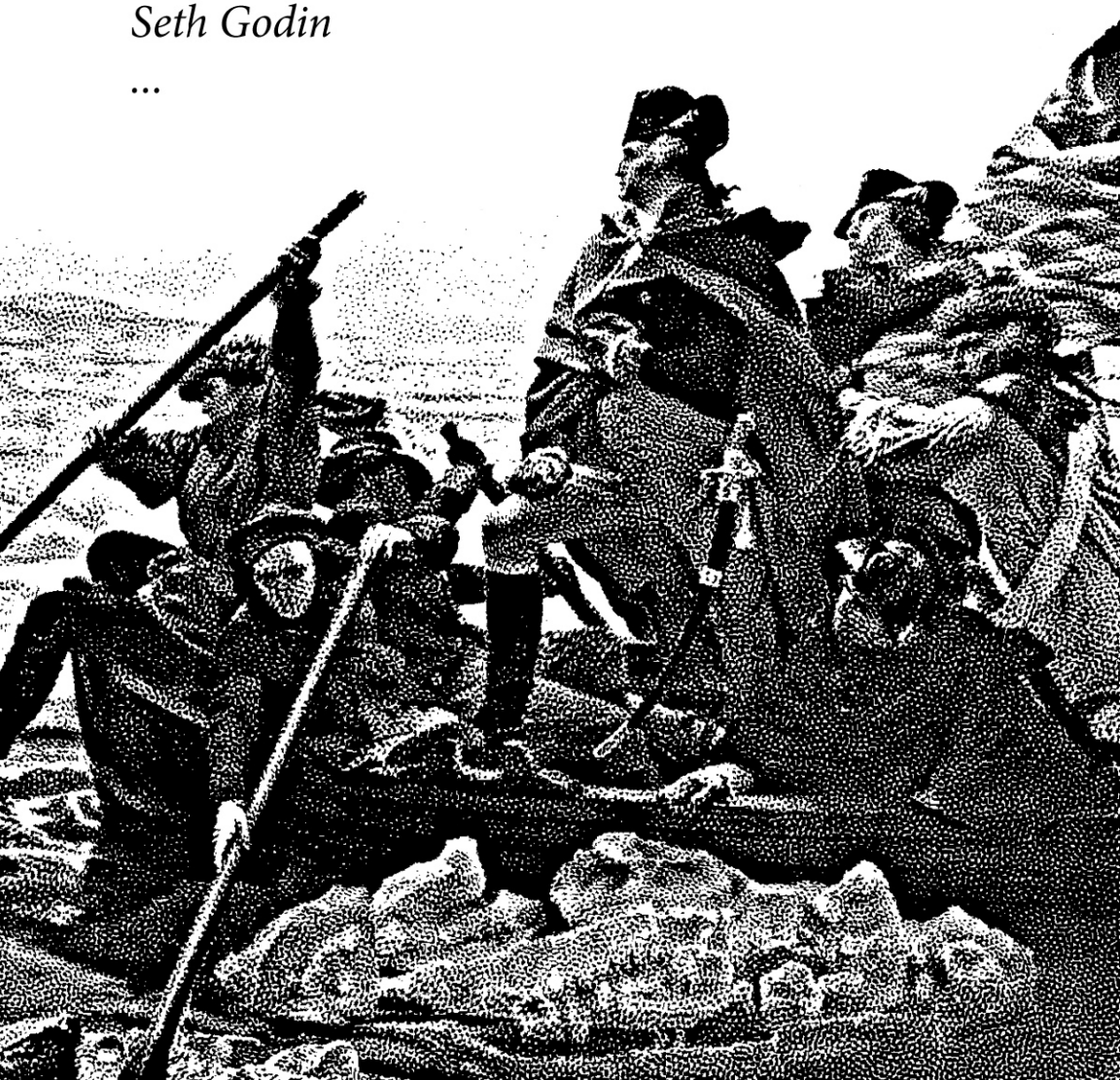
On Startups v2

Essays by
Paul Graham
Joel Spolsky
Seth Godin
...



Compiled and created by
Max Olson

TABLE OF CONTENTS

Part III

Part IV

Part I

Hackers and Painters

BY PAUL GRAHAM
MAY 2003

When I finished grad school in computer science I went to art school to study painting. A lot of people seemed surprised that someone interested in computers would also be interested in painting. They seemed to think that hacking and painting were very different kinds of work—that hacking was cold, precise, and methodical, and that painting was the frenzied expression of some primal urge.

Both of these images are wrong. Hacking and painting have a lot in common. In fact, of all the different types of people I've known, hackers and painters are among the most alike.

What hackers and painters have in common is that they're both makers. Along with composers, architects, and writers, what hackers and painters are trying to do is make good things. They're not doing research per se, though if in the course of trying to make good things they discover some new technique, so much the better.

I've never liked the term "computer science." The main reason I don't like it is that there's no such thing. Computer science is a grab bag of tenuously related areas thrown together by an accident of history, like Yugoslavia. At one end you have people who are really mathematicians, but call what they're doing computer science so they can get

DARPA grants. In the middle you have people working on something like the natural history of computers—studying the behavior of algorithms for routing data through networks, for example. And then at the other extreme you have the hackers, who are trying to write interesting software, and for whom computers are just a medium of expression, as concrete is for architects or paint for painters. It's as if mathematicians, physicists, and architects all had to be in the same department.

Sometimes what the hackers do is called "software engineering," but this term is just as misleading. Good software designers are no more engineers than architects are. The border between architecture and engineering is not sharply defined, but it's there. It falls between what and how: architects decide what to do, and engineers figure out how to do it.

What and how should not be kept too separate. You're asking for trouble if you try to decide what to do without understanding how to do it. But hacking can certainly be more than just deciding how to implement some spec. At its best, it's creating the spec—though it turns out the best way to do that is to implement it.

Perhaps one day "computer science" will, like Yugoslavia, get broken up into its component parts. That might be a good thing. Especially if it meant independence for my native land, hacking.

Bundling all these different types of work together in one department may be convenient administratively, but it's confusing intellectually. That's the other reason I don't like the name "computer science." Arguably the people in the middle are doing something like an experimental science. But the people at either end, the hackers and the mathematicians, are not actually doing science.

The mathematicians don't seem bothered by this. They happily set to work proving theorems like the other mathematicians over in the math department, and probably soon stop noticing that the building they work in says "computer science" on the outside. But for the hackers this label is a problem. If what they're doing is called science, it makes them feel they ought to be acting scientific. So instead

of doing what they really want to do, which is to design beautiful software, hackers in universities and research labs feel they ought to be writing research papers.

In the best case, the papers are just a formality. Hackers write cool software, and then write a paper about it, and the paper becomes a proxy for the achievement represented by the software. But often this mismatch causes problems. It's easy to drift away from building beautiful things toward building ugly things that make more suitable subjects for research papers.

Unfortunately, beautiful things don't always make the best subjects for papers. Number one, research must be original—and as anyone who has written a PhD dissertation knows, the way to be sure that you're exploring virgin territory is to stake out a piece of ground that no one wants. Number two, research must be substantial—and awkward systems yield meatier papers, because you can write about the obstacles you have to overcome in order to get things done. Nothing yields meaty problems like starting with the wrong assumptions. Most of AI is an example of this rule; if you assume that knowledge can be represented as a list of predicate logic expressions whose arguments represent abstract concepts, you'll have a lot of papers to write about how to make this work. As Ricky Ricardo used to say, "Lucy, you got a lot of explaining to do."

The way to create something beautiful is often to make subtle tweaks to something that already exists, or to combine existing ideas in a slightly new way. This kind of work is hard to convey in a research paper.

So why do universities and research labs continue to judge hackers by publications? For the same reason that "scholastic aptitude" gets measured by simple-minded standardized tests, or the productivity of programmers gets measured in lines of code. These tests are easy to apply, and there is nothing so tempting as an easy test that kind of works.

Measuring what hackers are actually trying to do, designing beautiful software, would be much more difficult. You need a good

sense of design to judge good design. And there is no correlation, except possibly a negative one, between people's ability to recognize good design and their confidence that they can.

The only external test is time. Over time, beautiful things tend to thrive, and ugly things tend to get discarded. Unfortunately, the amounts of time involved can be longer than human lifetimes. Samuel Johnson said it took a hundred years for a writer's reputation to converge. You have to wait for the writer's influential friends to die, and then for all their followers to die.

I think hackers just have to resign themselves to having a large random component in their reputations. In this they are no different from other makers. In fact, they're lucky by comparison. The influence of fashion is not nearly so great in hacking as it is in painting.

There are worse things than having people misunderstand your work. A worse danger is that you will yourself misunderstand your work. Related fields are where you go looking for ideas. If you find yourself in the computer science department, there is a natural temptation to believe, for example, that hacking is the applied version of what theoretical computer science is the theory of. All the time I was in graduate school I had an uncomfortable feeling in the back of my mind that I ought to know more theory, and that it was very remiss of me to have forgotten all that stuff within three weeks of the final exam.

Now I realize I was mistaken. Hackers need to understand the theory of computation about as much as painters need to understand paint chemistry. You need to know how to calculate time and space complexity and about Turing completeness. You might also want to remember at least the concept of a state machine, in case you have to write a parser or a regular expression library. Painters in fact have to remember a good deal more about paint chemistry than that.

I've found that the best sources of ideas are not the other fields that have the word "computer" in their names, but the other fields inhabited by makers. Painting has been a much richer source of ideas than the theory of computation.

For example, I was taught in college that one ought to figure out a program completely on paper before even going near a computer. I found that I did not program this way. I found that I liked to program sitting in front of a computer, not a piece of paper. Worse still, instead of patiently writing out a complete program and assuring myself it was correct, I tended to just spew out code that was hopelessly broken, and gradually beat it into shape. Debugging, I was taught, was a kind of final pass where you caught typos and oversights. The way I worked, it seemed like programming consisted of debugging.

For a long time I felt bad about this, just as I once felt bad that I didn't hold my pencil the way they taught me to in elementary school. If I had only looked over at the other makers, the painters or the architects, I would have realized that there was a name for what I was doing: sketching. As far as I can tell, the way they taught me to program in college was all wrong. You should figure out programs as you're writing them, just as writers and painters and architects do.

Realizing this has real implications for software design. It means that a programming language should, above all, be malleable. A programming language is for thinking of programs, not for expressing programs you've already thought of. It should be a pencil, not a pen. Static typing would be a fine idea if people actually did write programs the way they taught me to in college. But that's not how any of the hackers I know write programs. We need a language that lets us scribble and smudge and smear, not a language where you have to sit with a teacup of types balanced on your knee and make polite conversation with a strict old aunt of a compiler.

While we're on the subject of static typing, identifying with the makers will save us from another problem that afflicts the sciences: math envy. Everyone in the sciences secretly believes that mathematicians are smarter than they are. I think mathematicians also believe this. At any rate, the result is that scientists tend to make their work look as mathematical as possible. In a field like physics this probably doesn't do much harm, but the further you get from the natural sci-

ences, the more of a problem it becomes.

A page of formulas just looks so impressive. (Tip: for extra impressiveness, use Greek variables.) And so there is a great temptation to work on problems you can treat formally, rather than problems that are, say, important.

If hackers identified with other makers, like writers and painters, they wouldn't feel tempted to do this. Writers and painters don't suffer from math envy. They feel as if they're doing something completely unrelated. So are hackers, I think.

If universities and research labs keep hackers from doing the kind of work they want to do, perhaps the place for them is in companies. Unfortunately, most companies won't let hackers do what they want either. Universities and research labs force hackers to be scientists, and companies force them to be engineers.

I only discovered this myself quite recently. When Yahoo bought Viaweb, they asked me what I wanted to do. I had never liked the business side very much, and said that I just wanted to hack. When I got to Yahoo, I found that what hacking meant to them was implementing software, not designing it. Programmers were seen as technicians who translated the visions (if that is the word) of product managers into code.

This seems to be the default plan in big companies. They do it because it decreases the standard deviation of the outcome. Only a small percentage of hackers can actually design software, and it's hard for the people running a company to pick these out. So instead of entrusting the future of the software to one brilliant hacker, most companies set things up so that it is designed by committee, and the hackers merely implement the design.

If you want to make money at some point, remember this, because this is one of the reasons startups win. Big companies want to decrease the standard deviation of design outcomes because they want to avoid disasters. But when you damp oscillations, you lose the high points as well as the low. This is not a problem for big companies, because they don't win by making great products. Big compa-

nies win by sucking less than other big companies.

So if you can figure out a way to get in a design war with a company big enough that its software is designed by product managers, they'll never be able to keep up with you. These opportunities are not easy to find, though. It's hard to engage a big company in a design war, just as it's hard to engage an opponent inside a castle in hand to hand combat. It would be pretty easy to write a better word processor than Microsoft Word, for example, but Microsoft, within the castle of their operating system monopoly, probably wouldn't even notice if you did.

The place to fight design wars is in new markets, where no one has yet managed to establish any fortifications. That's where you can win big by taking the bold approach to design, and having the same people both design and implement the product. Microsoft themselves did this at the start. So did Apple. And Hewlett-Packard. I suspect almost every successful startup has.

So one way to build great software is to start your own startup. There are two problems with this, though. One is that in a startup you have to do so much besides write software. At Viaweb I considered myself lucky if I got to hack a quarter of the time. And the things I had to do the other three quarters of the time ranged from tedious to terrifying. I have a benchmark for this, because I once had to leave a board meeting to have some cavities filled. I remember sitting back in the dentist's chair, waiting for the drill, and feeling like I was on vacation.

The other problem with startups is that there is not much overlap between the kind of software that makes money and the kind that's interesting to write. Programming languages are interesting to write, and Microsoft's first product was one, in fact, but no one will pay for programming languages now. If you want to make money, you tend to be forced to work on problems that are too nasty for anyone to solve for free.

All makers face this problem. Prices are determined by supply and demand, and there is just not as much demand for things that are fun to work on as there is for things that solve the mundane

problems of individual customers. Acting in off-Broadway plays just doesn't pay as well as wearing a gorilla suit in someone's booth at a trade show. Writing novels doesn't pay as well as writing ad copy for garbage disposals. And hacking programming languages doesn't pay as well as figuring out how to connect some company's legacy database to their Web server.

I think the answer to this problem, in the case of software, is a concept known to nearly all makers: the day job. This phrase began with musicians, who perform at night. More generally, it means that you have one kind of work you do for money, and another for love.

Nearly all makers have day jobs early in their careers. Painters and writers notoriously do. If you're lucky you can get a day job that's closely related to your real work. Musicians often seem to work in record stores. A hacker working on some programming language or operating system might likewise be able to get a day job using it.*

When I say that the answer is for hackers to have day jobs, and work on beautiful software on the side, I'm not proposing this as a new idea. This is what open-source hacking is all about. What I'm saying is that open-source is probably the right model, because it has been independently confirmed by all the other makers.

It seems surprising to me that any employer would be reluctant to let hackers work on open-source projects. At Viaweb, we would have been reluctant to hire anyone who didn't. When we interviewed programmers, the main thing we cared about was what kind of software they wrote in their spare time. You can't do anything really well unless you love it, and if you love to hack you'll inevitably be working on projects of your own.†

---

* The greatest damage that photography has done to painting may be the fact that it killed the best day job. Most of the great painters in history supported themselves by painting portraits.

† I've been told that Microsoft discourages employees from contributing to open-source projects, even in their spare time. But so many of the best hackers work on open-source projects now that the main effect of this policy may be to ensure that they won't be able to hire any first-rate programmers.

Because hackers are makers rather than scientists, the right place to look for metaphors is not in the sciences, but among other kinds of makers. What else can painting teach us about hacking?

One thing we can learn, or at least confirm, from the example of painting is how to learn to hack. You learn to paint mostly by doing it. Ditto for hacking. Most hackers don't learn to hack by taking college courses in programming. They learn to hack by writing programs of their own at age thirteen. Even in college classes, you learn to hack mostly by hacking.*

Because painters leave a trail of work behind them, you can watch them learn by doing. If you look at the work of a painter in chronological order, you'll find that each painting builds on things that have been learned in previous ones. When there's something in a painting that works very well, you can usually find version 1 of it in a smaller form in some earlier painting.

I think most makers work this way. Writers and architects seem to as well. Maybe it would be good for hackers to act more like painters, and regularly start over from scratch, instead of continuing to work for years on one project, and trying to incorporate all their later ideas as revisions.

The fact that hackers learn to hack by doing it is another sign of how different hacking is from the sciences. Scientists don't learn science by doing it, but by doing labs and problem sets. Scientists start out doing work that's perfect, in the sense that they're just trying to reproduce work someone else has already done for them. Eventually, they get to the point where they can do original work. Whereas hackers, from the start, are doing original work; it's just very bad. So hackers start original, and get good, and scientists start good, and get original.

The other way makers learn is from examples. For a painter, a museum is a reference library of techniques. For hundreds of years it has

* What you learn about programming in college is much like what you learn about books or clothes or dating: what bad taste you had in high school.

been part of the traditional education of painters to copy the works of the great masters, because copying forces you to look closely at the way a painting is made.

Writers do this too. Benjamin Franklin learned to write by summarizing the points in the essays of Addison and Steele and then trying to reproduce them. Raymond Chandler did the same thing with detective stories.

Hackers, likewise, can learn to program by looking at good programs—not just at what they do, but the source code too. One of the less publicized benefits of the open-source movement is that it has made it easier to learn to program. When I learned to program, we had to rely mostly on examples in books. The one big chunk of code available then was Unix, but even this was not open source. Most of the people who read the source read it in illicit photocopies of John Lions' book, which though written in 1977 was not allowed to be published until 1996.

Another example we can take from painting is the way that paintings are created by gradual refinement. Paintings usually begin with a sketch. Gradually the details get filled in. But it is not merely a process of filling in. Sometimes the original plans turn out to be mistaken. Countless paintings, when you look at them in xrays, turn out to have limbs that have been moved or facial features that have been readjusted.

Here's a case where we can learn from painting. I think hacking should work this way too. It's unrealistic to expect that the specifications for a program will be perfect. You're better off if you admit this up front, and write programs in a way that allows specifications to change on the fly.

(The structure of large companies makes this hard for them to do, so here is another place where startups have an advantage.)

Everyone by now presumably knows about the danger of premature optimization. I think we should be just as worried about premature design—deciding too early what a program should do.

The right tools can help us avoid this danger. A good program-

ming language should, like oil paint, make it easy to change your mind. Dynamic typing is a win here because you don't have to commit to specific data representations up front. But the key to flexibility, I think, is to make the language very abstract. The easiest program to change is one that's very short.

This sounds like a paradox, but a great painting has to be better than it has to be. For example, when Leonardo painted the portrait of Ginevra de Benci in the National Gallery, he put a juniper bush behind her head. In it he carefully painted each individual leaf. Many painters might have thought, this is just something to put in the background to frame her head. No one will look that closely at it.

Not Leonardo. How hard he worked on part of a painting didn't depend at all on how closely he expected anyone to look at it. He was like Michael Jordan. Relentless.

Relentlessness wins because, in the aggregate, unseen details become visible. When people walk by the portrait of Ginevra de Benci, their attention is often immediately arrested by it, even before they look at the label and notice that it says Leonardo da Vinci. All those unseen details combine to produce something that's just stunning, like a thousand barely audible voices all singing in tune.

Great software, likewise, requires a fanatical devotion to beauty. If you look inside good software, you find that parts no one is ever supposed to see are beautiful too. I'm not claiming I write great software, but I know that when it comes to code I behave in a way that would make me eligible for prescription drugs if I approached everyday life the same way. It drives me crazy to see code that's badly indented, or that uses ugly variable names.

If a hacker were a mere implementor, turning a spec into code, then he could just work his way through it from one end to the other like someone digging a ditch. But if the hacker is a creator, we have to take inspiration into account.

In hacking, like painting, work comes in cycles. Sometimes you get excited about some new project and you want to work sixteen

hours a day on it. Other times nothing seems interesting.

To do good work you have to take these cycles into account, because they're affected by how you react to them. When you're driving a car with a manual transmission on a hill, you have to back off the clutch sometimes to avoid stalling. Backing off can likewise prevent ambition from stalling. In both painting and hacking there are some tasks that are terrifyingly ambitious, and others that are comfortingly routine. It's a good idea to save some easy tasks for moments when you would otherwise stall.

In hacking, this can literally mean saving up bugs. I like debugging: it's the one time that hacking is as straightforward as people think it is. You have a totally constrained problem, and all you have to do is solve it. Your program is supposed to do x. Instead it does y. Where does it go wrong? You know you're going to win in the end. It's as relaxing as painting a wall.

The example of painting can teach us not only how to manage our own work, but how to work together. A lot of the great art of the past is the work of multiple hands, though there may only be one name on the wall next to it in the museum. Leonardo was an apprentice in the workshop of Verrocchio and painted one of the angels in his Baptism of Christ. This sort of thing was the rule, not the exception. Michelangelo was considered especially dedicated for insisting on painting all the figures on the ceiling of the Sistine Chapel himself.

As far as I know, when painters worked together on a painting, they never worked on the same parts. It was common for the master to paint the principal figures and for assistants to paint the others and the background. But you never had one guy painting over the work of another.

I think this is the right model for collaboration in software too. Don't push it too far. When a piece of code is being hacked by three or four different people, no one of whom really owns it, it will end up being like a common-room. It will tend to feel bleak and abandoned, and accumulate cruft. The right way to collaborate, I think, is to di-

vide projects into sharply defined modules, each with a definite owner, and with interfaces between them that are as carefully designed and, if possible, as articulated as programming languages.

Like painting, most software is intended for a human audience. And so hackers, like painters, must have empathy to do really great work. You have to be able to see things from the user's point of view.

When I was a kid I was always being told to look at things from someone else's point of view. What this always meant in practice was to do what someone else wanted, instead of what I wanted. This of course gave empathy a bad name, and I made a point of not cultivating it.

Boy, was I wrong. It turns out that looking at things from other people's point of view is practically the secret of success. It doesn't necessarily mean being self-sacrificing. Far from it. Understanding how someone else sees things doesn't imply that you'll act in his interest; in some situations—in war, for example—you want to do exactly the opposite.*

Most makers make things for a human audience. And to engage an audience you have to understand what they need. Nearly all the greatest paintings are paintings of people, for example, because people are what people are interested in.

Empathy is probably the single most important difference between a good hacker and a great one. Some hackers are quite smart, but when it comes to empathy are practically solipsists. It's hard for such people to design great software†, because they can't see things

* Here's an example of applied empathy. At Viaweb, if we couldn't decide between two alternatives, we'd ask, what would our competitors hate most? At one point a competitor added a feature to their software that was basically useless, but since it was one of few they had that we didn't, they made much of it in the trade press. We could have tried to explain that the feature was useless, but we decided it would annoy our competitor more if we just implemented it ourselves, so we hacked together our own version that afternoon.

† Except text editors and compilers. Hackers don't need empathy to design these, because they are themselves typical users.

from the user's point of view.

One way to tell how good people are at empathy is to watch them explain a technical question to someone without a technical background. We probably all know people who, though otherwise smart, are just comically bad at this. If someone asks them at a dinner party what a programming language is, they'll say something like ``Oh, a high-level language is what the compiler uses as input to generate object code." High-level language? Compiler? Object code? Someone who doesn't know what a programming language is obviously doesn't know what these things are, either.

Part of what software has to do is explain itself. So to write good software you have to understand how little users understand. They're going to walk up to the software with no preparation, and it had better do what they guess it will, because they're not going to read the manual. The best system I've ever seen in this respect was the original Macintosh, in 1985. It did what software almost never does: it just worked.*

Source code, too, should explain itself. If I could get people to remember just one quote about programming, it would be the one at the beginning of *Structure and Interpretation of Computer Programs.*

Programs should be written for people to read, and only incidentally for machines to execute.

You need to have empathy not just for your users, but for your readers. It's in your interest, because you'll be one of them. Many a hacker has written a program only to find on returning to it six months later that he has no idea how it works. I know several people who've sworn off Perl after such experiences.†

---

* Well, almost. They overshot the available RAM somewhat, causing much inconvenient disk swapping, but this could be fixed within a few months by buying an additional disk drive.

† The way to make programs easy to read is not to stuff them with comments. I would take Abelson and Sussman's quote a step further. Programming languages should be designed to express algorithms, and only incidentally to tell computers how to execute them. A good programming language ought to be better for explaining software than English. You should only need comments

Lack of empathy is associated with intelligence, to the point that there is even something of a fashion for it in some places. But I don't think there's any correlation. You can do well in math and the natural sciences without having to learn empathy, and people in these fields tend to be smart, so the two qualities have come to be associated. But there are plenty of dumb people who are bad at empathy too. Just listen to the people who call in with questions on talk shows. They ask whatever it is they're asking in such a roundabout way that the hosts often have to rephrase the question for them.

So, if hacking works like painting and writing, is it as cool? After all, you only get one life. You might as well spend it working on something great.

Unfortunately, the question is hard to answer. There is always a big time lag in prestige. It's like light from a distant star. Painting has prestige now because of great work people did five hundred years ago. At the time, no one thought these paintings were as important as we do today. It would have seemed very odd to people at the time that Federico da Montefeltro, the Duke of Urbino, would one day be known mostly as the guy with the strange nose in a painting by Piero della Francesca.

So while I admit that hacking doesn't seem as cool as painting now, we should remember that painting itself didn't seem as cool in its glory days as it does now.

What we can say with some confidence is that these are the glory days of hacking. In most fields the great work is done early on. The paintings made between 1430 and 1500 are still unsurpassed. Shakespeare appeared just as professional theater was being born, and pushed the medium so far that every playwright since has had to live in his shadow. Albrecht Durer did the same thing with engraving, and Jane Austen with the novel.

Over and over we see the same pattern. A new medium appears, and people are so excited about it that they explore most of its possi-

---

when there is some kind of kludge you need to warn readers about, just as on a road there are only arrows on parts with unexpectedly sharp curves.

bilities in the first couple generations. Hacking seems to be in this phase now.

Painting was not, in Leonardo's time, as cool as his work helped make it. How cool hacking turns out to be will depend on what we can do with this new medium.

How to Make Wealth

BY PAUL GRAHAM
MAY 2004

If you wanted to get rich, how would you do it? I think your best bet would be to start or join a startup. That's been a reliable way to get rich for hundreds of years. The word "startup" dates from the 1960s, but what happens in one is very similar to the venture-backed trading voyages of the Middle Ages.

Startups usually involve technology, so much so that the phrase "high-tech startup" is almost redundant. A startup is a small company that takes on a hard technical problem.

Lots of people get rich knowing nothing more than that. You don't have to know physics to be a good pitcher. But I think it could give you an edge to understand the underlying principles. Why do startups have to be small? Will a startup inevitably stop being a startup as it grows larger? And why do they so often work on developing new technology? Why are there so many startups selling new drugs or computer software, and none selling corn oil or laundry detergent?

The Proposition

Economically, you can think of a startup as a way to compress your whole working life into a few years. Instead of working at a low intensity for forty years, you work as hard as you possibly can for four. This pays especially well in technology, where you earn a premium

for working fast.

Here is a brief sketch of the economic proposition. If you're a good hacker in your mid-twenties, you can get a job paying about $80,000 per year. So on average such a hacker must be able to do at least $80,000 worth of work per year for the company just to break even. You could probably work twice as many hours as a corporate employee, and if you focus you can probably get three times as much done in an hour.* You should get another multiple of two, at least, by eliminating the drag of the pointy-haired middle manager who would be your boss in a big company. Then there is one more multiple: how much smarter are you than your job description expects you to be? Suppose another multiple of three. Combine all these multipliers, and I'm claiming you could be 36 times more productive than you're expected to be in a random corporate job.† If a fairly

---

* One valuable thing you tend to get only in startups is *uninterruptability*. Different kinds of work have different time quanta. Someone proofreading a manuscript could probably be interrupted every fifteen minutes with little loss of productivity. But the time quantum for hacking is very long: it might take an hour just to load a problem into your head. So the cost of having someone from personnel call you about a form you forgot to fill out can be huge.

This is why hackers give you such a baleful stare as they turn from their screen to answer your question. Inside their heads a giant house of cards is tottering.

The mere possibility of being interrupted deters hackers from starting hard projects. This is why they tend to work late at night, and why it's next to impossible to write great software in a cubicle (except late at night).

One great advantage of startups is that they don't yet have any of the people who interrupt you. There is no personnel department, and thus no form nor anyone to call you about it.

† Faced with the idea that people working for startups might be 20 or 30 times as productive as those working for large companies, executives at large companies will naturally wonder, how could I get the people working for me to do that? The answer is simple: pay them to.

Internally most companies are run like Communist states. If you believe in free markets, why not turn your company into one?

Hypothesis: A company will be maximally profitable when each employee is paid in proportion to the wealth they generate.

good hacker is worth $80,000 a year at a big company, then a smart hacker working very hard without any corporate bullshit to slow him down should be able to do work worth about $3 million a year.

Like all back-of-the-envelope calculations, this one has a lot of wiggle room. I wouldn't try to defend the actual numbers. But I stand by the structure of the calculation. I'm not claiming the multiplier is precisely 36, but it is certainly more than 10, and probably rarely as high as 100.

If $3 million a year seems high, remember that we're talking about the limit case: the case where you not only have zero leisure time but indeed work so hard that you endanger your health.

Startups are not magic. They don't change the laws of wealth creation. They just represent a point at the far end of the curve. There is a conservation law at work here: if you want to make a million dollars, you have to endure a million dollars' worth of pain. For example, one way to make a million dollars would be to work for the Post Office your whole life, and save every penny of your salary. Imagine the stress of working for the Post Office for fifty years. In a startup you compress all this stress into three or four years. You do tend to get a certain bulk discount if you buy the economy-size pain, but you can't evade the fundamental conservation law. If starting a startup were easy, everyone would do it.

Millions, not Billions

If $3 million a year seems high to some people, it will seem low to others. Three *million?* How do I get to be a billionaire, like Bill Gates?

So let's get Bill Gates out of the way right now. It's not a good idea to use famous rich people as examples, because the press only write about the very richest, and these tend to be outliers. Bill Gates is a smart, determined, and hardworking man, but you need more than that to make as much money as he has. You also need to be very lucky.

There is a large random factor in the success of any company. So the guys you end up reading about in the papers are the ones who are very smart, totally dedicated, *and* win the lottery. Certainly Bill is

smart and dedicated, but Microsoft also happens to have been the beneficiary of one of the most spectacular blunders in the history of business: the licensing deal for DOS. No doubt Bill did everything he could to steer IBM into making that blunder, and he has done an excellent job of exploiting it, but if there had been one person with a brain on IBM's side, Microsoft's future would have been very different. Microsoft at that stage had little leverage over IBM. They were effectively a component supplier. If IBM had required an exclusive license, as they should have, Microsoft would still have signed the deal. It would still have meant a lot of money for them, and IBM could easily have gotten an operating system elsewhere.

Instead IBM ended up using all its power in the market to give Microsoft control of the PC standard. From that point, all Microsoft had to do was execute. They never had to bet the company on a bold decision. All they had to do was play hardball with licensees and copy more innovative products reasonably promptly.

If IBM hadn't made this mistake, Microsoft would still have been a successful company, but it could not have grown so big so fast. Bill Gates would be rich, but he'd be somewhere near the bottom of the Forbes 400 with the other guys his age.

There are a lot of ways to get rich, and this essay is about only one of them. This essay is about how to make money by creating wealth and getting paid for it. There are plenty of other ways to get money, including chance, speculation, marriage, inheritance, theft, extortion, fraud, monopoly, graft, lobbying, counterfeiting, and prospecting. Most of the greatest fortunes have probably involved several of these.

The advantage of creating wealth, as a way to get rich, is not just that it's more legitimate (many of the other methods are now illegal) but that it's more *straightforward*. You just have to do something people want.

Money Is Not Wealth

If you want to create wealth, it will help to understand what it is.

Wealth is not the same thing as money.* Wealth is as old as human history. Far older, in fact; ants have wealth. Money is a comparatively recent invention.

Wealth is the fundamental thing. Wealth is stuff we want: food, clothes, houses, cars, gadgets, travel to interesting places, and so on. You can have wealth without having money. If you had a magic machine that could on command make you a car or cook you dinner or do your laundry, or do anything else you wanted, you wouldn't need money. Whereas if you were in the middle of Antarctica, where there is nothing to buy, it wouldn't matter how much money you had.

Wealth is what you want, not money. But if wealth is the important thing, why does everyone talk about making money? It is a kind of shorthand: money is a way of moving wealth, and in practice they are usually interchangeable. But they are not the same thing, and unless you plan to get rich by counterfeiting, talking about *making money* can make it harder to understand how to make money.

Money is a side effect of specialization. In a specialized society, most of the things you need, you can't make for yourself. If you want a potato or a pencil or a place to live, you have to get it from someone else.

How do you get the person who grows the potatoes to give you some? By giving him something he wants in return. But you can't get very far by trading things directly with the people who need them. If you make violins, and none of the local farmers wants one, how will you eat?

The solution societies find, as they get more specialized, is to make the trade into a two-step process. Instead of trading violins directly for potatoes, you trade violins for, say, silver, which you can

* Until recently even governments sometimes didn't grasp the distinction between money and wealth. Adam Smith (*Wealth of Nations*, v:i) mentions several that tried to preserve their "wealth" by forbidding the export of gold or silver. But having more of the medium of exchange would not make a country richer; if you have more money chasing the same amount of material wealth, the only result is higher prices.

then trade again for anything else you need. The intermediate stuff—the *medium of exchange*—can be anything that's rare and portable. Historically metals have been the most common, but recently we've been using a medium of exchange, called the *dollar*, that doesn't physically exist. It works as a medium of exchange, however, because its rarity is guaranteed by the U.S. Government.

The advantage of a medium of exchange is that it makes trade work. The disadvantage is that it tends to obscure what trade really means. People think that what a business does is make money. But money is just the intermediate stage—just a shorthand—for whatever people want. What most businesses really do is make wealth. They do something people want.*

The Pie Fallacy

A surprising number of people retain from childhood the idea that there is a fixed amount of wealth in the world. There is, in any normal family, a fixed amount of *money* at any moment. But that's not the same thing.

When wealth is talked about in this context, it is often described

---

* There are many senses of the word "wealth," not all of them material. I'm not trying to make a deep philosophical point here about which is the true kind. I'm writing about one specific, rather technical sense of the word "wealth." What people will give you money for. This is an interesting sort of wealth to study, because it is the kind that prevents you from starving. And what people will give you money for depends on them, not you.

When you're starting a business, it's easy to slide into thinking that customers want what you do. During the Internet Bubble I talked to a woman who, because she liked the outdoors, was starting an "outdoor portal." You know what kind of business you should start if you like the outdoors? One to recover data from crashed hard disks.

What's the connection? None at all. Which is precisely my point. If you want to create wealth (in the narrow technical sense of not starving) then you should be especially skeptical about any plan that centers on things you like doing. That is where your idea of what's valuable is least likely to coincide with other people's.

as a pie. "You can't make the pie larger," say politicians. When you're talking about the amount of money in one family's bank account, or the amount available to a government from one year's tax revenue, this is true. If one person gets more, someone else has to get less.

I can remember believing, as a child, that if a few rich people had all the money, it left less for everyone else. Many people seem to continue to believe something like this well into adulthood. This fallacy is usually there in the background when you hear someone talking about how *x* percent of the population have *y* percent of the wealth. If you plan to start a startup, then whether you realize it or not, you're planning to disprove the Pie Fallacy.

What leads people astray here is the abstraction of money. Money is not wealth. It's just something we use to move wealth around. So although there may be, in certain specific moments (like your family, this month) a fixed amount of money available to trade with other people for things you want, there is not a fixed amount of wealth in the world. *You can make more wealth.* Wealth has been getting created and destroyed (but on balance, created) for all of human history.

Suppose you own a beat-up old car. Instead of sitting on your butt next summer, you could spend the time restoring your car to pristine condition. In doing so you create wealth. The world is—and you specifically are—one pristine old car the richer. And not just in some metaphorical way. If you sell your car, you'll get more for it.

In restoring your old car you have made yourself richer. You haven't made anyone else poorer. So there is obviously not a fixed pie. And in fact, when you look at it this way, you wonder why anyone would think there was.*

Kids know, without knowing they know, that they can create wealth. If you need to give someone a present and don't have any

* In the average car restoration you probably do make everyone else microscopically poorer, by doing a small amount of damage to the environment. While environmental costs should be taken into account, they don't make wealth a zero-sum game. For example, if you repair a machine that's broken because a part has come unscrewed, you create wealth with no environmental cost.

money, you make one. But kids are so bad at making things that they consider home-made presents to be a distinct, inferior, sort of thing to store-bought ones—a mere expression of the proverbial thought that counts. And indeed, the lumpy ashtrays we made for our parents did not have much of a resale market.

Craftsmen

The people most likely to grasp that wealth can be created are the ones who are good at making things, the craftsmen. Their hand-made objects become store-bought ones. But with the rise of industrialization there are fewer and fewer craftsmen. One of the biggest remaining groups is computer programmers.

A programmer can sit down in front of a computer and *create wealth*. A good piece of software is, in itself, a valuable thing. There is no manufacturing to confuse the issue. Those characters you type are a complete, finished product. If someone sat down and wrote a web browser that didn't suck (a fine idea, by the way), the world would be that much richer.*

Everyone in a company works together to create wealth, in the sense of making more things people want. Many of the employees (e.g. the people in the mailroom or the personnel department) work at one remove from the actual making of stuff. Not the programmers. They literally think the product, one line at a time. And so it's clearer to programmers that wealth is something that's made, rather than being distributed, like slices of a pie, by some imaginary Daddy.

It's also obvious to programmers that there are huge variations in the rate at which wealth is created. At Viaweb we had one programmer who was a sort of monster of productivity. I remember watching what he did one long day and estimating that he had added several hundred thousand dollars to the market value of the company. A great programmer, on a roll, could create a million dollars' worth of wealth in a couple weeks. A mediocre programmer over the same period will generate zero or even negative wealth (e.g. by introducing

* This essay was written before Firefox.

bugs).

This is why so many of the best programmers are libertarians. In our world, you sink or swim, and there are no excuses. When those far removed from the creation of wealth—undergraduates, reporters, politicians—hear that the richest 5% of the people have half the total wealth, they tend to think *injustice!* An experienced programmer would be more likely to think *is that all?* The top 5% of programmers probably write 99% of the good software.

Wealth can be created without being sold. Scientists, till recently at least, effectively donated the wealth they created. We are all richer for knowing about penicillin, because we're less likely to die from infections. Wealth is whatever people want, and not dying is certainly something we want. Hackers often donate their work by writing open source software that anyone can use for free. I am much the richer for the operating system FreeBSD, which I'm running on the computer I'm using now, and so is Yahoo, which runs it on all their servers.

What a Job Is

In industrialized countries, people belong to one institution or another at least until their twenties. After all those years you get used to the idea of belonging to a group of people who all get up in the morning, go to some set of buildings, and do things that they do not, ordinarily, enjoy doing. Belonging to such a group becomes part of your identity: name, age, role, institution. If you have to introduce yourself, or someone else describes you, it will be as something like, John Smith, age 10, a student at such and such elementary school, or John Smith, age 20, a student at such and such college.

When John Smith finishes school he is expected to get a job. And what getting a job seems to mean is joining another institution. Superficially it's a lot like college. You pick the companies you want to work for and apply to join them. If one likes you, you become a member of this new group. You get up in the morning and go to a new set of buildings, and do things that you do not, ordinarily, enjoy doing. There are a few differences: life is not as much fun, and you

get paid, instead of paying, as you did in college. But the similarities feel greater than the differences. John Smith is now John Smith, 22, a software developer at such and such corporation.

In fact John Smith's life has changed more than he realizes. Socially, a company looks much like college, but the deeper you go into the underlying reality, the more different it gets.

What a company does, and has to do if it wants to continue to exist, is earn money. And the way most companies make money is by creating wealth. Companies can be so specialized that this similarity is concealed, but it is not only manufacturing companies that create wealth. A big component of wealth is location. Remember that magic machine that could make you cars and cook you dinner and so on? It would not be so useful if it delivered your dinner to a random location in central Asia. If wealth means what people want, companies that move things also create wealth. Ditto for many other kinds of companies that don't make anything physical. Nearly all companies exist to do something people want.

And that's what you do, as well, when you go to work for a company. But here there is another layer that tends to obscure the underlying reality. In a company, the work you do is averaged together with a lot of other people's. You may not even be aware you're doing something people want. Your contribution may be indirect. But the company as a whole must be giving people something they want, or they won't make any money. And if they are paying you x dollars a year, then on average you must be contributing at least x dollars a year worth of work, or the company will be spending more than it makes, and will go out of business.

Someone graduating from college thinks, and is told, that he needs to get a job, as if the important thing were becoming a member of an institution. A more direct way to put it would be: you need to start doing something people want. You don't need to join a company to do that. All a company is is a group of people working together to do something people want. It's doing something people

want that matters, not joining the group.*

For most people the best plan probably is to go to work for some existing company. But it is a good idea to understand what's happening when you do this. A job means doing something people want, averaged together with everyone else in that company.

Working Harder

That averaging gets to be a problem. I think the single biggest problem afflicting large companies is the difficulty of assigning a value to each person's work. For the most part they punt. In a big company you get paid a fairly predictable salary for working fairly hard. You're expected not to be obviously incompetent or lazy, but you're not expected to devote your whole life to your work.

It turns out, though, that there are economies of scale in how much of your life you devote to your work. In the right kind of business, someone who really devoted himself to work could generate ten or even a hundred times as much wealth as an average employee. A programmer, for example, instead of chugging along maintaining and updating an existing piece of software, could write a whole new piece of software, and with it create a new source of revenue.

Companies are not set up to reward people who want to do this. You can't go to your boss and say, I'd like to start working ten times as hard, so will you please pay me ten times as much? For one thing, the official fiction is that you are already working as hard as you can. But a more serious problem is that the company has no way of measuring the value of your work.

Salesmen are an exception. It's easy to measure how much revenue they generate, and they're usually paid a percentage of it. If a

---

* Many people feel confused and depressed in their early twenties. Life seemed so much more fun in college. Well, of course it was. Don't be fooled by the surface similarities. You've gone from guest to servant. It's possible to have fun in this new world. Among other things, you now get to go behind the doors that say "authorized personnel only." But the change is a shock at first, and all the worse if you're not consciously aware of it.

salesman wants to work harder, he can just start doing it, and he will automatically get paid proportionally more.

There is one other job besides sales where big companies can hire first-rate people: in the top management jobs. And for the same reason: their performance can be measured. The top managers are held responsible for the performance of the entire company. Because an ordinary employee's performance can't usually be measured, he is not expected to do more than put in a solid effort. Whereas top management, like salespeople, have to actually come up with the numbers. The CEO of a company that tanks cannot plead that he put in a solid effort. If the company does badly, he's done badly.

A company that could pay all its employees so straightforwardly would be enormously successful. Many employees would work harder if they could get paid for it. More importantly, such a company would attract people who wanted to work especially hard. It would crush its competitors.

Unfortunately, companies can't pay everyone like salesmen. Salesmen work alone. Most employees' work is tangled together. Suppose a company makes some kind of consumer gadget. The engineers build a reliable gadget with all kinds of new features; the industrial designers design a beautiful case for it; and then the marketing people convince everyone that it's something they've got to have. How do you know how much of the gadget's sales are due to each group's efforts? Or, for that matter, how much is due to the creators of past gadgets that gave the company a reputation for quality? There's no way to untangle all their contributions. Even if you could read the minds of the consumers, you'd find these factors were all blurred together.

