A Student's Guide to Startups

paulgraham.com · Paul Graham · 2006-10 · source

|

October 2006
(This essay is derived from a talk at MIT.)

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

I'm sure the default will always be to get a job, but starting a startup could well become as popular as grad school. In the late 90s my professor friends used to complain that they couldn't get grad students, because all the undergrads were going to work for startups. I wouldn't be surprised if that situation returns, but with one difference: this time they'll be starting their own instead of going to work for other people's.

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

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

I now realize that something does change at graduation: you lose a

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

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

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

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

You probably can't change that. Even if you could, I don't think you'd want to; someone who really, truly doesn't care what his peers

think of him is probably a psychopath. So the best you can do is consider this force like a wind, and set up your boat accordingly. If you know your peers are going to push you in some direction, choose good peers, and position yourself so they push you in a direction you like.

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

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

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

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

succeed anyway."

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

In that case, you might ask, why not wait longer? Why not go work for a while, or go to grad school, and then start a startup? And indeed, that might be a good idea. If I had to pick the sweet spot for startup founders, based on who we're most excited to see applications from, I'd say it's probably the mid-twenties. Why? What advantages does someone in their mid-twenties have over someone who's 21? And why isn't it older? What can 25 year olds do that 32 year olds can't? Those turn out to be questions worth examining.

Plus

If you start a startup soon after college, you'll be a young founder by present standards, so you should know what the relative advantages of young founders are. They're not what you might think. As a young founder your strengths are: stamina, poverty, rootlessness, colleagues, and ignorance.

The importance of stamina shouldn't be surprising. If you've heard anything about startups you've probably heard about the long hours. As far as I can tell these are universal. I can't think of any successful startups whose founders worked 9 to 5. And it's particularly necessary for younger founders to work long hours because they're probably not as efficient as they'll be later.

Your second advantage, poverty, might not sound like an advantage, but it is a huge one. Poverty implies you can live cheaply, and this is critically important for startups. Nearly every startup that fails, fails by running out of money. It's a little misleading to put it this way, because there's usually some other underlying

cause. But regardless of the source of your problems, a low burn rate gives you more opportunity to recover from them. And since most startups make all kinds of mistakes at first, room to recover from mistakes is a valuable thing to have.

Most startups end up doing something different than they planned. The way the successful ones find something that works is by trying things that don't. So the worst thing you can do in a startup is to have a rigid, pre-ordained plan and then start spending a lot of money to implement it. Better to operate cheaply and give your ideas time to evolve.

Recent grads can live on practically nothing, and this gives you an edge over older founders, because the main cost in software startups is people. The guys with kids and mortgages are at a real disadvantage. This is one reason I'd bet on the 25 year old over the 32 year old. The 32 year old probably is a better programmer, but probably also has a much more expensive life. Whereas a 25 year old has some work experience (more on that later) but can live as cheaply as an undergrad.

Robert Morris and I were 29 and 30 respectively when we started Viaweb, but fortunately we still lived like 23 year olds. We both had roughly zero assets. I would have loved to have a mortgage, since that would have meant I had a house. But in retrospect having nothing turned out to be convenient. I wasn't tied down and I was used to living cheaply.

Even more important than living cheaply, though, is thinking cheaply. One reason the Apple II was so popular was that it was cheap. The computer itself was cheap, and it used cheap, off-the-shelf peripherals like a cassette tape recorder for data storage and a TV as a monitor. And you know why? Because Woz designed this computer for himself, and he couldn't afford anything more.

We benefitted from the same phenomenon. Our prices were daringly low for the time. The top level of service was $300 a month, which was an order of magnitude below the norm. In retrospect this was a smart move, but we didn't do it because we were smart. $300 a month seemed like a lot of money to us. Like

Apple, we created something inexpensive, and therefore popular, simply because we were poor.

A lot of startups have that form: someone comes along and makes something for a tenth or a hundredth of what it used to cost, and the existing players can't follow because they don't even want to think about a world in which that's possible. Traditional long distance carriers, for example, didn't even want to think about VoIP. (It was coming, all the same.) Being poor helps in this game, because your own personal bias points in the same direction technology evolves in.

The advantages of rootlessness are similar to those of poverty. When you're young you're more mobile—not just because you don't have a house or much stuff, but also because you're less likely to have serious relationships. This turns out to be important, because a lot of startups involve someone moving.

