easy. Few if any colleges have classes about startups. There may be business school classes on entrepreneurship, as they call it over there, but these are likely to be a waste of time. Business schools like to talk about startups, but philosophically they're at the opposite end of the spectrum. Most books on startups also seem to be useless. I've looked at a few and none get it right. Books in most fields are written by people who know the subject from experience, but for startups there's a unique problem: by definition the founders of successful startups don't need to write

books to make money. As a result most books on the subject end up being written by people who don't understand it.

So I'd be skeptical of classes and books. The way to learn about startups is by watching them in action, preferably by working at one. How do you do that as an undergrad? Probably by sneaking in through the back door. Just hang around a lot and gradually start doing things for them. Most startups are (or should be) very cautious about hiring. Every hire increases the burn rate, and bad hires early on are hard to recover from. However, startups usually have a fairly informal atmosphere, and there's always a lot that needs to be done. If you just start doing stuff for them, many will be too busy to shoo you away. You can thus gradually work your way into their confidence, and maybe turn it into an official job later, or not, whichever you prefer. This won't work for all startups, but it would work for most I've known.

Number two, make the most of the great advantage of school: the wealth of co-founders. Look at the people around you and ask yourself which you'd like to work with. When you apply that test, you may find you get surprising results. You may find you'd prefer the quiet guy you've mostly ignored to someone who seems impressive but has an attitude to match. I'm not suggesting you suck up to people you don't really like because you think one day they'll be successful. Exactly the opposite, in fact: you should only start a startup with someone you like, because a startup will put your friendship through a stress test. I'm just saying you should think about who you really admire and hang out with them, instead of whoever circumstances throw you together with.

Another thing you can do is learn skills that will be useful to you in a startup. These may be different from the skills you'd learn to get a job. For example, thinking about getting a job will make you want to learn programming languages you think employers want, like Java and C++. Whereas if you start a startup, you get to pick the language, so you have to think about which will actually let you get the most done. If you use that test you might end up learning Ruby or Python instead.

But the most important skill for a startup founder isn't a programming technique. It's a knack for understanding users and figuring out how to give them what they want. I know I repeat this, but that's because it's so important. And it's a skill you can learn, though perhaps habit might be a better word. Get into the habit of thinking of software as having users. What do those users want? What would make them say wow?

This is particularly valuable for undergrads, because the concept of users is missing from most college programming classes. The way you get taught programming in college would be like teaching writing as grammar, without mentioning that its purpose is to communicate something to an audience. Fortunately an audience for software is now only an http request away. So in addition to the programming you do for your classes, why not build some kind of website people will find useful? At the very least it will teach you how to write software with users. In the best case, it might not just be preparation for a startup, but the startup itself, like it was for Yahoo and Google.

Notes

[1] Even the desire to protect one's children seems weaker, judging from things people have historically done to their kids rather than risk their community's disapproval. (I assume we still do things that will be regarded in the future as barbaric, but historical abuses are easier for us to see.)

[2] Worrying that Y Combinator makes founders move for 3 months also suggests one underestimates how hard it is to start a startup. You're going to have to put up with much greater inconveniences than that.

[3] Most employee agreements say that any idea relating to the company's present or potential future business belongs to them. Often as not the second clause could include any possible startup, and anyone doing due diligence for an investor or acquirer will assume the worst.

To be safe either (a) don't use code written while you were still employed in your previous job, or (b) get your employer to renounce, in writing, any claim to the code you write for your side project. Many will consent to (b) rather than lose a prized employee. The downside is that you'll have to tell them exactly what your project does.

[4] Geshke and Warnock only founded Adobe because Xerox ignored them. If Xerox had used what they built, they would probably never have left PARC.

Thanks to Jessica Livingston and Robert Morris for reading drafts of this, and to Jeff Arnold and the SIPB for inviting me to speak.



[Comment](#) on this essay.

■ [Chinese Translation](#)

■ [Arabic Translation](#)

PAUL GRAHAM

THE 18 MISTAKES THAT KILL STARTUPS

Want to start a startup? Get funded by [Y Combinator](#).

October 2006

In 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 Founder

Have 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 Location

Startups 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 Niche

Most 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 Idea

Many 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. Obstinance

In 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 Programmers

I 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 lurches 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 Platform

A 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 Launching

Companies 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 Early

Launching 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 Mind

You 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 Money

Most 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 Much

It'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 Money

It'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 Management

As 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) Profit

When 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 Dirty

Nearly 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 Founders

Fights 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 Effort

The 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.