If you want to go faster, it's a problem to have your work tangled together with a large number of other people's. In a large group, your performance is not separately measurable—and the rest of the group slows you down.

Measurement and Leverage

To get rich you need to get yourself in a situation with two things,

measurement and leverage. You need to be in a position where your performance can be measured, or there is no way to get paid more by doing more. And you have to have leverage, in the sense that the decisions you make have a big effect.

Measurement alone is not enough. An example of a job with measurement but not leverage is doing piecework in a sweatshop. Your performance is measured and you get paid accordingly, but you have no scope for decisions. The only decision you get to make is how fast you work, and that can probably only increase your earnings by a factor of two or three.

An example of a job with both measurement and leverage would be lead actor in a movie. Your performance can be measured in the gross of the movie. And you have leverage in the sense that your performance can make or break it.

CEOs also have both measurement and leverage. They're measured, in that the performance of the company is their performance. And they have leverage in that their decisions set the whole company moving in one direction or another.

I think everyone who gets rich by their own efforts will be found to be in a situation with measurement and leverage. Everyone I can think of does: CEOs, movie stars, hedge fund managers, professional athletes. A good hint to the presence of leverage is the possibility of failure. Upside must be balanced by downside, so if there is big potential for gain there must also be a terrifying possibility of loss. CEOs, stars, fund managers, and athletes all live with the sword hanging over their heads; the moment they start to suck, they're out. If you're in a job that feels safe, you are not going to get rich, because if there is no danger there is almost certainly no leverage.

But you don't have to become a CEO or a movie star to be in a situation with measurement and leverage. All you need to do is be part of a small group working on a hard problem.

Smallness = Measurement

If you can't measure the value of the work done by individual employees, you can get close. You can measure the value of the work

done by small groups.

One level at which you can accurately measure the revenue generated by employees is at the level of the whole company. When the company is small, you are thereby fairly close to measuring the contributions of individual employees. A viable startup might only have ten employees, which puts you within a factor of ten of measuring individual effort.

Starting or joining a startup is thus as close as most people can get to saying to one's boss, I want to work ten times as hard, so please pay me ten times as much. There are two differences: you're not saying it to your boss, but directly to the customers (for whom your boss is only a proxy after all), and you're not doing it individually, but along with a small group of other ambitious people.

It will, ordinarily, be a group. Except in a few unusual kinds of work, like acting or writing books, you can't be a company of one person. And the people you work with had better be good, because it's their work that yours is going to be averaged with.

A big company is like a giant galley driven by a thousand rowers. Two things keep the speed of the galley down. One is that individual rowers don't see any result from working harder. The other is that, in a group of a thousand people, the average rower is likely to be pretty average.

If you took ten people at random out of the big galley and put them in a boat by themselves, they could probably go faster. They would have both carrot and stick to motivate them. An energetic rower would be encouraged by the thought that he could have a visible effect on the speed of the boat. And if someone was lazy, the others would be more likely to notice and complain.

But the real advantage of the ten-man boat shows when you take the ten *best* rowers out of the big galley and put them in a boat together. They will have all the extra motivation that comes from being in a small group. But more importantly, by selecting that small a group you can get the best rowers. Each one will be in the top 1%. It's a much better deal for them to average their work together with a small group of their peers than to average it with everyone.

That's the real point of startups. Ideally, you are getting together with a group of other people who also want to work a lot harder, and get paid a lot more, than they would in a big company. And because startups tend to get founded by self-selecting groups of ambitious people who already know one another (at least by reputation), the level of measurement is more precise than you get from smallness alone. A startup is not merely ten people, but ten people like you.

Steve Jobs once said that the success or failure of a startup depends on the first ten employees. I agree. If anything, it's more like the first five. Being small is not, in itself, what makes startups kick butt, but rather that small groups can be select. You don't want small in the sense of a village, but small in the sense of an all-star team.

The larger a group, the closer its average member will be to the average for the population as a whole. So all other things being equal, a very able person in a big company is probably getting a bad deal, because his performance is dragged down by the overall lower performance of the others. Of course, all other things often are not equal: the able person may not care about money, or may prefer the stability of a large company. But a very able person who does care about money will ordinarily do better to go off and work with a small group of peers.

Technology = Leverage

Startups offer anyone a way to be in a situation with measurement and leverage. They allow measurement because they're small, and they offer leverage because they make money by inventing new technology.

What is technology? It's *technique*. It's the way we all do things. And when you discover a new way to do things, its value is multiplied by all the people who use it. It is the proverbial fishing rod, rather than the fish. That's the difference between a startup and a restaurant or a barber shop. You fry eggs or cut hair one customer at a time. Whereas if you solve a technical problem that a lot of people care about, you help everyone who uses your solution. That's leverage.

If you look at history, it seems that most people who got rich by creating wealth did it by developing new technology. You just can't fry eggs or cut hair fast enough. What made the Florentines rich in 1200 was the discovery of new techniques for making the high-tech product of the time, fine woven cloth. What made the Dutch rich in 1600 was the discovery of shipbuilding and navigation techniques that enabled them to dominate the seas of the Far East.

Fortunately there is a natural fit between smallness and solving hard problems. The leading edge of technology moves fast. Technology that's valuable today could be worthless in a couple years. Small companies are more at home in this world, because they don't have layers of bureaucracy to slow them down. Also, technical advances tend to come from unorthodox approaches, and small companies are less constrained by convention.

Big companies can develop technology. They just can't do it quickly. Their size makes them slow and prevents them from rewarding employees for the extraordinary effort required. So in practice big companies only get to develop technology in fields where large capital requirements prevent startups from competing with them, like microprocessors, power plants, or passenger aircraft. And even in those fields they depend heavily on startups for components and ideas.

It's obvious that biotech or software startups exist to solve hard technical problems, but I think it will also be found to be true in businesses that don't seem to be about technology. McDonald's, for example, grew big by designing a system, the McDonald's franchise, that could then be reproduced at will all over the face of the earth. A McDonald's franchise is controlled by rules so precise that it is practically a piece of software. Write once, run everywhere. Ditto for Wal-Mart. Sam Walton got rich not by being a retailer, but by designing a new kind of store.

Use difficulty as a guide not just in selecting the overall aim of your company, but also at decision points along the way. At Viaweb one of our rules of thumb was *run upstairs*. Suppose you are a little, nimble guy being chased by a big, fat, bully. You open a door and

find yourself in a staircase. Do you go up or down? I say up. The bully can probably run downstairs as fast as you can. Going upstairs his bulk will be more of a disadvantage. Running upstairs is hard for you but even harder for him.

What this meant in practice was that we deliberately sought hard problems. If there were two features we could add to our software, both equally valuable in proportion to their difficulty, we'd always take the harder one. Not just because it was more valuable, but *because it was harder.* We delighted in forcing bigger, slower competitors to follow us over difficult ground. Like guerillas, startups prefer the difficult terrain of the mountains, where the troops of the central government can't follow. I can remember times when we were just exhausted after wrestling all day with some horrible technical problem. And I'd be delighted, because something that was hard for us would be impossible for our competitors.

This is not just a good way to run a startup. It's what a startup is. Venture capitalists know about this and have a phrase for it: *barriers to entry.* If you go to a VC with a new idea and ask him to invest in it, one of the first things he'll ask is, how hard would this be for someone else to develop? That is, how much difficult ground have you put between yourself and potential pursuers?* And you had better have a convincing explanation of why your technology would be hard to duplicate. Otherwise as soon as some big company becomes aware of it, they'll make their own, and with their brand name, capital, and distribution clout, they'll take away your market overnight. You'd be like guerillas caught in the open field by regular army forces.

One way to put up barriers to entry is through patents. But patents may not provide much protection. Competitors commonly find ways to work around a patent. And if they can't, they may simply violate it and invite you to sue them. A big company is not afraid to be sued; it's an everyday thing for them. They'll make sure that suing them is expensive and takes a long time. Ever heard of Philo Farns-

* When VCs asked us how long it would take another startup to duplicate our software, we used to reply that they probably wouldn't be able to at all. I think this made us seem naive, or liars.

worth? He invented television. The reason you've never heard of him is that his company was not the one to make money from it.* The company that did was RCA, and Farnsworth's reward for his efforts was a decade of patent litigation.

Here, as so often, the best defense is a good offense. If you can develop technology that's simply too hard for competitors to duplicate, you don't need to rely on other defenses. Start by picking a hard problem, and then at every decision point, take the harder choice.†

The Catch(es)

If it were simply a matter of working harder than an ordinary employee and getting paid proportionately, it would obviously be a good deal to start a startup. Up to a point it would be more fun. I don't think many people like the slow pace of big companies, the interminable meetings, the water-cooler conversations, the clueless middle managers, and so on.

Unfortunately there are a couple catches. One is that you can't choose the point on the curve that you want to inhabit. You can't decide, for example, that you'd like to work just two or three times as hard, and get paid that much more. When you're running a startup, your competitors decide how hard you work. And they pretty much all make the same decision: as hard as you possibly can.

The other catch is that the payoff is only on average proportion-

---

* Few technologies have one clear inventor. So as a rule, if you know the "inventor" of something (the telephone, the assembly line, the airplane, the light bulb, the transistor) it is because their company made money from it, and the company's PR people worked hard to spread the story. If you don't know who invented something (the automobile, the television, the computer, the jet engine, the laser), it's because other companies made all the money.

† This is a good plan for life in general. If you have two choices, choose the harder. If you're trying to decide whether to go out running or sit home and watch TV, go running. Probably the reason this trick works so well is that when you have two choices and one is harder, the only reason you're even considering the other is laziness. You know in the back of your mind what's the right thing to do, and this trick merely forces you to acknowledge it.

ate to your productivity. There is, as I said before, a large random multiplier in the success of any company. So in practice the deal is not that you're 30 times as productive and get paid 30 times as much. It is that you're 30 times as productive, and get paid between zero and a thousand times as much. If the mean is 30x, the median is probably zero. Most startups tank, and not just the dogfood portals we all heard about during the Internet Bubble. It's common for a startup to be developing a genuinely good product, take slightly too long to do it, run out of money, and have to shut down.

A startup is like a mosquito. A bear can absorb a hit and a crab is armored against one, but a mosquito is designed for one thing: to score. No energy is wasted on defense. The defense of mosquitos, as a species, is that there are a lot of them, but this is little consolation to the individual mosquito.

Startups, like mosquitos, tend to be an all-or-nothing proposition. And you don't generally know which of the two you're going to get till the last minute. Viaweb came close to tanking several times. Our trajectory was like a sine wave. Fortunately we got bought at the top of the cycle, but it was damned close. While we were visiting Yahoo in California to talk about selling the company to them, we had to borrow a conference room to reassure an investor who was about to back out of a new round of funding that we needed to stay alive.

The all-or-nothing aspect of startups was not something we wanted. Viaweb's hackers were all extremely risk-averse. If there had been some way just to work super hard and get paid for it, without having a lottery mixed in, we would have been delighted. We would have much preferred a 100% chance of $1 million to a 20% chance of $10 million, even though theoretically the second is worth twice as much. Unfortunately, there is not currently any space in the business world where you can get the first deal.

The closest you can get is by selling your startup in the early stages, giving up upside (and risk) for a smaller but guaranteed payoff. We had a chance to do this, and stupidly, as we then thought, let it slip by. After that we became comically eager to sell. For the next year or so, if anyone expressed the slightest curiosity about Viaweb

we would try to sell them the company. But there were no takers, so we had to keep going.

It would have been a bargain to buy us at an early stage, but companies doing acquisitions are not looking for bargains. A company big enough to acquire startups will be big enough to be fairly conservative, and within the company the people in charge of acquisitions will be among the more conservative, because they are likely to be business school types who joined the company late. They would rather overpay for a safe choice. So it is easier to sell an established startup, even at a large premium, than an early-stage one.

Get Users

I think it's a good idea to get bought, if you can. Running a business is different from growing one. It is just as well to let a big company take over once you reach cruising altitude. It's also financially wiser, because selling allows you to diversify. What would you think of a financial advisor who put all his client's assets into one volatile stock?

How do you get bought? Mostly by doing the same things you'd do if you didn't intend to sell the company. Being profitable, for example. But getting bought is also an art in its own right, and one that we spent a lot of time trying to master.

Potential buyers will always delay if they can. The hard part about getting bought is getting them to act. For most people, the most powerful motivator is not the hope of gain, but the fear of loss. For potential acquirers, the most powerful motivator is the prospect that one of their competitors will buy you. This, as we found, causes CEOs to take red-eyes. The second biggest is the worry that, if they don't buy you now, you'll continue to grow rapidly and will cost more to acquire later, or even become a competitor.

In both cases, what it all comes down to is users. You'd think that a company about to buy you would do a lot of research and decide for themselves how valuable your technology was. Not at all. What they go by is the number of users you have.

In effect, acquirers assume the customers know who has the best technology. And this is not as stupid as it sounds. Users are the only

real proof that you've created wealth. Wealth is what people want, and if people aren't using your software, maybe it's not just because you're bad at marketing. Maybe it's because you haven't made what they want.

Venture capitalists have a list of danger signs to watch out for. Near the top is the company run by techno-weenies who are obsessed with solving interesting technical problems, instead of making users happy. In a startup, you're not just trying to solve problems. You're trying to solve problems *that users care about.*

So I think you should make users the test, just as acquirers do. Treat a startup as an optimization problem in which performance is measured by number of users. As anyone who has tried to optimize software knows, the key is measurement. When you try to guess where your program is slow, and what would make it faster, you almost always guess wrong.

Number of users may not be the perfect test, but it will be very close. It's what acquirers care about. It's what revenues depend on. It's what makes competitors unhappy. It's what impresses reporters, and potential new users. Certainly it's a better test than your a priori notions of what problems are important to solve, no matter how technically adept you are.

Among other things, treating a startup as an optimization problem will help you avoid another pitfall that VCs worry about, and rightly—taking a long time to develop a product. Now we can recognize this as something hackers already know to avoid: premature optimization. Get a version 1.0 out there as soon as you can. Until you have some users to measure, you're optimizing based on guesses.

The ball you need to keep your eye on here is the underlying principle that wealth is what people want. If you plan to get rich by creating wealth, you have to know what people want. So few businesses really pay attention to making customers happy. How often do you walk into a store, or call a company on the phone, with a feeling of dread in the back of your mind? When you hear "your call is important to us, please stay on the line," do you think, oh good, now everything will be all right?

A restaurant can afford to serve the occasional burnt dinner. But in technology, you cook one thing and that's what everyone eats. So any difference between what people want and what you deliver is multiplied. You please or annoy customers wholesale. The closer you can get to what they want, the more wealth you generate.

Wealth and Power

Making wealth is not the only way to get rich. For most of human history it has not even been the most common. Until a few centuries ago, the main sources of wealth were mines, slaves and serfs, land, and cattle, and the only ways to acquire these rapidly were by inheritance, marriage, conquest, or confiscation. Naturally wealth had a bad reputation.

Two things changed. The first was the rule of law. For most of the world's history, if you did somehow accumulate a fortune, the ruler or his henchmen would find a way to steal it. But in medieval Europe something new happened. A new class of merchants and manufacturers began to collect in towns.* Together they were able to withstand the local feudal lord. So for the first time in our history, the bullies stopped stealing the nerds' lunch money. This was naturally a great incentive, and possibly indeed the main cause of the second big change, industrialization.

A great deal has been written about the causes of the Industrial Revolution. But surely a necessary, if not sufficient, condition was that people who made fortunes be able to enjoy them in peace.† One

---

* It is probably no accident that the middle class first appeared in northern Italy and the Low Countries, where there were no strong central governments. These two regions were the richest of their time and became the twin centers from which Renaissance civilization radiated. If they no longer play that role, it is because other places, like the United States, have been truer to the principles they discovered.

† It may indeed be a sufficient condition. But if so, why didn't the Industrial Revolution happen earlier? Two possible (and not incompatible) answers: (a) It did. The Industrial Revolution was one in a series. (b) Because in medieval towns, monopolies and guild regulations initially slowed the development of new

piece of evidence is what happened to countries that tried to return to the old model, like the Soviet Union, and to a lesser extent Britain under the labor governments of the 1960s and early 1970s. Take away the incentive of wealth, and technical innovation grinds to a halt.

Remember what a startup is, economically: a way of saying, I want to work faster. Instead of accumulating money slowly by being paid a regular wage for fifty years, I want to get it over with as soon as possible. So governments that forbid you to accumulate wealth are in effect decreeing that you work slowly. They're willing to let you earn $3 million over fifty years, but they're not willing to let you work so hard that you can do it in two. They are like the corporate boss that you can't go to and say, I want to work ten times as hard, so please pay me ten times a much. Except this is not a boss you can escape by starting your own company.

The problem with working slowly is not just that technical innovation happens slowly. It's that it tends not to happen at all. It's only when you're deliberately looking for hard problems, as a way to use speed to the greatest advantage, that you take on this kind of project. Developing new technology is a pain in the ass. It is, as Edison said, one percent inspiration and ninety-nine percent perspiration. Without the incentive of wealth, no one wants to do it. Engineers will work on sexy projects like fighter planes and moon rockets for ordinary salaries, but more mundane technologies like light bulbs or semiconductors have to be developed by entrepreneurs.

Startups are not just something that happened in Silicon Valley in the last couple decades. Since it became possible to get rich by creating wealth, everyone who has done it has used essentially the same recipe: measurement and leverage, where measurement comes from working with a small group, and leverage from developing new techniques. The recipe was the same in Florence in 1200 as it is in Santa Clara today.

Understanding this may help to answer an important question:

---

means of production.

why Europe grew so powerful. Was it something about the geography of Europe? Was it that Europeans are somehow racially superior? Was it their religion? The answer (or at least the proximate cause) may be that the Europeans rode on the crest of a powerful new idea: allowing those who made a lot of money to keep it.

Once you're allowed to do that, people who want to get rich can do it by generating wealth instead of stealing it. The resulting technological growth translates not only into wealth but into military power. The theory that led to the stealth plane was developed by a Soviet mathematician. But because the Soviet Union didn't have a computer industry, it remained for them a theory; they didn't have hardware capable of executing the calculations fast enough to design an actual airplane.

In that respect the Cold War teaches the same lesson as World War II and, for that matter, most wars in recent history. Don't let a ruling class of warriors and politicians squash the entrepreneurs. The same recipe that makes individuals rich makes countries powerful. Let the nerds keep their lunch money, and you rule the world.

The Power of the Marginal

BY PAUL GRAHAM
JUNE 2006

A couple years ago my friend Trevor and I went to look at the Apple garage. As we stood there, he said that as a kid growing up in Saskatchewan he'd been amazed at the dedication Jobs and Wozniak must have had to work in a garage. "Those guys must have been freezing!"

That's one of California's hidden advantages: the mild climate means there's lots of marginal space. In cold places that margin gets trimmed off. There's a sharper line between outside and inside, and only projects that are officially sanctioned—by organizations, or parents, or wives, or at least by oneself—get proper indoor space. That raises the activation energy for new ideas. You can't just tinker. You have to justify.

Some of Silicon Valley's most famous companies began in garages: Hewlett-Packard in 1938, Apple in 1976, Google in 1998. In Apple's case the garage story is a bit of an urban legend. Woz says all they did there was assemble some computers, and that he did all the actual design of the Apple I and Apple II in his apartment or his cube at HP. This was apparently too marginal even for Apple's PR people.

By conventional standards, Jobs and Wozniak were marginal people too. Obviously they were smart, but they can't have looked good on paper. They were at the time a pair of college dropouts with

about three years of school between them, and hippies to boot. Their previous business experience consisted of making "blue boxes" to hack into the phone system, a business with the rare distinction of being both illegal and unprofitable.

Outsiders

Now a startup operating out of a garage in Silicon Valley would feel part of an exalted tradition, like the poet in his garret, or the painter who can't afford to heat his studio and thus has to wear a beret indoors. But in 1976 it didn't seem so cool. The world hadn't yet realized that starting a computer company was in the same category as being a writer or a painter. It hadn't been for long. Only in the preceding couple years had the dramatic fall in the cost of hardware allowed outsiders to compete.

In 1976, everyone looked down on a company operating out of a garage, including the founders. One of the first things Jobs did when they got some money was to rent office space. He wanted Apple to seem like a real company.

They already had something few real companies ever have: a fabulously well designed product. You'd think they'd have had more confidence. But I've talked to a lot of startup founders, and it's always this way. They've built something that's going to change the world, and they're worried about some nit like not having proper business cards.

That's the paradox I want to explore: great new things often come from the margins, and yet the people who discover them are looked down on by everyone, including themselves.

It's an old idea that new things come from the margins. I want to examine its internal structure. Why do great ideas come from the margins? What kind of ideas? And is there anything we can do to encourage the process?

Insiders

One reason so many good ideas come from the margin is simply that

there's so much of it. There have to be more outsiders than insiders, if insider means anything. If the number of outsiders is huge it will always seem as if a lot of ideas come from them, even if few do per capita. But I think there's more going on than this. There are real disadvantages to being an insider, and in some kinds of work they can outweigh the advantages.

Imagine, for example, what would happen if the government decided to commission someone to write an official Great American Novel. First there'd be a huge ideological squabble over who to choose. Most of the best writers would be excluded for having offended one side or the other. Of the remainder, the smart ones would refuse such a job, leaving only a few with the wrong sort of ambition. The committee would choose one at the height of his career—that is, someone whose best work was behind him—and hand over the project with copious free advice about how the book should show in positive terms the strength and diversity of the American people, etc, etc.

The unfortunate writer would then sit down to work with a huge weight of expectation on his shoulders. Not wanting to blow such a public commission, he'd play it safe. This book had better command respect, and the way to ensure that would be to make it a tragedy. Audiences have to be enticed to laugh, but if you kill people they feel obliged to take you seriously. As everyone knows, America plus tragedy equals the Civil War, so that's what it would have to be about. Better stick to the standard cartoon version that the Civil War was about slavery; people would be confused otherwise; plus you can show a lot of strength and diversity. When finally completed twelve years later, the book would be a 900-page pastiche of existing popular novels—roughly *Gone with the Wind* plus*Roots*. But its bulk and celebrity would make it a bestseller for a few months, until blown out of the water by a talk-show host's autobiography. The book would be made into a movie and thereupon forgotten, except by the more waspish sort of reviewers, among whom it would be a byword for bogusness like Milli Vanilli or Battlefield Earth.

Maybe I got a little carried away with this example. And yet is

this not at each point the way such a project would play out? The government knows better than to get into the novel business, but in other fields where they have a natural monopoly, like nuclear waste dumps, aircraft carriers, and regime change, you'd find plenty of projects isomorphic to this one—and indeed, plenty that were less successful.

This little thought experiment suggests a few of the disadvantages of insider projects: the selection of the wrong kind of people, the excessive scope, the inability to take risks, the need to seem serious, the weight of expectations, the power of vested interests, the undiscerning audience, and perhaps most dangerous, the tendency of such work to become a duty rather than a pleasure.

Tests

A world with outsiders and insiders implies some kind of test for distinguishing between them. And the trouble with most tests for selecting elites is that there are two ways to pass them: to be good at what they try to measure, and to be good at hacking the test itself.

So the first question to ask about a field is how honest its tests are, because this tells you what it means to be an outsider. This tells you how much to trust your instincts when you disagree with authorities, whether it's worth going through the usual channels to become one yourself, and perhaps whether you want to work in this field at all.

Tests are least hackable when there are consistent standards for quality, and the people running the test really care about its integrity. Admissions to PhD programs in the hard sciences are fairly honest, for example. The professors will get whoever they admit as their own grad students, so they try hard to choose well, and they have a fair amount of data to go on. Whereas undergraduate admissions seem to be much more hackable.

One way to tell whether a field has consistent standards is the overlap between the leading practitioners and the people who teach the subject in universities. At one end of the scale you have fields like math and physics, where nearly all the teachers are among the best

practitioners. In the middle are medicine, law, history, architecture, and computer science, where many are. At the bottom are business, literature, and the visual arts, where there's almost no overlap between the teachers and the leading practitioners. It's this end that gives rise to phrases like "those who can't do, teach."

Incidentally, this scale might be helpful in deciding what to study in college. When I was in college the rule seemed to be that you should study whatever you were most interested in. But in retrospect you're probably better off studying something moderately interesting with someone who's good at it than something very interesting with someone who isn't. You often hear people say that you shouldn't major in business in college, but this is actually an instance of a more general rule: don't learn things from teachers who are bad at them.

How much you should worry about being an outsider depends on the quality of the insiders. If you're an amateur mathematician and think you've solved a famous open problem, better go back and check. When I was in grad school, a friend in the math department had the job of replying to people who sent in proofs of Fermat's last theorem and so on, and it did not seem as if he saw it as a valuable source of tips—more like manning a mental health hotline. Whereas if the stuff you're writing seems different from what English professors are interested in, that's not necessarily a problem.

Anti-Tests

Where the method of selecting the elite is thoroughly corrupt, most of the good people will be outsiders. In art, for example, the image of the poor, misunderstood genius is not just one possible image of a great artist: it's the *standard* image. I'm not saying it's correct, incidentally, but it is telling how well this image has stuck. You couldn't make a rap like that stick to math or medicine.*

* As usual the popular image is several decades behind reality. Now the misunderstood artist is not a chain-smoking drunk who pours his soul into big, messy canvases that philistines see and say "that's not art" because it isn't a picture of anything. The philistines have now been trained that anything hung on a wall is

If it's corrupt enough, a test becomes an anti-test, filtering out the people it should select by making them to do things only the wrong people would do. Popularity in high school seems to be such a test. There are plenty of similar ones in the grownup world. For example, rising up through the hierarchy of the average big company demands an attention to politics few thoughtful people could spare.* Someone like Bill Gates can grow a company under him, but it's hard to imagine him having the patience to climb the corporate ladder at General Electric—or Microsoft, actually.

It's kind of strange when you think about it, because lord-of-the-flies schools and bureaucratic companies are both the default. There are probably a lot of people who go from one to the other and never realize the whole world doesn't work this way.

I think that's one reason big companies are so often blindsided by startups. People at big companies don't realize the extent to which they live in an environment that is one large, ongoing test for the wrong qualities.

If you're an outsider, your best chances for beating insiders are obviously in fields where corrupt tests select a lame elite. But there's a catch: if the tests are corrupt, your victory won't be recognized, at least in your lifetime. You may feel you don't need that, but history suggests it's dangerous to work in fields with corrupt tests. You may beat the insiders, and yet not do as good work, on an absolute scale, as you would in a field that was more honest.

Standards in art, for example, were almost as corrupt in the first half of the eighteenth century as they are today. This was the era of those fluffy idealized portraits of countesses with their lapdogs. Chardin decided to skip all that and paint ordinary things as he saw them. He's now considered the best of that period—and yet not the equal of Leonardo or Bellini or Memling, who all had the additional

art. Now the misunderstood artist is a coffee-drinking vegan cartoonist whose work they see and say "that's not art" because it looks like stuff they've seen in the Sunday paper.

* In fact this would do fairly well as a definition of politics: what determines rank in the absence of objective tests.

encouragement of honest standards.

It can be worth participating in a corrupt contest, however, if it's followed by another that isn't corrupt. For example, it would be worth competing with a company that can spend more than you on marketing, as long as you can survive to the next round, when customers compare your actual products. Similarly, you shouldn't be discouraged by the comparatively corrupt test of college admissions, because it's followed immediately by less hackable tests.*

Risk

Even in a field with honest tests, there are still advantages to being an outsider. The most obvious is that outsiders have nothing to lose. They can do risky things, and if they fail, so what? Few will even notice.

The eminent, on the other hand, are weighed down by their eminence. Eminence is like a suit: it impresses the wrong people, and it constrains the wearer.

Outsiders should realize the advantage they have here. Being able to take risks is hugely valuable. Everyone values safety too much, both the obscure and the eminent. No one wants to look like a fool. But it's very useful to be able to. If most of your ideas aren't stupid, you're probably being too conservative. You're not bracketing the problem.

Lord Acton said we should judge talent at its best and character at its worst. For example, if you write one great book and ten bad ones, you still count as a great writer—or at least, a better writer than someone who wrote eleven that were merely good. Whereas if you're a quiet, law-abiding citizen most of the time but occasionally cut someone up and bury them in your backyard, you're a bad guy.

Almost everyone makes the mistake of treating ideas as if they

---

* In high school you're led to believe your whole future depends on where you go to college, but it turns out only to buy you a couple years. By your mid-twenties the people worth impressing already judge you more by what you've done than where you went to school.

were indications of character rather than talent—as if having a stupid idea made you stupid. There's a huge weight of tradition advising us to play it safe. "Even a fool is thought wise if he keeps silent," says the Old Testament (Proverbs 17:28).

Well, that may be fine advice for a bunch of goatherds in Bronze Age Palestine. There conservatism would be the order of the day. But times have changed. It might still be reasonable to stick with the Old Testament in political questions, but materially the world now has a lot more state. Tradition is less of a guide, not just because things change faster, but because the space of possibilities is so large. The more complicated the world gets, the more valuable it is to be willing to look like a fool.

Delegation

And yet the more successful people become, the more heat they get if they screw up—or even seem to screw up. In this respect, as in many others, the eminent are prisoners of their own success. So the best way to understand the advantages of being an outsider may be to look at the disadvantages of being an insider.

If you ask eminent people what's wrong with their lives, the first thing they'll complain about is the lack of time. A friend of mine at Google is fairly high up in the company and went to work for them long before they went public. In other words, he's now rich enough not to have to work. I asked him if he could still endure the annoyances of having a job, now that he didn't have to. And he said that there weren't really any annoyances, except—and he got a wistful look when he said this—that he got *so much email*.

The eminent feel like everyone wants to take a bite out of them. The problem is so widespread that people pretending to be eminent do it by pretending to be overstretched.

The lives of the eminent become scheduled, and that's not good for thinking. One of the great advantages of being an outsider is long, uninterrupted blocks of time. That's what I remember about grad school: apparently endless supplies of time, which I spent worrying about, but not writing, my dissertation. Obscurity is like health

food—unpleasant, perhaps, but good for you. Whereas fame tends to be like the alcohol produced by fermentation. When it reaches a certain concentration, it kills off the yeast that produced it.

The eminent generally respond to the shortage of time by turning into managers. They don't have time to work. They're surrounded by junior people they're supposed to help or supervise. The obvious solution is to have the junior people do the work. Some good stuff happens this way, but there are problems it doesn't work so well for: the kind where it helps to have everything in one head.

For example, it recently emerged that the famous glass artist Dale Chihuly hasn't actually blown glass for 27 years. He has assistants do the work for him. But one of the most valuable sources of ideas in the visual arts is the resistance of the medium. That's why oil paintings look so different from watercolors. In principle you could make any mark in any medium; in practice the medium steers you. And if you're no longer doing the work yourself, you stop learning from this.

So if you want to beat those eminent enough to delegate, one way to do it is to take advantage of direct contact with the medium. In the arts it's obvious how: blow your own glass, edit your own films, stage your own plays. And in the process pay close attention to accidents and to new ideas you have on the fly. This technique can be generalized to any sort of work: if you're an outsider, don't be ruled by plans. Planning is often just a weakness forced on those who delegate.

Is there a general rule for finding problems best solved in one head? Well, you can manufacture them by taking any project usually done by multiple people and trying to do it all yourself. Wozniak's work was a classic example: he did everything himself, hardware and software, and the result was miraculous. He claims not one bug was ever found in the Apple II, in either hardware or software.

Another way to find good problems to solve in one head is to focus on the grooves in the chocolate bar—the places where tasks are divided when they're split between several people. If you want to beat delegation, focus on a vertical slice: for example, be both writer and editor, or both design buildings and construct them.

One especially good groove to span is the one between tools and things made with them. For example, programming languages and applications are usually written by different people, and this is responsible for a lot of the worst flaws in programming languages. I think every language should be designed simultaneously with a large application written in it, the way C was with Unix.

Techniques for competing with delegation translate well into business, because delegation is endemic there. Instead of avoiding it as a drawback of senility, many companies embrace it as a sign of maturity. In big companies software is often designed, implemented, and sold by three separate types of people. In startups one person may have to do all three. And though this feels stressful, it's one reason startups win. The needs of customers and the means of satisfying them are all in one head.

Focus

The very skill of insiders can be a weakness. Once someone is good at something, they tend to spend all their time doing that. This kind of focus is very valuable, actually. Much of the skill of experts is the ability to ignore false trails. But focus has drawbacks: you don't learn from other fields, and when a new approach arrives, you may be the last to notice.

For outsiders this translates into two ways to win. One is to work on a variety of things. Since you can't derive as much benefit (yet) from a narrow focus, you may as well cast a wider net and derive what benefit you can from similarities between fields. Just as you can compete with delegation by working on larger vertical slices, you can compete with specialization by working on larger horizontal slices—by both writing and illustrating your book, for example.

The second way to compete with focus is to see what focus overlooks. In particular, new things. So if you're not good at anything yet, consider working on something so new that no one else is either. It won't have any prestige yet, if no one is good at it, but you'll have it all to yourself.

The potential of a new medium is usually underestimated, pre-

cisely because no one has yet explored its possibilities. Before Durer tried making engravings, no one took them very seriously. Engraving was for making little devotional images—basically fifteenth century baseball cards of saints. Trying to make masterpieces in this medium must have seemed to Durer's contemporaries that way that, say, making masterpieces in comics might seem to the average person today.

In the computer world we get not new mediums but new platforms: the minicomputer, the microprocessor, the web-based application. At first they're always dismissed as being unsuitable for real work. And yet someone always decides to try anyway, and it turns out you can do more than anyone expected. So in the future when you hear people say of a new platform: yeah, it's popular and cheap, but not ready yet for real work, jump on it.

As well as being more comfortable working on established lines, insiders generally have a vested interest in perpetuating them. The professor who made his reputation by discovering some new idea is not likely to be the one to discover its replacement. This is particularly true with companies, who have not only skill and pride anchoring them to the status quo, but money as well. The Achilles heel of successful companies is their inability to cannibalize themselves. Many innovations consist of replacing something with a cheaper alternative, and companies just don't want to see a path whose immediate effect is to cut an existing source of revenue.

So if you're an outsider you should actively seek out contrarian projects. Instead of working on things the eminent have made prestigious, work on things that could steal that prestige.

The really juicy new approaches are not the ones insiders reject as impossible, but those they ignore as undignified. For example, after Wozniak designed the Apple II he offered it first to his employer, HP. They passed. One of the reasons was that, to save money, he'd designed the Apple II to use a TV as a monitor, and HP felt they couldn't produce anything so declasse.

Less

Wozniak used a TV as a monitor for the simple reason that he couldn't afford a monitor. Outsiders are not merely free but compelled to make things that are cheap and lightweight. And both are good bets for growth: cheap things spread faster, and lightweight things evolve faster.

The eminent, on the other hand, are almost forced to work on a large scale. Instead of garden sheds they must design huge art museums. One reason they work on big things is that they can: like our hypothetical novelist, they're flattered by such opportunities. They also know that big projects will by their sheer bulk impress the audience. A garden shed, however lovely, would be easy to ignore; a few might even snicker at it. You can't snicker at a giant museum, no matter how much you dislike it. And finally, there are all those people the eminent have working for them; they have to choose projects that can keep them all busy.

Outsiders are free of all this. They can work on small things, and there's something very pleasing about small things. Small things can be perfect; big ones always have something wrong with them. But there's a magic in small things that goes beyond such rational explanations. All kids know it. Small things have more personality.

Plus making them is more fun. You can do what you want; you don't have to satisfy committees. And perhaps most important, small things can be done fast. The prospect of seeing the finished project hangs in the air like the smell of dinner cooking. If you work fast, maybe you could have it done tonight.

Working on small things is also a good way to learn. The most important kinds of learning happen one project at a time. ("Next time, I won't…") The faster you cycle through projects, the faster you'll evolve.

Plain materials have a charm like small scale. And in addition there's the challenge of making do with less. Every designer's ears perk up at the mention of that game, because it's a game you can't lose. Like the JV playing the varsity, if you even tie, you win. So paradoxically there are cases where fewer resources yield better results,

because the designers' pleasure at their own ingenuity more than compensates.*

So if you're an outsider, take advantage of your ability to make small and inexpensive things. Cultivate the pleasure and simplicity of that kind of work; one day you'll miss it.

Responsibility

When you're old and eminent, what will you miss about being young and obscure? What people seem to miss most is the lack of responsibilities.

Responsibility is an occupational disease of eminence. In principle you could avoid it, just as in principle you could avoid getting fat as you get old, but few do. I sometimes suspect that responsibility is a trap and that the most virtuous route would be to shirk it, but regardless it's certainly constraining.

When you're an outsider you're constrained too, of course. You're short of money, for example. But that constrains you in different ways. How does responsibility constrain you? The worst thing is that it allows you not to focus on real work. Just as the most dangerous forms of procrastination are those that seem like work, the danger of responsibilities is not just that they can consume a whole day, but that they can do it without setting off the kind of alarms you'd set off if you spent a whole day sitting on a park bench.

A lot of the pain of being an outsider is being aware of one's own procrastination. But this is actually a good thing. You're at least close enough to work that the smell of it makes you hungry.

As an outsider, you're just one step away from getting things done. A huge step, admittedly, and one that most people never seem to make, but only one step. If you can summon up the energy to get started, you can work on projects with an intensity (in both senses)

---

* Managers are presumably wondering, how can I make this miracle happen? How can I make the people working for me do more with less? Unfortunately the constraint probably has to be self-imposed. If you're *expected* to do more with less, then you're being starved, not eating virtuously.

that few insiders can match. For insiders work turns into a duty, laden with responsibilities and expectations. It's never so pure as it was when they were young.

Work like a dog being taken for a walk, instead of an ox being yoked to the plow. That's what they miss.

Audience

A lot of outsiders make the mistake of doing the opposite; they admire the eminent so much that they copy even their flaws. Copying is a good way to learn, but copy the right things. When I was in college I imitated the pompous diction of famous professors. But this wasn't what *made* them eminent—it was more a flaw their eminence had allowed them to sink into. Imitating it was like pretending to have gout in order to seem rich.

Half the distinguishing qualities of the eminent are actually disadvantages. Imitating these is not only a waste of time, but will make you seem a fool to your models, who are often well aware of it.

What are the genuine advantages of being an insider? The greatest is an audience. It often seems to outsiders that the great advantage of insiders is money—that they have the resources to do what they want. But so do people who inherit money, and that doesn't seem to help, not as much as an audience. It's good for morale to know people want to see what you're making; it draws work out of you.

If I'm right that the defining advantage of insiders is an audience, then we live in exciting times, because just in the last ten years the Internet has made audiences a lot more liquid. Outsiders don't have to content themselves anymore with a proxy audience of a few smart friends. Now, thanks to the Internet, they can start to grow themselves actual audiences. This is great news for the marginal, who retain the advantages of outsiders while increasingly being able to siphon off what had till recently been the prerogative of the elite.

Though the Web has been around for more than ten years, I think we're just beginning to see its democratizing effects. Outsiders are still learning how to steal audiences. But more importantly, audiences are still learning how to be stolen—they're still just beginning

to realize how much deeper bloggers can dig than journalists, how much more interesting a democratic news site can be than a front page controlled by editors, and how much funnier a bunch of kids with webcams can be than mass-produced sitcoms.

The big media companies shouldn't worry that people will post their copyrighted material on YouTube. They should worry that people will post their own stuff on YouTube, and audiences will watch that instead.

Hacking

If I had to condense the power of the marginal into one sentence it would be: just try hacking something together. That phrase draws in most threads I've mentioned here. Hacking something together means deciding what to do as you're doing it, not a subordinate executing the vision of his boss. It implies the result won't be pretty, because it will be made quickly out of inadequate materials. It may work, but it won't be the sort of thing the eminent would want to put their name on. Something hacked together means something that barely solves the problem, or maybe doesn't solve the problem at all, but another you discovered en route. But that's ok, because the main value of that initial version is not the thing itself, but what it leads to. Insiders who daren't walk through the mud in their nice clothes will never make it to the solid ground on the other side.

The word "try" is an especially valuable component. I disagree here with Yoda, who said there is no try. There is try. It implies there's no punishment if you fail. You're driven by curiosity instead of duty. That means the wind of procrastination will be in your favor: instead of avoiding this work, this will be what you do as a way of avoiding other work. And when you do it, you'll be in a better mood. The more the work depends on imagination, the more that matters, because most people have more ideas when they're happy.

If I could go back and redo my twenties, that would be one thing I'd do more of: just try hacking things together. Like many people that age, I spent a lot of time worrying about what I should do. I also spent some time trying to build stuff. I should have spent less time

worrying and more time building. If you're not sure what to do, make something.

Raymond Chandler's advice to thriller writers was "When in doubt, have a man come through a door with a gun in his hand." He followed that advice. Judging from his books, he was often in doubt. But though the result is occasionally cheesy, it's never boring. In life, as in books, action is underrated.

Fortunately the number of things you can just hack together keeps increasing. People fifty years ago would be astonished that one could just hack together a movie, for example. Now you can even hack together distribution. Just make stuff and put it online.

Inappropriate

If you really want to score big, the place to focus is the margin of the margin: the territories only recently captured from the insiders. That's where you'll find the juiciest projects still undone, either because they seemed too risky, or simply because there were too few insiders to explore everything.

This is why I spend most of my time writing essays lately. The writing of essays used to be limited to those who could get them published. In principle you could have written them and just shown them to your friends; in practice that didn't work.* An essayist needs the resistance of an audience, just as an engraver needs the resistance of the plate.

Up till a few years ago, writing essays was the ultimate insider's game. Domain experts were allowed to publish essays about their field, but the pool allowed to write on general topics was about eight people who went to the right parties in New York. Now the reconquista has overrun this territory, and, not surprisingly, found it sparsely cultivated. There are so many essays yet unwritten. They

* Without the prospect of publication, the closest most people come to writing essays is to write in a journal. I find I never get as deeply into subjects as I do in proper essays. As the name implies, you don't go back and rewrite journal entries over and over for two weeks.

tend to be the naughtier ones; the insiders have pretty much exhausted the motherhood and apple pie topics.

This leads to my final suggestion: a technique for determining when you're on the right track. You're on the right track when people complain that you're unqualified, or that you've done something inappropriate. If people are complaining, that means you're doing something rather than sitting around, which is the first step. And if they're driven to such empty forms of complaint, that means you've probably done something good.

If you make something and people complain that it doesn't *work*, that's a problem. But if the worst thing they can hit you with is your own status as an outsider, that implies that in every other respect you've succeeded. Pointing out that someone is unqualified is as desperate as resorting to racial slurs. It's just a legitimate sounding way of saying: we don't like your type around here.

But the best thing of all is when people call what you're doing inappropriate. I've been hearing this word all my life and I only recently realized that it is, in fact, the sound of the homing beacon. "Inappropriate" is the null criticism. It's merely the adjective form of "I don't like it."

So that, I think, should be the highest goal for the marginal. Be inappropriate. When you hear people saying that, you're golden. And they, incidentally, are busted.

How to Start a Startup

BY PAUL GRAHAM
MARCH 2005

You need three things to create a successful startup: to start with good people, to make something customers actually want, and to spend as little money as possible. Most startups that fail do it because they fail at one of these. A startup that does all three will probably succeed.

And that's kind of exciting, when you think about it, because all three are doable. Hard, but doable. And since a startup that succeeds ordinarily makes its founders rich, that implies getting rich is doable too. Hard, but doable.

If there is one message I'd like to get across about startups, that's it. There is no magically difficult step that requires brilliance to solve.