The founders of Kiko, for example, are now en route to the Bay Area to start their next startup. It's a better place for what they want to do. And it was easy for them to decide to go, because neither as far as I know has a serious girlfriend, and everything they own will fit in one car—or more precisely, will either fit in one car or is crappy enough that they don't mind leaving it behind.

They at least were in Boston. What if they'd been in Nebraska, like Evan Williams was at their age? Someone wrote recently that the drawback of Y Combinator was that you had to move to participate. It couldn't be any other way. The kind of conversations we have with founders, we have to have in person. We fund a dozen startups at a time, and we can't be in a dozen places at once. But even if we could somehow magically save people from moving, we wouldn't. We wouldn't be doing founders a favor by letting them stay in Nebraska. Places that aren't
startup hubs are toxic to startups.
You can tell that from indirect evidence. You can tell how hard it must be to start a startup in Houston or Chicago or Miami from the microscopically small number, per capita, that succeed there. I don't know exactly what's suppressing all the startups in these

towns—probably a hundred subtle little things—but something must be.
[2]

Maybe this will change. Maybe the increasing cheapness of startups will mean they'll be able to survive anywhere, instead of only in the most hospitable environments. Maybe 37signals is the pattern for the future. But maybe not. Historically there have always been certain towns that were centers for certain industries, and if you weren't in one of them you were at a disadvantage. So my guess is that 37signals is an anomaly. We're looking at a pattern much older than "Web 2.0" here.

Perhaps the reason more startups per capita happen in the Bay Area than Miami is simply that there are more founder-type people there. Successful startups are almost never started by one person. Usually they begin with a conversation in which someone mentions that something would be a good idea for a company, and his friend says, "Yeah, that is a good idea, let's try it." If you're missing that second person who says "let's try it," the startup never happens. And that is another area where undergrads have an edge. They're surrounded by people willing to say that. At a good college you're concentrated together with a lot of other ambitious and technically minded people—probably more concentrated than you'll ever be again. If your nucleus spits out a neutron, there's a good chance it will hit another nucleus.

The number one question people ask us at Y Combinator is: Where can I find a co-founder? That's the biggest problem for someone starting a startup at 30. When they were in school they knew a lot of good co-founders, but by 30 they've either lost touch with them or these people are tied down by jobs they don't want to leave.

Viaweb was an anomaly in this respect too. Though we were comparatively old, we weren't tied down by impressive jobs. I was trying to be an artist, which is not very constraining, and Robert, though 29, was still in grad school due to a little interruption in his academic career back in 1988. So arguably the Worm made Viaweb possible. Otherwise Robert would have been a junior professor at that age, and he wouldn't have had time to work on crazy speculative projects

with me.

Most of the questions people ask Y Combinator we have some kind of answer for, but not the co-founder question. There is no good answer. Co-founders really should be people you already know. And by far the best place to meet them is school. You have a large sample of smart people; you get to compare how they all perform on identical tasks; and everyone's life is pretty fluid. A lot of startups grow out of schools for this reason. Google, Yahoo, and Microsoft, among others, were all founded by people who met in school. (In Microsoft's case, it was high school.)

Many students feel they should wait and get a little more experience before they start a company. All other things being equal, they should. But all other things are not quite as equal as they look. Most students don't realize how rich they are in the scarcest ingredient in startups, co-founders. If you wait too long, you may find that your friends are now involved in some project they don't want to abandon. The better they are, the more likely this is to happen.

One way to mitigate this problem might be to actively plan your startup while you're getting those n years of experience. Sure, go off and get jobs or go to grad school or whatever, but get together regularly to scheme, so the idea of starting a startup stays alive in everyone's brain. I don't know if this works, but it can't hurt to try.

It would be helpful just to realize what an advantage you have as students. Some of your classmates are probably going to be successful startup founders; at a great technical university, that is a near certainty. So which ones? If I were you I'd look for the people who are not just smart, but incurable
builders.
Look
for the people who keep starting projects, and finish at least some of them. That's what we look for. Above all else, above academic credentials and even the idea you apply with, we look for people who build things.

The other place co-founders meet is at work. Fewer do than at school, but there are things you can do to improve the odds. The most important, obviously, is to work somewhere that has a lot of smart, young people. Another is to work for a company located in a startup hub. It will be easier to talk a co-worker into quitting with you in a place where startups are happening all around you.