PAUL GRAHAM

WHY TO NOT NOT START A STARTUP

Want to start a startup? Get funded by [Y Combinator](#).

March 2007

(This essay is derived from talks at the 2007 Startup School and the Berkeley CSUA.)

We've now been doing Y Combinator long enough to have some data about success rates. Our first batch, in the summer of 2005, had eight startups in it. Of those eight, it now looks as if at least four succeeded. Three have been acquired: [Reddit](#) was a merger of two, Reddit and Infogami, and a third was acquired that we can't talk about yet. Another from that batch was [Loopt](#), which is doing so well they could probably be acquired in about ten minutes if they wanted to.

So about half the founders from that first summer, less than

two years ago, are now rich, at least by their standards. (One thing you learn when you get rich is that there are many degrees of it.)

I'm not ready to predict our success rate will stay as high as 50%. That first batch could have been an anomaly. But we should be able to do better than the oft-quoted (and probably made up) standard figure of 10%. I'd feel safe aiming at 25%.

Even the founders who fail don't seem to have such a bad time. Of those first eight startups, three are now probably dead. In two cases the founders just went on to do other things at the end of the summer. I don't think they were traumatized by the experience. The closest to a traumatic failure was Kiko, whose founders kept working on their startup for a whole year before being squashed by Google Calendar. But they ended up happy. They sold their software on eBay for a quarter of a million dollars. After they paid back their angel investors, they had about a year's salary each. [1] Then they immediately went on to start a new and much more exciting startup, [Justin.TV](#).

So here is an even more striking statistic: 0% of that first batch had a terrible experience. They had ups and downs, like every startup, but I don't think any would have traded it for a job in a cubicle. And that statistic is probably not an anomaly. Whatever our long-term success rate ends up being, I think the rate of people who wish they'd gotten a regular job will stay close to 0%.

The big mystery to me is: why don't more people start startups? If nearly everyone who does it prefers it to a regular job, and a significant percentage get rich, why doesn't everyone want to do this? A lot of people think we get thousands of applications for each funding cycle. In fact we usually only get several hundred. Why don't more people apply? And while it must seem to anyone watching this world that startups are popping up like crazy, the number is small compared to the number of people with the necessary skills. The great majority of programmers still go straight from college to cubicle, and stay there.

It seems like people are not acting in their own interest. What's going on? Well, I can answer that. Because of Y Combinator's position at the very start of the venture funding process, we're probably the world's leading experts on the psychology of people who aren't sure if they want to start a company.

There's nothing wrong with being unsure. If you're a hacker thinking about starting a startup and hesitating before taking the leap, you're part of a grand tradition. Larry and Sergey seem to have felt the same before they started Google, and so did Jerry and Filo before they started Yahoo. In fact, I'd guess the most successful startups are the ones started by uncertain hackers rather than gung-ho business guys.

We have some evidence to support this. Several of the most successful startups we've funded told us later that they only

decided to apply at the last moment. Some decided only hours before the deadline.

The way to deal with uncertainty is to analyze it into components. Most people who are reluctant to do something have about eight different reasons mixed together in their heads, and don't know themselves which are biggest. Some will be justified and some bogus, but unless you know the relative proportion of each, you don't know whether your overall uncertainty is mostly justified or mostly bogus.

So I'm going to list all the components of people's reluctance to start startups, and explain which are real. Then would-be founders can use this as a checklist to examine their own feelings.

I admit my goal is to increase your self-confidence. But there are two things different here from the usual confidence-building exercise. One is that I'm motivated to be honest. Most people in the confidence-building business have already achieved their goal when you buy the book or pay to attend the seminar where they tell you how great you are. Whereas if I encourage people to start startups who shouldn't, I make my own life worse. If I encourage too many people to apply to Y Combinator, it just means more work for me, because I have to read all the applications.

The other thing that's going to be different is my approach. Instead of being positive, I'm going to be negative. Instead of telling you "come on, you can do it" I'm going to consider all the reasons you aren't doing it, and show why most (but not all) should be ignored. We'll start with the one everyone's born with.

1. Too young

A 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 inexperienced

I 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 enough

You 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 enough

You 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 business

This 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 cofounder

Not 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 idea

In 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 startups

A 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 support

This 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 wealthy

This 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 commitment

This 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 structure

I'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 uncertainty

Perhaps 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 avoiding

One 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 doctor

A 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 default

This 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.