The Idea

In particular, you don't need a brilliant idea to start a startup around. The way a startup makes money is to offer people better technology than they have now. But what people have now is often so bad that it doesn't take brilliance to do better.

Google's plan, for example, was simply to create a search site that didn't suck. They had three new ideas: index more of the Web, use links to rank search results, and have clean, simple web pages with unintrusive keyword-based ads. Above all, they were determined to

make a site that was good to use. No doubt there are great technical tricks within Google, but the overall plan was straightforward. And while they probably have bigger ambitions now, this alone brings them a billion dollars a year.*

There are plenty of other areas that are just as backward as search was before Google. I can think of several heuristics for generating ideas for startups, but most reduce to this: look at something people are trying to do, and figure out how to do it in a way that doesn't suck.

For example, dating sites currently suck far worse than search did before Google. They all use the same simple-minded model. They seem to have approached the problem by thinking about how to do database matches instead of how dating works in the real world. An undergrad could build something better as a class project. And yet there's a lot of money at stake. Online dating is a valuable business now, and it might be worth a hundred times as much if it worked.

An idea for a startup, however, is only a beginning. A lot of would-be startup founders think the key to the whole process is the initial idea, and from that point all you have to do is execute. Venture capitalists know better. If you go to VC firms with a brilliant idea that you'll tell them about if they sign a nondisclosure agreement, most will tell you to get lost. That shows how much a mere idea is worth. The market price is less than the inconvenience of signing an NDA.

Another sign of how little the initial idea is worth is the number of startups that change their plan en route. Microsoft's original plan was to make money selling programming languages, of all things. Their current business model didn't occur to them until IBM dropped it in their lap five years later.

Ideas for startups are worth something, certainly, but the trouble is, they're not transferrable. They're not something you could hand to someone else to execute. Their value is mainly as starting points: as questions for the people who had them to continue thinking about.

---

* Google's revenues are about two billion a year, but half comes from ads on other sites.

What matters is not ideas, but the people who have them. Good people can fix bad ideas, but good ideas can't save bad people.

People

What do I mean by good people? One of the best tricks I learned during our startup was a rule for deciding who to hire. Could you describe the person as an animal? It might be hard to translate that into another language, but I think everyone in the US knows what it means. It means someone who takes their work a little too seriously; someone who does what they do so well that they pass right through professional and cross over into obsessive.

What it means specifically depends on the job: a salesperson who just won't take no for an answer; a hacker who will stay up till 4:00 AM rather than go to bed leaving code with a bug in it; a PR person who will cold-call *New York Times* reporters on their cell phones; a graphic designer who feels physical pain when something is two millimeters out of place.

Almost everyone who worked for us was an animal at what they did. The woman in charge of sales was so tenacious that I used to feel sorry for potential customers on the phone with her. You could sense them squirming on the hook, but you knew there would be no rest for them till they'd signed up.

If you think about people you know, you'll find the animal test is easy to apply. Call the person's image to mind and imagine the sentence "so-and-so is an animal." If you laugh, they're not. You don't need or perhaps even want this quality in big companies, but you need it in a startup.

For programmers we had three additional tests. Was the person genuinely smart? If so, could they actually get things done? And finally, since a few good hackers have unbearable personalities, could we stand to have them around?

That last test filters out surprisingly few people. We could bear any amount of nerdiness if someone was truly smart. What we couldn't stand were people with a lot of attitude. But most of those weren't truly smart, so our third test was largely a restatement of the

first.

When nerds are unbearable it's usually because they're trying too hard to seem smart. But the smarter they are, the less pressure they feel to act smart. So as a rule you can recognize genuinely smart people by their ability to say things like "I don't know," "Maybe you're right," and "I don't understand x well enough."

This technique doesn't always work, because people can be influenced by their environment. In the MIT CS department, there seems to be a tradition of acting like a brusque know-it-all. I'm told it derives ultimately from Marvin Minsky, in the same way the classic airline pilot manner is said to derive from Chuck Yeager. Even genuinely smart people start to act this way there, so you have to make allowances.

It helped us to have Robert Morris, who is one of the readiest to say "I don't know" of anyone I've met. (At least, he was before he became a professor at MIT.) No one dared put on attitude around Robert, because he was obviously smarter than they were and yet had zero attitude himself.

Like most startups, ours began with a group of friends, and it was through personal contacts that we got most of the people we hired. This is a crucial difference between startups and big companies. Being friends with someone for even a couple days will tell you more than companies could ever learn in interviews.*

It's no coincidence that startups start around universities, because that's where smart people meet. It's not what people learn in classes at MIT and Stanford that has made technology companies spring up around them. They could sing campfire songs in the classes so long as admissions worked the same.

---

* One advantage startups have over established companies is that there are no discrimination laws about starting businesses. For example, I would be reluctant to start a startup with a woman who had small children, or was likely to have them soon. But you're not allowed to ask prospective employees if they plan to have kids soon. Believe it or not, under current US law, you're not even allowed to discriminate on the basis of intelligence. Whereas when you're starting a company, you can discriminate on any basis you want about who you start it with.

If you start a startup, there's a good chance it will be with people you know from college or grad school. So in theory you ought to try to make friends with as many smart people as you can in school, right? Well, no. Don't make a conscious effort to schmooze; that doesn't work well with hackers.

What you should do in college is work on your own projects. Hackers should do this even if they don't plan to start startups, because it's the only real way to learn how to program. In some cases you may collaborate with other students, and this is the best way to get to know good hackers. The project may even grow into a startup. But once again, I wouldn't aim too directly at either target. Don't force things; just work on stuff you like with people you like.

Ideally you want between two and four founders. It would be hard to start with just one. One person would find the moral weight of starting a company hard to bear. Even Bill Gates, who seems to be able to bear a good deal of moral weight, had to have a co-founder. But you don't want so many founders that the company starts to look like a group photo. Partly because you don't need a lot of people at first, but mainly because the more founders you have, the worse disagreements you'll have. When there are just two or three founders, you know you have to resolve disputes immediately or perish. If there are seven or eight, disagreements can linger and harden into factions. You don't want mere voting; you need unanimity.

In a technology startup, which most startups are, the founders should include technical people. During the Internet Bubble there were a number of startups founded by business people who then went looking for hackers to create their product for them. This doesn't work well. Business people are bad at deciding what to do with technology, because they don't know what the options are, or which kinds of problems are hard and which are easy. And when business people try to hire hackers, they can't tell which ones are good. Even other hackers have a hard time doing that. For business people it's roulette.

Do the founders of a startup have to include business people? That depends. We thought so when we started ours, and we asked

several people who were said to know about this mysterious thing called "business" if they would be the president. But they all said no, so I had to do it myself. And what I discovered was that business was no great mystery. It's not something like physics or medicine that requires extensive study. You just try to get people to pay you for stuff.

I think the reason I made such a mystery of business was that I was disgusted by the idea of doing it. I wanted to work in the pure, intellectual world of software, not deal with customers' mundane problems. People who don't want to get dragged into some kind of work often develop a protective incompetence at it. Paul Erdos was particularly good at this. By seeming unable even to cut a grapefruit in half (let alone go to the store and buy one), he forced other people to do such things for him, leaving all his time free for math. Erdos was an extreme case, but most husbands use the same trick to some degree.

Once I was forced to discard my protective incompetence, I found that business was neither so hard nor so boring as I feared. There are esoteric areas of business that are quite hard, like tax law or the pricing of derivatives, but you don't need to know about those in a startup. All you need to know about business to run a startup are commonsense things people knew before there were business schools, or even universities.

If you work your way down the Forbes 400 making an x next to the name of each person with an MBA, you'll learn something important about business school. After Warren Buffett, you don't hit another MBA till number 22, Phil Knight, the CEO of Nike. There are only 5 MBAs in the top 50. What you notice in the Forbes 400 are a lot of people with technical backgrounds. Bill Gates, Steve Jobs, Larry Ellison, Michael Dell, Jeff Bezos, Gordon Moore. The rulers of the technology business tend to come from technology, not business. So if you want to invest two years in something that will help you succeed in business, the evidence suggests you'd do better to learn how to hack than get an MBA.*

* Learning to hack is a lot cheaper than business school, because you can do it mostly on your own. For the price of a Linux box, a copy of K&R, and a few

There is one reason you might want to include business people in a startup, though: because you have to have at least one person willing and able to focus on what customers want. Some believe only business people can do this—that hackers can implement software, but not design it. That's nonsense. There's nothing about knowing how to program that prevents hackers from understanding users, or about not knowing how to program that magically enables business people to understand them.

If you can't understand users, however, you should either learn how or find a co-founder who can. That is the single most important issue for technology startups, and the rock that sinks more of them than anything else.

What Customers Want

It's not just startups that have to worry about this. I think most businesses that fail do it because they don't give customers what they want. Look at restaurants. A large percentage fail, about a quarter in the first year. But can you think of one restaurant that had really good food and went out of business?

Restaurants with great food seem to prosper no matter what. A restaurant with great food can be expensive, crowded, noisy, dingy, out of the way, and even have bad service, and people will keep coming. It's true that a restaurant with mediocre food can sometimes attract customers through gimmicks. But that approach is very risky. It's more straightforward just to make the food good.

It's the same with technology. You hear all kinds of reasons why startups fail. But can you think of one that had a massively popular product and still failed?

In nearly every failed startup, the real problem was that customers didn't want the product. For most, the cause of death is listed as "ran out of funding," but that's only the immediate cause. Why couldn't they get more funding? Probably because the product was a

---

hours of advice from your neighbor's fifteen year old son, you'll be well on your way.

dog, or never seemed likely to be done, or both.

When I was trying to think of the things every startup needed to do, I almost included a fourth: get a version 1 out as soon as you can. But I decided not to, because that's implicit in making something customers want. The only way to make something customers want is to get a prototype in front of them and refine it based on their reactions.

The other approach is what I call the "Hail Mary" strategy. You make elaborate plans for a product, hire a team of engineers to develop it (people who do this tend to use the term "engineer" for hackers), and then find after a year that you've spent two million dollars to develop something no one wants. This was not uncommon during the Bubble, especially in companies run by business types, who thought of software development as something terrifying that therefore had to be carefully planned.

We never even considered that approach. As a Lisp hacker, I come from the tradition of rapid prototyping. I would not claim (at least, not here) that this is the right way to write every program, but it's certainly the right way to write software for a startup. In a startup, your initial plans are almost certain to be wrong in some way, and your first priority should be to figure out where. The only way to do that is to try implementing them.

Like most startups, we changed our plan on the fly. At first we expected our customers to be Web consultants. But it turned out they didn't like us, because our software was easy to use and we hosted the site. It would be too easy for clients to fire them. We also thought we'd be able to sign up a lot of catalog companies, because selling online was a natural extension of their existing business. But in 1996 that was a hard sell. The middle managers we talked to at catalog companies saw the Web not as an opportunity, but as something that meant more work for them.

We did get a few of the more adventurous catalog companies. Among them was Frederick's of Hollywood, which gave us valuable experience dealing with heavy loads on our servers. But most of our users were small, individual merchants who saw the Web as an op-

portunity to build a business. Some had retail stores, but many only existed online. And so we changed direction to focus on these users. Instead of concentrating on the features Web consultants and catalog companies would want, we worked to make the software easy to use.

I learned something valuable from that. It's worth trying very, very hard to make technology easy to use. Hackers are so used to computers that they have no idea how horrifying software seems to normal people. Stephen Hawking's editor told him that every equation he included in his book would cut sales in half. When you work on making technology easier to use, you're riding that curve up instead of down. A 10% improvement in ease of use doesn't just increase your sales 10%. It's more likely to double your sales.

How do you figure out what customers want? Watch them. One of the best places to do this was at trade shows. Trade shows didn't pay as a way of getting new customers, but they were worth it as market research. We didn't just give canned presentations at trade shows. We used to show people how to build real, working stores. Which meant we got to watch as they used our software, and talk to them about what they needed.

No matter what kind of startup you start, it will probably be a stretch for you, the founders, to understand what users want. The only kind of software you can build without studying users is the sort for which you are the typical user. But this is just the kind that tends to be open source: operating systems, programming languages, editors, and so on. So if you're developing technology for money, you're probably not going to be developing it for people like you. Indeed, you can use this as a way to generate ideas for startups: what do people who are not like you want from technology?

When most people think of startups, they think of companies like Apple or Google. Everyone knows these, because they're big consumer brands. But for every startup like that, there are twenty more that operate in niche markets or live quietly down in the infrastructure. So if you start a successful startup, odds are you'll start one of those.

Another way to say that is, if you try to start the kind of startup

that has to be a big consumer brand, the odds against succeeding are steeper. The best odds are in niche markets. Since startups make money by offering people something better than they had before, the best opportunities are where things suck most. And it would be hard to find a place where things suck more than in corporate IT departments. You would not believe the amount of money companies spend on software, and the crap they get in return. This imbalance equals opportunity.

If you want ideas for startups, one of the most valuable things you could do is find a middle-sized non-technology company and spend a couple weeks just watching what they do with computers. Most good hackers have no more idea of the horrors perpetrated in these places than rich Americans do of what goes on in Brazilian slums.

Start by writing software for smaller companies, because it's easier to sell to them. It's worth so much to sell stuff to big companies that the people selling them the crap they currently use spend a lot of time and money to do it. And while you can outhack Oracle with one frontal lobe tied behind your back, you can't outsell an Oracle salesman. So if you want to win through better technology, aim at smaller customers.*

They're the more strategically valuable part of the market anyway. In technology, the low end always eats the high end. It's easier to make an inexpensive product more powerful than to make a powerful product cheaper. So the products that start as cheap, simple options tend to gradually grow more powerful till, like water rising in a room, they squash the "high-end" products against the ceiling. Sun did this to mainframes, and Intel is doing it to Sun. Microsoft Word did it to desktop publishing software like Interleaf and Framemaker. Mass-market digital cameras are doing it to the expensive models made for professionals. Avid did it to the manufacturers of specialized video editing systems, and now Apple is doing it to Avid. *Henry*

* Corollary: Avoid starting a startup to sell things to the biggest company of all, the government. Yes, there are lots of opportunities to sell them technology. But let someone else start those startups.

Ford did it to the car makers that preceded him. If you build the simple, inexpensive option, you'll not only find it easier to sell at first, but you'll also be in the best position to conquer the rest of the market.

It's very dangerous to let anyone fly under you. If you have the cheapest, easiest product, you'll own the low end. And if you don't, you're in the crosshairs of whoever does.

Raising Money

To make all this happen, you're going to need money. Some startups have been self-funding—Microsoft for example—but most aren't. I think it's wise to take money from investors. To be self-funding, you have to start as a consulting company, and it's hard to switch from that to a product company.

Financially, a startup is like a pass/fail course. The way to get rich from a startup is to maximize the company's chances of succeeding, not to maximize the amount of stock you retain. So if you can trade stock for something that improves your odds, it's probably a smart move.

To most hackers, getting investors seems like a terrifying and mysterious process. Actually it's merely tedious. I'll try to give an outline of how it works.

The first thing you'll need is a few tens of thousands of dollars to pay your expenses while you develop a prototype. This is called seed capital. Because so little money is involved, raising seed capital is comparatively easy—at least in the sense of getting a quick yes or no.

Usually you get seed money from individual rich people called "angels." Often they're people who themselves got rich from technology. At the seed stage, investors don't expect you to have an elaborate business plan. Most know that they're supposed to decide quickly. It's not unusual to get a check within a week based on a half-page agreement.

We started Viaweb with $10,000 of seed money from our friend Julian. But he gave us a lot more than money. He's a former CEO and also a corporate lawyer, so he gave us a lot of valuable advice about

business, and also did all the legal work of getting us set up as a company. Plus he introduced us to one of the two angel investors who supplied our next round of funding.

Some angels, especially those with technology backgrounds, may be satisfied with a demo and a verbal description of what you plan to do. But many will want a copy of your business plan, if only to remind themselves what they invested in.

Our angels asked for one, and looking back, I'm amazed how much worry it caused me. "Business plan" has that word "business" in it, so I figured it had to be something I'd have to read a book about business plans to write. Well, it doesn't. At this stage, all most investors expect is a brief description of what you plan to do and how you're going to make money from it, and the resumes of the founders. If you just sit down and write out what you've been saying to one another, that should be fine. It shouldn't take more than a couple hours, and you'll probably find that writing it all down gives you more ideas about what to do.

For the angel to have someone to make the check out to, you're going to have to have some kind of company. Merely incorporating yourselves isn't hard. The problem is, for the company to exist, you have to decide who the founders are, and how much stock they each have. If there are two founders with the same qualifications who are both equally committed to the business, that's easy. But if you have a number of people who are expected to contribute in varying degrees, arranging the proportions of stock can be hard. And once you've done it, it tends to be set in stone.

I have no tricks for dealing with this problem. All I can say is, try hard to do it right. I do have a rule of thumb for recognizing when you have, though. When everyone feels they're getting a slightly bad deal, that they're doing more than they should for the amount of stock they have, the stock is optimally apportioned.

There is more to setting up a company than incorporating it, of course: insurance, business license, unemployment compensation, various things with the IRS. I'm not even sure what the list is, because we, ah, skipped all that. When we got real funding near the end

of 1996, we hired a great CFO, who fixed everything retroactively. It turns out that no one comes and arrests you if you don't do everything you're supposed to when starting a company. And a good thing too, or a lot of startups would never get started.*

It can be dangerous to delay turning yourself into a company, because one or more of the founders might decide to split off and start another company doing the same thing. This does happen. So when you set up the company, as well as as apportioning the stock, you should get all the founders to sign something agreeing that everyone's ideas belong to this company, and that this company is going to be everyone's only job.

[If this were a movie, ominous music would begin here.]

While you're at it, you should ask what else they've signed. One of the worst things that can happen to a startup is to run into intellectual property problems. We did, and it came closer to killing us than any competitor ever did.

As we were in the middle of getting bought, we discovered that one of our people had, early on, been bound by an agreement that said all his ideas belonged to the giant company that was paying for him to go to grad school. In theory, that could have meant someone else owned big chunks of our software. So the acquisition came to a screeching halt while we tried to sort this out. The problem was, since we'd been about to be acquired, we'd allowed ourselves to run low on cash. Now we needed to raise more to keep going. But it's hard to raise money with an IP cloud over your head, because investors can't judge how serious it is.

Our existing investors, knowing that we needed money and had nowhere else to get it, at this point attempted certain gambits which I will not describe in detail, except to remind readers that the word "angel" is a metaphor. The founders thereupon proposed to walk away from the company, after giving the investors a brief tutorial on how to administer the servers themselves. And while this was hap-

* A friend who started a company in Germany told me they do care about the paperwork there, and that there's more of it. Which helps explain why there are not more startups in Germany.

pening, the acquirers used the delay as an excuse to welch on the deal.

Miraculously it all turned out ok. The investors backed down; we did another round of funding at a reasonable valuation; the giant company finally gave us a piece of paper saying they didn't own our software; and six months later we were bought by Yahoo for much more than the earlier acquirer had agreed to pay. So we were happy in the end, though the experience probably took several years off my life.

Don't do what we did. Before you consummate a startup, ask everyone about their previous IP history.

Once you've got a company set up, it may seem presumptuous to go knocking on the doors of rich people and asking them to invest tens of thousands of dollars in something that is really just a bunch of guys with some ideas. But when you look at it from the rich people's point of view, the picture is more encouraging. Most rich people are looking for good investments. If you really think you have a chance of succeeding, you're doing them a favor by letting them invest. Mixed with any annoyance they might feel about being approached will be the thought: are these guys the next Google?

Usually angels are financially equivalent to founders. They get the same kind of stock and get diluted the same amount in future rounds. How much stock should they get? That depends on how ambitious you feel. When you offer x percent of your company for y dollars, you're implicitly claiming a certain value for the whole company. Venture investments are usually described in terms of that number. If you give an investor new shares equal to 5% of those already outstanding in return for $100,000, then you've done the deal at a pre-money valuation of $2 million.

How do you decide what the value of the company should be? There is no rational way. At this stage the company is just a bet. I didn't realize that when we were raising money. Julian thought we ought to value the company at several million dollars. I thought it was preposterous to claim that a couple thousand lines of code, which was all we had at the time, were worth several million dollars.

Eventually we settled on one million, because Julian said no one would invest in a company with a valuation any lower.*

What I didn't grasp at the time was that the valuation wasn't just the value of the code we'd written so far. It was also the value of our ideas, which turned out to be right, and of all the future work we'd do, which turned out to be a lot.

The next round of funding is the one in which you might deal with actual venture capital firms. But don't wait till you've burned through your last round of funding to start approaching them. VCs are slow to make up their minds. They can take months. You don't want to be running out of money while you're trying to negotiate with them.

Getting money from an actual VC firm is a bigger deal than getting money from angels. The amounts of money involved are larger, millions usually. So the deals take longer, dilute you more, and impose more onerous conditions.

Sometimes the VCs want to install a new CEO of their own choosing. Usually the claim is that you need someone mature and experienced, with a business background. Maybe in some cases this is true. And yet Bill Gates was young and inexperienced and had no business background, and he seems to have done ok. Steve Jobs got booted out of his own company by someone mature and experienced, with a business background, who then proceeded to ruin the company. So I think people who are mature and experienced, with a business background, may be overrated. We used to call these guys "newscasters," because they had neat hair and spoke in deep, confident voices, and generally didn't know much more than they read on the teleprompter.

We talked to a number of VCs, but eventually we ended up financing our startup entirely with angel money. The main reason was that we feared a brand-name VC firm would stick us with a newscaster as part of the deal. That might have been ok if he was content

* At the seed stage our valuation was in principle $100,000, because Julian got 10% of the company. But this is a very misleading number, because the money was the least important of the things Julian gave us.

to limit himself to talking to the press, but what if he wanted to have a say in running the company? That would have led to disaster, because our software was so complex. We were a company whose whole m.o. was to win through better technology. The strategic decisions were mostly decisions about technology, and we didn't need any help with those.

This was also one reason we didn't go public. Back in 1998 our CFO tried to talk me into it. In those days you could go public as a dogfood portal, so as a company with a real product and real revenues, we might have done well. But I feared it would have meant taking on a newscaster—someone who, as they say, "can talk Wall Street's language."

I'm happy to see Google is bucking that trend. They didn't talk Wall Street's language when they did their IPO, and Wall Street didn't buy. And now Wall Street is collectively kicking itself. They'll pay attention next time. Wall Street learns new languages fast when money is involved.

You have more leverage negotiating with VCs than you realize. The reason is other VCs. I know a number of VCs now, and when you talk to them you realize that it's a seller's market. Even now there is too much money chasing too few good deals.

VCs form a pyramid. At the top are famous ones like Sequoia and Kleiner Perkins, but beneath those are a huge number you've never heard of. What they all have in common is that a dollar from them is worth one dollar. Most VCs will tell you that they don't just provide money, but connections and advice. If you're talking to Vinod Khosla or John Doerr or Mike Moritz, this is true. But such advice and connections can come very expensive. And as you go down the food chain the VCs get rapidly dumber. A few steps down from the top you're basically talking to bankers who've picked up a few new vocabulary words from reading *Wired*. (Does your product use *XML?*) So I'd advise you to be skeptical about claims of experience and connections. Basically, a VC is a source of money. I'd be inclined to go with whoever offered the most money the soonest with the least strings attached.

You may wonder how much to tell VCs. And you should, because some of them may one day be funding your competitors. I think the best plan is not to be overtly secretive, but not to tell them everything either. After all, as most VCs say, they're more interested in the people than the ideas. The main reason they want to talk about your idea is to judge you, not the idea. So as long as you seem like you know what you're doing, you can probably keep a few things back from them.*

Talk to as many VCs as you can, even if you don't want their money, because a) they may be on the board of someone who will buy you, and b) if you seem impressive, they'll be discouraged from investing in your competitors. The most efficient way to reach VCs, especially if you only want them to know about you and don't want their money, is at the conferences that are occasionally organized for startups to present to them.

Not Spending It

When and if you get an infusion of real money from investors, what should you do with it? Not spend it, that's what. In nearly every startup that fails, the proximate cause is running out of money. Usually there is something deeper wrong. But even a proximate cause of death is worth trying hard to avoid.

During the Bubble many startups tried to "get big fast." Ideally this meant getting a lot of customers fast. But it was easy for the meaning to slide over into hiring a lot of people fast.

Of the two versions, the one where you get a lot of customers fast is of course preferable. But even that may be overrated. The idea is to get there first and get all the users, leaving none for competitors. But I think in most businesses the advantages of being first to market are not so overwhelmingly great. Google is again a case in point. When

* The same goes for companies that seem to want to acquire you. There will be a few that are only pretending to in order to pick your brains. But you can never tell for sure which these are, so the best approach is to seem entirely open, but to fail to mention a few critical technical secrets.

they appeared it seemed as if search was a mature market, dominated by big players who'd spent millions to build their brands: Yahoo, Lycos, Excite, Infoseek, Altavista, Inktomi. Surely 1998 was a little late to arrive at the party.

But as the founders of Google knew, brand is worth next to nothing in the search business. You can come along at any point and make something better, and users will gradually seep over to you. As if to emphasize the point, Google never did any advertising. They're like dealers; they sell the stuff, but they know better than to use it themselves.

The competitors Google buried would have done better to spend those millions improving their software. Future startups should learn from that mistake. Unless you're in a market where products are as undifferentiated as cigarettes or vodka or laundry detergent, spending a lot on brand advertising is a sign of breakage. And few if any Web businesses are so undifferentiated. The dating sites are running big ad campaigns right now, which is all the more evidence they're ripe for the picking. (Fee, fie, fo, fum, I smell a company run by marketing guys.)

We were compelled by circumstances to grow slowly, and in retrospect it was a good thing. The founders all learned to do every job in the company. As well as writing software, I had to do sales and customer support. At sales I was not very good. I was persistent, but I didn't have the smoothness of a good salesman. My message to potential customers was: you'd be stupid not to sell online, and if you sell online you'd be stupid to use anyone else's software. Both statements were true, but that's not the way to convince people.

I was great at customer support though. Imagine talking to a customer support person who not only knew everything about the product, but would apologize abjectly if there was a bug, and then fix it immediately, while you were on the phone with them. Customers loved us. And we loved them, because when you're growing slow by word of mouth, your first batch of users are the ones who were smart enough to find you by themselves. There is nothing more valuable, in the early stages of a startup, than smart users. If you listen to them,

they'll tell you exactly how to make a winning product. And not only will they give you this advice for free, they'll pay you.

We officially launched in early 1996. By the end of that year we had about 70 users. Since this was the era of "get big fast," I worried about how small and obscure we were. But in fact we were doing exactly the right thing. Once you get big (in users or employees) it gets hard to change your product. That year was effectively a laboratory for improving our software. By the end of it, we were so far ahead of our competitors that they never had a hope of catching up. And since all the hackers had spent many hours talking to users, we understood online commerce way better than anyone else.

That's the key to success as a startup. There is nothing more important than understanding your business. You might think that anyone in a business must, ex officio, understand it. Far from it. Google's secret weapon was simply that they understood search. I was working for Yahoo when Google appeared, and Yahoo didn't understand search. I know because I once tried to convince the powers that be that we had to make search better, and I got in reply what was then the party line about it: that Yahoo was no longer a mere "search engine." Search was now only a small percentage of our page views, less than one month's growth, and now that we were established as a "media company," or "portal," or whatever we were, search could safely be allowed to wither and drop off, like an umbilical cord.

Well, a small fraction of page views they may be, but they are an important fraction, because they are the page views that Web sessions start with. I think Yahoo gets that now.

Google understands a few other things most Web companies still don't. The most important is that you should put users before advertisers, even though the advertisers are paying and users aren't. One of my favorite bumper stickers reads "if the people lead, the leaders will follow." Paraphrased for the Web, this becomes "get all the users, and the advertisers will follow." More generally, design your product to please users first, and then think about how to make money from it. If you don't put users first, you leave a gap for competitors who do.

To make something users love, you have to understand them.

And the bigger you are, the harder that is. So I say "get big slow." The slower you burn through your funding, the more time you have to learn.

The other reason to spend money slowly is to encourage a culture of cheapness. That's something Yahoo did understand. David Filo's title was "Chief Yahoo," but he was proud that his unofficial title was "Cheap Yahoo." Soon after we arrived at Yahoo, we got an email from Filo, who had been crawling around our directory hierarchy, asking if it was really necessary to store so much of our data on expensive RAID drives. I was impressed by that. Yahoo's market cap then was already in the billions, and they were still worrying about wasting a few gigs of disk space.

When you get a couple million dollars from a VC firm, you tend to feel rich. It's important to realize you're not. A rich company is one with large revenues. This money isn't revenue. It's money investors have given you in the hope you'll be able to generate revenues. So despite those millions in the bank, you're still poor.

For most startups the model should be grad student, not law firm. Aim for cool and cheap, not expensive and impressive. For us the test of whether a startup understood this was whether they had Aeron chairs. The Aeron came out during the Bubble and was very popular with startups. Especially the type, all too common then, that was like a bunch of kids playing house with money supplied by VCs. We had office chairs so cheap that the arms all fell off. This was slightly embarrassing at the time, but in retrospect the grad-studenty atmosphere of our office was another of those things we did right without knowing it.

Our offices were in a wooden triple-decker in Harvard Square. It had been an apartment until about the 1970s, and there was still a claw-footed bathtub in the bathroom. It must once have been inhabited by someone fairly eccentric, because a lot of the chinks in the walls were stuffed with aluminum foil, as if to protect against cosmic rays. When eminent visitors came to see us, we were a bit sheepish about the low production values. But in fact that place was the perfect space for a startup. We felt like our role was to be impudent un-

derdogs instead of corporate stuffed shirts, and that is exactly the spirit you want.

An apartment is also the right kind of place for developing software. Cube farms suck for that, as you've probably discovered if you've tried it. Ever notice how much easier it is to hack at home than at work? So why not make work more like home?

When you're looking for space for a startup, don't feel that it has to look professional. Professional means doing good work, not elevators and glass walls. I'd advise most startups to avoid corporate space at first and just rent an apartment. You want to live at the office in a startup, so why not have a place designed to be lived in as your office?

Besides being cheaper and better to work in, apartments tend to be in better locations than office buildings. And for a startup location is very important. The key to productivity is for people to come back to work after dinner. Those hours after the phone stops ringing are by far the best for getting work done. Great things happen when a group of employees go out to dinner together, talk over ideas, and then come back to their offices to implement them. So you want to be in a place where there are a lot of restaurants around, not some dreary office park that's a wasteland after 6:00 PM. Once a company shifts over into the model where everyone drives home to the suburbs for dinner, however late, you've lost something extraordinarily valuable. God help you if you actually start in that mode.

If I were going to start a startup today, there are only three places I'd consider doing it: on the Red Line near Central, Harvard, or Davis Squares (Kendall is too sterile); in Palo Alto on University or California Aves; and in Berkeley immediately north or south of campus. These are the only places I know that have the right kind of vibe.

The most important way to not spend money is by not hiring people. I may be an extremist, but I think hiring people is the worst thing a company can do. To start with, people are a recurring expense, which is the worst kind. They also tend to cause you to grow out of your space, and perhaps even move to the sort of uncool office building that will make your software worse. But worst of all, they

slow you down: instead of sticking your head in someone's office and checking out an idea with them, eight people have to have a meeting about it. So the fewer people you can hire, the better.

During the Bubble a lot of startups had the opposite policy. They wanted to get "staffed up" as soon as possible, as if you couldn't get anything done unless there was someone with the corresponding job title. That's big company thinking. Don't hire people to fill the gaps in some a priori org chart. The only reason to hire someone is to do something you'd like to do but can't.

If hiring unnecessary people is expensive and slows you down, why do nearly all companies do it? I think the main reason is that people like the idea of having a lot of people working for them. This weakness often extends right up to the CEO. If you ever end up running a company, you'll find the most common question people ask is how many employees you have. This is their way of weighing you. It's not just random people who ask this; even reporters do. And they're going to be a lot more impressed if the answer is a thousand than if it's ten.

This is ridiculous, really. If two companies have the same revenues, it's the one with fewer employees that's more impressive. When people used to ask me how many people our startup had, and I answered "twenty," I could see them thinking that we didn't count for much. I used to want to add "but our main competitor, whose ass we regularly kick, has a hundred and forty, so can we have credit for the larger of the two numbers?"

As with office space, the number of your employees is a choice between seeming impressive, and being impressive. Any of you who were nerds in high school know about this choice. Keep doing it when you start a company.

Should You?

But should you start a company? Are you the right sort of person to do it? If you are, is it worth it?

More people are the right sort of person to start a startup than realize it. That's the main reason I wrote this. There could be ten

times more startups than there are, and that would probably be a good thing.

I was, I now realize, exactly the right sort of person to start a startup. But the idea terrified me at first. I was forced into it because I was a Lisp hacker. The company I'd been consulting for seemed to be running into trouble, and there were not a lot of other companies using Lisp. Since I couldn't bear the thought of programming in another language (this was 1995, remember, when "another language" meant C++) the only option seemed to be to start a new company using Lisp.

I realize this sounds far-fetched, but if you're a Lisp hacker you'll know what I mean. And if the idea of starting a startup frightened me so much that I only did it out of necessity, there must be a lot of people who would be good at it but who are too intimidated to try.

So who should start a startup? Someone who is a good hacker, between about 23 and 38, and who wants to solve the money problem in one shot instead of getting paid gradually over a conventional working life.

I can't say precisely what a good hacker is. At a first rate university this might include the top half of computer science majors. Though of course you don't have to be a CS major to be a hacker; I was a philosophy major in college.

It's hard to tell whether you're a good hacker, especially when you're young. Fortunately the process of starting startups tends to select them automatically. What drives people to start startups is (or should be) looking at existing technology and thinking, don't these guys realize they should be doing x, y, and z? And that's also a sign that one is a good hacker.

I put the lower bound at 23 not because there's something that doesn't happen to your brain till then, but because you need to see what it's like in an existing business before you try running your own. The business doesn't have to be a startup. I spent a year working for a software company to pay off my college loans. It was the worst year of my adult life, but I learned, without realizing it at the time, a lot of valuable lessons about the software business. In this

case they were mostly negative lessons: don't have a lot of meetings; don't have chunks of code that multiple people own; don't have a sales guy running the company; don't make a high-end product; don't let your code get too big; don't leave finding bugs to QA people; don't go too long between releases; don't isolate developers from users; don't move from Cambridge to Route 128; and so on.* But negative lessons are just as valuable as positive ones. Perhaps even more valuable: it's hard to repeat a brilliant performance, but it's straightforward to avoid errors.†

The other reason it's hard to start a company before 23 is that people won't take you seriously. VCs won't trust you, and will try to reduce you to a mascot as a condition of funding. Customers will worry you're going to flake out and leave them stranded. Even you yourself, unless you're very unusual, will feel your age to some degree; you'll find it awkward to be the boss of someone much older than you, and if you're 21, hiring only people younger rather limits your options.

Some people could probably start a company at 18 if they wanted to. Bill Gates was 19 when he and Paul Allen started Microsoft. (Paul Allen was 22, though, and that probably made a difference.) So if you're thinking, I don't care what he says, I'm going to start a company now, you may be the sort of person who could get away with it.

The other cutoff, 38, has a lot more play in it. One reason I put it there is that I don't think many people have the physical stamina much past that age. I used to work till 2:00 or 3:00 AM every night, seven days a week. I don't know if I could do that now.

Also, startups are a big risk financially. If you try something that blows up and leaves you broke at 26, big deal; a lot of 26 year olds are broke. By 38 you can't take so many risks—especially if you have kids.

My final test may be the most restrictive. Do you actually want to

---

* I was as bad an employee as this place was a company. I apologize to anyone who had to work with me there.

† You could probably write a book about how to succeed in business by doing everything in exactly the opposite way from the DMV.

start a startup? What it amounts to, economically, is compressing your working life into the smallest possible space. Instead of working at an ordinary rate for 40 years, you work like hell for four. And maybe end up with nothing—though in that case it probably won't take four years.

During this time you'll do little but work, because when you're not working, your competitors will be. My only leisure activities were running, which I needed to do to keep working anyway, and about fifteen minutes of reading a night. I had a girlfriend for a total of two months during that three year period. Every couple weeks I would take a few hours off to visit a used bookshop or go to a friend's house for dinner. I went to visit my family twice. Otherwise I just worked.

Working was often fun, because the people I worked with were some of my best friends. Sometimes it was even technically interesting. But only about 10% of the time. The best I can say for the other 90% is that some of it is funnier in hindsight than it seemed then. Like the time the power went off in Cambridge for about six hours, and we made the mistake of trying to start a gasoline powered generator inside our offices. I won't try that again.

I don't think the amount of bullshit you have to deal with in a startup is more than you'd endure in an ordinary working life. It's probably less, in fact; it just seems like a lot because it's compressed into a short period. So mainly what a startup buys you is time. That's the way to think about it if you're trying to decide whether to start one. If you're the sort of person who would like to solve the money problem once and for all instead of working for a salary for 40 years, then a startup makes sense.

For a lot of people the conflict is between startups and graduate school. Grad students are just the age, and just the sort of people, to start software startups. You may worry that if you do you'll blow your chances of an academic career. But it's possible to be part of a startup and stay in grad school, especially at first. Two of our three original hackers were in grad school the whole time, and both got their degrees. There are few sources of energy so powerful as a procrastinating grad student.

If you do have to leave grad school, in the worst case it won't be for too long. If a startup fails, it will probably fail quickly enough that you can return to academic life. And if it succeeds, you may find you no longer have such a burning desire to be an assistant professor.

If you want to do it, do it. Starting a startup is not the great mystery it seems from outside. It's not something you have to know about "business" to do. Build something users love, and spend less than you make. How hard is that?

Why to Not Not Start a Startup

BY PAUL GRAHAM
MARCH 2007

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.* 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

* 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.

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. *

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

* 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.

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? *

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

* There need to be some number of big companies to acquire the startups, so the number of big companies couldn't decrease to zero.

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 ev-

idence: 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.* But not so lucrative or prestigious as it was when their opinions were formed. When I was a

* Thought experiment: If doctors did the same work, but as impoverished outcasts, which parents would still want their kids to be doctors?

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.

How to Get Startup Ideas

BY PAUL GRAHAM
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.*

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.†

---

* 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.

† I'm not saying, incidentally, that I know for sure a social network for pet owners

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

---

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.

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.*

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

* 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.

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.*

You don't have to learn programming to be at the leading edge of a domain that's changing fast. Other domains change fast. But while learning to hack is not necessary, it is for the forseeable future sufficient. As Marc Andreessen put it, software is eating the world, and this trend has decades left to run.

Knowing how to hack also means that when you have ideas, you'll be able to implement them. That's not absolutely necessary (Jeff Bezos couldn't) but it's an advantage. It's a big advantage, when you're considering an idea like putting a college facebook online, if instead of merely thinking "That's an interesting idea," you can think instead "That's an interesting idea. I'll try building an initial version tonight." It's even better when you're both a programmer and the target user, because then the cycle of generating new versions and testing them on users can happen inside one head.

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

---

* 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.

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 life 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.*

* It was also obvious to us that the web was going to be a big deal. Few nonprogrammers grasped that in 1995, but the programmers had seen what GUIs had done for desktop computers.

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 curiousity, but have a second self watching over your shoulder, taking note of gaps and anomalies.*

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.†

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

---

* 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.

† 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.

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. *

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.†

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

* 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.

† 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.

be something that could be turned into a startup.* 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. †

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

---

* I'm speaking here of IT startups; in biotech things are different.

† 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.

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). *

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

---

* 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.

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 fir-

ing 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. *

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.

---

* 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.

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. *

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. †

---

* 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.

† 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

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. *

---

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."

* 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.

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. *

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

* 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.

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.

IDEAS FOR STARTUPS

October 2005

How do you get good ideas for startups? That's probably the number one question people ask me.

I'd like to reply with another question: why do people think it's hard to come up with ideas for startups?

That might seem a stupid thing to ask. Why do they *think* it's hard? If people can't do it, then it *is* hard, at least for them. Right?

Well, maybe not. What people usually say is not that they can't think of ideas, but that they don't have any. That's not quite the same thing. It could be the reason they don't have any is that they haven't tried to generate them.

I think this is often the case. I think people believe that coming up with ideas for startups is very hard—that it *must* be very hard—and so they don't try do to it. They assume ideas are like miracles: they either pop into your head or they don't.

I also have a theory about why people think this. They overvalue ideas. They think creating a startup is just a matter of implementing some fabulous initial idea. And since a successful startup is worth millions of dollars, a good idea is therefore a million dollar idea.

If coming up with an idea for a startup equals coming up with a million dollar idea, then of course it's going to seem hard. Too hard to bother trying. Our instincts tell us something so valuable would not be just lying around for anyone to discover.

Actually, startup ideas are not million dollar ideas, and here's an experiment you can try to prove it: just try to sell one. Nothing evolves faster than markets. The fact that there's no market for

startup ideas suggests there's no demand. Which means, in the narrow sense of the word, that startup ideas are worthless.

Questions

The fact is, most startups end up nothing like the initial idea. It would be closer to the truth to say the main value of your initial idea is that, in the process of discovering it's broken, you'll come up with your real idea.

The initial idea is just a starting point—not a blueprint, but a question. It might help if they were expressed that way. Instead of saying that your idea is to make a collaborative, web-based spreadsheet, say: could one make a collaborative, web-based spreadsheet? A few grammatical tweaks, and a woefully incomplete idea becomes a promising question to explore.

There's a real difference, because an assertion provokes objections in a way a question doesn't. If you say: I'm going to build a web-based spreadsheet, then critics—the most dangerous of which are in your own head—will immediately reply that you'd be competing with Microsoft, that you couldn't give people the kind of UI they expect, that users wouldn't want to have their data on your servers, and so on.

A question doesn't seem so challenging. It becomes: let's try making a web-based spreadsheet and see how far we get. And everyone knows that if you tried this you'd be able to make *something* useful. Maybe what you'd end up with wouldn't even be a spreadsheet. Maybe it would be some kind of new spreasheet-like collaboration tool that doesn't even have a name yet. You wouldn't have thought of something like that except by implementing your way toward it.

Treating a startup idea as a question changes what you're looking for. If an idea is a blueprint, it has to be right. But if it's a question, it can be wrong, so long as it's wrong in a way that leads to more ideas.

One valuable way for an idea to be wrong is to be only a partial solution. When someone's working on a problem that seems too big, I always ask: is there some way to bite off some subset of the problem, then gradually expand from there? That will generally work un-

less you get trapped on a local maximum, like 1980s-style AI, or C.

Upwind

So far, we've reduced the problem from thinking of a million dollar idea to thinking of a mistaken question. That doesn't seem so hard, does it?

To generate such questions you need two things: to be familiar with promising new technologies, and to have the right kind of friends. New technologies are the ingredients startup ideas are made of, and conversations with friends are the kitchen they're cooked in.

Universities have both, and that's why so many startups grow out of them. They're filled with new technologies, because they're trying to produce research, and only things that are new count as research. And they're full of exactly the right kind of people to have ideas with: the other students, who will be not only smart but elastic-minded to a fault.

The opposite extreme would be a well-paying but boring job at a big company. Big companies are biased against new technologies, and the people you'd meet there would be wrong too.

In an essay I wrote for high school students, I said a good rule of thumb was to stay upwind—to work on things that maximize your future options. The principle applies for adults too, though perhaps it has to be modified to: stay upwind for as long as you can, then cash in the potential energy you've accumulated when you need to pay for kids.

I don't think people consciously realize this, but one reason downwind jobs like churning out Java for a bank pay so well is precisely that they are downwind. The market price for that kind of work is higher because it gives you fewer options for the future. A job that lets you work on exciting new stuff will tend to pay less, because part of the compensation is in the form of the new skills you'll learn.

