# Ideas for Startups

paulgraham.com · Paul Graham · 2005-10 · source

|

October 2005
(This essay is derived from a talk at the 2005
Startup School.)

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

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

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

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

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

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

If coming up with an idea for a startup equals coming up with a

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

Actually, startup ideas are not million dollar ideas, and here's an experiment you can try to prove it: just try to sell one. Nothing evolves faster than markets. The fact that there's no market for startup ideas suggests there's no demand. Which means, in the narrow sense of the word, that startup ideas are worthless.

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 spreadsheet-like collaboration tool that doesn't even have a name yet. You wouldn't have thought of something like that except by implementing your way toward it.

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

One valuable way for an idea to be wrong is to be only a partial solution. When someone's working on a problem that seems too big, I always ask: is there some way to bite off some subset of the problem, then gradually expand from there? That will generally work unless you get trapped on a local maximum, like 1980s-style AI, or C.

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.
[1]

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

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

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

What happens when your mind wanders? It may be like doodling. Most people have characteristic ways of doodling. This habit is unconscious, but not random: I found my doodles changed after I started studying painting. I started to make the kind of gestures I'd make if I were drawing from life. They were atoms of drawing, but arranged randomly.
[2]

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

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

of time thinking about language design, and my habit of always asking "would x be useful in a programming language" just got invoked.

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

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

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

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.
[3]

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

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

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

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

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

computers.

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

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

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

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

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

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

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

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

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

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

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

Design for 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.

Notes

[1]
This phenomenon may account for a number of discrepancies currently blamed on various forbidden isms. Never attribute to malice what can be explained by math.

[2]
A lot of classic abstract expressionism is doodling of this type: artists trained to paint from life using the same gestures but without using them to represent anything. This explains why such paintings are (slightly) more interesting than random marks would be.

[3]
Bill Yerazunis had solved the problem, but he got there by another path. He made a general-purpose file classifier so good that it also worked for spam.

|