Grad school is the other end of the spectrum from a coding job at a big company: the pay's low but you spend most of your time working on new stuff. And of course, it's called "school," which

makes that clear to everyone, though in fact all jobs are some percentage school.

The right environment for having startup ideas need not be a university per se. It just has to be a situation with a large percentage of school.

It's obvious why you want exposure to new technology, but why do you need other people? Can't you just think of new ideas yourself? The empirical answer is: no. Even Einstein needed people to bounce ideas off. Ideas get developed in the process of explaining them to the right kind of person. You need that resistance, just as a carver needs the resistance of the wood.

This is one reason Y Combinator has a rule against investing in startups with only one founder. Practically every successful company has at least two. And because startup founders work under great pressure, it's critical they be friends.

I didn't realize it till I was writing this, but that may help explain why there are so few female startup founders. I read on the Internet (so it must be true) that only 1.7% of VC-backed startups are founded by women. The percentage of female hackers is small, but not that small. So why the discrepancy?

When you realize that successful startups tend to have multiple founders who were already friends, a possible explanation emerges. People's best friends are likely to be of the same sex, and if one group is a minority in some population, *pairs* of them will be a minority squared.*

Doodling

What these groups of co-founders do together is more complicated than just sitting down and trying to think of ideas. I suspect the most productive setup is a kind of together-alone-together sandwich. Together you talk about some hard problem, probably getting nowhere.

---

* This phenomenon may account for a number of discrepancies currently blamed on various forbidden isms. Never attribute to malice what can be explained by math.

Then, the next morning, one of you has an idea in the shower about how to solve it. He runs eagerly to tell the others, and together they work out the kinks.

What happens in that shower? It seems to me that ideas just pop into my head. But can we say more than that?

Taking a shower is like a form of meditation. You're alert, but there's nothing to distract you. It's in a situation like this, where your mind is free to roam, that it bumps into new ideas.

What happens when your mind wanders? It may be like doodling. Most people have characteristic ways of doodling. This habit is unconscious, but not random: I found my doodles changed after I started studying painting. I started to make the kind of gestures I'd make if I were drawing from life. They were atoms of drawing, but arranged randomly.*

Perhaps letting your mind wander is like doodling with ideas. You have certain mental gestures you've learned in your work, and when you're not paying attention, you keep making these same gestures, but somewhat randomly. In effect, you call the same functions on random arguments. That's what a metaphor is: a function applied to an argument of the wrong type.

Conveniently, as I was writing this, my mind wandered: would it be useful to have metaphors in a programming language? I don't know; I don't have time to think about this. But it's convenient because this is an example of what I mean by habits of mind. I spend a lot of time thinking about language design, and my habit of always asking "would x be useful in a programming language" just got invoked.

If new ideas arise like doodles, this would explain why you have to work at something for a while before you have any. It's not just that you can't judge ideas till you're an expert in a field. You won't even generate ideas, because you won't have any habits of mind to invoke.

---

* A lot of classic abstract expressionism is doodling of this type: artists trained to paint from life using the same gestures but without using them to represent anything. This explains why such paintings are (slightly) more interesting than random marks would be.

Of course the habits of mind you invoke on some field don't have to be derived from working in that field. In fact, it's often better if they're not. You're not just looking for good ideas, but for good *new* ideas, and you have a better chance of generating those if you combine stuff from distant fields. As hackers, one of our habits of mind is to ask, could one open-source x? For example, what if you made an open-source operating system? A fine idea, but not very novel. Whereas if you ask, could you make an open-source play? you might be onto something.

Are some kinds of work better sources of habits of mind than others? I suspect harder fields may be better sources, because to attack hard problems you need powerful solvents. I find math is a good source of metaphors—good enough that it's worth studying just for that. Related fields are also good sources, especially when they're related in unexpected ways. Everyone knows computer science and electrical engineering are related, but precisely because everyone knows it, importing ideas from one to the other doesn't yield great profits. It's like importing something from Wisconsin to Michigan. Whereas (I claim) hacking and painting are also related, in the sense that hackers and painters are both makers, and this source of new ideas is practically virgin territory.

Problems

In theory you could stick together ideas at random and see what you came up with. What if you built a peer-to-peer dating site? Would it be useful to have an automatic book? Could you turn theorems into a commodity? When you assemble ideas at random like this, they may not be just stupid, but semantically ill-formed. What would it even mean to make theorems a commodity? You got me. I didn't think of that idea, just its name.

You might come up with something useful this way, but I never have. It's like knowing a fabulous sculpture is hidden inside a block of marble, and all you have to do is remove the marble that isn't part of it. It's an encouraging thought, because it reminds you there is an answer, but it's not much use in practice because the search space is

too big.

I find that to have good ideas I need to be working on some problem. You can't start with randomness. You have to start with a problem, then let your mind wander just far enough for new ideas to form.

In a way, it's harder to see problems than their solutions. Most people prefer to remain in denial about problems. It's obvious why: problems are irritating. They're problems! Imagine if people in 1700 saw their lives the way we'd see them. It would have been unbearable. This denial is such a powerful force that, even when presented with possible solutions, people often prefer to believe they wouldn't work.

I saw this phenomenon when I worked on spam filters. In 2002, most people preferred to ignore spam, and most of those who didn't preferred to believe the heuristic filters then available were the best you could do.

I found spam intolerable, and I felt it had to be possible to recognize it statistically. And it turns out that was all you needed to solve the problem. The algorithm I used was ridiculously simple. Anyone who'd really tried to solve the problem would have found it. It was just that no one had really tried to solve the problem.*

Let me repeat that recipe: finding the problem intolerable and feeling it must be possible to solve it. Simple as it seems, that's the recipe for a lot of startup ideas.

Wealth

So far most of what I've said applies to ideas in general. What's special about startup ideas? Startup ideas are ideas for companies, and companies have to make money. And the way to make money is to make something people want.

Wealth is what people want. I don't mean that as some kind of philosophical statement; I mean it as a tautology.

So an idea for a startup is an idea for something people want.

---

* Bill Yerazunis had solved the problem, but he got there by another path. He made a general-purpose file classifier so good that it also worked for spam.

Wouldn't any good idea be something people want? Unfortunately not. I think new theorems are a fine thing to create, but there is no great demand for them. Whereas there appears to be great demand for celebrity gossip magazines. Wealth is defined democratically. Good ideas and valuable ideas are not quite the same thing; the difference is individual tastes.

But valuable ideas are very close to good ideas, especially in technology. I think they're so close that you can get away with working as if the goal were to discover good ideas, so long as, in the final stage, you stop and ask: will people actually pay for this? Only a few ideas are likely to make it that far and then get shot down; RPN calculators might be one example.

One way to make something people want is to look at stuff people use now that's broken. Dating sites are a prime example. They have millions of users, so they must be promising something people want. And yet they work horribly. Just ask anyone who uses them. It's as if they used the worse-is-better approach but stopped after the first stage and handed the thing over to marketers.

Of course, the most obvious breakage in the average computer user's life is Windows itself. But this is a special case: you can't defeat a monopoly by a frontal attack. Windows can and will be overthrown, but not by giving people a better desktop OS. The way to kill it is to redefine the problem as a superset of the current one. The problem is not, what operating system should people use on desktop computers? but how should people use applications? There are answers to that question that don't even involve desktop computers.

Everyone thinks Google is going to solve this problem, but it is a very subtle one, so subtle that a company as big as Google might well get it wrong. I think the odds are better than 50-50 that the Windows killer—or more accurately, Windows transcender—will come from some little startup.

Another classic way to make something people want is to take a luxury and make it into a commodity. People must want something if they pay a lot for it. And it is a very rare product that can't be made dramatically cheaper if you try.

This was Henry Ford's plan. He made cars, which had been a luxury item, into a commodity. But the idea is much older than Henry Ford. Water mills transformed mechanical power from a luxury into a commodity, and they were used in the Roman empire. Arguably pastoralism transformed a luxury into a commodity.

When you make something cheaper you can sell more of them. But if you make something dramatically cheaper you often get qualitative changes, because people start to use it in different ways. For example, once computers get so cheap that most people can have one of their own, you can use them as communication devices.

Often to make something dramatically cheaper you have to redefine the problem. The Model T didn't have all the features previous cars did. It only came in black, for example. But it solved the problem people cared most about, which was getting from place to place.

One of the most useful mental habits I know I learned from Michael Rabin: that the best way to solve a problem is often to redefine it. A lot of people use this technique without being consciously aware of it, but Rabin was spectacularly explicit. You need a big prime number? Those are pretty expensive. How about if I give you a big number that only has a 10 to the minus 100 chance of not being prime? Would that do? Well, probably; I mean, that's probably smaller than the chance that I'm imagining all this anyway.

Redefining the problem is a particularly juicy heuristic when you have competitors, because it's so hard for rigid-minded people to follow. You can work in plain sight and they don't realize the danger. Don't worry about us. We're just working on search. Do one thing and do it well, that's our motto.

Making things cheaper is actually a subset of a more general technique: making things easier. For a long time it was most of making things easier, but now that the things we build are so complicated, there's another rapidly growing subset: making things easier to *use*.

This is an area where there's great room for improvement. What you want to be able to say about technology is: it just works. How often do you say that now?

Simplicity takes effort—genius, even. The average programmer seems to produce UI designs that are almost willfully bad. I was trying to use the stove at my mother's house a couple weeks ago. It was a new one, and instead of physical knobs it had buttons and an LED display. I tried pressing some buttons I thought would cause it to get hot, and you know what it said? "Err." Not even "Error." "Err." You can't just say "Err" to the user of a *stove*. You should design the UI so that errors are impossible. And the boneheads who designed this stove even had an example of such a UI to work from: the old one. You turn one knob to set the temperature and another to set the timer. What was wrong with that? It just worked.

It seems that, for the average engineer, more options just means more rope to hang yourself. So if you want to start a startup, you can take almost any existing technology produced by a big company, and assume you could build something way easier to use.

Design for Exit

Success for a startup approximately equals getting bought. You need some kind of exit strategy, because you can't get the smartest people to work for you without giving them options likely to be worth something. Which means you either have to get bought or go public, and the number of startups that go public is very small.

If success probably means getting bought, should you make that a conscious goal? The old answer was no: you were supposed to pretend that you wanted to create a giant, public company, and act surprised when someone made you an offer. Really, you want to buy us? Well, I suppose we'd consider it, for the right price.

I think things are changing. If 98% of the time success means getting bought, why not be open about it? If 98% of the time you're doing product development on spec for some big company, why not think of that as your task? One advantage of this approach is that it gives you another source of ideas: look at big companies, think what they should be doing, and do it yourself. Even if they already know it, you'll probably be done faster.

Just be sure to make something multiple acquirers will want.

Don't fix Windows, because the only potential acquirer is Microsoft, and when there's only one acquirer, they don't have to hurry. They can take their time and copy you instead of buying you. If you want to get market price, work on something where there's competition.

If an increasing number of startups are created to do product development on spec, it will be a natural counterweight to monopolies. Once some type of technology is captured by a monopoly, it will only evolve at big company rates instead of startup rates, whereas alternatives will evolve with especial speed. A free market interprets monopoly as damage and routes around it.

The Woz Route

The most productive way to generate startup ideas is also the most unlikely-sounding: by accident. If you look at how famous startups got started, a lot of them weren't initially supposed to be startups. Lotus began with a program Mitch Kapor wrote for a friend. Apple got started because Steve Wozniak wanted to build microcomputers, and his employer, Hewlett-Packard, wouldn't let him do it at work. Yahoo began as David Filo's personal collection of links.

This is not the only way to start startups. You can sit down and consciously come up with an idea for a company; we did. But measured in total market cap, the build-stuff-for-yourself model might be more fruitful. It certainly has to be the most fun way to come up with startup ideas. And since a startup ought to have multiple founders who were already friends before they decided to start a company, the rather surprising conclusion is that the best way to generate startup ideas is to do what hackers do for fun: cook up amusing hacks with your friends.

It seems like it violates some kind of conservation law, but there it is: the best way to get a "million dollar idea" is just to do what hackers enjoy doing anyway.

ORGANIC STARTUP IDEAS

April 2010

The best way to come up with startup ideas is to ask yourself the question: what do you wish someone would make for you?

There are two types of startup ideas: those that grow organically out of your own life, and those that you decide, from afar, are going to be necessary to some class of users other than you. Apple was the first type. Apple happened because Steve Wozniak wanted a computer. Unlike most people who wanted computers, he could design one, so he did. And since lots of other people wanted the same thing, Apple was able to sell enough of them to get the company rolling. They still rely on this principle today, incidentally. The iPhone is the phone Steve Jobs wants.*

Our own startup, Viaweb, was of the second type. We made software for building online stores. We didn't need this software ourselves. We weren't direct marketers. We didn't even know when we started that our users were called "direct marketers." But we were comparatively old when we started the company (I was 30 and Robert Morris was 29), so we'd seen enough to know users would need this type of software.†

There is no sharp line between the two types of ideas, but the most successful startups seem to be closer to the Apple type than the Viaweb type. When he was writing that first Basic interpreter for the Altair, Bill Gates was writing something he would use, as were Larry and Sergey when they wrote the first versions of Google.

Organic ideas are generally preferable to the made up kind, but particularly so when the founders are young. It takes experience to

---

* This suggests a way to predict areas where Apple will be weak: things Steve Jobs doesn't use. E.g. I doubt he is much into gaming.

† In retrospect, we should have *become* direct marketers. If I were doing Viaweb again, I'd open our own online store. If we had, we'd have understood users a lot better. I'd encourage anyone starting a startup to become one of its users, however unnatural it seems.

predict what other people will want. The worst ideas we see at Y Combinator are from young founders making things they think other people will want.

So if you want to start a startup and don't know yet what you're going to do, I'd encourage you to focus initially on organic ideas. What's missing or broken in your daily life? Sometimes if you just ask that question you'll get immediate answers. It must have seemed obviously broken to Bill Gates that you could only program the Altair in machine language.

You may need to stand outside yourself a bit to see brokenness, because you tend to get used to it and take it for granted. You can be sure it's there, though. There are always great ideas sitting right under our noses. In 2004 it was ridiculous that Harvard undergrads were still using a Facebook printed on paper. Surely that sort of thing should have been online.

There are ideas that obvious lying around now. The reason you're overlooking them is the same reason you'd have overlooked the idea of building Facebook in 2004: organic startup ideas usually don't seem like startup ideas at first. We know now that Facebook was very successful, but put yourself back in 2004. Putting undergraduates' profiles online wouldn't have seemed like much of a startup idea. And in fact, it wasn't initially a startup idea. When Mark spoke at a YC dinner this winter he said he wasn't trying to start a company when he wrote the first version of Facebook. It was just a project. So was the Apple I when Woz first started working on it. He didn't think he was starting a company. If these guys had thought they were starting companies, they might have been tempted to do something more "serious," and that would have been a mistake.

So if you want to come up with organic startup ideas, I'd encourage you to focus more on the idea part and less on the startup part. Just fix things that seem broken, regardless of whether it seems like the problem is important enough to build a company on. If you keep pursuing such threads it would be hard not to end up making something of value to a lot of people, and when you do, surprise, you've

got a company.*

Don't be discouraged if what you produce initially is something other people dismiss as a toy. In fact, that's a good sign. That's probably why everyone else has been overlooking the idea. The first microcomputers were dismissed as toys. And the first planes, and the first cars. At this point, when someone comes to us with something that users like but that we could envision forum trolls dismissing as a toy, it makes us especially likely to invest.

While young founders are at a disadvantage when coming up with made-up ideas, they're the best source of organic ones, because they're at the forefront of technology. They use the latest stuff. They only just decided what to use, so why wouldn't they? And because they use the latest stuff, they're in a position to discover valuable types of fixable brokenness first.

There's nothing more valuable than an unmet need that is just becoming fixable. If you find something broken that you can fix for a lot of people, you've found a gold mine. As with an actual gold mine, you still have to work hard to get the gold out of it. But at least you know where the seam is, and that's the hard part.

FRIGHTENINGLY AMBITIOUS STARTUP IDEAS

March 2012

One of the more surprising things I've noticed while working on Y Combinator is how frightening the most ambitious startup ideas are. In this essay I'm going to demonstrate this phenomenon by describing some. Any one of them could make you a billionaire. That might sound like an attractive prospect, and yet when I describe these ideas you may notice you find yourself shrinking away from them.

---

* Possible exception: It's hard to compete directly with open source software. You can build things for programmers, but there has to be some part you can charge for.

Don't worry, it's not a sign of weakness. Arguably it's a sign of sanity. The biggest startup ideas are terrifying. And not just because they'd be a lot of work. The biggest ideas seem to threaten your identity: you wonder if you'd have enough ambition to carry them through.

There's a scene in *Being John Malkovich* where the nerdy hero encounters a very attractive, sophisticated woman. She says to him:

> Here's the thing: If you ever got me, you wouldn't have a clue what to do with me.

That's what these ideas say to us.

This phenomenon is one of the most important things you can understand about startups.* You'd expect big startup ideas to be attractive, but actually they tend to repel you. And that has a bunch of consequences. It means these ideas are invisible to most people who try to think of startup ideas, because their subconscious filters them out. Even the most ambitious people are probably best off approaching them obliquely.

1. A New Search Engine

The best ideas are just on the right side of impossible. I don't know if this one is possible, but there are signs it might be. Making a new search engine means competing with Google, and recently I've noticed some cracks in their fortress.

The point when it became clear to me that Microsoft had lost their way was when they decided to get into the search business. That was not a natural move for Microsoft. They did it because they were afraid of Google, and Google was in the search business. But this meant (a) Google was now setting Microsoft's agenda, and (b)

---

* It's also one of the most important things VCs fail to understand about startups. Most expect founders to walk in with a clear plan for the future, and judge them based on that. Few consciously realize that in the biggest successes there is the least correlation between the initial plan and what the startup eventually becomes.

Microsoft's agenda consisted of stuff they weren't good at.

Microsoft : Google :: Google : Facebook.

That does not by itself mean there's room for a new search engine, but lately when using Google search I've found myself nostalgic for the old days, when Google was true to its own slightly aspy self. Google used to give me a page of the right answers, fast, with no clutter. Now the results seem inspired by the Scientologist principle that what's true is what's true for you. And the pages don't have the clean, sparse feel they used to. Google search results used to look like the output of a Unix utility. Now if I accidentally put the cursor in the wrong place, anything might happen.

The way to win here is to build the search engine all the hackers use. A search engine whose users consisted of the top 10,000 hackers and no one else would be in a very powerful position despite its small size, just as Google was when it was that search engine. And for the first time in over a decade the idea of switching seems thinkable to me.

Since anyone capable of starting this company is one of those 10,000 hackers, the route is at least straightforward: make the search engine you yourself want. Feel free to make it excessively hackerish. Make it really good for code search, for example. Would you like search queries to be Turing complete? Anything that gets you those 10,000 users is ipso facto good.

Don't worry if something you want to do will constrain you in the long term, because if you don't get that initial core of users, there won't be a long term. If you can just build something that you and your friends genuinely prefer to Google, you're already about 10% of the way to an IPO, just as Facebook was (though they probably didn't realize it) when they got all the Harvard undergrads.

2. Replace Email

Email was not designed to be used the way we use it now. Email is not a messaging protocol. It's a todo list. Or rather, my inbox is a todo list, and email is the way things get onto it. But it is a disastrously bad todo list.

I'm open to different types of solutions to this problem, but I suspect that tweaking the inbox is not enough, and that email has to be replaced with a new protocol. This new protocol should be a todo list protocol, not a messaging protocol, although there is a degenerate case where what someone wants you to do is: read the following text.

As a todo list protocol, the new protocol should give more power to the recipient than email does. I want there to be more restrictions on what someone can put on my todo list. And when someone can put something on my todo list, I want them to tell me more about what they want from me. Do they want me to do something beyond just reading some text? How important is it? (There obviously has to be some mechanism to prevent people from saying everything is important.) When does it have to be done?

This is one of those ideas that's like an irresistible force meeting an immovable object. On one hand, entrenched protocols are impossible to replace. On the other, it seems unlikely that people in 100 years will still be living in the same email hell we do now. And if email is going to get replaced eventually, why not now?

If you do it right, you may be able to avoid the usual chicken and egg problem new protocols face, because some of the most powerful people in the world will be among the first to switch to it. They're all at the mercy of email too.

Whatever you build, make it fast. GMail has become painfully slow. If you made something no better than GMail, but fast, that alone would let you start to pull users away from GMail.

GMail is slow because Google can't afford to spend a lot on it. But people will pay for this. I'd have no problem paying $50 a month. Considering how much time I spend in email, it's kind of scary to think how much I'd be justified in paying. At least $1000 a month. If I spend several hours a day reading and writing email, that would be a cheap way to make my life better.

3. Replace Universities

People are all over this idea lately, and I think they're onto something. I'm reluctant to suggest that an institution that's been around

for a millennium is finished just because of some mistakes they made in the last few decades, but certainly in the last few decades US universities seem to have been headed down the wrong path. One could do a lot better for a lot less money.

I don't think universities will disappear. They won't be replaced wholesale. They'll just lose the de facto monopoly on certain types of learning that they once had. There will be many different ways to learn different things, and some may look quite different from universities. Y Combinator itself is arguably one of them.

Learning is such a big problem that changing the way people do it will have a wave of secondary effects. For example, the name of the university one went to is treated by a lot of people (correctly or not) as a credential in its own right. If learning breaks up into many little pieces, credentialling may separate from it. There may even need to be replacements for campus social life (and oddly enough, YC even has aspects of that).

You could replace high schools too, but there you face bureaucratic obstacles that would slow down a startup. Universities seem the place to start.

4. Internet Drama

Hollywood has been slow to embrace the Internet. That was a mistake, because I think we can now call a winner in the race between delivery mechanisms, and it is the Internet, not cable.

A lot of the reason is the horribleness of cable clients, also known as TVs. Our family didn't wait for Apple TV. We hated our last TV so much that a few months ago we replaced it with an iMac bolted to the wall. It's a little inconvenient to control it with a wireless mouse, but the overall experience is much better than the nightmare UI we had to deal with before.

Some of the attention people currently devote to watching movies and TV can be stolen by things that seem completely unrelated, like social networking apps. More can be stolen by things that are a little more closely related, like games. But there will probably always remain some residual demand for conventional drama, where you sit

passively and watch as a plot happens. So how do you deliver drama via the Internet? Whatever you make will have to be on a larger scale than Youtube clips. When people sit down to watch a show, they want to know what they're going to get: either part of a series with familiar characters, or a single longer "movie" whose basic premise they know in advance.

There are two ways delivery and payment could play out. Either some company like Netflix or Apple will be the app store for entertainment, and you'll reach audiences through them. Or the would-be app stores will be too overreaching, or too technically inflexible, and companies will arise to supply payment and streaming a la carte to the producers of drama. If that's the way things play out, there will also be a need for such infrastructure companies.

5. The Next Steve Jobs

I was talking recently to someone who knew Apple well, and I asked him if the people now running the company would be able to keep creating new things the way Apple had under Steve Jobs. His answer was simply "no." I already feared that would be the answer. I asked more to see how he'd qualify it. But he didn't qualify it at all. No, there will be no more great new stuff beyond whatever's currently in the pipeline. Apple's revenues may continue to rise for a long time, but as Microsoft shows, revenue is a lagging indicator in the technology business.

So if Apple's not going to make the next iPad, who is? None of the existing players. None of them are run by product visionaries, and empirically you can't seem to get those by hiring them. Empirically the way you get a product visionary as CEO is for him to found the company and not get fired. So the company that creates the next wave of hardware is probably going to have to be a startup.

I realize it sounds preposterously ambitious for a startup to try to become as big as Apple. But no more ambitious than it was for Apple to become as big as Apple, and they did it. Plus a startup taking on this problem now has an advantage the original Apple didn't: the example of Apple. Steve Jobs has shown us what's possible. That helps

would-be successors both directly, as Roger Bannister did, by showing how much better you can do than people did before, and indirectly, as Augustus did, by lodging the idea in users' minds that a single person could unroll the future for them.*

Now Steve is gone there's a vacuum we can all feel. If a new company led boldly into the future of hardware, users would follow. The CEO of that company, the "next Steve Jobs," might not measure up to Steve Jobs. But he wouldn't have to. He'd just have to do a better job than Samsung and HP and Nokia, and that seems pretty doable.

6. Bring Back Moore's Law

The last 10 years have reminded us what Moore's Law actually says. Till about 2002 you could safely misinterpret it as promising that clock speeds would double every 18 months. Actually what it says is that circuit densities will double every 18 months. It used to seem pedantic to point that out. Not anymore. Intel can no longer give us faster CPUs, just more of them.

This Moore's Law is not as good as the old one. Moore's Law used to mean that if your software was slow, all you had to do was wait, and the inexorable progress of hardware would solve your problems. Now if your software is slow you have to rewrite it to do more things in parallel, which is a lot more work than waiting.

It would be great if a startup could give us something of the old Moore's Law back, by writing software that could make a large number of CPUs look to the developer like one very fast CPU. There are several ways to approach this problem. The most ambitious is to try to do it automatically: to write a compiler that will parallelize our code for us. There's a name for this compiler, *the sufficiently smart compiler,* and it is a byword for impossibility. But is it really impossible? Is there no configuration of the bits in memory of a present day

---

* Roger Bannister is famous as the first person to run a mile in under 4 minutes. But his world record only lasted 46 days. Once he showed it could be done, lots of others followed. Ten years later Jim Ryun ran a 3:59 mile as a high school junior.

computer that is this compiler? If you really think so, you should try to prove it, because that would be an interesting result. And if it's not impossible but simply very hard, it might be worth trying to write it. The expected value would be high even if the chance of succeeding was low.

The reason the expected value is so high is web services. If you could write software that gave programmers the convenience of the way things were in the old days, you could offer it to them as a web service. And that would in turn mean that you got practically all the users.

Imagine there was another processor manufacturer that could still translate increased circuit densities into increased clock speeds. They'd take most of Intel's business. And since web services mean that no one sees their processors anymore, by writing the sufficiently smart compiler you could create a situation indistinguishable from you being that manufacturer, at least for the server market.

The least ambitious way of approaching the problem is to start from the other end, and offer programmers more parallelizable Lego blocks to build programs out of, like Hadoop and MapReduce. Then the programmer still does much of the work of optimization.

There's an intriguing middle ground where you build a semi-automatic weapon—where there's a human in the loop. You make something that looks to the user like the sufficiently smart compiler, but inside has people, using highly developed optimization tools to find and eliminate bottlenecks in users' programs. These people might be your employees, or you might create a marketplace for optimization.

An optimization marketplace would be a way to generate the sufficiently smart compiler piecemeal, because participants would immediately start writing bots. It would be a curious state of affairs if you could get to the point where everything could be done by bots, because then you'd have made the sufficiently smart compiler, but no one person would have a complete copy of it.

I realize how crazy all this sounds. In fact, what I like about this idea is all the different ways in which it's wrong. The whole idea of

focusing on optimization is counter to the general trend in software development for the last several decades. Trying to write the sufficiently smart compiler is by definition a mistake. And even if it weren't, compilers are the sort of software that's supposed to be created by open source projects, not companies. Plus if this works it will deprive all the programmers who take pleasure in making multithreaded apps of so much amusing complexity. The forum troll I have by now internalized doesn't even know where to begin in raising objections to this project. Now that's what I call a startup idea.

7. Ongoing Diagnosis

But wait, here's another that could face even greater resistance: ongoing, automatic medical diagnosis.

One of my tricks for generating startup ideas is to imagine the ways in which we'll seem backward to future generations. And I'm pretty sure that to people 50 or 100 years in the future, it will seem barbaric that people in our era waited till they had symptoms to be diagnosed with conditions like heart disease and cancer.

For example, in 2004 Bill Clinton found he was feeling short of breath. Doctors discovered that several of his arteries were over 90% blocked and 3 days later he had a quadruple bypass. It seems reasonable to assume Bill Clinton has the best medical care available. And yet even he had to wait till his arteries were over 90% blocked to learn that the number was over 90%. Surely at some point in the future we'll know these numbers the way we now know something like our weight. Ditto for cancer. It will seem preposterous to future generations that we wait till patients have physical symptoms to be diagnosed with cancer. Cancer will show up on some sort of radar screen immediately.

(Of course, what shows up on the radar screen may be different from what we think of now as cancer. I wouldn't be surprised if at any given time we have ten or even hundreds of microcancers going at once, none of which normally amount to anything.)

A lot of the obstacles to ongoing diagnosis will come from the fact that it's going against the grain of the medical profession. The

way medicine has always worked is that patients come to doctors with problems, and the doctors figure out what's wrong. A lot of doctors don't like the idea of going on the medical equivalent of what lawyers call a "fishing expedition," where you go looking for problems without knowing what you're looking for. They call the things that get discovered this way "incidentalomas," and they are something of a nuisance.

For example, a friend of mine once had her brain scanned as part of a study. She was horrified when the doctors running the study discovered what appeared to be a large tumor. After further testing, it turned out to be a harmless cyst. But it cost her a few days of terror. A lot of doctors worry that if you start scanning people with no symptoms, you'll get this on a giant scale: a huge number of false alarms that make patients panic and require expensive and perhaps even dangerous tests to resolve. But I think that's just an artifact of current limitations. If people were scanned all the time and we got better at deciding what was a real problem, my friend would have known about this cyst her whole life and known it was harmless, just as we do a birthmark.

There is room for a lot of startups here. In addition to the technical obstacles all startups face, and the bureaucratic obstacles all medical startups face, they'll be going against thousands of years of medical tradition. But it will happen, and it will be a great thing—so great that people in the future will feel as sorry for us as we do for the generations that lived before anaesthesia and antibiotics.

Tactics

Let me conclude with some tactical advice. If you want to take on a problem as big as the ones I've discussed, don't make a direct frontal attack on it. Don't say, for example, that you're going to replace email. If you do that you raise too many expectations. Your employees and investors will constantly be asking "are we there yet?" and you'll have an army of haters waiting to see you fail. Just say you're building to-do-list software. That sounds harmless. People can notice you've re-

placed email when it's a *fait accompli*.*

Empirically, the way to do really big things seems to be to start with deceptively small things. Want to dominate microcomputer software? Start by writing a Basic interpreter for a machine with a few thousand users. Want to make the universal web site? Start by building a site for Harvard undergrads to stalk one another.

Empirically, it's not just for other people that you need to start small. You need to for your own sake. Neither Bill Gates nor Mark Zuckerberg knew at first how big their companies were going to get. All they knew was that they were onto something. Maybe it's a bad idea to have really big ambitions initially, because the bigger your ambition, the longer it's going to take, and the further you project into the future, the more likely you'll get it wrong.

I think the way to use these big ideas is not to try to identify a precise point in the future and then ask yourself how to get from here to there, like the popular image of a visionary. You'll be better off if you operate like Columbus and just head in a general westerly direction. Don't try to construct the future like a building, because your current blueprint is almost certainly mistaken. Start with something you know works, and when you expand, expand westward.

The popular image of the visionary is someone with a clear view of the future, but empirically it may be better to have a blurry one.

* If you want to be the next Apple, maybe you don't even want to start with consumer electronics. Maybe at first you make something hackers use. Or you make something popular but apparently unimportant, like a headset or router. All you need is a bridgehead.

The Idea Maze

BY CHRIS DIXON
AUGUST 4, 2013

The pop culture view of startups is that they're all about coming up with a great product idea. After the eureka moment, the outcome is preordained. This neglects the years of toil that entrepreneurs endure, and also the fact that the vast majority of startups change over time, often dramatically.

In response to this pop culture misconception, it has become popular in the startup community to say things like "execution is everything" and "ideas don't matter".

But the reality is that ideas do matter, just not in the narrow sense in which startup ideas are popularly defined. Good startup ideas are well developed, multi-year plans that contemplate many possible paths according to how the world changes. Balaji Srinivasan calls this the idea maze:

> A good founder is capable of anticipating which turns lead to treasure and which lead to certain death. A bad founder is just running to the entrance of (say) the "movies/music/filesharing/P2P" maze or the "photosharing" maze without any sense for the history of the industry, the players in the maze, the casualties of the past, and the technologies that are likely to move walls and change assumptions.

Imagine, for example, that you were thinking of starting Netflix

back when it was founded in 1997. How would content providers, distribution channels, and competitors respond? How soon would technology develop to open a hidden door and let you distribute online instead of by mail? Or consider Dropbox in 2007. Dozens of cloud storage companies had been started before. What mistakes had they made? How would incumbents like Amazon and Google respond? How would new platforms like mobile affect you?

When you're starting out, it's impossible to completely map out the idea maze. But there are some places you can look for help:

1) History. If your idea has been tried before (and almost all good ideas have), you should figure out what the previous attempts did right and wrong. A lot of this knowledge exists only in the brains of practitioners, which is one of many reasons why "stealth mode" is a bad idea. The benefits of learning about the maze generally far outweigh the risks of having your idea stolen.

2) Analogy. You can also build the maze by analogy to similar businesses. If you are building a "peer economy" company it can be useful to look at what Airbnb did right. If you are building a marketplace you should understand eBay's beginnings. Etc.

3) Theories. There are now decades of historical data on tech startups, and smart observers have sifted through to develop theories that generalize that data. Some of these theories come from academia (e.g. Clay Christensen) but increasingly they come from investors and entrepreneurs on blogs.

4) Direct experience. A lot of good startup founders figure out the maze through direct experience, often at work. The key here is to put yourself in interesting mazes and give yourself time to figure it out.

The metaphor of a maze also helps you think about competition. Competition from other startups is usually just a distraction. In all likelihood, they won't take the same path, and the presence of others in your maze means you might be onto something. Your real competition—and what you should worry about—is the years you could waste going down the wrong path.

Strategy Letter VI.

BY JOEL SPOLSKY
SEPTEMBER 18, 2007

IBM just released an open-source office suite called IBM Lotus Symphony. Sounds like Yet Another StarOffice distribution. But I suspect they're probably trying to wipe out the memory of the original Lotus Symphony, which had been hyped as the Second Coming and which fell totally flat. It was the software equivalent of Gigli.

In the late 80s, Lotus was trying very hard to figure out what to do next with their flagship spreadsheet and graphics product, Lotus 1-2-3. There were two obvious ideas: first, they could add more features. Word processing, say. This product was called Symphony. Another idea which seemed obvious was to make a 3-D spreadsheet. That became 1-2-3 version 3.0.

Both ideas ran head-first into a serious problem: the old DOS 640K memory limitation. IBM was starting to ship a few computers with 80286 chips, which could address more memory, but Lotus didn't think there was a big enough market for software that needed a $10,000 computer to run. So they squeezed and squeezed. They spent 18 months cramming 1-2-3 for DOS into 640K, and eventually, after a lot of wasted time, had to give up the 3D feature to get it to fit. In the case of Symphony, they just chopped features left and right.

Neither strategy was right. By the time 123 3.0 was shipping, eve-

rybody had 80386s with 2M or 4M of RAM. And Symphony had an inadequate spreadsheet, an inadequate word processor, and some other inadequate bits.

"That's nice, old man," you say. "Who gives a fart about some old character mode software?"

Humor me for a minute, because history is repeating itself, in three different ways, and the smart strategy is to bet on the same results.

Limited-memory, limited-CPU environments

From the beginning of time until about, say, 1989, programmers were extremely concerned with efficiency. There just wasn't that much memory and there just weren't that many CPU cycles.

In the late 90s a couple of companies, including Microsoft and Apple, noticed (just a little bit sooner than anyone else) that Moore's Law meant that they shouldn't think too hard about performance and memory usage… just build cool stuff, and wait for the hardware to catch up. Microsoft first shipped Excel for Windows when 80386s were too expensive to buy, but they were patient. Within a couple of years, the 80386SX came out, and anybody who could afford a $1500 clone could run Excel.

As a programmer, thanks to plummeting memory prices, and CPU speeds doubling every year, you had a choice. You could spend six months rewriting your inner loops in Assembler, or take six months off to play drums in a rock and roll band, and in either case, your program would run faster. Assembler programmers don't have groupies.

So, we don't care about performance or optimization much anymore.

Except in one place: JavaScript running on browsers in AJAX applications. And since that's the direction almost all software development is moving, that's a big deal.

A lot of today's AJAX applications have a meg or more of client side code. This time, it's not the RAM or CPU cycles that are scarce: it's the download bandwidth and the compile time. Either way, you

really have to squeeze to get complex AJAX apps to perform well.

History, though, is repeating itself. Bandwidth is getting cheaper. People are figuring out how to precompile JavaScript.

The developers who put a lot of effort into optimizing things and making them tight and fast will wake up to discover that effort was, more or less, wasted, or, at the very least, you could say that it "conferred no long term competitive advantage," if you're the kind of person who talks like an economist.

The developers who ignored performance and blasted ahead adding cool features to their applications will, in the long run, have better applications.

A portable programming language

The C programming language was invented with the explicit goal of making it easy to port applications from one instruction set to another. And it did a fine job, but wasn't really 100% portable, so we got Java, which was even more portable than C. Mmmhmm.

Right now the big hole in the portability story is—tada!—client-side JavaScript, and especially the DOM in web browsers. Writing applications that work in all different browsers is a friggin' nightmare. There is simply no alternative but to test exhaustively on Firefox, IE6, IE7, Safari, and Opera, and guess what? I don't have time to test on Opera. Sucks to be Opera. Startup web browsers don't stand a chance.

What's going to happen? Well, you can try begging Microsoft and Firefox to be more compatible. Good luck with that. You can follow the p-code/Java model and build a little sandbox on top of the underlying system. But sandboxes are penalty boxes; they're slow and they suck, which is why Java Applets are dead, dead, dead. To build a sandbox you pretty much doom yourself to running at 1/10th the speed of the underlying platform, and you doom yourself to never supporting any of the cool features that show up on one of the platforms but not the others. (I'm still waiting for someone to show me a Java applet for phones that can access *any* of the phone's features, like the camera, the contacts list, the SMS messages, or the GPS receiver.)

Sandboxes didn't work then and they're not working now.

What's going to happen? The winners are going to do what worked at Bell Labs in 1978: build a programming language, like C, that's portable and efficient. It should compile down to "native" code (native code being JavaScript and DOMs) with different backends for different target platforms, where the compiler writers obsess about performance so you don't have to. It'll have all the same performance as native JavaScript with full access to the DOM in a consistent fashion, and it'll compile down to IE native and Firefox native portably and automatically. And, yes, it'll go into your CSS and muck around with it in some frightening but provably-correct way so you never have to think about CSS incompatibilities ever again. Ever. Oh joyous day that will be.

High interactivity and UI standards

The IBM 360 mainframe computer system used a user interface called CICS, which you can still see at the airport if you lean over the checkin counter. There's an 80 character by 24 character green screen, character mode only, of course. The mainframe sends down a form to the "client" (the client being a 3270 smart terminal). The terminal is smart; it knows how to present the form to you and let you input data into the form without talking to the mainframe at all. This was one reason mainframes were so much more powerful than Unix: the CPU didn't have to handle your line editing; it was offloaded to a smart terminal. (If you couldn't afford smart terminals for everyone, you bought a System/1 minicomputer to sit between the dumb terminals and the mainframe and handle the form editing for you).

Anyhoo, after you filled out your form, you pressed SEND, and all your answers were sent back to the server to process. Then it sent you another form. And on and on.

Awful. How do you make a word processor in that kind of environment? (You really can't. There never was a decent word processor for mainframes).

That was the first stage. It corresponds precisely to the HTML

phase of the Internet. HTML is CICS with fonts.

In the second stage, everybody bought PCs for their desks, and suddenly, programmers could poke text anywhere on the screen wily-nily, anywhere they wanted, any time they wanted, and you could actually read every keystroke from the users as they typed, so you could make a nice fast application that didn't have to wait for you to hit SEND before the CPU could get involved. So, for example, you could make a word processor that automatically wrapped, moving a word down to the next line when the current line filled up. Right away. Oh my god. You can do that?

The trouble with the second stage was that there were no clear UI standards… the programmers almost had too much flexibility, so everybody did things in different ways, which made it hard, if you knew how to use program X, to also use program Y. WordPerfect and Lotus 1-2-3 had completely different menu systems, keyboard interfaces, and command structures. And copying data between them was out of the question.

And that's exactly where we are with Ajax development today. Sure, yeah, the usability is much better than the first generation DOS apps, because we've learned some things since then. But Ajax apps can be inconsistent, and have a lot of trouble working together—you can't really cut and paste objects from one Ajax app to another, for example, so I'm not sure how you get a picture from Gmail to Flickr. Come on guys, Cut and Paste was invented 25 years ago.

The third phase with PCs was Macintosh and Windows. A standard, consistent user interface with features like multiple windows and the Clipboard designed so that applications could work together. The increased usability and power we got out of the new GUIs made personal computing explode.

So if history repeats itself, we can expect some standardization of Ajax user interfaces to happen in the same way we got Microsoft Windows. Somebody is going to write a compelling SDK that you can use to make powerful Ajax applications with common user interface elements that work together. And whichever SDK wins the most developer mindshare will have the same kind of competitive strong-

hold as Microsoft had with their Windows API.

If you're a web app developer, and you don't want to support the SDK everybody else is supporting, you'll increasingly find that people won't use your web app, because it doesn't, you know, cut and paste and support address book synchronization and whatever weird new interop features we'll want in 2010.

Imagine, for example, that you're Google with GMail, and you're feeling rather smug. But then somebody you've never heard of, some bratty Y Combinator startup, maybe, is gaining ridiculous traction selling NewSDK, which combines a great portable programming language that compiles to JavaScript, and even better, a huge Ajaxy library that includes all kinds of clever interop features. Not just cut 'n' paste: cool mashup features like synchronization and single-point identity management (so you don't have to tell Facebook and Twitter what you're doing, you can just enter it in one place). And you laugh at them, for their NewSDK is a honking 232 megabytes … 232 megabytes! … of JavaScript, and it takes 76 seconds to load a page. And your app, GMail, doesn't lose any customers.

But then, while you're sitting on your googlechair in the googleplex sipping googleccinos and feeling smuggy smug smug smug, new versions of the browsers come out that support cached, compiled JavaScript. And suddenly NewSDK is really fast. And Paul Graham gives them another 6000 boxes of instant noodles to eat, so they stay in business another three years perfecting things.

And your programmers are like, jeez louise, GMail is huge, we can't port GMail to this stupid NewSDK. We'd have to change every line of code. Heck it'd be a complete rewrite; the whole programming model is upside down and recursive and the portable programming language has more parentheses than even Google can buy. The last line of almost every function consists of a string of 3,296 right parentheses. You have to buy a special editor to count them.

And the NewSDK people ship a pretty decent word processor and a pretty decent email app and a killer Facebook/Twitter event publisher that synchronizes with everything, so people start using it.

And while you're not paying attention, everybody starts writing

NewSDK apps, and they're really good, and suddenly businesses ONLY want NewSDK apps, and all those old-school Plain Ajax apps look pathetic and won't cut and paste and mash and sync and play drums nicely with one another. And Gmail becomes a legacy. The WordPerfect of Email. And you'll tell your children how excited you were to get 2GB to store email, and they'll laugh at you. Their *nail polish* has more than 2GB.

Crazy story? Substitute "Google Gmail" with "Lotus 1-2-3". The NewSDK will be the second coming of Microsoft Windows; this is exactly how Lotus lost control of the spreadsheet market. And it's going to happen again on the web because all the same dynamics and forces are in place. The only thing we don't know yet are the particulars, but it'll happen.

Part II

On Business Models

BY SETH GODIN
2009

About Business Models

A business model is the architecture of a business or project. It has four elements:

1. What compelling reason exists for people to give you money? (or votes or donations)
2. How do you acquire what you're selling for less than it costs to sell it?
3. What structural insulation do you have from relentless commoditization and a price war?
4. How will strangers find out about the business and decide to become customers?

The internet 1.0 was a fascinating place because business models were in flux. Suddenly, it was possible to have costless transactions, which meant that doing something at a huge scale was very cheap. That means that #2 was really cheap, so #1 didn't have to be very big at all.

Some people got way out of hand and decided that costs were so low, they didn't have to worry about revenue at all. There are still

some internet hotshot companies that are operating under this scenario, which means that it's fair to say that they don't actually have a business model.

The idea of connecting people, of building tribes, of the natural monopoly provided by online communities means that the internet is the best friend of people focusing on the third element, insulation from competition. Once you build a network, it's extremely difficult for someone else to disrupt it.

As the internet has spread into all aspects of our culture, it is affecting business models offline as well. Your t-shirt shop or consulting firm or political campaign has a different business model than it did ten years ago, largely because viral marketing and the growth of cash-free marketing means that you can spread an idea farther and faster than ever before. It also makes it far cheaper for a competitor to enter the market (#3) putting existing players under significant pressure from newcomers.

This business model revolution is just getting started. It's' not too late to invent a better one.

The Modern Business Plan

It's not clear to me why business plans are the way they are, but they're often misused to obfuscate, bore and show an ability to comply with expectations. If I want the real truth about a business and where it's going, I'd rather see something else. I'd divide the modern business plan into five sections:

- Truth
- Assertions
- Alternatives
- People
- Money

The **truth** section describes the world as it is. Footnote if you want to, but tell me about the market you are entering, the needs that already exist, the competitors in your space, technology standards,

the way others have succeeded and failed in the past. The more specific the better. The more ground knowledge the better. The more visceral the stories, the better. The point of this section is to be sure that you're clear about the way you see the world, and that you and I agree on your assumptions. This section isn't partisan, it takes no positions, it just states how things are.

Truth can take as long as you need to tell it. It can include spreadsheets, market share analysis and anything I need to know about how the world works.

The **assertions** section is your chance to describe how you're going to change things. We will do X, and then Y will happen. We will build Z with this much money in this much time. We will present Q to the market and the market will respond by taking this action.

This is the heart of the modern business plan. The only reason to launch a project is to change something, and I want to know what you're going to do and what impact it's going to have.

Of course, this section will be incorrect. You will make assertions that won't pan out. You'll miss budgets and deadlines and sales. So the **alternatives** section tells me what you'll do if that happens. How much flexibility does your product or team have? If your assertions don't pan out, is it over?

The **people** section rightly highlights the key element… who is on your team, who is going to join your team. 'Who' doesn't mean their resume, who means their attitudes and abilities and track record in shipping.

And the last section is all about **money**. How much do you need, how will you spend it, what does cash flow look like, P&Ls, balance sheets, margins and exit strategies.

Your local VC might not like this format, but I'm betting it will help your team think through the hard issues more clearly.

How to Convince Investors

BY PAUL GRAHAM
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.*

(There are a handful of angels who'd be interested in a company

* 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.

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.* 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

---

* 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.

x." (Whereas when they don't like you, they'll be saying "but what about x?")

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.* 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?†

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

---

* The two are not mutually exclusive. There are people who are both genuinely formidable, and also really good at acting that way.

† 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.

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.* 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.†

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.

---

* 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?

† 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.

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.*

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 convincing 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.

---

* 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.

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.*

---

* 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

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

---

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.

Microsoft.* 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?†

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

* 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.

† 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.

gotcha left to be discovered by the investors you're currently talking to, who will be proud of and thus attached to their discovery.*

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.†

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

* 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.

† Dropbox wasn't rejected by all the East Coast VCs. There was one firm that wanted to invest but tried to lowball them.

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.*

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.

---

* 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.

How to Fund a Startup

BY PAUL GRAHAM
NOVEMBER 2005

Venture funding works like gears. A typical startup goes through several rounds of funding, and at each round you want to take just enough money to reach the speed where you can shift into the next gear.

Few startups get it quite right. Many are underfunded. A few are overfunded, which is like trying to start driving in third gear.

I think it would help founders to understand funding better—not just the mechanics of it, but what investors are thinking. I was surprised recently when I realized that all the worst problems we faced in our startup were due not to competitors, but investors. Dealing with competitors was easy by comparison.

I don't mean to suggest that our investors were nothing but a drag on us. They were helpful in negotiating deals, for example. I mean more that conflicts with investors are particularly nasty. Competitors punch you in the jaw, but investors have you by the balls.

Apparently our situation was not unusual. And if trouble with investors is one of the biggest threats to a startup, managing them is one of the most important skills founders need to learn.

Let's start by talking about the five sources of startup funding. Then we'll trace the life of a hypothetical (very fortunate) startup as it shifts gears through successive rounds.

Friends and Family

A lot of startups get their first funding from friends and family. Excite did, for example: after the founders graduated from college, they borrowed $15,000 from their parents to start a company. With the help of some part-time jobs they made it last 18 months.

If your friends or family happen to be rich, the line blurs between them and angel investors. At Viaweb we got our first $10,000 of seed money from our friend Julian, but he was sufficiently rich that it's hard to say whether he should be classified as a friend or angel. He was also a lawyer, which was great, because it meant we didn't have to pay legal bills out of that initial small sum.

The advantage of raising money from friends and family is that they're easy to find. You already know them. There are three main disadvantages: you mix together your business and personal life; they will probably not be as well connected as angels or venture firms; and they may not be accredited investors, which could complicate your life later.

The SEC defines an "accredited investor" as someone with over a million dollars in liquid assets or an income of over $200,000 a year. The regulatory burden is much lower if a company's shareholders are all accredited investors. Once you take money from the general public you're more restricted in what you can do. *

A startup's life will be more complicated, legally, if any of the investors aren't accredited. In an IPO, it might not merely add expense, but change the outcome. A lawyer I asked about it said:

When the company goes public, the SEC will carefully study all prior issuances of stock by the company and demand that it take immediate action to cure any past violations of securities laws. Those remedial actions can delay, stall or even kill the IPO.

---

* The aim of such regulations is to protect widows and orphans from crooked investment schemes; people with a million dollars in liquid assets are assumed to be able to protect themselves. The unintended consequence is that the investments that generate the highest returns, like hedge funds, are available only to the rich.

Of course the odds of any given startup doing an IPO are small. But not as small as they might seem. A lot of startups that end up going public didn't seem likely to at first. (Who could have guessed that the company Wozniak and Jobs started in their spare time selling plans for microcomputers would yield one of the biggest IPOs of the decade?) Much of the value of a startup consists of that tiny probability multiplied by the huge outcome.

It wasn't because they weren't accredited investors that I didn't ask my parents for seed money, though. When we were starting Viaweb, I didn't know about the concept of an accredited investor, and didn't stop to think about the value of investors' connections. The reason I didn't take money from my parents was that I didn't want them to lose it.

Consulting

Another way to fund a startup is to get a job. The best sort of job is a consulting project in which you can build whatever software you wanted to sell as a startup. Then you can gradually transform yourself from a consulting company into a product company, and have your clients pay your development expenses.

This is a good plan for someone with kids, because it takes most of the risk out of starting a startup. There never has to be a time when you have no revenues. Risk and reward are usually proportionate, however: you should expect a plan that cuts the risk of starting a startup also to cut the average return. In this case, you trade decreased financial risk for increased risk that your company won't succeed as a startup.

But isn't the consulting company itself startup? No, not generally. A company has to be more than small and newly founded to be a startup. There are millions of small businesses in America, but only a few thousand are startups. To be a startup, a company has to be a product business, not a service business. By which I mean not that it has to make something physical, but that it has to have one thing it sells to many people, rather than doing custom work for individual clients. Custom work doesn't scale. To be a startup you need to be the

band that sells a million copies of a song, not the band that makes money by playing at individual weddings and bar mitzvahs.

The trouble with consulting is that clients have an awkward habit of calling you on the phone. Most startups operate close to the margin of failure, and the distraction of having to deal with clients could be enough to put you over the edge. Especially if you have competitors who get to work full time on just being a startup.

So you have to be very disciplined if you take the consulting route. You have to work actively to prevent your company growing into a "weed tree," dependent on this source of easy but low-margin money.*

Indeed, the biggest danger of consulting may be that it gives you an excuse for failure. In a startup, as in grad school, a lot of what ends up driving you are the expectations of your family and friends. Once you start a startup and tell everyone that's what you're doing, you're now on a path labeled "get rich or bust." You now have to get rich, or you've failed.

Fear of failure is an extraordinarily powerful force. Usually it prevents people from starting things, but once you publish some definite ambition, it switches directions and starts working in your favor. I think it's a pretty clever piece of jiu-jitsu to set this irresistible force against the slightly less immovable object of becoming rich. You won't have it driving you if your stated ambition is merely to start a consulting company that you will one day morph into a startup.

An advantage of consulting, as a way to develop a product, is that you know you're making something at least one customer wants. But if you have what it takes to start a startup you should have sufficient vision not to need this crutch.

---

* Consulting is where product companies go to die. IBM is the most famous example. So starting as a consulting company is like starting out in the grave and trying to work your way up into the world of the living.

Angel Investors

Angels are individual rich people. The word was first used for backers of Broadway plays, but now applies to individual investors generally. Angels who've made money in technology are preferable, for two reasons: they understand your situation, and they're a source of contacts and advice.

The contacts and advice can be more important than the money. When del.icio.us took money from investors, they took money from, among others, Tim O'Reilly. The amount he put in was small compared to the VCs who led the round, but Tim is a smart and influential guy and it's good to have him on your side.

You can do whatever you want with money from consulting or friends and family. With angels we're now talking about venture funding proper, so it's time to introduce the concept of *exit strategy*. Younger would-be founders are often surprised that investors expect them either to sell the company or go public. The reason is that investors need to get their capital back. They'll only consider companies that have an exit strategy—meaning companies that could get bought or go public.

This is not as selfish as it sounds. There are few large, private technology companies. Those that don't fail all seem to get bought or go public. The reason is that employees are investors too—of their time—and they want just as much to be able to cash out. If your competitors offer employees stock options that might make them rich, while you make it clear you plan to stay private, your competitors will get the best people. So the principle of an "exit" is not just something forced on startups by investors, but part of what it means to be a startup.

Another concept we need to introduce now is valuation. When someone buys shares in a company, that implicitly establishes a value for it. If someone pays $20,000 for 10% of a company, the company is in theory worth $200,000. I say "in theory" because in early stage investing, valuations are voodoo. As a company gets more established, its valuation gets closer to an actual market value. But in a newly founded startup, the valuation number is just an artifact of the re-

spective contributions of everyone involved.

Startups often "pay" investors who will help the company in some way by letting them invest at low valuations. If I had a startup and Steve Jobs wanted to invest in it, I'd give him the stock for $10, just to be able to brag that he was an investor. Unfortunately, it's impractical (if not illegal) to adjust the valuation of the company up and down for each investor. Startups' valuations are supposed to rise over time. So if you're going to sell cheap stock to eminent angels, do it early, when it's natural for the company to have a low valuation.

Some angel investors join together in syndicates. Any city where people start startups will have one or more of them. In Boston the biggest is the Common Angels. In the Bay Area it's the Band of Angels. You can find groups near you through the Angel Capital Association.* However, most angel investors don't belong to these groups. In fact, the more prominent the angel, the less likely they are to belong to a group.

Some angel groups charge you money to pitch your idea to them. Needless to say, you should never do this.

One of the dangers of taking investment from individual angels, rather than through an angel group or investment firm, is that they have less reputation to protect. A big-name VC firm will not screw you too outrageously, because other founders would avoid them if word got out. With individual angels you don't have this protection, as we found to our dismay in our own startup. In many startups' lives there comes a point when you're at the investors' mercy—when you're out of money and the only place to get more is your existing investors. When we got into such a scrape, our investors took advantage of it in a way that a name-brand VC probably wouldn't have.

Angels have a corresponding advantage, however: they're also not bound by all the rules that VC firms are. And so they can, for example, allow founders to cash out partially in a funding round, by selling some of their stock directly to the investors. I think this will become more common; the average founder is eager to do it, and

* If "near you" doesn't mean the Bay Area, Boston, or Seattle, consider moving. It's not a coincidence you haven't heard of many startups from Philadelphia.

selling, say, half a million dollars worth of stock will not, as VCs fear, cause most founders to be any less committed to the business.

The same angels who tried to screw us also let us do this, and so on balance I'm grateful rather than angry. (As in families, relations between founders and investors can be complicated.)

The best way to find angel investors is through personal introductions. You could try to cold-call angel groups near you, but angels, like VCs, will pay more attention to deals recommended by someone they respect.

Deal terms with angels vary a lot. There are no generally accepted standards. Sometimes angels' deal terms are as fearsome as VCs'. Other angels, particularly in the earliest stages, will invest based on a two-page agreement.

Angels who only invest occasionally may not themselves know what terms they want. They just want to invest in this startup. What kind of anti-dilution protection do they want? Hell if they know. In these situations, the deal terms tend to be random: the angel asks his lawyer to create a vanilla agreement, and the terms end up being whatever the lawyer considers vanilla. Which in practice usually means, whatever existing agreement he finds lying around his firm. (Few legal documents are created from scratch.)

These heaps o' boilerplate are a problem for small startups, because they tend to grow into the union of all preceding documents. I know of one startup that got from an angel investor what amounted to a five hundred pound handshake: after deciding to invest, the angel presented them with a 70-page agreement. The startup didn't have enough money to pay a lawyer even to read it, let alone negotiate the terms, so the deal fell through.

One solution to this problem would be to have the startup's lawyer produce the agreement, instead of the angel's. Some angels might balk at this, but others would probably welcome it.

Inexperienced angels often get cold feet when the time comes to write that big check. In our startup, one of the two angels in the initial round took months to pay us, and only did after repeated nagging from our lawyer, who was also, fortunately, his lawyer.

It's obvious why investors delay. Investing in startups is risky! When a company is only two months old, every *day* you wait gives you 1.7% more data about their trajectory. But the investor is already being compensated for that risk in the low price of the stock, so it is unfair to delay.

Fair or not, investors do it if you let them. Even VCs do it. And funding delays are a big distraction for founders, who ought to be working on their company, not worrying about investors. What's a startup to do? With both investors and acquirers, the only leverage you have is competition. If an investor knows you have other investors lined up, he'll be a lot more eager to close—and not just because he'll worry about losing the deal, but because if other investors are interested, you must be worth investing in. It's the same with acquisitions. No one wants to buy you till someone else wants to buy you, and then everyone wants to buy you.

The key to closing deals is never to stop pursuing alternatives. When an investor says he wants to invest in you, or an acquirer says they want to buy you, *don't believe it till you get the check.* Your natural tendency when an investor says yes will be to relax and go back to writing code. Alas, you can't; you have to keep looking for more investors, if only to get this one to act.*

Seed Funding Firms

Seed firms are like angels in that they invest relatively small amounts at early stages, but like VCs in that they're companies that do it as a business, rather than individuals making occasional investments on the side.

Till now, nearly all seed firms have been so-called "incubators," so Y Combinator gets called one too, though the only thing we have

* Investors are often compared to sheep. And they are like sheep, but that's a rational response to their situation. Sheep act the way they do for a reason. If all the other sheep head for a certain field, it's probably good grazing. And when a wolf appears, is he going to eat a sheep in the middle of the flock, or one near the edge?

in common is that we invest in the earliest phase.

According to the National Association of Business Incubators, there are about 800 incubators in the US. This is an astounding number, because I know the founders of a lot of startups, and I can't think of one that began in an incubator.

What is an incubator? I'm not sure myself. The defining quality seems to be that you work in their space. That's where the name "incubator" comes from. They seem to vary a great deal in other respects. At one extreme is the sort of pork-barrel project where a town gets money from the state government to renovate a vacant building as a "high-tech incubator," as if it were merely lack of the right sort of office space that had till now prevented the town from becoming a startup hub. At the other extreme are places like Idealab, which generates ideas for new startups internally and hires people to work for them.

The classic Bubble incubators, most of which now seem to be dead, were like VC firms except that they took a much bigger role in the startups they funded. In addition to working in their space, you were supposed to use their office staff, lawyers, accountants, and so on.

Whereas incubators tend (or tended) to exert more control than VCs, Y Combinator exerts less. And we think it's better if startups operate out of their own premises, however crappy, than the offices of their investors. So it's annoying that we keep getting called an "incubator," but perhaps inevitable, because there's only one of us so far and no word yet for what we are. If we have to be called something, the obvious name would be "excubator." (The name is more excusable if one considers it as meaning that we enable people to escape cubicles.)

Because seed firms are companies rather than individual people, reaching them is easier than reaching angels. Just go to their web site and send them an email. The importance of personal introductions varies, but is less than with angels or VCs.

The fact that seed firms are companies also means the investment process is more standardized. (This is generally true with angel

groups too.) Seed firms will probably have set deal terms they use for every startup they fund. The fact that the deal terms are standard doesn't mean they're favorable to you, but if other startups have signed the same agreements and things went well for them, it's a sign the terms are reasonable.

Seed firms differ from angels and VCs in that they invest exclusively in the earliest phases—often when the company is still just an idea. Angels and even VC firms occasionally do this, but they also invest at later stages.

The problems are different in the early stages. For example, in the first couple months a startup may completely redefine their idea. So seed investors usually care less about the idea than the people. This is true of all venture funding, but especially so in the seed stage.

Like VCs, one of the advantages of seed firms is the advice they offer. But because seed firms operate in an earlier phase, they need to offer different kinds of advice. For example, a seed firm should be able to give advice about how to approach VCs, which VCs obviously don't need to do; whereas VCs should be able to give advice about how to hire an "executive team," which is not an issue in the seed stage.

In the earliest phases, a lot of the problems are technical, so seed firms should be able to help with technical as well as business problems.

Seed firms and angel investors generally want to invest in the initial phases of a startup, then hand them off to VC firms for the next round. Occasionally startups go from seed funding direct to acquisition, however, and I expect this to become increasingly common.

Google has been aggressively pursuing this route, and now Yahoo is too. Both now compete directly with VCs. And this is a smart move. Why wait for further funding rounds to jack up a startup's price? When a startup reaches the point where VCs have enough information to invest in it, the acquirer should have enough information to buy it. More information, in fact; with their technical depth, the acquirers should be better at picking winners than VCs.

Venture Capital Funds

VC firms are like seed firms in that they're actual companies, but they invest other people's money, and much larger amounts of it. VC investments average several million dollars. So they tend to come later in the life of a startup, are harder to get, and come with tougher terms.

The word "venture capitalist" is sometimes used loosely for any venture investor, but there is a sharp difference between VCs and other investors: VC firms are organized as *funds*, much like hedge funds or mutual funds. The fund managers, who are called "general partners," get about 2% of the fund annually as a management fee, plus about 20% of the fund's gains.

There is a very sharp dropoff in performance among VC firms, because in the VC business both success and failure are self-perpetuating. When an investment scores spectacularly, as Google did for Kleiner and Sequoia, it generates a lot of good publicity for the VCs. And many founders prefer to take money from successful VC firms, because of the legitimacy it confers. Hence a vicious (for the losers) cycle: VC firms that have been doing badly will only get the deals the bigger fish have rejected, causing them to continue to do badly.

As a result, of the thousand or so VC funds in the US now, only about 50 are likely to make money, and it is very hard for a new fund to break into this group.

In a sense, the lower-tier VC firms are a bargain for founders. They may not be quite as smart or as well connected as the big-name firms, but they are much hungrier for deals. This means you should be able to get better terms from them.

Better how? The most obvious is valuation: they'll take less of your company. But as well as money, there's power. I think founders will increasingly be able to stay on as CEO, and on terms that will make it fairly hard to fire them later.

The most dramatic change, I predict, is that VCs will allow founders to cash out partially by selling some of their stock direct to the VC firm. VCs have traditionally resisted letting founders get any-

thing before the ultimate "liquidity event." But they're also desperate for deals. And since I know from my own experience that the rule against buying stock from founders is a stupid one, this is a natural place for things to give as venture funding becomes more and more a seller's market.

The disadvantage of taking money from less known firms is that people will assume, correctly or not, that you were turned down by the more exalted ones. But, like where you went to college, the name of your VC stops mattering once you have some performance to measure. So the more confident you are, the less you need a brand-name VC. We funded Viaweb entirely with angel money; it never occurred to us that the backing of a well known VC firm would make us seem more impressive.*

Another danger of less known firms is that, like angels, they have less reputation to protect. I suspect it's the lower-tier firms that are responsible for most of the tricks that have given VCs such a bad reputation among hackers. They are doubly hosed: the general partners themselves are less able, and yet they have harder problems to solve, because the top VCs skim off all the best deals, leaving the lower-tier firms exactly the startups that are likely to blow up.

For example, lower-tier firms are much more likely to pretend to want to do a deal with you just to lock you up while they decide if they really want to. One experienced CFO said:

The better ones usually will not give a term sheet unless they really want to do a deal. The second or third tier firms have a much higher break rate—it could be as high as 50%.

It's obvious why: the lower-tier firms' biggest fear, when chance throws them a bone, is that one of the big dogs will notice and take it away. The big dogs don't have worry about that.

Falling victim to this trick could really hurt you. As one VC told

---

* This was partly confidence, and partly simple ignorance. We didn't know ourselves which VC firms were the impressive ones. We thought software was all that mattered. But that turned out to be the right direction to be naive in: it's much better to overestimate than underestimate the importance of making a good product.

me:

If you were talking to four VCs, told three of them that you accepted a term sheet, and then have to call them back to tell them you were just kidding, you are absolutely damaged goods.

Here's a partial solution: when a VC offers you a term sheet, ask how many of their last 10 term sheets turned into deals. This will at least force them to lie outright if they want to mislead you.

Not all the people who work at VC firms are partners. Most firms also have a handful of junior employees called something like associates or analysts. If you get a call from a VC firm, go to their web site and check whether the person you talked to is a partner. Odds are it will be a junior person; they scour the web looking for startups their bosses could invest in. The junior people will tend to seem very positive about your company. They're not pretending; they *want* to believe you're a hot prospect, because it would be a huge coup for them if their firm invested in a company they discovered. Don't be misled by this optimism. It's the partners who decide, and they view things with a colder eye.

Because VCs invest large amounts, the money comes with more restrictions. Most only come into effect if the company gets into trouble. For example, VCs generally write it into the deal that in any sale, they get their investment back first. So if the company gets sold at a low price, the founders could get nothing. Some VCs now require that in any sale they get 4x their investment back before the common stock holders (that is, you) get anything, but this is an abuse that should be resisted.

Another difference with large investments is that the founders are usually required to accept "vesting"—to surrender their stock and earn it back over the next 4-5 years. VCs don't want to invest millions in a company the founders could just walk away from. Financially, vesting has little effect, but in some situations it could mean founders will have less power. If VCs got de facto control of the company and fired one of the founders, he'd lose any unvested stock unless there was specific protection against this. So vesting would in that situation force founders to toe the line.

The most noticeable change when a startup takes serious funding is that the founders will no longer have complete control. Ten years ago VCs used to insist that founders step down as CEO and hand the job over to a business guy they supplied. This is less the rule now, partly because the disasters of the Bubble showed that generic business guys don't make such great CEOs.

But while founders will increasingly be able to stay on as CEO, they'll have to cede some power, because the board of directors will become more powerful. In the seed stage, the board is generally a formality; if you want to talk to the other board members, you just yell into the next room. This stops with VC-scale money. In a typical VC funding deal, the board of directors might be composed of two VCs, two founders, and one outside person acceptable to both. The board will have ultimate power, which means the founders now have to convince instead of commanding.

This is not as bad as it sounds, however. Bill Gates is in the same position; he doesn't have majority control of Microsoft; in principle he also has to convince instead of commanding. And yet he seems pretty commanding, doesn't he? As long as things are going smoothly, boards don't interfere much. The danger comes when there's a bump in the road, as happened to Steve Jobs at Apple.

Like angels, VCs prefer to invest in deals that come to them through people they know. So while nearly all VC funds have some address you can send your business plan to, VCs privately admit the chance of getting funding by this route is near zero. One recently told me that he did not know a single startup that got funded this way.

I suspect VCs accept business plans "over the transom" more as a way to keep tabs on industry trends than as a source of deals. In fact, I would strongly advise against mailing your business plan randomly to VCs, because they treat this as evidence of laziness. Do the extra work of getting personal introductions. As one VC put it:

I'm not hard to find. I know a lot of people. If you can't find some way to reach me, how are you going to create a successful company?

One of the most difficult problems for startup founders is deciding when to approach VCs. You really only get one chance, because

they rely heavily on first impressions. And you can't approach some and save others for later, because (a) they ask who else you've talked to and when and (b) they talk among themselves. If you're talking to one VC and he finds out that you were rejected by another several months ago, you'll definitely seem shopworn.

So when do you approach VCs? When you can convince them. If the founders have impressive resumes and the idea isn't hard to understand, you could approach VCs quite early. Whereas if the founders are unknown and the idea is very novel, you might have to launch the thing and show that users loved it before VCs would be convinced.

If several VCs are interested in you, they will sometimes be willing to split the deal between them. They're more likely to do this if they're close in the VC pecking order. Such deals may be a net win for founders, because you get multiple VCs interested in your success, and you can ask each for advice about the other. One founder I know wrote:

Two-firm deals are great. It costs you a little more equity, but being able to play the two firms off each other (as well as ask one if the other is being out of line) is invaluable.

When you do negotiate with VCs, remember that they've done this a lot more than you have. They've invested in dozens of startups, whereas this is probably the first you've founded. But don't let them or the situation intimidate you. The average founder is smarter than the average VC. So just do what you'd do in any complex, unfamiliar situation: proceed deliberately, and question anything that seems odd.

It is, unfortunately, common for VCs to put terms in an agreement whose consequences surprise founders later, and also common for VCs to defend things they do by saying that they're standard in the industry. Standard, schmandard; the whole industry is only a few decades old, and rapidly evolving. The concept of "standard" is a useful one when you're operating on a small scale (Y Combinator uses identical terms for every deal because for tiny seed-stage investments it's not worth the overhead of negotiating individual deals), but it

doesn't apply at the VC level. On that scale, every negotiation is unique.

Most successful startups get money from more than one of the preceding five sources.* And, confusingly, the names of funding sources also tend to be used as the names of different rounds. The best way to explain how it all works is to follow the case of a hypothetical startup.

Stage 1: Seed Round

Our startup begins when a group of three friends have an idea—either an idea for something they might build, or simply the idea "let's start a company." Presumably they already have some source of food and shelter. But if you have food and shelter, you probably also have something you're supposed to be working on: either classwork, or a job. So if you want to work full-time on a startup, your money situation will probably change too.

A lot of startup founders say they started the company without any idea of what they planned to do. This is actually less common than it seems: many have to claim they thought of the idea after quitting because otherwise their former employer would own it.

The three friends decide to take the leap. Since most startups are

---

* I've omitted one source: government grants. I don't think these are even worth thinking about for the average startup. Governments may mean well when they set up grant programs to encourage startups, but what they give with one hand they take away with the other: the process of applying is inevitably so arduous, and the restrictions on what you can do with the money so burdensome, that it would be easier to take a job to get the money.

You should be especially suspicious of grants whose purpose is some kind of social engineering—e.g. to encourage more startups to be started in Mississippi. Free money to start a startup in a place where few succeed is hardly free.

Some government agencies run venture funding groups, which make investments rather than giving grants. For example, the CIA runs a venture fund called In-Q-Tel that is modeled on private sector funds and apparently generates good returns. They would probably be worth approaching—if you don't mind taking money from the CIA.

in competitive businesses, you not only want to work full-time on them, but more than full-time. So some or all of the friends quit their jobs or leave school. (Some of the founders in a startup can stay in grad school, but at least one has to make the company his full-time job.)

They're going to run the company out of one of their apartments at first, and since they don't have any users they don't have to pay much for infrastructure. Their main expenses are setting up the company, which costs a couple thousand dollars in legal work and registration fees, and the living expenses of the founders.

The phrase "seed investment" covers a broad range. To some VC firms it means $500,000, but to most startups it means several months' living expenses. We'll suppose our group of friends start with $15,000 from their friend's rich uncle, who they give 5% of the company in return. There's only common stock at this stage. They leave 20% as an options pool for later employees (but they set things up so that they can issue this stock to themselves if they get bought early and most is still unissued), and the three founders each get 25%.

By living really cheaply they think they can make the remaining money last five months. When you have five months' runway left, how soon do you need to start looking for your next round? Answer: immediately. It takes time to find investors, and time (always more than you expect) for the deal to close even after they say yes. So if our group of founders know what they're doing they'll start sniffing around for angel investors right away. But of course their main job is to build version 1 of their software.

The friends might have liked to have more money in this first phase, but being slightly underfunded teaches them an important lesson. For a startup, cheapness is power. The lower your costs, the more options you have—not just at this stage, but at every point till you're profitable. When you have a high "burn rate," you're always under time pressure, which means (a) you don't have time for your ideas to evolve, and (b) you're often forced to take deals you don't like.

Every startup's rule should be: spend little, and work fast.

After ten weeks' work the three friends have built a prototype that gives one a taste of what their product will do. It's not what they originally set out to do—in the process of writing it, they had some new ideas. And it only does a fraction of what the finished product will do, but that fraction includes stuff that no one else has done before.

They've also written at least a skeleton business plan, addressing the five fundamental questions: what they're going to do, why users need it, how large the market is, how they'll make money, and who the competitors are and why this company is going to beat them. (That last has to be more specific than "they suck" or "we'll work really hard.")

If you have to choose between spending time on the demo or the business plan, spend most on the demo. Software is not only more convincing, but a better way to explore ideas.

Stage 2: Angel Round

While writing the prototype, the group has been traversing their network of friends in search of angel investors. They find some just as the prototype is demoable. When they demo it, one of the angels is willing to invest. Now the group is looking for more money: they want enough to last for a year, and maybe to hire a couple friends. So they're going to raise $200,000.

The angel agrees to invest at a pre-money valuation of $1 million. The company issues $200,000 worth of new shares to the angel; if there were 1000 shares before the deal, this means 200 additional shares. The angel now owns 200/1200 shares, or a sixth of the company, and all the previous shareholders' percentage ownership is diluted by a sixth. After the deal, the capitalization table looks like this:

Shareholder	Shares	Percent
Angel	200	16.7
Uncle	50	4.2
Each founder	250	20.8
Option pool	200	16.7
Total	1200	100.0

To keep things simple, I had the angel do a straight cash for stock deal. In reality the angel might be more likely to make the investment in the form of a convertible loan. A convertible loan is a loan that can be converted into stock later; it works out the same as a stock purchase in the end, but gives the angel more protection against being squashed by VCs in future rounds.

Who pays the legal bills for this deal? The startup, remember, only has a couple thousand left. In practice this turns out to be a sticky problem that usually gets solved in some improvised way. Maybe the startup can find lawyers who will do it cheaply in the hope of future work if the startup succeeds. Maybe someone has a lawyer friend. Maybe the angel pays for his lawyer to represent both sides. (Make sure if you take the latter route that the lawyer is *representing* you rather than merely advising you, or his only duty is to the investor.)

An angel investing $200k would probably expect a seat on the board of directors. He might also want preferred stock, meaning a special class of stock that has some additional rights over the common stock everyone else has. Typically these rights include vetoes over major strategic decisions, protection against being diluted in future rounds, and the right to get one's investment back first if the company is sold.

Some investors might expect the founders to accept vesting for a sum this size, and others wouldn't. VCs are more likely to require vesting than angels. At Viaweb we managed to raise $2.5 million from angels without ever accepting vesting, largely because we were so inexperienced that we were appalled at the idea. In practice this turned out to be good, because it made us harder to push around.

Our experience was unusual; vesting is the norm for amounts that size. Y Combinator doesn't require vesting, because (a) we invest such small amounts, and (b) we think it's unnecessary, and that the hope of getting rich is enough motivation to keep founders at work. But maybe if we were investing millions we would think differently.

I should add that vesting is also a way for founders to protect themselves against one another. It solves the problem of what to do if one of the founders quits. So some founders impose it on themselves when they start the company.

The angel deal takes two weeks to close, so we are now three months into the life of the company.

The point after you get the first big chunk of angel money will usually be the happiest phase in a startup's life. It's a lot like being a postdoc: you have no immediate financial worries, and few responsibilities. You get to work on juicy kinds of work, like designing software. You don't have to spend time on bureaucratic stuff, because you haven't hired any bureaucrats yet. Enjoy it while it lasts, and get as much done as you can, because you will never again be so productive.

With an apparently inexhaustible sum of money sitting safely in the bank, the founders happily set to work turning their prototype into something they can release. They hire one of their friends—at first just as a consultant, so they can try him out—and then a month later as employee #1. They pay him the smallest salary he can live on, plus 3% of the company in restricted stock, vesting over four years. (So after this the option pool is down to 13.7%).* They also spend a little money on a freelance graphic designer.

How much stock do you give early employees? That varies so much that there's no conventional number. If you get someone really good, really early, it might be wise to give him as much stock as the founders. The one universal rule is that the amount of stock an em-

* Options have largely been replaced with restricted stock, which amounts to the same thing. Instead of earning the right to buy stock, the employee gets the stock up front, and earns the right not to have to give it back. The shares set aside for this purpose are still called the "option pool."

ployee gets decreases polynomially with the age of the company. In other words, you get rich as a power of how early you were. So if some friends want you to come work for their startup, don't wait several months before deciding.

A month later, at the end of month four, our group of founders have something they can launch. Gradually through word of mouth they start to get users. Seeing the system in use by real users—people they don't know—gives them lots of new ideas. Also they find they now worry obsessively about the status of their server. (How relaxing founders' lives must have been when startups wrote VisiCalc.)

By the end of month six, the system is starting to have a solid core of features, and a small but devoted following. People start to write about it, and the founders are starting to feel like experts in their field.

We'll assume that their startup is one that could put millions more to use. Perhaps they need to spend a lot on marketing, or build some kind of expensive infrastructure, or hire highly paid salesmen. So they decide to start talking to VCs. They get introductions to VCs from various sources: their angel investor connects them with a couple; they meet a few at conferences; a couple VCs call them after reading about them.

Stage 3: Series A Round

Armed with their now somewhat fleshed-out business plan and able to demo a real, working system, the founders visit the VCs they have introductions to. They find the VCs intimidating and inscrutable. They all ask the same question: who else have you pitched to? (VCs are like high school girls: they're acutely aware of their position in the VC pecking order, and their interest in a company is a function of the interest other VCs show in it.)

One of the VC firms says they want to invest and offers the founders a term sheet. A term sheet is a summary of what the deal terms will be when and if they do a deal; lawyers will fill in the details later. By accepting the term sheet, the startup agrees to turn away other VCs for some set amount of time while this firm does the

"due diligence" required for the deal. Due diligence is the corporate equivalent of a background check: the purpose is to uncover any hidden bombs that might sink the company later, like serious design flaws in the product, pending lawsuits against the company, intellectual property issues, and so on. VCs' legal and financial due diligence is pretty thorough, but the technical due diligence is generally a joke.*

The due diligence discloses no ticking bombs, and six weeks later they go ahead with the deal. Here are the terms: a $2 million investment at a pre-money valuation of $4 million, meaning that after the deal closes the VCs will own a third of the company (2 / (4 + 2)). The VCs also insist that prior to the deal the option pool be enlarged by an additional hundred shares. So the total number of new shares issued is 750, and the cap table becomes:

Shareholder	Shares	Percent
VCs	650	33.3
Angel	200	10.3
Uncle	50	2.6
Each founder	250	12.8
Employee	36*	1.8
Option pool	264	13.5
Total	1950	100.0

* Unvested

This picture is unrealistic in several respects. For example, while the percentages might end up looking like this, it's unlikely that the VCs would keep the existing numbers of shares. In fact, every bit of the startup's paperwork would probably be replaced, as if the company were being founded anew. Also, the money might come in several tranches, the later ones subject to various conditions—though this is apparently more common in deals with lower-tier VCs (whose lot in life is to fund more dubious startups) than with the top firms.

---

* First-rate technical people do not generally hire themselves out to do due diligence for VCs. So the most difficult part for startup founders is often responding politely to the inane questions of the "expert" they send to look you over.

And of course any VCs reading this are probably rolling on the floor laughing at how my hypothetical VCs let the angel keep his 10.3 of the company. I admit, this is the Bambi version; in simplifying the picture, I've also made everyone nicer. In the real world, VCs regard angels the way a jealous husband feels about his wife's previous boyfriends. To them the company didn't exist before they invested in it.*

I don't want to give the impression you have to do an angel round before going to VCs. In this example I stretched things out to show multiple sources of funding in action. Some startups could go directly from seed funding to a VC round; several of the companies we've funded have.

The founders are required to vest their shares over four years, and the board is now reconstituted to consist of two VCs, two founders, and a fifth person acceptable to both. The angel investor cheerfully surrenders his board seat.

At this point there is nothing new our startup can teach us about funding—or at least, nothing good.† The startup will almost certainly hire more people at this point; those millions must be put to work,

---

* VCs regularly wipe out angels by issuing arbitrary amounts of new stock. They seem to have a standard piece of casuistry for this situation: that the angels are no longer working to help the company, and so don't deserve to keep their stock. This of course reflects a willful misunderstanding of what investment means; like any investor, the angel is being compensated for risks he took earlier. By a similar logic, one could argue that the VCs should be deprived of their shares when the company goes public.

† One new thing the company might encounter is a *down round*, or a funding round at valuation lower than the previous round. Down rounds are bad news; it is generally the common stock holders who take the hit. Some of the most fearsome provisions in VC deal terms have to do with down rounds—like "full ratchet anti-dilution," which is as frightening as it sounds.

Founders are tempted to ignore these clauses, because they think the company will either be a big success or a complete bust. VCs know otherwise: it's not uncommon for startups to have moments of adversity before they ultimately succeed. So it's worth negotiating anti-dilution provisions, even though you don't think you need to, and VCs will try to make you feel that you're being gratuitously troublesome.

after all. The company may do additional funding rounds, presumably at higher valuations. They may if they are extraordinarily fortunate do an IPO, which we should remember is also in principle a round of funding, regardless of its de facto purpose. But that, if not beyond the bounds of possibility, is beyond the scope of this article.

Deals Fall Through

Anyone who's been through a startup will find the preceding portrait to be missing something: disasters. If there's one thing all startups have in common, it's that something is always going wrong. And nowhere more than in matters of funding.

For example, our hypothetical startup never spent more than half of one round before securing the next. That's more ideal than typical. Many startups—even successful ones—come close to running out of money at some point. Terrible things happen to startups when they run out of money, because they're designed for growth, not adversity.

But the most unrealistic thing about the series of deals I've described is that they all closed. In the startup world, closing is not what deals do. What deals do is fall through. If you're starting a startup you would do well to remember that. Birds fly; fish swim; deals fall through.

Why? Partly the reason deals seem to fall through so often is that you lie to yourself. You want the deal to close, so you start to believe it will. But even correcting for this, startup deals fall through alarmingly often—far more often than, say, deals to buy real estate. The reason is that it's such a risky environment. People about to fund or acquire a startup are prone to wicked cases of buyer's remorse. They don't really grasp the risk they're taking till the deal's about to close. And then they panic. And not just inexperienced angel investors, but big companies too.

So if you're a startup founder wondering why some angel investor isn't returning your phone calls, you can at least take comfort in the thought that the same thing is happening to other deals a hundred times the size.

The example of a startup's history that I've presented is like a

skeleton—accurate so far as it goes, but needing to be fleshed out to be a complete picture. To get a complete picture, just add in every possible disaster.

A frightening prospect? In a way. And yet also in a way encouraging. The very uncertainty of startups frightens away almost everyone. People overvalue stability—especially young people, who ironically need it least. And so in starting a startup, as in any really bold undertaking, merely deciding to do it gets you halfway there. On the day of the race, most of the other runners won't show up.

FOUNDER CONTROL

December 2010

Someone we funded is talking to VCs now, and asked me how common it was for a startup's founders to retain control of the board after a series A round. He said VCs told him this almost never happened.

Ten years ago that was true. In the past, founders rarely kept control of the board through a series A. The traditional series A board consisted of two founders, two VCs, and one independent member. More recently the recipe is often one founder, one VC, and one independent. In either case the founders lose their majority.

But not always. Mark Zuckerberg kept control of Facebook's board through the series A and still has it today. Mark Pincus has kept control of Zynga's too. But are these just outliers? How common is it for founders to keep control after an A round? I'd heard of several cases among the companies we've funded, but I wasn't sure how many there were, so I emailed the ycfounders list.

The replies surprised me. In a dozen companies we've funded, the founders still had a majority of the board seats after the series A round.

I feel like we're at a tipping point here. A lot of VCs still act as if founders retaining board control after a series A is unheard-of. A lot of them try to make you feel bad if you even ask—as if you're a noob

or a control freak for wanting such a thing. But the founders I heard from aren't noobs or control freaks. Or if they are, they are, like Mark Zuckerberg, the kind of noobs and control freaks VCs should be trying to fund more of.

Founders retaining control after a series A is clearly heard-of. And barring financial catastrophe, I think in the coming year it will become the norm.

Control of a company is a more complicated matter than simply outvoting other parties in board meetings. Investors usually get vetos over certain big decisions, like selling the company, regardless of how many board seats they have. And board votes are rarely split. Matters are decided in the discussion preceding the vote, not in the vote itself, which is usually unanimous. But if opinion is divided in such discussions, the side that knows it would lose in a vote will tend to be less insistent. That's what board control means in practice. You don't simply get to do whatever you want; the board still has to act in the interest of the shareholders; but if you have a majority of board seats, then your opinion about what's in the interest of the shareholders will tend to prevail.

So while board control is not total control, it's not imaginary either. There's inevitably a difference in how things feel within the company. Which means if it becomes the norm for founders to retain board control after a series A, that will change the way things feel in the whole startup world.

The switch to the new norm may be surprisingly fast, because the startups that can retain control tend to be the best ones. They're the ones that set the trends, both for other startups and for VCs.

A lot of the reason VCs are harsh when negotiating with startups is that they're embarrassed to go back to their partners looking like they got beaten. When they sign a termsheet, they want to be able to brag about the good terms they got. A lot of them don't care that much personally about whether founders keep board control. They just don't want to seem like they had to make concessions. Which means if letting the founders keep control stops being perceived as a concession, it will rapidly become much more common.

Like a lot of changes that have been forced on VCs, this change won't turn out to be as big a problem as they might think. VCs will still be able to convince; they just won't be able to compel. And the startups where they have to resort to compulsion are not the ones that matter anyway. VCs make most of their money from a few big hits, and those aren't them.

Knowing that founders will keep control of the board may even help VCs pick better. If they know they can't fire the founders, they'll have to choose founders they can trust. And that's who they should have been choosing all along.

How to Raise Money

BY PAUL GRAHAM
SEPTEMBER 2013

Most startups that raise money do it more than once. A typical trajectory might be (1) to get started with a few tens of thousands from something like Y Combinator or individual angels, then (2) raise a few hundred thousand to a few million to build the company, and then (3) once the company is clearly succeeding, raise one or more later rounds to accelerate growth.

Reality can be messier. Some companies raise money twice in phase 2. Others skip phase 1 and go straight to phase 2. And at Y Combinator we get an increasing number of companies that have already raised amounts in the hundreds of thousands. But the three phase path is at least the one about which individual startups' paths oscillate.

This essay focuses on phase 2 fundraising. That's the type the startups we fund are doing on Demo Day, and this essay is the advice we give them.

Forces

Fundraising is hard in both senses: hard like lifting a heavy weight, and hard like solving a puzzle. It's hard like lifting a weight because it's intrinsically hard to convince people to part with large sums of money. That problem is irreducible; it should be hard. But much of

the other kind of difficulty can be eliminated. Fundraising only seems a puzzle because it's an alien world to most founders, and I hope to fix that by supplying a map through it.

To founders, the behavior of investors is often opaque—partly because their motivations are obscure, but partly because they deliberately mislead you. And the misleading ways of investors combine horribly with the wishful thinking of inexperienced founders. At YC we're always warning founders about this danger, and investors are probably more circumspect with YC startups than with other companies they talk to, and even so we witness a constant series of explosions as these two volatile components combine.*

If you're an inexperienced founder, the only way to survive is by imposing external constraints on yourself. You can't trust your intuitions. I'm going to give you a set of rules here that will get you through this process if anything will. At certain moments you'll be tempted to ignore them. So rule number zero is: these rules exist for a reason. You wouldn't need a rule to keep you going in one direction if there weren't powerful forces pushing you in another.

The ultimate source of the forces acting on you are the forces acting on investors. Investors are pinched between two kinds of fear: fear of investing in startups that fizzle, and fear of missing out on startups that take off. The cause of all this fear is the very thing that makes startups such attractive investments: the successful ones grow very fast. But that fast growth means investors can't wait around. If you wait till a startup is obviously a success, it's too late. To get the really high returns, you have to invest in startups when it's still unclear

---

* The worst explosions happen when unpromising-seeming startups encounter mediocre investors. Good investors don't lead startups on; their reputations are too valuable. And startups that seem promising can usually get enough money from good investors that they don't have to talk to mediocre ones. It is the unpromising-seeming startups that have to resort to raising money from mediocre investors. And it's particularly damaging when these investors flake, because unpromising-seeming startups are usually more desperate for money.

(Not all unpromising-seeming startups do badly. Some are merely ugly ducklings in the sense that they violate current startup fashions.)

how they'll do. But that in turn makes investors nervous they're about to invest in a flop. As indeed they often are.

What investors would like to do, if they could, is wait. When a startup is only a few months old, every week that passes gives you significantly more information about them. But if you wait too long, other investors might take the deal away from you. And of course the other investors are all subject to the same forces. So what tends to happen is that they all wait as long as they can, then when some act the rest have to.

Don't raise money unless you want it and it wants you.

Such a high proportion of successful startups raise money that it might seem fundraising is one of the defining qualities of a startup. Actually it isn't. Rapid growth is what makes a company a startup. Most companies in a position to grow rapidly find that (a) taking outside money helps them grow faster, and (b) their growth potential makes it easy to attract such money. It's so common for both (a) and (b) to be true of a successful startup that practically all do raise outside money. But there may be cases where a startup either wouldn't want to grow faster, or outside money wouldn't help them to, and if you're one of them, don't raise money.

The other time not to raise money is when you won't be able to. If you try to raise money before you can convince investors, you'll not only waste your time, but also burn your reputation with those investors.

Be in fundraising mode or not.

One of the things that surprises founders most about fundraising is how distracting it is. When you start fundraising, everything else grinds to a halt. The problem is not the time fundraising consumes but that it becomes the top idea in your mind. A startup can't endure that level of distraction for long. An early stage startup grows mostly because the founders make it grow, and if the founders look away, growth usually drops sharply.

Because fundraising is so distracting, a startup should either be

in fundraising mode or not. And when you do decide to raise money, you should focus your whole attention on it so you can get it done quickly and get back to work.*

You can take money from investors when you're not in fundraising mode. You just can't expend any attention on it. There are two things that take attention: convincing investors, and negotiating with them. So when you're not in fundraising mode, you should take money from investors only if they require no convincing, and are willing to invest on terms you'll take without negotiation. For example, if a reputable investor is willing to invest on a convertible note, using standard paperwork, that is either uncapped or capped at a good valuation, you can take that without having to think.† The terms will be whatever they turn out to be in your next equity round. And "no convincing" means just that: zero time spent meeting with investors or preparing materials for them. If an investor says they're ready to invest, but they need you to come in for one meeting to meet some of the partners, tell them no, if you're not in fundraising mode, because that's fundraising.‡ Tell them politely; tell them you're focusing on the company right now, and that you'll get back to them when you're fundraising; but do not get sucked down the slippery slope.

Investors will try to lure you into fundraising when you're not. It's great for them if they can, because they can thereby get a shot at you before everyone else. They'll send you emails saying they want to meet to learn more about you. If you get cold-emailed by an associ-

---

* One YC founder told me:

I think in general we've done ok at fundraising, but I managed to screw up twice at the exact same thing—trying to focus on building the company and fundraising at the same time.

† There is one subtle danger you have to watch out for here, which I warn about later: beware of getting too high a valuation from an eager investor, lest that set an impossibly high target when raising additional money.

‡ If they really need a meeting, then they're not ready to invest, regardless of what they say. They're still deciding, which means you're being asked to come in and convince them. Which is fundraising.

ate at a VC firm, you shouldn't meet even if you are in fundraising mode. Deals don't happen that way.* But even if you get an email from a partner you should try to delay meeting till you're in fundraising mode. They may say they just want to meet and chat, but investors never just want to meet and chat. What if they like you? What if they start to talk about giving you money? Will you be able to resist having that conversation? Unless you're experienced enough at fundraising to have a casual conversation with investors that stays casual, it's safer to tell them that you'd be happy to later, when you're fundraising, but that right now you need to focus on the company.†

Companies that are successful at raising money in phase 2 sometimes tack on a few investors after leaving fundraising mode. This is fine; if fundraising went well, you'll be able to do it without spending

---

* Associates at VC firms regularly cold email startups. Naive founders think "Wow, a VC is interested in us!" But an associate is not a VC. They have no decision-making power. And while they may introduce startups they like to partners at their firm, the partners discriminate against deals that come to them this way. I don't know of a single VC investment that began with an associate cold-emailing a startup. If you want to approach a specific firm, get an intro to a partner from someone they respect.

  It's ok to talk to an associate if you get an intro to a VC firm or they see you at a Demo Day and they begin by having an associate vet you. That's not a promising lead and should therefore get low priority, but it's not as completely worthless as a cold email.

  Because the title "associate" has gotten a bad reputation, a few VC firms have started to give their associates the title "partner," which can make things very confusing. If you're a YC startup you can ask us who's who; otherwise you may have to do some research online. There may be a special title for actual partners. If someone speaks for the firm in the press or a blog on the firm's site, they're probably a real partner. If they're on boards of directors they're probably a real partner.

  There are titles between "associate" and "partner," including "principal" and "venture partner." The meanings of these titles vary too much to generalize.

† For similar reasons, avoid casual conversations with potential acquirers. They can lead to distractions even more dangerous than fundraising. Don't even take a meeting with a potential acquirer unless you want to sell your company right now.

time convincing them or negotiating about terms.

Get introductions to investors.

Before you can talk to investors, you have to be introduced to them. If you're presenting at a Demo Day, you'll be introduced to a whole bunch simultaneously. But even if you are, you should supplement these with intros you collect yourself.

Do you have to be introduced? In phase 2, yes. Some investors will let you email them a business plan, but you can tell from the way their sites are organized that they don't really want startups to approach them directly.

Intros vary greatly in effectiveness. The best type of intro is from a well-known investor who has just invested in you. So when you get an investor to commit, ask them to introduce you to other investors they respect.* The next best type of intro is from a founder of a company they've funded. You can also get intros from other people in the startup community, like lawyers and reporters.

There are now sites like AngelList, FundersClub, and WeFunder that can introduce you to investors. We recommend startups treat them as auxiliary sources of money. Raise money first from leads you get yourself. Those will on average be better investors. Plus you'll have an easier time raising money on these sites once you can say you've already raised some from well-known investors.

Hear no till you hear yes.

Treat investors as saying no till they unequivocally say yes, in the form of a definite offer with no contingencies.

I mentioned earlier that investors prefer to wait if they can. What's particularly dangerous for founders is the way they wait. Essentially, they lead you on. They seem like they're about to invest right up till the moment they say no. If they even say no. Some of the worse ones never actually do say no; they just stop replying to your

* Joshua Reeves specifically suggests asking each investor to intro you to two more investors. Don't ask investors who say no for introductions to other investors. That will in many cases be an anti-recommendation.

emails. They hope that way to get a free option on investing. If they decide later that they want to invest—usually because they've heard you're a hot deal—they can pretend they just got distracted and then restart the conversation as if they'd been about to.*

That's not the worst thing investors will do. Some will use language that makes it sound as if they're committing, but which doesn't actually commit them. And wishful thinking founders are happy to meet them half way.†

Fortunately, the next rule is a tactic for neutralizing this behavior. But to work it depends on you not being tricked by the no that sounds like yes. It's so common for founders to be misled/mistaken about this that we designed a protocol to fix the problem. If you believe an investor has committed, get them to confirm it. If you and they have different views of reality, whether the source of the discrepancy is their sketchiness or your wishful thinking, the prospect of confirming a commitment in writing will flush it out. And till they confirm, regard them as saying no.

Do breadth-first search weighted by expected value.

When you talk to investors your m.o. should be breadth-first search, weighted by expected value. You should always talk to investors in parallel rather than serially. You can't afford the time it takes to talk to investors serially, plus if you only talk to one investor at a time, they don't have the pressure of other investors to make them act. But you shouldn't pay the same attention to every investor, because some are more promising prospects than others. The optimal solution is to

---

* This is not always as deliberate as its sounds. A lot of the delays and disconnects between founders and investors are induced by the customs of the venture business, which have evolved the way they have because they suit investors' interests.

† One YC founder who read a draft of this essay wrote:

> *This is the most important section. I think it might bear stating even more clearly. "Investors will deliberately affect more interest than they have to preserve optionality. If an investor seems very interested in you, they still probably won't invest. The solution for this is to assume the worst—that an investor is just feigning interest—until you get a definite commitment."*

talk to all potential investors in parallel, but give higher priority to the more promising ones.*

Expected value = how likely an investor is to say yes, multiplied by how good it would be if they did. So for example, an eminent investor who would invest a lot, but will be hard to convince, might have the same expected value as an obscure angel who won't invest much, but will be easy to convince. Whereas an obscure angel who will only invest a small amount, and yet needs to meet multiple times before making up his mind, has very low expected value. Meet such investors last, if at all.†

Doing breadth-first search weighted by expected value will save you from investors who never explicitly say no but merely drift away, because you'll drift away from them at the same rate. It protects you from investors who flake in much the same way that a distributed algorithm protects you from processors that fail. If some investor isn't returning your emails, or wants to have lots of meetings but isn't progressing toward making you an offer, you automatically focus less on them. But you have to be disciplined about assigning probabilities. You can't let how much you want an investor influence your estimate of how much they want you.

Know where you stand.

How do you judge how well you're doing with an investor, when investors habitually seem more positive than they are? By looking at their actions rather than their words. Every investor has some track they need to move along from the first conversation to wiring the money, and you should always know what that track consists of, where you are on it, and how fast you're moving forward.

---

* Though you should probably pack investor meetings as closely as you can, Jeff Byun mentions one reason not to: if you pack investor meetings too closely, you'll have less time for your pitch to evolve.

  Some founders deliberately schedule a handful of lame investors first, to get the bugs out of their pitch.

† There is not an efficient market in this respect. Some of the most useless investors are also the highest maintenance.

Never leave a meeting with an investor without asking what happens next. What more do they need in order to decide? Do they need another meeting with you? To talk about what? And how soon? Do they need to do something internally, like talk to their partners, or investigate some issue? How long do they expect it to take? Don't be too pushy, but know where you stand. If investors are vague or resist answering such questions, assume the worst; investors who are seriously interested in you will usually be happy to talk about what has to happen between now and wiring the money, because they're already running through that in their heads.*

If you're experienced at negotiations, you already know how to ask such questions.† If you're not, there's a trick you can use in this situation. Investors know you're inexperienced at raising money. Inexperience there doesn't make you unattractive. Being a noob at technology would, if you're starting a technology startup, but not being a noob at fundraising. Larry and Sergey were noobs at fundraising. So you can just confess that you're inexperienced at this and ask how their process works and where you are in it.‡

Get the first commitment.

The biggest factor in most investors' opinions of you is the opinion of other investors. Once you start getting investors to commit, it becomes increasingly easy to get more to. But the other side of this coin is that it's often hard to get the first commitment.

---

* Incidentally, this paragraph is sales 101. If you want to see it in action, go talk to a car dealer.

† I know one very smooth founder who used to end investor meetings with "So, can I count you in?" delivered as if it were "Can you pass the salt?" Unless you're very smooth (if you're not sure. . .), do not do this yourself. There is nothing more unconvincing, for an investor, than a nerdy founder trying to deliver the lines meant for a smooth one.

Investors are fine with funding nerds. So if you're a nerd, just try to be a good nerd, rather than doing a bad imitation of a smooth salesman.

‡ Ian Hogarth suggests a good way to tell how serious potential investors are: the resources they expend on you after the first meeting. An investor who's seriously interested will already be working to help you even before they've committed.

Getting the first substantial offer can be half the total difficulty of fundraising. What counts as a substantial offer depends on who it's from and how much it is. Money from friends and family doesn't usually count, no matter how much. But if you get $50k from a well known VC firm or angel investor, that will usually be enough to set things rolling.*

Close committed money.

It's not a deal till the money's in the bank. I often hear inexperienced founders say things like "We've raised $800,000," only to discover that zero of it is in the bank so far. Remember the twin fears that torment investors? The fear of missing out that makes them jump early, and the fear of jumping onto a turd that results? This is a market where people are exceptionally prone to buyer's remorse. And it's also one that furnishes them plenty of excuses to gratify it. The public markets snap startup investing around like a whip. If the Chinese economy blows up tomorrow, all bets are off. But there are lots of surprises for individual startups too, and they tend to be concentrated around fundraising. Tomorrow a big competitor could appear, or you could get C&Ded, or your cofounder could quit.†

---

* In principle you might have to think about so-called "signaling risk." If a prestigious VC makes a small seed investment in you, what if they don't want to invest the next time you raise money? Other investors might assume that the VC knows you well, since they're an existing investor, and if they don't want to invest in your next round, that must mean you suck. The reason I say "in principle" is that in practice signaling hasn't been much of a problem so far. It rarely arises, and in the few cases where it does, the startup in question usually is doing badly and is doomed anyway.

If you have the luxury of choosing among seed investors, you can play it safe by excluding VC firms. But it isn't critical to.

† Sometimes a competitor will deliberately threaten you with a lawsuit just as you start fundraising, because they know you'll have to disclose the threat to potential investors and they hope this will make it harder for you to raise money. If this happens it will probably frighten you more than investors. Experienced investors know about this trick, and know the actual lawsuits rarely happen. So if you're attacked in this way, be forthright with investors. They'll be more alarmed

Even a day's delay can bring news that causes an investor to change their mind. So when someone commits, get the money. Knowing where you stand doesn't end when they say they'll invest. After they say yes, know what the timetable is for getting the money, and then babysit that process till it happens. Institutional investors have people in charge of wiring money, but you may have to hunt angels down in person to collect a check.

Inexperienced investors are the ones most likely to get buyer's remorse. Established ones have learned to treat saying yes as like diving off a diving board, and they also have more brand to preserve. But I've heard of cases of even top-tier VC firms welching on deals.

Avoid investors who don't "lead."

Since getting the first offer is most of the difficulty of fundraising, that should be part of your calculation of expected value when you start. You have to estimate not just the probability that an investor will say yes, but the probability that they'd be the *first* to say yes, and the latter is not simply a constant fraction of the former. Some investors are known for deciding quickly, and those are extra valuable early on.

Conversely, an investor who will only invest once other investors have is worthless initially. And while most investors are influenced by how interested other investors are in you, there are some who have an explicit policy of only investing after other investors have. You can recognize this contemptible subspecies of investor because they often talk about "leads." They say that they don't lead, or that they'll invest once you have a lead. Sometimes they even claim to be willing to lead themselves, by which they mean they won't invest till you get $x from other investors. (It's great if by "lead" they mean they'll invest unilaterally, and in addition will help you raise more. What's lame is when they use the term to mean they won't invest unless you can raise more elsewhere.)*

---

if you seem evasive than if you tell them everything.

* A related trick is to claim that they'll only invest contingently on other investors doing so because otherwise you'd be "undercapitalized." This is almost always

Where does this term "lead" come from? Up till a few years ago, startups raising money in phase 2 would usually raise equity rounds in which several investors invested at the same time using the same paperwork. You'd negotiate the terms with one "lead" investor, and then all the others would sign the same documents and all the money change hands at the closing.

Series A rounds still work that way, but things now work differently for most fundraising prior to the series A. Now there are rarely actual rounds before the A round, or leads for them. Now startups simply raise money from investors one at a time till they feel they have enough.

Since there are no longer leads, why do investors use that term? Because it's a more legitimate-sounding way of saying what they really mean. All they really mean is that their interest in you is a function of other investors' interest in you. I.e. the spectral signature of all mediocre investors. But when phrased in terms of leads, it sounds like there is something structural and therefore legitimate about their behavior.

When an investor tells you "I want to invest in you, but I don't lead," translate that in your mind to "No, except yes if you turn out to be a hot deal." And since that's the default opinion of any investor about any startup, they've essentially just told you nothing.

When you first start fundraising, the expected value of an investor who won't "lead" is zero, so talk to such investors last if at all.

Have multiple plans.

Many investors will ask how much you're planning to raise. This question makes founders feel they should be planning to raise a specific amount. But in fact you shouldn't. It's a mistake to have fixed plans in an undertaking as unpredictable as fundraising.

So why do investors ask how much you plan to raise? For much the same reasons a salesperson in a store will ask "How much were you planning to spend?" if you walk in looking for a gift for a friend.

---

bullshit. They can't estimate your minimum capital needs that precisely.

You probably didn't have a precise amount in mind; you just want to find something good, and if it's inexpensive, so much the better. The salesperson asks you this not because you're supposed to have a plan to spend a specific amount, but so they can show you only things that cost the most you'll pay.

Similarly, when investors ask how much you plan to raise, it's not because you're supposed to have a plan. It's to see whether you'd be a suitable recipient for the size of investment they like to make, and also to judge your ambition, reasonableness, and how far you are along with fundraising.

If you're a wizard at fundraising, you can say "We plan to raise a $7 million series A round, and we'll be accepting termsheets next Tuesday." I've known a handful of founders who could pull that off without having VCs laugh in their faces. But if you're in the inexperienced but earnest majority, the solution is analogous to the solution I recommend for pitching your startup: do the right thing and then just tell investors what you're doing.

And the right strategy, in fundraising, is to have multiple plans depending on how much you can raise. Ideally you should be able to tell investors something like: we can make it to profitability without raising any more money, but if we raise a few hundred thousand we can hire one or two smart friends, and if we raise a couple million, we can hire a whole engineering team, etc.

Different plans match different investors. If you're talking to a VC firm that only does series A rounds (though there are few of those left), it would be a waste of time talking about any but your most expensive plan. Whereas if you're talking to an angel who invests $20k at a time and you haven't raised any money yet, you probably want to focus on your least expensive plan.

If you're so fortunate as to have to think about the upper limit on what you should raise, a good rule of thumb is to multiply the number of people you want to hire times $15k times 18 months. In most startups, nearly all the costs are a function of the number of people, and $15k per month is the conventional total cost (including benefits and even office space) per person. $15k per month is high, so don't

actually spend that much. But it's ok to use a high estimate when fundraising to add a margin for error. If you have additional expenses, like manufacturing, add in those at the end. Assuming you have none and you think you might hire 20 people, the most you'd want to raise is 20 x $15k x 18 = $5.4 million.*

Underestimate how much you want.

Though you can focus on different plans when talking to different types of investors, you should on the whole err on the side of underestimating the amount you hope to raise.

For example, if you'd like to raise $500k, it's better to say initially that you're trying to raise $250k. Then when you reach $150k you're more than half done. That sends two useful signals to investors: that you're doing well, and that they have to decide quickly because you're running out of room. Whereas if you'd said you were raising $500k, you'd be less than a third done at $150k. If fundraising stalled there for an appreciable time, you'd start to read as a failure.

Saying initially that you're raising $250k doesn't limit you to raising that much. When you reach your initial target and you still have investor interest, you can just decide to raise more. Startups do that all the time. In fact, most startups that are very successful at fundraising end up raising more than they originally intended.

I'm not saying you should lie, but that you should lower your expectations initially. There is almost no downside in starting with a low number. It not only won't cap the amount you raise, but will on the whole tend to increase it.

A good metaphor here is angle of attack. If you try to fly at too steep an angle of attack, you just stall. If you say right out of the gate that you want to raise a $5 million series A round, unless you're in a very strong position, you not only won't get that but won't get anything. Better to start at a low angle of attack, build up speed, and then gradually increase the angle if you want.

---

* You won't hire all those 20 people at once, and you'll probably have some revenues before 18 months are out. But those too are acceptable or at least accepted additions to the margin for error.

Be profitable if you can.

You will be in a much stronger position if your collection of plans includes one for raising zero dollars—i.e. if you can make it to profitability without raising any additional money. Ideally you want to be able to say to investors "We'll succeed no matter what, but raising money will help us do it faster."

There are many analogies between fundraising and dating, and this is one of the strongest. No one wants you if you seem desperate. And the best way not to seem desperate is not to *be* desperate. That's one reason we urge startups during YC to keep expenses low and to try to make it to ramen profitability before Demo Day. Though it sounds slightly paradoxical, if you want to raise money, the best thing you can do is get yourself to the point where you don't need to.

There are almost two distinct modes of fundraising: one in which founders who need money knock on doors seeking it, knowing that otherwise the company will die or at the very least people will have to be fired, and one in which founders who don't need money take some to grow faster than they could merely on their own revenues. To emphasize the distinction I'm going to name them: type A fundraising is when you don't need money, and type B fundraising is when you do.

Inexperienced founders read about famous startups doing what was type A fundraising, and decide they should raise money too, since that seems to be how startups work. Except when they raise money they don't have a clear path to profitability and are thus doing type B fundraising. And they are then surprised how difficult and unpleasant it is.

Of course not all startups can make it to ramen profitability in a few months. And some that don't still manage to have the upper hand over investors, if they have some other advantage like extraordinary growth numbers or exceptionally formidable founders. But as time passes it gets increasingly difficult to fundraise from a position of strength without being profitable.*

* Type A fundraising is so much better that it might even be worth doing some-

Don't optimize for valuation.

When you raise money, what should your valuation be? The most important thing to understand about valuation is that it's not that important.

Founders who raise money at high valuations tend to be unduly proud of it. Founders are often competitive people, and since valuation is usually the only visible number attached to a startup, they end up competing to raise money at the highest valuation. This is stupid, because fundraising is not the test that matters. The real test is revenue. Fundraising is just a means to that end. Being proud of how well you did at fundraising is like being proud of your college grades.

Not only is fundraising not the test that matters, valuation is not even the thing to optimize about fundraising. The number one thing you want from phase 2 fundraising is to get the money you need, so you can get back to focusing on the real test, the success of your company. Number two is good investors. Valuation is at best third.

The empirical evidence shows just how unimportant it is. Dropbox and Airbnb are the most successful companies we've funded so far, and they raised money after Y Combinator at premoney valuations of $4 million and $2.6 million respectively. Prices are so much higher now that if you can raise money at all you'll probably raise it at higher valuations than Dropbox and Airbnb. So let that satisfy your competitiveness. You're doing better than Dropbox and Airbnb! At a test that doesn't matter.

When you start fundraising, your initial valuation (or valuation cap) will be set by the deal you make with the first investor who commits. You can increase the price for later investors, if you get a lot of interest, but by default the valuation you got from the first investor becomes your asking price.

So if you're raising money from multiple investors, as most companies do in phase 2, you have to be careful to avoid raising the first

---

thing different if it gets you there sooner. One YC founder told me that if he were a first-time founder again he'd "leave ideas that are up-front capital intensive to founders with established reputations."

from an over-eager investor at a price you won't be able to sustain. You can of course lower your price if you need to (in which case you should give the same terms to investors who invested earlier at a higher price), but you may lose a bunch of leads in the process of realizing you need to do this.

What you can do if you have eager first investors is raise money from them on an uncapped convertible note with an MFN clause. This is essentially a way of saying that the valuation cap of the note will be determined by the next investors you raise money from.

It will be easier to raise money at a lower valuation. It shouldn't be, but it is. Since phase 2 prices vary at most 10x and the big successes generate returns of at least 100x, investors should pick startups entirely based on their estimate of the probability that the company will be a big success and hardly at all on price. But although it's a mistake for investors to care about price, a significant number do. A startup that investors seem to like but won't invest in at a cap of $x will have an easier time at $x/2.*

Yes/no before valuation.

Some investors want to know what your valuation is before they even talk to you about investing. If your valuation has already been set by a prior investment at a specific valuation or cap, you can tell them that number. But if it isn't set because you haven't closed anyone yet, and they try to push you to name a price, resist doing so. If this would be the first investor you've closed, then this could be the tipping point of fundraising. That means closing this investor is the first priority, and you need to get the conversation onto that instead of being dragged sideways into a discussion of price.

Fortunately there is a way to avoid naming a price in this situation. And it is not just a negotiating trick; it's how you (both) should be operating. Tell them that valuation is not the most important thing to you and that you haven't thought much about it, that you are

* I don't know whether this happens because they're innumerate, or because they believe they have zero ability to predict startup outcomes (in which case this behavior at least wouldn't be irrational). In either case the implications are similar.

looking for investors you want to partner with and who want to partner with you, and that you should talk first about whether they want to invest at all. Then if they decide they do want to invest, you can figure out a price. But first things first.

Since valuation isn't that important and getting fundraising rolling is, we usually tell founders to give the first investor who commits as low a price as they need to. This is a safe technique so long as you combine it with the next one.*

Beware "valuation sensitive" investors.

Occasionally you'll encounter investors who describe themselves as "valuation sensitive." What this means in practice is that they are compulsive negotiators who will suck up a lot of your time trying to push your price down. You should therefore never approach such investors first. While you shouldn't chase high valuations, you also don't want your valuation to be set artificially low because the first investor who committed happened to be a compulsive negotiator. Some such investors have value, but the time to approach them is near the end of fundraising, when you're in a position to say "this is the price everyone else has paid; take it or leave it" and not mind if they leave it. This way, you'll not only get market price, but it will also take less time.

Ideally you know which investors have a reputation for being "valuation sensitive" and can postpone dealing with them till last, but occasionally one you didn't know about will pop up early on. The rule of doing breadth first search weighted by expected value already tells you what to do in this case: slow down your interactions with them.

There are a handful of investors who will try to invest at a lower valuation even when your price has already been set. Lowering your price is a backup plan you resort to when you discover you've let the price get set too high to close all the money you need. So you'd only want to talk to this sort of investor if you were about to do that any-

* If you're a YC startup and you have an investor who for some reason insists that you decide the price, any YC partner can estimate a market price for you.

way. But since investor meetings have to be arranged at least a few days in advance and you can't predict when you'll need to resort to lowering your price, this means in practice that you should approach this type of investor last if at all.

If you're surprised by a lowball offer, treat it as a backup offer and delay responding to it. When someone makes an offer in good faith, you have a moral obligation to respond in a reasonable time. But lowballing you is a dick move that should be met with the corresponding countermove.

Accept offers greedily.

I'm a little leery of using the term "greedily" when writing about fundraising lest non-programmers misunderstand me, but a greedy algorithm is simply one that doesn't try to look into the future. A greedy algorithm takes the best of the options in front of it right now. And that is how startups should approach fundraising in phases 2 and later. Don't try to look into the future because (a) the future is unpredictable, and indeed in this business you're often being deliberately misled about it and (b) your first priority in fundraising should be to get it finished and get back to work anyway.

If someone makes you an acceptable offer, take it. If you have multiple incompatible offers, take the best. Don't reject an acceptable offer in the hope of getting a better one in the future.

These simple rules cover a wide variety of cases. If you're raising money from many investors, roll them up as they say yes. As you start to feel you've raised enough, the threshold for acceptable will start to get higher.

In practice offers exist for stretches of time, not points. So when you get an acceptable offer that would be incompatible with others (e.g. an offer to invest most of the money you need), you can tell the other investors you're talking to that you have an offer good enough to accept, and give them a few days to make their own. This could lose you some that might have made an offer if they had more time. But by definition you don't care; the initial offer was acceptable.

Some investors will try to prevent others from having time to de-

cide by giving you an "exploding" offer, meaning one that's only valid for a few days. Offers from the very best investors explode less frequently and less rapidly—Fred Wilson never gives exploding offers, for example—because they're confident you'll pick them. But lower-tier investors sometimes give offers with very short fuses, because they believe no one who had other options would choose them. A deadline of three working days is acceptable. You shouldn't need more than that if you've been talking to investors in parallel. But a deadline any shorter is a sign you're dealing with a sketchy investor. You can usually call their bluff, and you may need to.*

It might seem that instead of accepting offers greedily, your goal should be to get the best investors as partners. That is certainly a good goal, but in phase 2 "get the best investors" only rarely conflicts with "accept offers greedily," because the best investors don't usually take any longer to decide than the others. The only case where the two strategies give conflicting advice is when you have to forgo an offer from an acceptable investor to see if you'll get an offer from a better one. If you talk to investors in parallel and push back on exploding offers with excessively short deadlines, that will almost never happen. But if it does, "get the best investors" is in the average case bad advice. The best investors are also the most selective, because they get their pick of all the startups. They reject nearly everyone they talk to, which means in the average case it's a bad trade to exchange a definite offer from an acceptable investor for a potential offer from a better one.

(The situation is different in phase 1. You can't apply to all the incubators in parallel, because some offset their schedules to prevent this. In phase 1, "accept offers greedily" and "get the best investors" do conflict, so if you want to apply to multiple incubators, you should do it in such a way that the ones you want most decide first.)

Sometimes when you're raising money from multiple investors, a series A will emerge out of those conversations, and these rules even

* You should respond in kind when investors behave upstandingly too. When an investor makes you a clean offer with no deadline, you have a moral obligation to respond promptly.

cover what to do in that case. When an investor starts to talk to you about a series A, keep taking smaller investments till they actually give you a termsheet. There's no practical difficulty. If the smaller investments are on convertible notes, they'll just convert into the series A round. The series A investor won't like having all these other random investors as bedfellows, but if it bothers them so much they should get on with giving you a termsheet. Till they do, you don't know for sure they will, and the greedy algorithm tells you what to do.*

Don't sell more than 25% in phase 2.

If you do well, you will probably raise a series A round eventually. I say probably because things are changing with series A rounds. Startups may start to skip them. But only one company we've funded has so far, so tentatively assume the path to huge passes through an A round.†

Which means you should avoid doing things in earlier rounds that will mess up raising an A round. For example, if you've sold more than about 40% of your company total, it starts to get harder to raise an A round, because VCs worry there will not be enough stock left to keep the founders motivated.

Our rule of thumb is not to sell more than 25% in phase 2, on top of whatever you sold in phase 1, which should be less than 15%. If

---

* Tell the investors talking to you about an A round about the smaller investments you raise as you raise them. You owe them such updates on your cap table, and this is also a good way to pressure them to act. They won't like you raising other money and may pressure you to stop, but they can't legitimately ask you to commit to them till they also commit to you. If they want you to stop raising money, the way to do it is to give you a series A termsheet with a no-shop clause

  You can relent a little if the potential series A investor has a great reputation and they're clearly working fast to get you a termsheet, particularly if a third party like YC is involved to ensure there are no misunderstandings. But be careful.

† The company is Weebly, which made it to profitability on a seed investment of $650k. They did try to raise a series A in the fall of 2008 but (no doubt partly because it was the fall of 2008) the terms they were offered were so bad that they decided to skip raising an A round.

you're raising money on uncapped notes, you'll have to guess what the eventual equity round valuation might be. Guess conservatively.

(Since the goal of this rule is to avoid messing up the series A, there's obviously an exception if you end up raising a series A in phase 2, as a handful of startups do.)

Have one person handle fundraising.

If you have multiple founders, pick one to handle fundraising so the other(s) can keep working on the company. And since the danger of fundraising is not the time taken up by the actual meetings but that it becomes the top idea in your mind, the founder who handles fundraising should make a conscious effort to insulate the other founder(s) from the details of the process.*

(If the founders mistrust one another, this could cause some friction. But if the founders mistrust one another, you have worse problems to worry about than how to organize fundraising.)

The founder who handles fundraising should be the CEO, who should in turn be the most formidable of the founders. Even if the CEO is a programmer and another founder is a salesperson? Yes. If you happen to be that type of founding team, you're effectively a single founder when it comes to fundraising.

It's ok to bring all the founders to meet an investor who will invest a lot, and who needs this meeting as the final step before deciding. But wait till that point. Introducing an investor to your cofounder(s) should be like introducing a girl/boyfriend to your parents—something you do only when things reach a certain stage of seriousness.

Even if there are still one or more founders focusing on the company during fundraising, growth will slow. But try to get as much

---

* Another advantage of having one founder take fundraising meetings is that you never have to negotiate in real time, which is something inexperienced founders should avoid. One YC founder told me:

> *Investors are professional negotiators and can negotiate on the spot very easily. If only one founder is in the room, you can say "I need to circle back with my cofounder" before making any commitments. I used to do this all the time.*

growth as you can, because fundraising is a segment of time, not a point, and what happens to the company during that time affects the outcome. If your numbers grow significantly between two investor meetings, investors will be hot to close, and if your numbers are flat or down they'll start to get cold feet.

You'll need an executive summary and (maybe) a deck.

Traditionally phase 2 fundraising consists of presenting a slide deck in person to investors. Sequoia describes what such a deck should contain, and since they're the customer you can take their word for it.

I say "traditionally" because I'm ambivalent about decks, and (though perhaps this is wishful thinking) they seem to be on the way out. A lot of the most successful startups we fund never make decks in phase 2. They just talk to investors and explain what they plan to do. Fundraising usually takes off fast for the startups that are most successful at it, and they're thus able to excuse themselves by saying that they haven't had time to make a deck.

You'll also want an executive summary, which should be no more than a page long and describe in the most matter of fact language what you plan to do, why it's a good idea, and what progress you've made so far. The point of the summary is to remind the investor (who may have met many startups that day) what you talked about.

Assume that if you give someone a copy of your deck or executive summary, it will be passed on to whoever you'd least like to have it. But don't refuse on that account to give copies to investors you meet. You just have to treat such leaks as a cost of doing business. In practice it's not that high a cost. Though founders are rightly indignant when their plans get leaked to competitors, I can't think of a startup whose outcome has been affected by it.

Sometimes an investor will ask you to send them your deck and/or executive summary before they decide whether to meet with you. I wouldn't do that. It's a sign they're not really interested.

Stop fundraising when it stops working.

When do you stop fundraising? Ideally when you've raised enough. But what if you haven't raised as much as you'd like? When do you give up?

It's hard to give general advice about this, because there have been cases of startups that kept trying to raise money even when it seemed hopeless, and miraculously succeeded. But what I usually tell founders is to stop fundraising when you start to get a lot of air in the straw. When you're drinking through a straw, you can tell when you get to the end of the liquid because you start to get a lot of air in the straw. When your fundraising options run out, they usually run out in the same way. Don't keep sucking on the straw if you're just getting air. It's not going to get better.

Don't get addicted to fundraising.

Fundraising is a chore for most founders, but some find it more interesting than working on their startup. The work at an early stage startup often consists of unglamorous schleps. Whereas fundraising, when it's going well, can be quite the opposite. Instead of sitting in your grubby apartment listening to users complain about bugs in your software, you're being offered millions of dollars by famous investors over lunch at a nice restaurant.*

The danger of fundraising is particularly acute for people who are good at it. It's always fun to work on something you're good at. If you're one of these people, beware. Fundraising is not what will make

---

* You'll be lucky if fundraising feels pleasant enough to become addictive. More often you have to worry about the other extreme—becoming demoralized when investors reject you. As one (very successful) YC founder wrote after reading a draft of this:

> *It's hard to mentally deal with the sheer scale of rejection in fundraising and if you are not in the right mindset you will fail. Users may love you but these supposedly smart investors may not understand you at all. At this point for me, rejection still rankles but I've come to accept that investors are just not super thoughtful for the most part and you need to play the game according to certain somewhat depressing rules (many of which you are listing) in order to win.*

your company successful. Listening to users complain about bugs in your software is what will make you successful. And the big danger of getting addicted to fundraising is not merely that you'll spend too long on it or raise too much money. It's that you'll start to think of yourself as being already successful, and lose your taste for the schleps you need to undertake to actually be successful. Startups can be destroyed by this.

When I see a startup with young founders that is fabulously successful at fundraising, I mentally decrease my estimate of the probability that they'll succeed. The press may be writing about them as if they'd been anointed as the next Google, but I'm thinking "this is going to end badly."

Don't raise too much.

Though only a handful of startups have to worry about this, it is possible to raise too much. The dangers of raising too much are subtle but insidious. One is that it will set impossibly high expectations. If you raise an excessive amount of money, it will be at a high valuation, and the danger of raising money at too high a valuation is that you won't be able to increase it sufficiently the next time you raise money.

A company's valuation is expected to rise each time it raises money. If not it's a sign of a company in trouble, which makes you unattractive to investors. So if you raise money in phase 2 at a post-money valuation of $30 million, the pre-money valuation of your next round, if you want to raise one, is going to have to be at least $50 million. And you have to be doing really, really well to raise money at $50 million.

It's very dangerous to let the competitiveness of your current round set the performance threshold you have to meet to raise your next one, because the two are only loosely coupled.

But the money itself may be more dangerous than the valuation. The more you raise, the more you spend, and spending a lot of money can be disastrous for an early stage startup. Spending a lot makes it harder to become profitable, and perhaps even worse, it makes you more rigid, because the main way to spend money is people, and the

more people you have, the harder it is to change directions. So if you do raise a huge amount of money, don't spend it. (You will find that advice almost impossible to follow, so hot will be the money burning a hole in your pocket, but I feel obliged at least to try.)

Be nice.

Startups raising money occasionally alienate investors by seeming arrogant. Sometimes because they are arrogant, and sometimes because they're noobs clumsily attempting to mimic the toughness they've observed in experienced founders.

It's a mistake to behave arrogantly to investors. While there are certain situations in which certain investors like certain kinds of arrogance, investors vary greatly in this respect, and a flick of the whip that will bring one to heel will make another roar with indignation. The only safe strategy is never to seem arrogant at all.

That will require some diplomacy if you follow the advice I've given here, because the advice I've given is essentially how to play hardball back. When you refuse to meet an investor because you're not in fundraising mode, or slow down your interactions with an investor who moves too slow, or treat a contingent offer as the no it actually is and then, by accepting offers greedily, end up leaving that investor out, you're going to be doing things investors don't like. So you must cushion the blow with soft words. At YC we tell startups they can blame us. And now that I've written this, everyone else can blame me if they want. That plus the inexperience card should work in most situations: sorry, we think you're great, but PG said startups shouldn't ___, and since we're new to fundraising, we feel like we have to play it safe.

The danger of behaving arrogantly is greatest when you're doing well. When everyone wants you, it's hard not to let it go to your head. Especially if till recently no one wanted you. But restrain yourself. The startup world is a small place, and startups have lots of ups and downs. This is a domain where it's more true than usual that pride

goeth before a fall.*

Be nice when investors reject you as well. The best investors are not wedded to their initial opinion of you. If they reject you in phase 2 and you end up doing well, they'll often invest in phase 3. In fact investors who reject you are some of your warmest leads for future fundraising. Any investor who spent significant time deciding probably came close to saying yes. Often you have some internal champion who only needs a little more evidence to convince the skeptics. So it's wise not merely to be nice to investors who reject you, but (unless they behaved badly) to treat it as the beginning of a relationship.

The bar will be higher next time.

Assume the money you raise in phase 2 will be the last you ever raise. You must make it to profitability on this money if you can.

Over the past several years, the investment community has evolved from a strategy of anointing a small number of winners early and then supporting them for years to a strategy of spraying money at early stage startups and then ruthlessly culling them at the next stage. This is probably the optimal strategy for investors. It's too hard to pick winners early on. Better to let the market do it for you. But it often comes as a surprise to startups how much harder it is to raise money in phase 3.

When your company is only a couple months old, all it has to be is a promising experiment that's worth funding to see how it turns out. The next time you raise money, the experiment has to have worked. You have to be on a trajectory that leads to going public. And while there are some ideas where the proof that the experiment worked might consist of e.g. query response times, usually the proof is profitability. Usually phase 3 fundraising has to be type A fundraising.

In practice there are two ways startups hose themselves between phases 2 and 3. Some are just too slow to become profitable. They raise enough money to last for two years. There doesn't seem any

* The actual sentence in the King James Bible is "Pride goeth before destruction, and an haughty spirit before a fall."

particular urgency to be profitable. So they don't make any effort to make money for a year. But by that time, not making money has become habitual. When they finally decide to try, they find they can't.

The other way companies hose themselves is by letting their expenses grow too fast. Which almost always means hiring too many people. You usually shouldn't go out and hire 8 people as soon as you raise money at phase 2. Usually you want to wait till you have growth (and thus usually revenues) to justify them. A lot of VCs will encourage you to hire aggressively. VCs generally tell you to spend too much, partly because as money people they err on the side of solving problems by spending money, and partly because they want you to sell them more of your company in subsequent rounds. Don't listen to them.

Don't make things complicated.

I realize it may seem odd to sum up this huge treatise by saying that my overall advice is not to make fundraising too complicated, but if you go back and look at this list you'll see it's basically a simple recipe with a lot of implications and edge cases. Avoid investors till you decide to raise money, and then when you do, talk to them all in parallel, prioritized by expected value, and accept offers greedily. That's fundraising in one sentence. Don't introduce complicated optimizations, and don't let investors introduce complications either.

Fundraising is not what will make you successful. It's just a means to an end. Your primary goal should be to get it over with and get back to what will make you successful—making things and talking to users—and the path I've described will for most startups be the surest way to that destination.

Be good, take care of yourselves, and *don't leave the path*.

INVESTOR HERD DYNAMICS

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.*

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.†

---

* 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.

† 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 inter-

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

---

ests. 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.

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.*

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.

* If an investor pushes you hard to tell them about your conversations with other investors, is this someone you want as an investor?

Part III

Startup = Growth

BY PAUL GRAHAM
SEPTEMBER 2012

A startup is a company designed to grow fast. Being newly founded does not in itself make a company a startup. Nor is it necessary for a startup to work on technology, or take venture funding, or have some sort of "exit." The only essential thing is growth. Everything else we associate with startups follows from growth.

If you want to start one it's important to understand that. Startups are so hard that you can't be pointed off to the side and hope to succeed. You have to know that growth is what you're after. The good news is, if you get growth, everything else tends to fall into place. Which means you can use growth like a compass to make almost every decision you face.

Redwoods

Let's start with a distinction that should be obvious but is often overlooked: not every newly founded company is a startup. Millions of companies are started every year in the US. Only a tiny fraction are startups. Most are service businesses—restaurants, barbershops, plumbers, and so on. These are not startups, except in a few unusual cases. A barbershop isn't designed to grow fast. Whereas a search engine, for example, is.

When I say startups are designed to grow fast, I mean it in two senses. Partly I mean designed in the sense of intended, because most startups fail. But I also mean startups are different by nature, in the same way a redwood seedling has a different destiny from a bean sprout.

That difference is why there's a distinct word, "startup," for companies designed to grow fast. If all companies were essentially similar, but some through luck or the efforts of their founders ended up growing very fast, we wouldn't need a separate word. We could just talk about super-successful companies and less successful ones. But in fact startups do have a different sort of DNA from other businesses. Google is not just a barbershop whose founders were unusually lucky and hard-working. Google was different from the beginning.

To grow rapidly, you need to make something you can sell to a big market. That's the difference between Google and a barbershop. A barbershop doesn't scale.

For a company to grow really big, it must (a) make something lots of people want, and (b) reach and serve all those people. Barbershops are doing fine in the (a) department. Almost everyone needs their hair cut. The problem for a barbershop, as for any retail establishment, is (b). A barbershop serves customers in person, and few will travel far for a haircut. And even if they did the barbershop couldn't accommodate them. *

Writing software is a great way to solve (b), but you can still end up constrained in (a). If you write software to teach Tibetan to Hungarian speakers, you'll be able to reach most of the people who want it, but there won't be many of them. If you make software to teach English to Chinese speakers, however, you're in startup territory.

Most businesses are tightly constrained in (a) or (b). The distinc-

---

* Strictly speaking it's not lots of customers you need but a big market, meaning a high product of number of customers times how much they'll pay. But it's dangerous to have too few customers even if they pay a lot, or the power that individual customers have over you could turn you into a de facto consulting firm. So whatever market you're in, you'll usually do best to err on the side of making the broadest type of product for it.

tive feature of successful startups is that they're not.

Ideas

It might seem that it would always be better to start a startup than an ordinary business. If you're going to start a company, why not start the type with the most potential? The catch is that this is a (fairly) efficient market. If you write software to teach Tibetan to Hungarians, you won't have much competition. If you write software to teach English to Chinese speakers, you'll face ferocious competition, precisely because that's such a larger prize.*

The constraints that limit ordinary companies also protect them. That's the tradeoff. If you start a barbershop, you only have to compete with other local barbers. If you start a search engine you have to compete with the whole world.

The most important thing that the constraints on a normal business protect it from is not competition, however, but the difficulty of coming up with new ideas. If you open a bar in a particular neighborhood, as well as limiting your potential and protecting you from competitors, that geographic constraint also helps define your company. Bar + neighborhood is a sufficient idea for a small business. Similarly for companies constrained in (a). Your niche both protects and defines you.

Whereas if you want to start a startup, you're probably going to have to think of something fairly novel. A startup has to make something it can deliver to a large market, and ideas of that type are so valuable that all the obvious ones are already taken.

That space of ideas has been so thoroughly picked over that a startup generally has to work on something everyone else has overlooked. I was going to write that one has to make a conscious effort

---

* One year at Startup School David Heinemeier Hansson encouraged programmers who wanted to start businesses to use a restaurant as a model. What he meant, I believe, is that it's fine to start software companies constrained in (a) in the same way a restaurant is constrained in (b). I agree. Most people should not try to start startups.

to find ideas everyone else has overlooked. But that's not how most startups get started. Usually successful startups happen because the founders are sufficiently different from other people that ideas few others can see seem obvious to them. Perhaps later they step back and notice they've found an idea in everyone else's blind spot, and from that point make a deliberate effort to stay there.* But at the moment when successful startups get started, much of the innovation is unconscious.

What's different about successful founders is that they can see different problems. It's a particularly good combination both to be good at technology and to face problems that can be solved by it, because technology changes so rapidly that formerly bad ideas often become good without anyone noticing. Steve Wozniak's problem was that he wanted his own computer. That was an unusual problem to have in 1975. But technological change was about to make it a much more common one. Because he not only wanted a computer but knew how to build them, Wozniak was able to make himself one. And the problem he solved for himself became one that Apple solved for millions of people in the coming years. But by the time it was obvious to ordinary people that this was a big market, Apple was already established.

Google has similar origins. Larry Page and Sergey Brin wanted to search the web. But unlike most people they had the technical expertise both to notice that existing search engines were not as good as they could be, and to know how to improve them. Over the next few years their problem became everyone's problem, as the web grew to a size where you didn't have to be a picky search expert to notice the old algorithms weren't good enough. But as happened with Apple, by the time everyone else realized how important search was, Google was entrenched.

That's one connection between startup ideas and technology.

---

* That sort of stepping back is one of the things we focus on at Y Combinator. It's common for founders to have discovered something intuitively without understanding all its implications. That's probably true of the biggest discoveries in any field.

Rapid change in one area uncovers big, soluble problems in other areas. Sometimes the changes are advances, and what they change is solubility. That was the kind of change that yielded Apple; advances in chip technology finally let Steve Wozniak design a computer he could afford. But in Google's case the most important change was the growth of the web. What changed there was not solubility but bigness.

The other connection between startups and technology is that startups create new ways of doing things, and new ways of doing things are, in the broader sense of the word, new technology. When a startup both begins with an idea exposed by technological change and makes a product consisting of technology in the narrower sense (what used to be called "high technology"), it's easy to conflate the two. But the two connections are distinct and in principle one could start a startup that was neither driven by technological change, nor whose product consisted of technology except in the broader sense.*

Rate

How fast does a company have to grow to be considered a startup? There's no precise answer to that. "Startup" is a pole, not a threshold. Starting one is at first no more than a declaration of one's ambitions. You're committing not just to starting a company, but to starting a fast growing one, and you're thus committing to search for one of the rare ideas of that type. But at first you have no more than commitment. Starting a startup is like being an actor in that respect. "Actor" too is a pole rather than a threshold. At the beginning of his career, an actor is a waiter who goes to auditions. Getting work makes him a successful actor, but he doesn't only become an actor when he's successful.

So the real question is not what growth rate makes a company a startup, but what growth rate successful startups tend to have. For

* I got it wrong in "How to Make Wealth" when I said that a startup was a small company that takes on a hard technical problem. That is the most common recipe but not the only one.

founders that's more than a theoretical question, because it's equivalent to asking if they're on the right path.

The growth of a successful startup usually has three phases:

1. There's an initial period of slow or no growth while the startup tries to figure out what it's doing.
2. As the startup figures out how to make something lots of people want and how to reach those people, there's a period of rapid growth.
3. Eventually a successful startup will grow into a big company. Growth will slow, partly due to internal limits and partly because the company is starting to bump up against the limits of the markets it serves. *

Together these three phases produce an S-curve. The phase whose growth defines the startup is the second one, the ascent. Its length and slope determine how big the company will be.

The slope is the company's growth rate. If there's one number every founder should always know, it's the company's growth rate. That's the measure of a startup. If you don't know that number, you don't even know if you're doing well or badly.

When I first meet founders and ask what their growth rate is, sometimes they tell me "we get about a hundred new customers a month." That's not a rate. What matters is not the absolute number of new customers, but the ratio of new customers to existing ones. If you're really getting a constant number of new customers every month, you're in trouble, because that means your growth rate is decreasing.

---

* In principle companies aren't limited by the size of the markets they serve, because they could just expand into new markets. But there seem to be limits on the ability of big companies to do that. Which means the slowdown that comes from bumping up against the limits of one's markets is ultimately just another way in which internal limits are expressed.

  It may be that some of these limits could be overcome by changing the shape of the organization—specifically by sharding it.

During Y Combinator we measure growth rate per week, partly because there is so little time before Demo Day, and partly because startups early on need frequent feedback from their users to tweak what they're doing.*

A good growth rate during YC is 5-7% a week. If you can hit 10% a week you're doing exceptionally well. If you can only manage 1%, it's a sign you haven't yet figured out what you're doing.

The best thing to measure the growth rate of is revenue. The next best, for startups that aren't charging initially, is active users. That's a reasonable proxy for revenue growth because whenever the startup does start trying to make money, their revenues will probably be a constant multiple of active users.†

Compass

We usually advise startups to pick a growth rate they think they can hit, and then just try to hit it every week. The key word here is "just." If they decide to grow at 7% a week and they hit that number, they're successful for that week. There's nothing more they need to do. But if they don't hit it, they've failed in the only thing that mattered, and should be correspondingly alarmed.

Programmers will recognize what we're doing here. We're turning starting a startup into an optimization problem. And anyone

---

* This is, obviously, only for startups that have already launched or can launch during YC. A startup building a new database will probably not do that. On the other hand, launching something small and then using growth rate as evolutionary pressure is such a valuable technique that any company that could start this way probably should.

† If the startup is taking the Facebook/Twitter route and building something they hope will be very popular but from which they don't yet have a definite plan to make money, the growth rate has to be higher, even though it's a proxy for revenue growth, because such companies need huge numbers of users to succeed at all.

Beware too of the edge case where something spreads rapidly but the churn is high as well, so that you have good net growth till you run through all the potential users, at which point it suddenly stops.

who has tried optimizing code knows how wonderfully effective that sort of narrow focus can be. Optimizing code means taking an existing program and changing it to use less of something, usually time or memory. You don't have to think about what the program should do, just make it faster. For most programmers this is very satisfying work. The narrow focus makes it a sort of puzzle, and you're generally surprised how fast you can solve it.

Focusing on hitting a growth rate reduces the otherwise bewilderingly multifarious problem of starting a startup to a single problem. You can use that target growth rate to make all your decisions for you; anything that gets you the growth you need is ipso facto right. Should you spend two days at a conference? Should you hire another programmer? Should you focus more on marketing? Should you spend time courting some big customer? Should you add x feature? Whatever gets you your target growth rate.*

Judging yourself by weekly growth doesn't mean you can look no more than a week ahead. Once you experience the pain of missing your target one week (it was the only thing that mattered, and you failed at it), you become interested in anything that could spare you such pain in the future. So you'll be willing for example to hire another programmer, who won't contribute to this week's growth but perhaps in a month will have implemented some new feature that will get you more users. But only if (a) the distraction of hiring someone won't make you miss your numbers in the short term, and (b) you're sufficiently worried about whether you can keep hitting your numbers without hiring someone new.

It's not that you don't think about the future, just that you think about it no more than necessary.

In theory this sort of hill-climbing could get a startup into trou-

---

* Within YC when we say it's ipso facto right to do whatever gets you growth, it's implicit that this excludes trickery like buying users for more than their lifetime value, counting users as active when they're really not, bleeding out invites at a regularly increasing rate to manufacture a perfect growth curve, etc. Even if you were able to fool investors with such tricks, you'd ultimately be hurting yourself, because you're throwing off your own compass.

ble. They could end up on a local maximum. But in practice that never happens. Having to hit a growth number every week forces founders to act, and acting versus not acting is the high bit of succeeding. Nine times out of ten, sitting around strategizing is just a form of procrastination. Whereas founders' intuitions about which hill to climb are usually better than they realize. Plus the maxima in the space of startup ideas are not spiky and isolated. Most fairly good ideas are adjacent to even better ones.

The fascinating thing about optimizing for growth is that it can actually discover startup ideas. You can use the need for growth as a form of evolutionary pressure. If you start out with some initial plan and modify it as necessary to keep hitting, say, 10% weekly growth, you may end up with a quite different company than you meant to start. But anything that grows consistently at 10% a week is almost certainly a better idea than you started with.

There's a parallel here to small businesses. Just as the constraint of being located in a particular neighborhood helps define a bar, the constraint of growing at a certain rate can help define a startup.

You'll generally do best to follow that constraint wherever it leads rather than being influenced by some initial vision, just as a scientist is better off following the truth wherever it leads rather than being influenced by what he wishes were the case. When Richard Feynman said that the imagination of nature was greater than the imagination of man, he meant that if you just keep following the truth you'll discover cooler things than you could ever have made up. For startups, growth is a constraint much like truth. Every successful startup is at least partly a product of the imagination of growth.*

---

* Which is why it's such a dangerous mistake to believe that successful startups are simply the embodiment of some brilliant initial idea. What you're looking for initially is not so much a great idea as an idea that could evolve into a great one. The danger is that promising ideas are not merely blurry versions of great ones. They're often different in kind, because the early adopters you evolve the idea upon have different needs from the rest of the market. For example, the idea that evolves into Facebook isn't merely a subset of Facebook; the idea that evolves into Facebook is a site for Harvard undergrads.

Value

It's hard to find something that grows consistently at several percent a week, but if you do you may have found something surprisingly valuable. If we project forward we see why.

weekly	yearly
1%	1.7x
2%	2.8x
5%	12.6x
7%	33.7x
10%	142.0x

A company that grows at 1% a week will grow 1.7x a year, whereas a company that grows at 5% a week will grow 12.6x. A company making $1000 a month (a typical number early in YC) and growing at 1% a week will 4 years later be making $7900 a month, which is less than a good programmer makes in salary in Silicon Valley. A startup that grows at 5% a week will in 4 years be making $25 million a month.*

Our ancestors must rarely have encountered cases of exponential growth, because our intuitions are no guide here. What happens to fast growing startups tends to surprise even the founders.

Small variations in growth rate produce qualitatively different outcomes. That's why there's a separate word for startups, and why startups do things that ordinary companies don't, like raising money and getting acquired. And, strangely enough, it's also why they fail so

* What if a company grew at 1.7x a year for a really long time? Could it not grow just as big as any successful startup? In principle yes, of course. If our hypothetical company making $1000 a month grew at 1% a week for 19 years, it would grow as big as a company growing at 5% a week for 4 years. But while such trajectories may be common in, say, real estate development, you don't see them much in the technology business. In technology, companies that grow slowly tend not to grow as big.

frequently.

Considering how valuable a successful startup can become, anyone familiar with the concept of expected value would be surprised if the failure rate weren't high. If a successful startup could make a founder $100 million, then even if the chance of succeeding were only 1%, the expected value of starting one would be $1 million. And the probability of a group of sufficiently smart and determined founders succeeding on that scale might be significantly over 1%. For the right people—e.g. the young Bill Gates—the probability might be 20% or even 50%. So it's not surprising that so many want to take a shot at it. In an efficient market, the number of failed startups should be proportionate to the size of the successes. And since the latter is huge the former should be too.*

What this means is that at any given time, the great majority of startups will be working on something that's never going to go anywhere, and yet glorifying their doomed efforts with the grandiose title of "startup."

This doesn't bother me. It's the same with other high-beta vocations, like being an actor or a novelist. I've long since gotten used to it. But it seems to bother a lot of people, particularly those who've started ordinary businesses. Many are annoyed that these so-called startups get all the attention, when hardly any of them will amount to anything.

If they stepped back and looked at the whole picture they might be less indignant. The mistake they're making is that by basing their opinions on anecdotal evidence they're implicitly judging by the median rather than the average. If you judge by the median startup, the whole concept of a startup seems like a fraud. You have to invent a bubble to explain why founders want to start them or investors want

---

* Any expected value calculation varies from person to person depending on their utility function for money. I.e. the first million is worth more to most people than subsequent millions. How much more depends on the person. For founders who are younger or more ambitious the utility function is flatter. Which is probably part of the reason the founders of the most successful startups of all tend to be on the young side.

to fund them. But it's a mistake to use the median in a domain with so much variation. If you look at the average outcome rather than the median, you can understand why investors like them, and why, if they aren't median people, it's a rational choice for founders to start them.

Deals

Why do investors like startups so much? Why are they so hot to invest in photo-sharing apps, rather than solid money-making businesses? Not only for the obvious reason.

The test of any investment is the ratio of return to risk. Startups pass that test because although they're appallingly risky, the returns when they do succeed are so high. But that's not the only reason investors like startups. An ordinary slower-growing business might have just as good a ratio of return to risk, if both were lower. So why are VCs interested only in high-growth companies? The reason is that they get paid by getting their capital back, ideally after the startup IPOs, or failing that when it's acquired.

The other way to get returns from an investment is in the form of dividends. Why isn't there a parallel VC industry that invests in ordinary companies in return for a percentage of their profits? Because it's too easy for people who control a private company to funnel its revenues to themselves (e.g. by buying overpriced components from a supplier they control) while making it look like the company is making little profit. Anyone who invested in private companies in return for dividends would have to pay close attention to their books.

The reason VCs like to invest in startups is not simply the returns, but also because such investments are so easy to oversee. The founders can't enrich themselves without also enriching the investors.*

---

* More precisely, this is the case in the biggest winners, which is where all the returns come from. A startup founder could pull the same trick of enriching himself at the company's expense by selling them overpriced components. But it wouldn't be worth it for the founders of Google to do that. Only founders of fail-

Why do founders want to take the VCs' money? Growth, again. The constraint between good ideas and growth operates in both directions. It's not merely that you need a scalable idea to grow. If you have such an idea and don't grow fast enough, competitors will. Growing too slowly is particularly dangerous in a business with network effects, which the best startups usually have to some degree.

Almost every company needs some amount of funding to get started. But startups often raise money even when they are or could be profitable. It might seem foolish to sell stock in a profitable company for less than you think it will later be worth, but it's no more foolish than buying insurance. Fundamentally that's how the most successful startups view fundraising. They could grow the company on its own revenues, but the extra money and help supplied by VCs will let them grow even faster. Raising money lets you *choose* your growth rate.

Money to grow faster is always at the command of the most successful startups, because the VCs need them more than they need the VCs. A profitable startup could if it wanted just grow on its own revenues. Growing slower might be slightly dangerous, but chances are it wouldn't kill them. Whereas VCs need to invest in startups, and in particular the most successful startups, or they'll be out of business. Which means that any sufficiently promising startup will be offered money on terms they'd be crazy to refuse. And yet because of the scale of the successes in the startup business, VCs can still make money from such investments. You'd have to be crazy to believe your company was going to become as valuable as a high growth rate can make it, but some do.

Pretty much every successful startup will get acquisition offers too. Why? What is it about startups that makes other companies want to buy them?*

---

ing startups would even be tempted, but those are writeoffs from the VCs' point of view anyway.

* Acquisitions fall into two categories: those where the acquirer wants the business, and those where the acquirer just wants the employees. The latter type is sometimes called an HR acquisition. Though nominally acquisitions and some-

Fundamentally the same thing that makes everyone else want the stock of successful startups: a rapidly growing company is valuable. It's a good thing eBay bought Paypal, for example, because Paypal is now responsible for 43% of their sales and probably more of their growth.

But acquirers have an additional reason to want startups. A rapidly growing company is not merely valuable, but dangerous. If it keeps expanding, it might expand into the acquirer's own territory. Most product acquisitions have some component of fear. Even if an acquirer isn't threatened by the startup itself, they might be alarmed at the thought of what a competitor could do with it. And because startups are in this sense doubly valuable to acquirers, acquirers will often pay more than an ordinary investor would.*

Understand

The combination of founders, investors, and acquirers forms a natural ecosystem. It works so well that those who don't understand it are driven to invent conspiracy theories to explain how neatly things sometimes turn out. Just as our ancestors did to explain the apparently too neat workings of the natural world. But there is no secret cabal making it all work.

If you start from the mistaken assumption that Instagram was worthless, you have to invent a secret boss to force Mark Zuckerberg to buy it. To anyone who knows Mark Zuckerberg that is the reduc-

---

times on a scale that has a significant effect on the expected value calculation for potential founders, HR acquisitions are viewed by acquirers as more akin to hiring bonuses.

* I once explained this to some founders who had recently arrived from Russia. They found it novel that if you threatened a company they'd pay a premium for you. "In Russia they just kill you," they said, and they were only partly joking. Economically, the fact that established companies can't simply eliminate new competitors may be one of the most valuable aspects of the rule of law. And so to the extent we see incumbents suppressing competitors via regulations or patent suits, we should worry, not because it's a departure from the rule of law per se but from what the rule of law is aiming at.

tio ad absurdum of the initial assumption. The reason he bought Instagram was that it was valuable and dangerous, and what made it so was growth.

If you want to understand startups, understand growth. Growth drives everything in this world. Growth is why startups usually work on technology—because ideas for fast growing companies are so rare that the best way to find new ones is to discover those recently made viable by change, and technology is the best source of rapid change. Growth is why it's a rational choice economically for so many founders to try starting a startup: growth makes the successful companies so valuable that the expected value is high even though the risk is too. Growth is why VCs want to invest in startups: not just because the returns are high but also because generating returns from capital gains is easier to manage than generating returns from dividends. Growth explains why the most successful startups take VC money even if they don't need to: it lets them choose their growth rate. And growth explains why successful startups almost invariably get acquisition offers. To acquirers a fast-growing company is not merely valuable but dangerous too.

It's not just that if you want to succeed in some domain, you have to understand the forces driving it. Understanding growth is what starting a startup *consists* of. What you're really doing (and to the dismay of some observers, all you're really doing) when you start a startup is committing to solve a harder type of problem than ordinary businesses do. You're committing to search for one of the rare ideas that generates rapid growth. Because these ideas are so valuable, finding one is hard. The startup is the embodiment of your discoveries so far. Starting a startup is thus very much like deciding to be a research scientist: you're not committing to solve any specific problem; you don't know for sure which problems are soluble; but you're committing to try to discover something no one knew before. A startup founder is in effect an economic research scientist. Most don't discover anything that remarkable, but some discover relativity.

Strategy Letter I: Ben & Jerry's vs. Amazon

BY JOEL SPOLSKY
MAY 12, 2000

Building a company? You've got one very important decision to make, because it affects everything else you do. No matter what else you do, you absolutely *must* figure out which camp you're in, and gear everything you do accordingly, or you're going to have a disaster on your hands.

The decision? Whether to grow slowly, organically, and profitably, or whether to have a big bang with very fast growth and lots of capital.

The organic model is to start small, with limited goals, and slowly build a business over a long period of time. I'm going to call this the Ben and Jerry's model, because Ben and Jerry's fits this model pretty well.

The other model, popularly called "Get Big Fast" (a.k.a. "Land Grab"), requires you to raise a lot of capital, and work as quickly as possible to get big fast without concern for profitability. I'm going to call this the Amazon model, because Jeff Bezos, the founder of Amazon, has practically become the celebrity spokesmodel for Get Big Fast.

Let's look at some of the differences between these models. The

first thing to ask is: are you going into a business that has competition, or not?

Ben and Jerry's	Amazon
Lots of established competitors	New technology, no competition at first

If you don't have any real competition, like Amazon, there is a chance that you can succeed at a "land grab", that is, get as many customers as quickly as possible, so that later competitors will have a serious barrier to entry. But if you're going into an industry where there is already a well-established set of competitors, the land-grab idea doesn't make sense. You need to create your customer base by getting customers to switch over from competitors.

In general, venture capitalists aren't too enthusiastic about the idea of going into a market with pesky competitors. Personally, I'm not so scared of established competition; perhaps because I worked on Microsoft Excel during a period when it almost completely took over Lotus 123, which virtually had the market to themselves. The number one word processor, Word, displaced WordPerfect, which displaced WordStar, all of which had been near monopolies at one time or another. And Ben and Jerry's grew to be a fabulous business, even though it's not like you couldn't get ice cream before they came along. It's not impossible to displace a competitor, if that's what you want to do. (I'll talk about how to do that in a future Strategy Letter).

Another question about displacing competitors has to do with network effects and lock-in:

Ben and Jerry's	Amazon
No network effect; weak customer lock-in	Strong network effect, strong customer lock-in

A "network effect" is a situation where the more customers you have, the more customers you will get. It's based on Metcalfe's Law: the value of a network is equal to the number of users squared.

A good example is eBay. If you want to sell your old Patek Philippe watch, you're going to get a better price on eBay, because there are more buyers there. If you want to buy a Patek Philippe watch, you're going to look on eBay, because there are more sellers there.

Another extremely strong network effect is proprietary chat systems like ICQ or AOL Instant Messenger. If you want to chat with people, you have to go where they are, and ICQ and AOL have the most people by far. Chances are, your friends are using one of those services, not one of the smaller ones like MSN Instant Messenger. With all of Microsoft's muscle, money, and marketing skill, they are just not going to be able to break into auctions or instant messaging, because the network effects there are so strong.

"Lock-in" is where there is something about the business that makes people not want to switch. Nobody wants to switch their Internet provider, even if the service isn't very good, because of the hassle of changing your email address and notifying everyone of the new email address. People don't want to switch word processors if their old files can't be read by the new word processor.

Even better than lock-in is the sneaky version I call *stealth lock-in*: services which lock you in without your even realizing it. For example, all those new services like PayMyBills.com which receive your bills for you, scan them in, and show them to you on the Internet. They usually come with three months free service. But when the three months are up, if you don't want to continue with the service, you have no choice but to contact every single bill provider and ask them to change the billing address back to your house. The sheer chore of doing this is likely to prevent you from switching away from PayMyBills.com—better just to let them keep sucking $8.95 out of your bank account every month. Gotcha!

If you are going into a business that has natural network effects and lock-in, and there are no established competitors, then you *better* use the Amazon model, or somebody else will, and you simply won't be able to get a toehold.

Quick case study. In 1998, AOL was spending massively to grow

at a rate of a million customers every five weeks. AOL has nice features like chat rooms and instant messaging that provide stealth lock-in. Once you've found a group of friends you like to chat with, you are simply *not* going to switch Internet providers. That's like trying to get all new friends. In my mind that's the key reason that AOL can charge around $22 a month when there are plenty of $10 a month Internet providers.

While I was working at Juno, management just failed to understand this point, and they missed their best opportunity to overtake AOL during a land rush when everyone was coming online: they didn't spend strongly enough on customer acquisition because they didn't want to dilute existing shareholders by raising more capital, and they didn't think strategically about chat and IM, so they never developed any software features to provide the kind of stealth lock-in that AOL has. Now Juno has around 3 million people paying them an average of $5.50 a month, while AOL has around 21 million people paying them an average of $17 a month. "Oops."

Ben and Jerry's	Amazon
Little capital required; break even fast	Outrageous amounts of capital required; profitability can take years

Ben and Jerry's companies start on somebody's credit card. In their early months and years, they have to use a business model that becomes profitable extremely quickly, which may not be the ultimate business model that they want to achieve. For example, you may *want*to become a giant ice cream company with $200,000,000 in annual sales, but for now, you're going to have to *settle* for opening a little ice cream shop in Vermont, hope that it's profitable, and, if it is, reinvest the profits to expand business steadily. The Ben and Jerry's corporate history says they started with a $12,000 investment. ArsDigita says that they started with an $11,000 investment. These numbers sound like a typical MasterCard credit limit. Hmmm.

Amazon companies raise money practically as fast as anyone can spend it. There's a reason for this. They are in a terrible rush. If they are in a business with no competitors and network effects, they better get big super-fast. Every day matters. And there are *lots* of ways to substitute money for time (see sidebar). Nearly all of them are fun.

Ways to substitute money for time:

- Use prebuilt, furnished executive offices instead of traditional office space. Cost: about 3 times as much. Time saved: several months to a year, depending on market.
- Pay outrageous salaries or offer programmers BMWs as starting bonuses. Cost: about 25% extra for technical staff. Time saved: you can fill openings in 3 weeks instead of the more typical 6 months.
- Hire consultants instead of employees. Cost: about 3 times as much. Time saved: you can get consultants up and running right away.
- Having trouble getting your consultants to give you the time and attention you need? Bribe them with cash until they only want to work for you.
- Spend cash freely to spot-solve problems. If your new star programmer isn't getting a lot of work done because they are busy setting up their new house and relocating, hire a high class relocation service to do it for them. If it's taking forever to get phones installed in your new offices, buy a couple of dozen cellular phones. Internet access problems slowing people down? Just get two redundant providers. Provide a concierge available to all employees for picking up dry cleaning, getting reservations, arranging for limos to the airport, etc.

Ben and Jerry's companies just can't afford to do this, so they have to settle for growing slowly.

Ben and Jerry's	Amazon
Corporate culture is important	Corporate culture is impossible

When you are growing faster than about 100% per year, it is simply impossible for mentors to transmit corporate values to new hires. If a programmer is promoted to manager and suddenly has 5 new reports, hired just yesterday, it is simply impossible for there to be very much mentoring. Netscape is the most egregious example of this, growing from 5 to about 2000 programmers in one year. As a result, their culture was a mishmash of different people with different values about the company, all tugging in different directions.

For some companies, this might be OK. For other companies, the corporate culture is an important part of the raison-d'ĕtre of the company. Ben and Jerry's *exists* because of the values of the founders, who would not accept growing faster than the rate at which that culture can be promulgated.

Let's take a hypothetical software example. Suppose you want to break into the market for word processors. Now, this market seems to be pretty sewn up by Microsoft, but you see a niche for people who, for whatever reason, absolutely cannot have their word processors crashing on them. You are going to make a super-robust, industrial strength word processor that just won't go down and sell it at a premium to people who simply depend on word processors for their *lives*. (OK, it's a stretch. I *said* this was a hypothetical example).

Now, your corporate culture probably includes all kinds of techniques for writing highly-robust code: unit testing, formal code reviews, coding conventions, large QA departments, and so on. These techniques are not trivial; they must be learned over a period of time. While a new programmer is learning how to write robust code, they need to be mentored and coached by someone more experienced.

As soon as you try to grow so fast that mentoring and coaching is impossible, you are simply going to stop transmitting those values. New hires won't know better and will write unreliable code. They won't check the return value from malloc(), and their code will fail in

some bizarre case that they never thought about, and nobody will have time to review their code and teach them the right way to do it, and your entire competitive advantage over Microsoft Word has been squandered.

Ben and Jerry's	Amazon
Mistakes become valuable lessons	Mistakes are not really noticed

A company that is growing too fast will simply not notice when it makes a big mistake, especially of the spend-too-much-money kind. Amazon buys Junglee, a comparison shopping service, for around $180,000,000 in stock, and then suddenly realizes that comparison shopping services are not very good for their business, so they just shut it down. Having piles and piles of cash makes stupid mistakes easy to cover up.

Ben and Jerry's	Amazon
It takes a long time to get big	You get big very fast

Getting big fast gives the *impression* (if not the reality) of being successful. When prospective employees see that you're hiring 30 new people a week, they will feel like they are part of something big and exciting and successful which will IPO. They may not be as impressed by a "sleepy little company" with 12 employees and a dog, even if the sleepy company is profitable and is building a better long-term company.

As a rule of thumb, you can make a nice place to work, or you can promise people they'll get rich quick. But you have to do one of those, or you won't be able to hire.

Some of your employees will be impressed by a company with a high chance of an IPO that gives out lots of stock options. Such people will be willing to put in three or four years at a company like this, even if they hate every minute of their working days, because they

see the pot at the end of the rainbow.

If you're growing slowly and organically, the pot may be farther off. In that case, you have no choice but to make a work environment where the journey is the reward. It can't be hectic 80 hour workweeks. The office can't be a big noisy loft jammed full of folding tables and hard wooden chairs. You have to give people decent vacations. People have to be friends with their co-workers, not just coworkers. Sociology and community at work matter. Managers have to be enlightened and get off people's backs, they can't be Dilbertesque micromanagers. If you do all this, you'll attract plenty of people who have been fooled too many times by dreams of becoming a millionaire in the next IPO; now they are just looking for something *sustainable*.

Ben and Jerry's	Amazon
You'll probably succeed. You certainly won't lose *too* much money.	You have a tiny chance of becoming a billionaire, and a high chance of just failing.

With the Ben and Jerry's model, if you're even reasonably smart, you're going to succeed. It may be a bit of a struggle, there may be good years and bad years, but unless we have another *depression*, you're certainly not going to lose *too* much money, because you didn't put in too much to begin with.

The trouble with the Amazon model is that all anybody thinks about is Amazon. And there's only one Amazon. You have to think of the other 95% of companies which spend an astonishing amount of venture capital and then simply fail because nobody wants to buy their product. At least, if you follow the Ben and Jerry's model, you'll know that nobody wants your product long before you spend more than one MasterCard's worth of credit limit on it.

The Worst Thing You Can Do

The worst thing you can do is fail to decide whether you're going to be a Ben and Jerry's company or an Amazon company.

If you're going into a market with no existing competition, lock-in, and network effects, you *better* use the Amazon model, or you're going the way of Wordsworth.com, which started two years before Amazon, and nobody's ever heard of them. Or even worse, you're going to be a ghost site like MSN Auctions with virtually no chance of ever overcoming eBay.

If you're going into an established market, getting big fast is a fabulous way of wasting tons of money, as did BarnesandNoble.com. Your best hope is to do something **sustainable** and **profitable**, so that you have years to slowly take over your competition.

Still can't decide? There are other things to consider. Think of your personal values. Would you rather have a company like Amazon or a company like Ben and Jerry's? Read a couple of corporate histories -Amazon and Ben and Jerry's for starters, even though they are blatant hagiographies, and see which one jibes more with your set of core values. Actually, an even better model for a Ben and Jerry's company is Microsoft, and there are lots of histories of Microsoft. Microsoft was, in a sense, "lucky" to land the PC-DOS deal, but the company was profitable and growing all along, so they could have hung around indefinitely waiting for their big break.

Think of your risk/reward profile. Do you want to take a shot at being a billionaire by the time you're 35, even if the chances of doing that make the lottery look like a good deal? Ben and Jerry's companies are not going to do that for you.

Probably the worst thing you can do is to decide that you have to be an Amazon company, and then act like a Ben and Jerry's company (while in denial all the time). Amazon companies absolutely *must* substitute cash for time whenever they can. You may think you're smart and frugal by insisting on finding programmers who will work at market rates. But you're not so smart, because that's going to take you six months, not two months, and those 4 months might mean you miss the Christmas shopping season, so now it cost you a year, and probably made your whole business plan unviable. You may

think that it's smart to have a Mac version of your software, as well as a Windows version, but if it takes you twice as long to ship while your programmers build a compatibility layer, and you only get 15% more customers, well, you're not going to look so smart, then, are you?

Both models work, but you've got to pick one and stick to it, or you'll find things mysteriously going wrong and you won't quite know why.

Do Things That Don't Scale

BY PAUL GRAHAM
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.*

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

* 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."

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.*

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

* 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.

and then gradually switch to less manual methods.*

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

* 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."

be.*

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.†

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

---

* 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.

† 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?

out to be surprisingly long), Wufoo sent each new user a handwritten 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 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 handwritten 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.*

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,

* 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.

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.*

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

* 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.

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.*

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 histo-

* 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.

ry. 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 things he learned was "how valuable it was to source good screws." Who knew?

Consult

Sometimes we advise founders of B2B startups to take overengagement to an extreme, and to pick a single user and act as if they were consultants building something just for that one user. The initial user serves as the form for your mold; keep tweaking till you fit their needs perfectly, and you'll usually find you've made something other users want too. Even if there aren't many of them, there are probably adjacent territories that have more. As long as you can find just one user who really needs something and can act on that need, you've got a toehold in making something people want, and that's as much as any startup needs initially.*

* 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 inexperi-

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

---

enced founders. So if it's not obvious you'd be making something for a market of one, don't worry about that danger.

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.*

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

* 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.

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.*

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.†

In the best case, both components of the vector contribute to your company's DNA: the unscalable things you have to do to get

---

* Google grew big on the back of Yahoo, but that wasn't a partnership. Yahoo was their customer.

† 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.

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.

I Spread Your Idea Because…

BY SETH GODIN
OCTOBER 2010

Ideas spread when people choose to spread them. Here are some reasons why:

1. I spread your idea because it makes me feel generous.
2. …because I feel smart alerting others to what I discovered.
3. …because I care about the outcome and want you (the creator of the idea) to succeed.
4. …because I have no choice. Every time I use your product, I spread the idea (Hotmail, iPad, a tattoo).
5. …because there's a financial benefit directly to me (Amazon affiliates, multi-level marketing).
6. …because it's funny and laughing alone is no fun.
7. …because I'm lonely and sharing an idea solves that problem, at least for a while.
8. …because I'm angry and I want to enlist others in my outrage (or in shutting you down).
9. …because both my friend and I will benefit if I share the idea (Groupon).
10. …because you asked me to, and it's hard to say no to you.

11. …because I can use the idea to introduce people to one another, and making a match is both fun in the short run and community-building.
12. …because your service works better if all my friends use it (email, Facebook).
13. …because if everyone knew this idea, I'd be happier.
14. …because your idea says something that I have trouble saying directly (AA, a blog post, a book).
15. …because I care about someone and this idea will make them happier or healthier.
16. …because it's fun to make another teen snicker about prurient stuff we're not supposed to see.
17. …because the tribe needs to know about this if we're going to avoid an external threat.
18. …because the tribe needs to know about this if we're going to maintain internal order.
19. …because it's my job.
20. I spread your idea because I'm in awe of your art and the only way I can repay you is to share that art with others.

Strategy Letter II:

Chicken and Egg Problems

BY JOEL SPOLSKY
MAY 24, 2000

The idea of advertising is to lie without getting caught. Most companies, when they run an advertising campaign, simply take the most unfortunate truth about their company, turn it upside down ("lie"), and drill that lie home. Let's call it "proof by repeated assertion." For example, plane travel is cramped and uncomfortable and airline employees are rude and unpleasant, indeed the whole commercial air *system* is designed as a means of torture. So almost all airline ads are going to be about how *comfortable* and *pleasant* it is to fly and how *pampered* you will be every step of the way. When British airways showed an ad with a businessman in a plane seat dreaming that he was a baby in a basket, all sense of reasonableness was gone for good.

Need another example? Paper companies are completely devastating our national forests, clear cutting old growth forest which they don't even own. So when they advertise, they inevitably show some nice old pine forest and talk about how much they care about the environment. Cigarettes cause death, so their ads show life, like all the ads with happy smiling healthy people exercising outdoors. And so on.

When the Macintosh first came out, there was no software avail-

able for it. So obviously, Apple created a giant glossy catalog listing all the great software that was "available". Half of the items listed said, in fine print, "under development," and the other half couldn't be had for love or money. Some were such lame products nobody would buy them. But even having a thick glossy catalog with one software "product" per page described in glowing prose couldn't disguise the fact that you just could not buy a word processor or spreadsheet to run on your 128KB Macintosh. There were similar "software product guides" for NeXT and BeOS. (Attention, NeXT and BeOS bigots: I don't need any flak about your poxy operating systems, OK? Write your own column.) The only thing a software product guide tells you is that there is no software available for the system. When you see one of these beasts, run fleeing in the opposite direction.

Amiga, Atari ST, Gem, IBM TopView, NeXT, BeOS, Windows CE, General Magic, the list of failed "new platforms" goes on and on. Because they are *platforms*, they are, by definition, not very interesting in and of themselves without juicy software to run on them. But, with very few exceptions (and I'm sure I'll get a whole *host* of email from tedious supporters of arcane and unloved platforms like the Amiga or RSTS-11), no software developer with the least bit of common sense would intentionally write software for a platform with 100,000 users on a *good* day, like BeOS, when they could do the same amount of work and create software for a platform with 100,000,000 users, like Windows. The fact that anybody writes software for those oddball systems at all proves that the profit motive isn't everything: religious fervor is still alive and well. Good for you, darling. You wrote a nice microEmacs clone for the Timex Sinclair 1000. Bravo. Here's a quarter, buy yourself a treat.

So. If you're in the platform creation business, you are probably going to suffer from what is commonly known as the *chicken and egg problem*. Nobody is going to buy your platform until there's good software that runs on it, and nobody is going to write software until you have a big installed base. Ooops. It's sort of like a Gordian Knot, although a Gordian Death Spiral might be more descriptive.

The chicken and egg problem, and variants thereof, is *the* most

important element of strategy to understand. Well, OK, you can probably live without understanding it: Steve Jobs practically made a*career* out of not understanding the chicken and egg problem, *twice*. But the rest of us don't have Jobs' Personal Reality Distortion Field at our disposal, so we'll have to buckle down and study hard.

Lesson one. The classic domain of chicken and egg problems is in software platforms. But here's another chicken and egg problem: every month, *millions* of credit card companies mail out *zillions* of bills to consumers in the mail. People write paper checks, stuff them in trillions of envelopes, and mail them back. The envelopes are put in big boxes and taken to countries where labor is cheap to be opened and processed. But the whole operation costs quite a bit: the last figure I heard was that it is more than $1 per bill.

To us Internet wise-guys, that's a joke. "Email me my bill", you say. "I'll pay it online!" You say. "It'll only cost, say, 1/100000th of a penny. You'll save *millions*" Or something like that.

And you're right. So a lot of companies have tried to get into this field, which is technically known as *Bill Presentment*. One example is (guess who) Microsoft. Their solution, TransPoint, looks like this: it's a web site. You go there, and it shows you your bills. You pay them.

So, now, if you get your bills on this Microsoft system, you have to visit the web page every few days to see if any bills have arrived so you don't miss them. If you get, say, 10 bills a month, this might not be too big a hassle. Therein lies the other problem: there are only a small handful of merchants that will bill you over this system. So for all your other bills, you'll have to go elsewhere.

End result? It's not worth it. I would be surprised if 10,000 people are using this system. Now, Microsoft has to go to merchants and say, "bill your customers over our system!" And the merchants will say, "OK! How much will it cost?" And Microsoft will say, "50 cents! But it's a lot cheaper than $1!" And the merchants will say, "OK. Anything else?" And Microsoft will say, "Oh yes, it will cost you about $250,000 to set up the software, connect our systems to your systems, and get everything working."

And since Microsoft has so few dang users on this system, it's

hard to imagine why anyone would pay $250,000 to save 50 cents on 37 users. Aha! The chicken and egg problem has reared its ugly head! Customers won't show up until you have merchants, and merchants won't show up until you have customers! Eventually, Microsoft is just going to spend their way out of this predicament. For smaller companies, that's not an option. So what can you do?

Software platforms actually gives us some nice hints as to how to roast your chicken and egg problem. Let's look a bit at the history of personal computer software platforms in the years since the IBM-PC came out; maybe we'll discover something!

Most people think that the IBM-PC required PC-DOS. Not true. When the IBM-PC first came out, you had a choice of three operating systems: PC-DOS, XENIX (a wimpy 8 bit version of UNIX published by, and I am not making this up, Microsoft), and something called UCSD P-System, which was, if you can believe this, just like Java: nice, slow, portable bytecodes, about 20 years before Java.

Now, most people have never heard of XENIX or UCSD's weirdo stuff. You kids today probably think that this is because Microsoft took over the market for dinky operating systems through marketing muscle or something. Absolutely not true; Microsoft was tiny in those days. The company with the marketing muscle was Digital Research, which had a different operating system. So, why was PC-DOS the winner of the three way race?

Before the PC, the only real operating system you could get was CP/M, although the market for CP/M based computers, which cost about $10,000, was too small. They were cranky and expensive and not very user friendly. But those who did buy them, did so to use as word processors, because you could get a pretty good word processor called WordStar for CP/M, and the Apple II just could *not* do word processing (it didn't have lower case, to begin with).

Now, here's a little known fact: even DOS 1.0 was designed with a CP/M backwards compatibility mode *built in*. Not only did it have its own spiffy new programming interface, known to hard core programmers as INT 21, but it fully supported the old CP/M programming interface. It could *almost* run CP/M software. In fact, WordStar

was ported to DOS by changing *one single byte* in the code. (Real Programmers can tell you what that byte was, I've long since forgotten).

That bears mentioning again. WordStar was ported to DOS by changing *one single byte* in the code. Let that sink in.

There.

Got it?

DOS was popular *because it had software from day one*. And it had software because Tim Paterson had thought to include a CP/M compatibility feature in it, because way back in the dark ages somebody was smart about chicken and egg problems.

Fast forward. In the entire *history* of the PC platform, there have only been two major paradigm shifts that took along almost every PC user: we all switched to Windows 3.x, and then we all switched to Windows 95. Only a tiny number of people ever switched to anything else on the way. Microsoft conspiracy to take over the world? Fine, you're welcome to think that. I think it's for another, more interesting reason, which just comes back to the chicken and the egg.

We all switched to Windows 3.x. The important clue in that sentence is the *3*. Why didn't we all switch to Windows 1.0? Or Windows 2.0? Or Windows 286 or Windows 386 which followed? Is it because it takes Microsoft five releases to "get it right"? **No.**

The actual reason was even more subtle than that, and it has to do with a very arcane hardware features that first showed up on the Intel 80386 chip which Windows 3.0 required.

- Feature one: old DOS programs put things on the screen by writing directly to memory locations which corresponded to character cells on the screen. This was the only way to do output fast enough to make your program look good. But Windows ran in graphics mode. On older Intel chips, the Microsoft engineers had no choice but to flip into full screen mode when they were running DOS programs. But on the 80386, they could set up virtual memory blocks and set interrupts so that the operating system was *notified* whenever a program tried to write to screen

memory. Windows could then write the equivalent text into a graphical window on the screen instantly.

- Feature two: old DOS programs assumed they had the run of the chip. As a result, they didn't play well together. But the Intel 80386 had the ability to create "virtual" PCs, each of them acting like a complete 8086, so old PC programs could pretend like they had the computer to themselves, even while other programs were running and, themselves, pretending they had the whole computer to themselves.

So **Windows 3.x on Intel 80386s was the first version that could run multiple DOS programs respectably.** (Technically, Windows 386 could too, but 80386s were rare and expensive until about the time that Windows 3.0 came out.) Windows 3.0 was the first version that could actually do a reasonable job running all your old software.

Windows 95? No problem. Nice new 32 bit API, but it still ran old 16 bit software perfectly. Microsoft obsessed about this, spending a big chunk of change testing every old program they could find with Windows 95. Jon Ross, who wrote the original version of SimCity for Windows 3.x, told me that he accidentally left a bug in SimCity where he read memory that he had just freed. Yep. It worked fine on Windows 3.x, because the memory never went anywhere. Here's the amazing part: On beta versions of Windows 95, SimCity wasn't working in testing. Microsoft tracked down the bug and *added specific code to Windows 95 that looks for SimCity*. If it finds SimCity running, it runs the memory allocator in a special mode that doesn't free memory right away. That's the kind of obsession with backward compatibility that made people willing to upgrade to Windows 95.

You should be starting to get some ideas about how to break the chicken and egg problem: provide a backwards compatibility mode which either delivers a truckload of chickens, or a truckload of eggs, depending on how you look at it, and sit back and rake in the bucks.

Ah. Now back to bill presentment. Remember bill presentment? The chicken-egg problem is that you can only get your Con Ed bills,

so you won't use the service. How can you solve it? Microsoft couldn't figure it out. PayMyBills.com (and a half dozen other Silicon Valley startups) all figured it out at the same time. You provide a *backwards compatibility mode*: if the merchant won't support the system, just get the merchant to mail their damn paper bills to University Avenue, in Palo Alto, where a bunch of actual human beings will open them and scan them in. Now you can get *all* your bills on their web site. Since every merchant on earth is available on the system, customers are happy to use it, even if it is running in this weird backwards compatibility mode where stupid Visa member banks send the bill electronically to a printer, print it out on paper, stuff it in an envelope, ship it 1500 miles to California, where it is cut open, the stupid flyers harping worthless "free" AM clock radios that actually cost $9.95 are thrown into a landfill somewhere, and the paper bill is scanned back into a computer and stuck up on the web where it should have been sent in the first place. But the stupid backwards compatibility mode will eventually go away, because PayMyBills.com, unlike Microsoft, can actually get customers to use their system, so pretty soon they'll be able to go to the stupid Visa member banks and say, "hey, I've got 93,400 of your customers. Why don't you save yourselves $93,400 each month with a direct wire connection to me?" And suddenly PayMyBills.com is very profitable while Microsoft is still struggling to sign up their second electric utility, maybe one serving Georgia would be a nice change of pace.

Companies that fail to recognize the Chicken and Egg problem can be thought of as *boil the ocean* companies: their business plan requires 93,000,000 humans to cooperate with their crazy business scheme before it actually works. One of the most outrageously stupid ideas I ever encountered was called ActiveNames. Their boneheaded idea was that everybody in the world would install a little add-in to their email client which looked up people's names on their central servers to get the actual email address. Then instead of telling people that your email address is kermit@sesame-street.com, you would tell them that your ActiveName is "spolsky", and if they want to email you, they need to install this special software. Bzzzzzt. Wrong an-

swer. I can't even *begin* to list all the reasons this idea is never going to work.

Conclusion: if you're in a market with a chicken and egg problem, you *better* have a backwards-compatibility answer that dissolves the problem, or it's going to take you a loooong time to get going (like, forever).

There are a lot of other companies that recognized the chicken and egg problem face on and defeated it intelligently. When Transmeta unveiled their new CPU, it was the first time in a *long* time that a company that was *not* Intel finally admitted that if you're a CPU, and you want a zillion people to buy you, you gotta run x86 code. This after Hitachi, Motorola, IBM, MIPS, National Semiconductor, and who knows how many other companies deceived themselves into thinking that they had the right to invent a new instruction set. The Transmeta architecture assumes from day one that any business plan that calls for making a computer that doesn't run Excel is just not going anywhere.

The Hierarchy of Success

BY SETH GODIN
SEPTEMBER 2009

I think it looks like this:

1. Attitude
2. Approach
3. Goals
4. Strategy
5. Tactics
6. Execution

We spend all our time on execution. Use this word instead of that one. This web host. That color. This material or that frequency of mailing.

Big news: No one ever succeeded because of execution tactics learned from a Dummies book.

Tactics tell you what to execute. They're important, but dwarfed by strategy. Strategy determines which tactics might work.

But what's the point of a strategy if your goals aren't clear, or contradict?

Which leads the first two, the two we almost never hear about.

Approach determines how you look at the project (or your career). Do you read a lot of books? Ask a lot of questions? Use science

and testing or go with your hunches? Are you imperious? A lifehacker? When was the last time you admitted an error and made a dramatic course correction? Most everyone has a style, and if you pick the wrong one, then all the strategy, tactics and execution in the world won't work nearly as well.

As far as I'm concerned, the most important of all, the top of the hierarchy is attitude. Why are you doing this at all? What's your bias in dealing with people and problems?

Some more questions:

- How do you deal with failure?
- When will you quit?
- How do you treat competitors?
- What personality are you looking for in the people you hire?
- What's it like to work for you? Why? Is that a deliberate choice?
- What sort of decisions do you make when no one is looking?

Sure, you can start at the bottom by focusing on execution and credentials. Reading a typical blog (or going to a typical school for 16 years), it seems like that's what you're supposed to do. What a waste.

Isn't it odd that these six questions are so important and yet we almost never talk or write about them?

If the top of the hierarchy is messed up, no amount of brilliant tactics or execution is going to help you at all.

Strategy Letter III: Let Me Go Back

BY JOEL SPOLSKY
JUNE 3, 2000

When you're trying to get people to switch from a competitor to your product, you need to understand *barriers to entry*, and you need to understand them a lot better than you think, or people won't switch and you'll be waiting tables.

In an earlier letter, I wrote about the difference between two kinds of companies: the Ben and Jerry's kind of company which is trying to take over from established competition, versus the Amazon.com kind of company which is trying a "land grab" in a new field where there is no established competition. When I worked on Microsoft Excel in the early 90's, it was a card-carrying member of the Ben and Jerry's camp. Lotus 123, the established competitor, had an almost complete monopoly in the market for spreadsheets. Sure, there were new users buying computers who started out with Excel, but for the most part, if Microsoft wanted to sell spreadsheets, they were going to have to get people to switch.

The most important thing to do when you're in this position is to *admit it*. Some companies can't even do this. The management at my last employer, Juno, was unwilling to admit that AOL had already achieved a dominant position. They spoke of the "millions of people

not yet online." They said that "in every market, there is room for two players: Time and Newsweek, Coke and Pepsi, etc." The only thing they wouldn't say is "we have to get people to switch away from AOL." I'm not sure what they were afraid of. Perhaps they thought they were afraid to "wake up the sleeping bear". When one of Juno's star programmers (no, not me) had the *chutzpah*, the unmitigated *gall* to ask a simple question at a company meeting: "Why aren't we doing more to get AOL users to switch?" they hauled him off, screamed at him for an hour, and denied him a promotion he had been promised. (Guess who took his talent elsewhere?)

There's nothing wrong with being in a market that has established competition. In fact, even if your product is radically new, like eBay, you probably have competition: garage sales! Don't stress too much. If your product *is* better in some way, you actually have a pretty good chance of getting people to switch. But you have to think strategically about it, and thinking strategically means thinking *one step beyond* the obvious.

The *only strategy* in getting people to switch to your product is to *eliminate barriers*. Imagine that it's 1991. The dominant spreadsheet, with 100% market share, is Lotus 123. You're the product manager for Microsoft Excel. Ask yourself: what are the barriers to switching? What keeps users from becoming Excel customers tomorrow?

Barrier
1. They have to know about Excel and know that it's better
2. They have to buy Excel
3. They have to buy Windows to run Excel
4. They have to convert their existing spreadsheets from 123 to Excel
5. They have to rewrite their keyboard macros which won't run in Excel
6. They have to learn a new user interface
7. They need a faster computer with more memory

And so on, and so on. Think of these barriers as an obstacle course that people have to run before you can count them as your customers. If you start out with a field of 1000 runners, about half of them will trip on the tires; half of the survivors won't be strong enough to jump the wall; half of *those* survivors will fall off the rope ladder into the mud, and so on, until only 1 or 2 people actually overcome all the hurdles. With 8 or 9 barriers, *everybody* will have one non-negotiable deal killer.

This calculus means that **eliminating barriers to switching** is the most important thing you have to do if you want to take over an existing market, because eliminating *just one barrier* will likely *double* your sales. Eliminate two barriers, and you'll double your sales again. Microsoft looked at the list of barriers and worked on *all of them*:

Barrier	Solution
1. They have to know about Excel and know that it's better	Advertise Excel, send out demo disks, and tour the country showing it off
2. They have to buy Excel	Offer a special discount for former 123 users to switch to Excel
3. They have to buy Windows to run Excel	Make a runtime version of Windows which ships free with Excel
4. They have to convert their existing spreadsheets from 123 to Excel	Give Excel the capability to read 123 spreadsheets
5. They have to rewrite their keyboard macros which won't run in Excel	Give Excel the capability to run 123 macros
6. They have to learn a new user interface	Give Excel the ability to understand Lotus keystrokes, in case you were used to the old way of doing things
7. They need a faster computer with more memory	Wait for Moore's law to solve the problem of computer power

And it worked pretty well. By incessant pounding on eliminating barriers, they slowly pried some market share away from Lotus.

One thing you see a lot when there is a transition from an old monopoly to a new monopoly is that there is a magic "tipping point": one morning, you wake up and your product has 80% market share instead of 20% market share. This flip tends to happen *very* quickly (VisiCalc to 123 to Excel, WordStar to WordPerfect to Word, Mosaic to Netscape to Internet Explorer, dBase to Access, and so on). It usually happens because the very last barrier to entry has fallen and suddenly it's logical for everyone to switch.

Obviously, it's important to work on fixing the obvious barriers to entry, but once you think you've addressed those, you need to figure out what the non-so-obvious ones are. And this is where strategy

becomes tricky, because there are some non-obvious things that keep people from switching.

Here's an example. This summer I'm spending most of my time in a house near the beach, but my bills still go to the apartment in New York City. And I travel a lot. There's a nice web service, PayMyBills.com, which is supposed to simplify your life: you have *all* your bills sent to them, and they scan them and put them on the web for you to see wherever you may be.

Now, PayMyBills costs about $9 a month, which sounds reasonable, and I would consider using it, but in the past, I've had pretty bad luck with financial services on the Internet, like Datek, which made so many *arithmetic* mistakes in my statements I couldn't believe they were licensed. So I'm willing to *try* PayMyBills, but if I don't like it, I want to be able to go back to the old way.

The trouble is, after I use PayMyBills, if I don't like it, I need to call every damn credit card company and change my address *again*. That's a lot of work. And so the *fear* of how hard it will be *to switch back* is keeping me from using their service. Earlier I called this "stealth lock-in," and sort of praised it, but if potential customers figure it out, oh boy are you in trouble.

That's the barrier to entry. Not how hard it is to switch *in*: it's how hard it might be to switch *out*.

And this reminded me of Excel's tipping point, which happened around the time of Excel 4.0. And the biggest reason was that Excel 4.0 was the first version of Excel that could **write** Lotus spreadsheets transparently.

Yep, you heard me. **Write**. Not read. It turns out that what was stopping people from switching to Excel was that everybody else they worked with was still using Lotus 123. They didn't want a product that would create spreadsheets that nobody else could read: a classic Chicken and Egg problem. When you're the lone Excel fan in a company where everyone else is using 123, even if you love Excel, you can't switch until you can participate in the 123 ecology.

To take over a market, you have to address *every* barrier to entry. If you forget just one barrier which trips up 50% of your potential

customers, then *by definition*, you can't have more than 50% market share, and you will never displace the dominant player, and you'll be stuck on the sad (omelet) side of chicken and egg problems.

The trouble is that most managers only think about strategy one step at a time, like chess players who refuse to think one move ahead. Most of them will say, "it's important to let people convert *into* your product, but why should I waste my limited engineering budget letting people convert *out*?"

That's a childish approach to strategy. It reminds me of independent booksellers, who said "why should I make it comfortable for people to read books in my store? I want them to buy the books!" And then one day Barnes and Nobles puts *couches* and *cafes* in the stores and practically *begged* people to read books in their store without buying them. Now you've got all these customers sitting in their stores for *hours* at a time, mittengrabben all the books with their filthy hands, and the probability that they find something they want to buy is linearly proportional to the amount of time they spend in the store, and even the dinkiest Barnes and Nobles superstore in Iowa City rakes in hundreds of dollars a *minute* while the independent booksellers are going out of business. Honey, Shakespeare and Company on Manhattan's Upper West Side did *not* close because Barnes and Nobles had cheaper prices, it closed because Barnes and Nobles had *more human beings in the building*.

The mature approach to strategy is not to try to force things on potential customers. If somebody *isn't even your customer yet*, trying to lock them in just isn't a good idea. When you have 100% market share, come talk to me about lock-in. Until then, if you try to lock them in now, it's too early, and if any customer catches you in the act, you'll just wind up locking them *out*. Nobody wants to switch to a product that is going to eliminate their freedom in the future.

Let's take a more current example: ISPs, a highly competitive market. Something that virtually no ISP offers is the ability to get your email forwarded to another email address *after you quit their service*. This is small-minded thinking of the worst sort, and I'm pretty surprised nobody has figured it out. If you're a small ISP try-

ing to get people to switch, they are going to be worrying about the biggest barrier: telling all their friends their new email address. So they won't even want to try your service. If they do try it, they won't tell their friends the new address for a while, just in case it doesn't work out. Which means they won't be getting much email at the new address, which means they won't really be trying out the service and seeing how much better they like it. Lose-lose.

Now suppose one brave ISP would make the following promise: "Try us. If you don't like us, we'll keep your email address functioning, and we'll forward your email for free to any other ISP. For life. Hop around from ISP to ISP as many times as you want, just let us know, and we'll be your permanent forwarding service."

Of course, the business managers would have fits. Why should we make it *easy* for customers to leave the service? That's because they are short sighted. These are *not your customers now*. Try to lock them in *before* they become your customers, and you'll just lock them *out*. But if you make an honest promise that it will be easy to back out of the service if they're not happy, and suddenly you eliminate one more barrier to entry. And, as we learned, eliminating even a single barrier to entry can have a dramatic effect on conversions, and over time, when you knock down that last barrier to entry, people will start flooding in, and life will be good for a while. Until somebody does the same thing to you.

Strategy Letter V.

BY JOEL SPOLSKY
JUNE 12, 2002

When I was in college I took two intro economics courses: macroeconomics and microeconomics. Macro was full of theories like "low unemployment causes inflation" that never quite stood up to reality. But the micro stuff was both cool and useful. It was full of interesting concepts about the relationships between supply and demand that really did work. For example, if you have a competitor who lowers their prices, the demand for your product will go down unless you match them.

In today's episode, I'll show how one of those concepts explains a lot about some familiar computer companies. Along the way, I noticed something interesting about open source software, which is this: most of the companies spending big money to develop open source software are doing it because it's a good business strategy for them, not because they suddenly stopped believing in capitalism and fell in love with freedom-as-in-speech.

Every product in the marketplace has *substitutes* and *complements*. A substitute is another product you might buy if the first product is too expensive. Chicken is a substitute for beef. If you're a chicken farmer and the price of beef goes up, the people will want more chicken, and you will sell more.

A complement is a product that you usually buy together with

another product. Gas and cars are complements. Computer hardware is a classic complement of computer operating systems. And babysitters are a complement of dinner at fine restaurants. In a small town, when the local five star restaurant has a two-for-one Valentine's day special, the local babysitters double their rates. (Actually, the nine-year-olds get roped into early service.)

All else being equal, demand for a product increases when the prices of its complements decrease.

Let me repeat that because you might have dozed off, and it's important. Demand for a product increases when the prices of its complements decrease. For example, if flights to Miami become cheaper, demand for hotel rooms in Miami goes up—because more people are flying to Miami and need a room. When computers become cheaper, more people buy them, and they all need operating systems, so demand for operating systems goes up, which means the price of operating systems can go up.

At this point, it's pretty common for people to try to confuse things by saying, "aha! But Linux is FREE!" OK. First of all, when an economist considers price, they consider the total price, including some intangible things like the time it takes to set up, reeducate everyone, and convert existing processes. All the things that we like to call "total cost of ownership."

Secondly, by using the free-as-in-beer argument, these advocates try to believe that they are not subject to the rules of economics because they've got a nice zero they can multiply everything by. Here's an example. When Slashdot asked Linux developer Moshe Bar if future Linux kernels would be compatible with existing device drivers, he said that they didn't need to. "Proprietary software goes at the tariff of US$ 50-200 per line of debugged code. No such price applies to OpenSource software." Moshe goes on to claim that it's OK for every Linux kernel revision to make all existing drivers obsolete, because the cost of rewriting all those existing drivers is zero. This is completely wrong. He's basically claiming that spending a small amount of programming time making the kernel backwards compatible is equivalent to spending a huge amount of programming time rewrit-

ing every driver, because both numbers are multiplied by their "cost," which he believes to be zero. This is a prima facie fallacy. The thousands or millions of developer hours it takes to revise every existing device driver are going to have to come at the expense of something. And until that's done, Linux will be once again handicapped in the marketplace because it doesn't support existing hardware. Wouldn't it be better to use all that "zero cost" effort making Gnome better? Or supporting new hardware?

Debugged code is NOT free, whether proprietary or open source. Even if you don't pay cash dollars for it, it has opportunity cost, and it has time cost. There is a finite amount of volunteer programming talent available for open source work, and each open source project competes with each other open source project for the same limited programming resource, and only the sexiest projects really have more volunteer developers than they can use. To summarize, I'm not very impressed by people who try to prove wild economic things about free-as-in-beer software, because they're just getting divide-by-zero errors as far as I'm concerned.

Open source is not exempt from the laws of gravity or economics. We saw this with Eazel, ArsDigita, The Company Formerly Known as VA Linux and a lot of other attempts. But something is still going on which very few people in the open source world really understand: a lot of very large public companies, with responsibilities to maximize shareholder value, are investing a lot of money in supporting open source software, usually by paying large teams of programmers to work on it. And that's what the principle of complements explains.

Once again: demand for a product increases when the price of its complements decreases. In general, a company's strategic interest is going to be to get the price of their complements as low as possible. The lowest theoretically sustainable price would be the "commodity price"—the price that arises when you have a bunch of competitors offering indistinguishable goods. So:

Smart companies try to commoditize their products' complements.

If you can do this, demand for your product will increase and you will be able to charge more and make more.

When IBM designed the PC architecture, they used off-the-shelf parts instead of custom parts, and they carefully documented the interfaces between the parts in the (revolutionary) IBM-PC Technical Reference Manual. Why? So that other manufacturers could join the party. As long as you match the interface, you can be used in PCs. **IBM's goal was to commoditize the add-in market,** which is a complement of the PC market, and they did this quite successfully. Within a short time scrillions of companies sprung up offering memory cards, hard drives, graphics cards, printers, etc. Cheap add-ins meant more demand for PCs.

When IBM licensed the operating system PC-DOS from Microsoft, Microsoft was very careful not to sell an exclusive license. This made it possible for Microsoft to license the same thing to Compaq and the other hundreds of OEMs who had legally cloned the IBM PC using IBM's own documentation. **Microsoft's goal was to commoditize the PC market.** Very soon the PC itself was basically a commodity, with ever decreasing prices, consistently increasing power, and fierce margins that make it extremely hard to make a profit. The low prices, of course, increase demand. Increased demand for PCs meant increased demand for their complement, MS-DOS. All else being equal, the greater the demand for a product, the more money it makes for you. And that's why Bill Gates can buy Sweden and you can't.

This year Microsoft's trying to do it again: their new game console, the XBox, uses commodity PC hardware instead of custom parts. The theory was that commodity hardware gets cheaper every year, so the XBox could ride down the prices. Unfortunately it seems to have backfired: apparently commodity PC hardware has already been squeezed down to commodity prices, and so the price of making an XBox isn't declining as fast as Microsoft would like. The other part of Microsoft's XBox strategy was to use DirectX, a graphics library that can be used to write code that runs on all kinds of video chips. The goal here is to make the video chip a commodity, to lower

its price, so that more games are sold, where the real profits occur. And why don't the video chip vendors of the world try to commoditize the games, somehow? That's a *lot* harder. If the game *Halo* is selling like crazy, it doesn't really *have* any substitutes. You're not going to go to the movie theatre to see Star Wars: Attack of the Clones and decide instead that you would be satisfied with a Woody Allen movie. They may both be great movies, but they're not perfect substitutes. Now: who would you rather be, a game publisher or a video chip vendor?

Commoditize your complements.

Understanding this strategy actually goes a long, long way in explaining why many commercial companies are making big contributions to open source. Let's go over these.

Headline: IBM Spends Millions to Develop Open Source Software.

Myth: They're doing this because Lou Gerstner read the GNU Manifesto and decided he doesn't actually like capitalism.

Reality: They're doing this because IBM is becoming an IT consulting company. IT consulting is a complement of enterprise software. Thus IBM needs to commoditize enterprise software, and the best way to do this is by supporting open source. Lo and behold, their consulting division is winning big with this strategy.

Headline: Netscape Open Sources Their Web Browser.

Myth: They're doing this to get free source code contributions from people in cyber cafes in New Zealand.

Reality: They're doing this to commoditize the web browser.

This has been Netscape's strategy *from day one*. Have a look at the very first Netscape press release: the browser is "freeware." Netscape gave away the browser so they could make money on servers. Browsers and servers are classic complements. The cheaper the browsers, the more servers you sell. This was never as true as it was in October 1994. (Netscape was actually surprised when MCI came in the door and dumped so much money in their laps that they realized they could make money off of the browser, too. This wasn't required by the business plan.)

When Netscape released Mozilla as Open Source, it was because they saw an opportunity to lower the cost of developing the browser. So they could get the commodity benefits at a lower cost.

Later AOL/Time Warner acquired Netscape. The server software, which was supposed to be the beneficiary of commodity browsers, wasn't doing all that well, and was jettisoned. Now: why would AOL/Time Warner continue to invest anything in open source?

AOL/Time Warner is an entertainment company. Entertainment companies are the complement of entertainment delivery platforms of all types, including web browsers. This giant conglomerate's strategic interest is to make entertainment delivery—web browsers—a commodity for which nobody can charge money.

My argument is a little bit tortured by the fact that Internet Explorer is free-as-in-beer. Microsoft wanted to make web browsers a commodity, too, so they can sell desktop and server operating systems. They went a step further and delivered a collection of components which anyone could use to throw together a web browser. Neoplanet, AOL, and Juno used these components to build their own web browsers. Given that IE is free, what is the incentive for Netscape to make the browser "even cheaper"? It's a preemptive move. They need to prevent Microsoft getting a complete monopoly in web browsers, even free web browsers, because that would theoretically give Microsoft an opportunity to increase the cost of web browsing in other ways—say, by increasing the price of Windows.

(My argument is even more shaky because it's pretty clear that Netscape in the days of Barksdale didn't exactly know what it was doing. A more likely explanation for what Netscape did is that upper management was technologically inept, and they had no choice but to go along with whatever scheme the developers came up with. The developers were hackers, not economists, and only coincidentally came up with a scheme which serves their strategy. But let's give them the benefit of the doubt.)

Headline: Transmeta Hires Linus, Pays Him To Hack on Linux.

Myth: They just did it to get publicity. Would you have heard of

Transmeta otherwise?

Reality: Transmeta is a CPU company. The natural complement of a CPU is an operating system. Transmeta wants OSs to be a commodity.

Headline: Sun and HP Pay Ximian To Hack on Gnome.

Myth: Sun and HP are supporting free software because they like Bazaars, not Cathedrals.

Reality: Sun and HP are hardware companies. They make boxen. In order to make money on the desktop, they need for windowing systems, which are a complement of desktop computers, to be a commodity. Why don't they take the money they're paying Ximian and use it to develop a proprietary windowing system? They tried this (Sun had NeWS and HP had New Wave), but these are really hardware companies at heart with pretty crude software skills, and they need windowing systems to be a *cheap commodity,* not a proprietary advantage which they have to pay for. So they hired the nice guys at Ximian to do this for the same reason that Sun bought Star Office and open sourced it: to commoditize software and make more money on hardware.

Headline: Sun Develops Java; New "Bytecode" System Means Write Once, Run Anywhere.

The bytecode idea is not new—programmers have always tried to make their code run on as many machines as possible. (That's how you commoditize your complement). For years Microsoft had its own p-code compiler and portable windowing layer which let Excel run on Mac, Windows, and OS/2, and on Motorola, Intel, Alpha, MIPS and PowerPC chips. Quark has a layer which runs Macintosh code on Windows. The C programming language is best described as a hardware-independent assembler language. It's not a new idea to software developers.

If you can run your software anywhere, that makes hardware more of a commodity. As hardware prices go down, the market expands, driving more demand for software (and leaving customers with extra money to spend on software which can now be more expensive.)

Sun's enthusiasm for WORA is, um, *strange*, because Sun is a hardware company. Making hardware a commodity is the *last* thing they want to do.

Ooooooooooooooooooooops!

Sun is the loose cannon of the computer industry. Unable to see past their raging fear and loathing of Microsoft, they adopt strategies based on anger rather than self-interest. Sun's two strategies are (a) make software a commodity by promoting and developing free software (Star Office, Linux, Apache, Gnome, etc), and (b) make hardware a commodity by promoting Java, with its bytecode architecture and WORA. OK, Sun, pop quiz: when the music stops, where are you going to sit down? Without proprietary advantages in hardware or software, you're going to have to take the commodity price, which barely covers the cost of cheap factories in Guadalajara, not your cushy offices in Silicon Valley.

"But Joel!" Jared says. "Sun is trying to commoditize the operating system, like Transmeta, not the hardware." Maybe, but the fact that Java bytecode also commoditizes the hardware is some pretty significant collateral damage to sustain.

An important thing you notice from all these examples is that it's easy for software to commoditize hardware (you just write a little hardware abstraction layer, like Windows NT's HAL, which is a tiny piece of code), but it's incredibly hard for hardware to commoditize software. Software is not interchangable, as the StarOffice marketing team is learning. Even when the price is zero, the cost of switching from Microsoft Office is non-zero. Until the switching cost becomes zero, desktop office software is not truly a commodity. And even the smallest differences can make two software packages a pain to switch between. Despite the fact that Mozilla has all the features I want and I'd love to use it if only to avoid the whack-a-mole pop-up-ad game, I'm too used to hitting Alt+D to go to the address bar. So sue me. One tiny difference and you lose your commodity status. But I've pulled hard drives out of IBM computers and slammed them into Dell computers and, boom, the system comes up perfectly and runs as if it were still in the old computer.

Amos Michelson, the CEO of Creo, told me that every employee in his firm is required to take a course in what he calls "economic thinking." Great idea. Even simple concepts in basic microeconomics go a long way to understanding some of the fundamental shifts going on today.

Part IV

Do Something!

BY SETH GODIN
2003

Here's a question that you should clip out and tape to your bathroom mirror. It might save you some angst 15 years from now. The question is, What did you do back when interest rates were at their lowest in 50 years, crime was close to zero, great employees were looking for good jobs, computers made product development and marketing easier than ever, and there was almost no competition for good news about great ideas?

Many people will have to answer that question by saying, "I spent my time waiting, whining, worrying, and wishing." Because that's what seems to be going around these days. Fortunately, though, not everyone will have to confess to having made such a bad choice.

While your company has been waiting for the economy to rebound, Reebok has launched Travel Trainers, a very cool-looking lightweight sneaker for travelers. They are selling out in Japan—from vending machines in airports!

While Detroit's car companies have been whining about gas prices and bad publicity for SUVs (SUVs are among their most profitable products), Honda has been busy building cars that look like SUVs but get twice the gas mileage. The Honda Pilot was so popular, it had

a waiting list.

While Africa's economic plight gets a fair amount of worry, a little startup called ApproTEC is actually doing something about it. The new income that its products generate accounts for 0.5% of the entire GDP of Kenya. How? It manufactures a $75 device that looks a lot like a StairMaster. But it's not for exercise. Instead, ApproTEC sells the machine to subsistence farmers, who use its stair-stepping feature to irrigate their land. People who buy it can move from subsistence farming to selling the additional produce that their land yields—and triple their annual income in the first year of using the product.

While you've been wishing for the inspiration to start something great, thousands of entrepreneurs have used the prevailing sense of uncertainty to start truly remarkable companies. Lucrative Web businesses, successful tool catalogs, fast-growing PR firms—all have started on a shoestring, and all have been profitable ahead of schedule. The Web is dead, right? Well, try telling that to Meetup.com, a new Web site that helps organize meetings anywhere and on any topic. It has 200,000 registered users—and counting.

Maybe you already have a clipping on your mirror that asks you what you did during the 1990s. What's your biggest regret about that decade? Do you wish that you had started, joined, invested in, or built something? Are you left wishing that you'd at least had the courage to try? In hindsight, the 1990s were the good old days. Yet so many people missed out. Why? Because it's always possible to find a reason to stay put, to skip an opportunity, or to decline an offer. And yet, in retrospect, it's hard to remember why we said no and easy to wish that we had said yes.

The thing is, we still live in a world that's filled with opportunity. In fact, we have more than an opportunity—we have an obligation. An obligation to spend our time doing great things. To find ideas that matter and to share them. To push ourselves and the people around us to demonstrate gratitude, insight, and inspiration. To take risks and to make the world better by being amazing.

Are these crazy times? You bet they are. But so were the days

when we were doing duck-and-cover air-raid drills in school, or going through the scares of Three Mile Island and Love Canal. There will always be crazy times.

So stop thinking about how crazy the times are, and start thinking about what the crazy times demand. There has never been a worse time for business as usual. Business as usual is sure to fail, sure to disappoint, sure to numb our dreams. That's why there has never been a better time for the new. Your competitors are too afraid to spend money on new productivity tools. Your bankers have no idea where they can safely invest. Your potential employees are desperately looking for something exciting, something they feel passionate about, something they can genuinely engage in and engage with.

You get to make a choice. You can remake that choice every day, in fact. It's never too late to choose optimism, to choose action, to choose excellence. The best thing is that it only takes a moment—just one second—to decide.

Before you finish this paragraph, you have the power to change everything that's to come. And you can do that by asking yourself (and your colleagues) the one question that every organization and every individual needs to ask today: Why not be great?

Rifting: Disney, Jobs, and You

BY SETH GODIN
FEBRUARY 2000

After the death of Walt Disney the man, something happened to Walt Disney the company. You see, Walt Disney was a three-time rifter. He was one of the few people who have successfully managed to find a rift in the continuum of life, to bet everything on it, and to make a profit by doing so. And he did it three times.

What's a rift? It's a big tear in the fabric of the rules that we live by. It's a fundamental change in the game, one that creates a bunch of new losers—and a handful of new winners.

Most people who build important businesses build them on a rift, usually one that they find by accident, and usually only once. Sometimes, after they've succeeded once, they fool themselves into thinking that they're so gifted that everywhere they look, they can see a rift. But Disney was different: He really was rift gifted. After all, he did it three times.

First, he noticed early on that movies would change the world of entertainment. Realizing that there would soon be a huge demand for family entertainment, he pioneered the development of the ani-

mated movie, perfecting the form with "Snow White and the Seven Dwarfs" (1937). The film was the beginning of a huge organization that would grow to dominate this new marketplace.

Unlike most folks who are lucky enough to catch a rift at the right moment, Disney didn't just declare himself a genius, collect his stock options, and relax. Nope. He looked for another rift—another change in the rules that he could turn into an opportunity.

That second rift came in the form of the automobile. Disney realized that the car was going to change the way that the American family got its entertainment. He believed that a strategically located, extravagantly designed theme park could reinvent family travel. And he was right. So, beginning with California's Disneyland in 1955, he built another huge organization around this rift—and it has dominated the theme-park industry ever since.

Once Disney was into this rift thing, he saw a third opportunity: television. Although many people regarded television simply as in-home movies, or as radio with a screen, Disney saw in it an entirely different medium. So, with properties like the Mickey Mouse Club, he set out to build a third organization, one that would produce a never-ending stream of content for this market.

A three-time winner: Someone who saw rifts and who mobilized an entire organization to take advantage of them. Someone who combined clarity of vision with tenacity of purpose. Unfortunately, since Disney's demise, his company hasn't really displayed that same rift-hungry attitude. The motto of most rifters ought to be WWWD: What Would Walt Do? I often wonder what Walt would have done with the Internet. Or with cable TV. Or with home shopping, home video, and DVD.

Another of my favorite rifters is Steve Jobs. Jobs is already much celebrated, but several of his successful rift moments are still worth a look. Here are three.

First, he realized that personal computers could serve as a tool in the home as well as in business, and he was smart enough to find the right people to build the Apple I and II. At the time, there were no headlines about how brilliant Jobs was, but he paved the way for eve-

ry single desktop computer in existence today.

Jobs's second rift was actually more difficult to seize, because it wasn't an obvious rift. Realizing that the graphical user interface that was developed for the Xerox Star could permanently change the way that computers worked, Jobs took a huge risk and came up with the Mac. Most entrepreneurs and virtually every large company would have laughed at the sheer hubris of it: to get lucky once and then to risk it all on a rift as narrow as this! Of course, we know what happened with the graphical user interface.

Jobs's third rift was, in fact, reminiscent of one that Disney would have jumped on. Jobs saw that computers would forever change the way that animated movies are made. And Pixar, the production company behind "Toy Story" and "A Bug's Life," was his bet on that rift. Having just taken my family to see "Toy Story 2," I can tell you that Jobs is on his way to a payoff of Disney-like proportions.

The surprising thing is that just about anyone could have seized any of those rifts and built hugely successful companies out of them. Jobs didn't know anyone in Hollywood—and he didn't need to. His success wasn't about connections or reputation or access to capital. In fact, being part of the company that sold the Apple II actually hindered his ability to launch the Mac, because his shareholders and employees fought the idea for years. No, Jobs succeeded because, like all rifters, when he saw an opportunity, he was single-minded in his focus and in his desire to take advantage of it.

My mom was also a rifter, though you've probably never heard of her. She saw and took advantage of two rifts that were probably bigger in scope than even Disney's rifts, albeit more prosaic in execution. First, a few decades ago, she saw that society was not only permitting women to go back to work—but it was also encouraging this behavior. Some women were going back to work because they needed the money; others were doing so because they wanted mental stimulation and social interaction.

Taking advantage of an opportunity that this rift created, my mom started hiring paid and volunteer workers for her nonprofit gift shop at the Albright-Knox Art Gallery, in Buffalo, New York. Her

overeducated, underpaid, super-dedicated workforce had extremely low turnover, was responsible for essentially no "shrinkage" (internal shoplifting), and displayed astonishing customer-service skills.

She was at the forefront of reinventing the way that museums and other institutions staffed and ran their stores. Not content to have a little shop that sold a few postcards each day, owners of such shops turned their businesses into full-fledged, cash-flow-positive enterprises.

My mom then foresaw a rift that would change the business of retail forever: People were no longer buying things only when they needed them. Instead, they were now shopping for fun. The experiential retail environment—stores that were destinations for people who were bored with TV—became an incredibly profitable phenomenon for almost every nonprofit museum store in the country. By watching for such rifts and then taking advantage of them, my mom was able to change fundamentally the marketing equation for her industry.

Was my mom the first person to notice these two rifts? Not at all! But she was a pioneer in executing against each one. And she did it with confidence and without hesitation.

So why doesn't everyone do this? If Disney and Jobs and my mom can become successful rifters, why can't you and your colleagues do the same? What stops existing companies from grabbing hold of rifts? Why didn't an established coffee company like Maxwell House foresee a rift in the way that adults would spend time and money? (If you don't drink alcohol, where do you go to hang out?) Why did it take a startup called Starbucks to see that rift? Industry by industry, we're seeing more and more startups catching up to—and then destroying—the old-guard market leaders.

Why do companies have to be destroyed before the way that we serve markets can evolve? Why don't the existing players see a rift when it's right in front of their eyes—and then jump into it? One reason why companies and individuals hesitate is because they don't know how to zoom. Big, successful companies aren't organized around the concept of change, and they don't reward people who

want to change the way that they do business. To them, change is bad, change is evil, change is to be feared. They have enough trouble coping with shift—but rift? Forget about it! You can't just cope with rift. Coping is out of the question. Rift appears out of nowhere and waits for a rifter to find it, grab it, and exploit it. And companies that resist zooming and insist on merely coping will always be last to see and to profit from a rift.

But there's also an underlying architectural problem. Market leaders will always be willing to make incremental changes that please customers, employees, and shareholders. (At the very least, they want to have the support of two out of the three constituencies.) Installing air bags in cars was a smart move on the part of car companies, but it had nothing to do with a rift. Putting Federal Express tracking information on the Net was great for you, me, and the beleaguered operators at FedEx, but it didn't fundamentally change the shipping business.

The problem for established companies is that when faced with a rift, they have to make a choice. They can't please all three constituencies. And, faced with that choice, most companies just say maybe. They wait. They hope that the rift will go away, quietly. They confidently project that the startups that are jumping into the rift are overvalued, overhyped, and sure to fail.

Sometimes the old guard is right: Sometimes a rift isn't a rift at all. But, as Disney and Jobs and my mom have demonstrated, if you take advantage of all potential opportunities (and that's exactly what the venture-capital community has done with the Internet), you might find a rift—and nail it before someone else does.

Take a look at my three-step guide to rifting for entrepreneurs, employees, and CEOs.

1. Make sure that it's really a rift—and not just a hiccup. A rift is characterized by a fundamental change in one of the basic rules of the game. You can usually expand the first rip in the fabric by discussing it in hypothetical terms: "What if the transaction cost of auctions became zero?" or "What if everyone had a television?"

2. Answer every objection with a "why?" And repeat that "why?"

until you get to the core of your hesitation. Then you'll know what's really causing the discomfort, and you'll be able to deal with it.

3. Maybe-proof your organization when it comes to rifts. Require someone, anyone, in the company to sign a piece of paper that says, "I heard about this rift from so-and-so, but we're not going to do anything about it, and here's why." Allow people not to sign the paper, but require those people to give the unsigned sheet to their boss, thereby passing the buck—all the way to the president of the company, if necessary. It will take only about a week for the president to become acutely aware of the opportunities that the company is not taking advantage of.

Walt Disney, Steve Jobs, my mom—and now you—have shared the secret of rifting. My mom taught me how to rift, and now I'm passing on the secret—no, the responsibility—to you. Go rift!

Fuck Everything, We're Doing Five Blades

BY JAMES M. KILTS
CEO AND PRESIDENT, THE GILLETTE COMPANY
THE ONION
FEBRUARY 18, 2004

Would someone tell me how this happened? We were the fucking vanguard of shaving in this country. The Gillette Mach3 was *the* razor to own. Then the other guy came out with a three-blade razor. Were we scared? Hell, no. Because we hit back with a little thing called the Mach3Turbo. That's three blades *and* an aloe strip. For moisture. But you know what happened next? Shut up, I'm telling you what happened—the bastards went to four blades. Now we're standing around with our cocks in our hands, selling three blades and a strip. Moisture or no, suddenly we're the chumps. Well, fuck it. We're going to five blades.

Sure, we could go to four blades next, like the competition. That seems like the logical thing to do. After all, three worked out pretty well, and four is the next number after three. So let's play it safe. Let's make a thicker aloe strip and call it the Mach3SuperTurbo. Why innovate when we can follow? Oh, I know why: Because we're a *business*, that's why!

You think it's crazy? It *is* crazy. But I don't give a shit. From now on, we're the ones who have the edge in the multi-blade game. Are they the best a man can get? Fuck, no. *Gillette* is the best a man can get.

What part of this don't you understand? If two blades is good, and three blades is better, obviously five blades would make us the best fucking razor that ever existed. *Comprende?* We didn't claw our way to the top of the razor game by clinging to the two-blade industry standard. We got here by taking chances. Well, five blades is the biggest chance of all.

Here's the report from Engineering. Someone put it in the bathroom: I want to wipe my ass with it. They don't tell me what to invent—I tell *them*. And I'm *telling* them to stick two more blades in there. I don't care *how*. Make the blades so thin they're invisible. Put some on the handle. I don't care if they have to cram the fifth blade in perpendicular to the other four, just do it!

You're taking the "safety" part of "safety razor" too literally, grandma. Cut the strings and soar. Let's hit it. Let's roll. This is our chance to make razor history. Let's dream big. All you have to do is say that five blades can happen, and it *will* happen. If you aren't on board, then fuck you. And if you're on the board, then fuck you and your father. Hey, if I'm the only one who'll take risks, I'm sure as hell happy to hog all the glory when the five-blade razor becomes the shaving tool for the U.S. of "this is how we shave now" A.

People said we couldn't go to three. It'll cost a fortune to manufacture, they said. Well, we did it. Now some egghead in a lab is screaming "Five's crazy?" Well, perhaps he'd be more comfortable in the labs at Norelco, working on fucking electrics. Rotary blades, my white ass!

Maybe I'm wrong. Maybe we should just ride in Bic's wake and make pens. Ha! Not on your fucking life! The day I shadow a penny-ante outfit like Bic is the day I leave the razor game for good, and that won't happen until the day I die!

The market? Listen, *we* make the market. All we have to do is put her out there with a little jingle. It's as easy as, "Hey, shaving with an-

ything less than five blades is like scraping your beard off with a dull hatchet." Or "You'll be so smooth, I could snort lines off of your chin." Try "Your neck is going to be so friggin' soft, someone's gonna walk up and tie a goddamn Cub Scout kerchief under it."

I know what you're thinking now: What'll people say? Mew mew mew. Oh, no, what will people say?! Grow the *fuck* up. When you're on top, people talk. That's the price you pay for being on top. Which Gillette is, always has been, and forever shall be, Amen, five blades, sweet Jesus in heaven.

Stop. I just had a stroke of genius. Are you ready? Open your mouth, baby birds, cause Mama's about to drop you one sweet, fat nightcrawler. Here she comes: Put another aloe strip on that fucker, too. That's right. Five blades, two strips, and make the second one lather. You heard me—the second strip *lathers*. It's a whole new way to think about shaving. Don't question it. Don't say a word. Just key the music, and call the chorus girls, because we're on the edge—the razor's edge—and I feel like dancing.