You might also want to look at the employment agreement you sign when you get hired. Most will say that any ideas you think of while you're employed by the company belong to them. In practice it's hard for anyone to prove what ideas you had when, so the line gets drawn at code. If you're going to start a startup, don't write any of the code while you're still employed. Or at least discard any code you wrote while still employed and start over. It's not so much that your employer will find out and sue you. It won't come to that; investors or acquirers or (if you're so lucky) underwriters will nail you first. Between t = 0 and when you buy that yacht, someone is going to ask if any of your code legally belongs to anyone else, and you need to be able to say no.
[3]

The most overreaching employee agreement I've seen so far is Amazon's. In addition to the usual clauses about owning your ideas, you also can't be a founder of a startup that has another founder who worked at Amazon—even if you didn't know them or even work there at the same time. I suspect they'd have a hard time enforcing this, but it's a bad sign they even try. There are plenty of other places to work; you may as well choose one that keeps more of your options open.

Speaking of cool places to work, there is of course Google. But I notice something slightly frightening about Google: zero startups come out of there. In that respect it's a black hole. People seem to like working at Google too much to leave. So if you hope to start a startup one day, the evidence so far suggests you shouldn't work there.

I realize this seems odd advice. If they make your life so good that you don't want to leave, why not work there? Because, in effect, you're probably getting a local maximum. You need a certain

activation energy to start a startup. So an employer who's fairly pleasant to work for can lull you into staying indefinitely, even if it would be a net win for you to leave.
[4]

The best place to work, if you want to start a startup, is probably a startup. In addition to being the right sort of experience, one way or another it will be over quickly. You'll either end up rich, in which case problem solved, or the startup will get bought, in which case it it will start to suck to work there and it will be easy to leave, or most likely, the thing will blow up and you'll be free again.

Your final advantage, ignorance, may not sound very useful. I deliberately used a controversial word for it; you might equally call it innocence. But it seems to be a powerful force. My Y Combinator co-founder Jessica Livingston is just about to publish a book of interviews
with startup founders, and I noticed a remarkable pattern in them. One after another said that if they'd known how hard it would be, they would have been too intimidated to start.

Ignorance can be useful when it's a counterweight to other forms of stupidity. It's useful in starting startups because you're capable of more than you realize. Starting startups is harder than you expect, but you're also capable of more than you expect, so they balance out.

Most people look at a company like Apple and think, how could I ever make such a thing? Apple is an institution, and I'm just a person. But every institution was at one point just a handful of people in a room deciding to start something. Institutions are made up, and made up by people no different from you.

I'm not saying everyone could start a startup. I'm sure most people couldn't; I don't know much about the population at large. When you get to groups I know well, like hackers, I can say more precisely. At the top schools, I'd guess as many as a quarter of the CS majors could make it as startup founders if they wanted.

That "if they wanted" is an important qualification—so important that it's almost cheating to append it like that—because once you get over a certain threshold of intelligence, which most CS majors at top schools are past, the deciding factor in whether you succeed as a founder is how much you want to. You don't have to be that smart. If you're not a genius, just start a startup in some unsexy field where you'll have less competition, like software for human resources departments. I picked that example at random, but I feel safe in predicting that whatever they have now, it wouldn't take genius to do better. There are a lot of people out there working on boring stuff who are desperately in need of better software, so however short you think you fall of Larry and Sergey, you can ratchet down the coolness of the idea far enough to compensate.

As well as preventing you from being intimidated, ignorance can sometimes help you discover new ideas. Steve Wozniak put this very strongly:

> All the best things that I did at Apple came from (a) not having money and (b) not having done it before, ever. Every single thing that we came out with that was really great, I'd never once done that thing in my life.

When you know nothing, you have to reinvent stuff for yourself, and if you're smart your reinventions may be better than what preceded them. This is especially true in fields where the rules change. All our ideas about software were developed in a time when processors were slow, and memories and disks were tiny. Who knows what obsolete assumptions are embedded in the conventional wisdom? And the way these assumptions are going to get fixed is not by explicitly deallocating them, but by something more akin to garbage collection. Someone ignorant but smart will come along and reinvent everything, and in the process simply fail to reproduce certain existing ideas.

Minus

So much for the advantages of young founders. What about the disadvantages? I'm going to start with what goes wrong and try to trace it back to the root causes.

What goes wrong with young founders is that they build stuff that looks like class projects. It was only recently that we figured

this out ourselves. We noticed a lot of similarities between the startups that seemed to be falling behind, but we couldn't figure out how to put it into words. Then finally we realized what it was: they were building class projects.

But what does that really mean? What's wrong with class projects? What's the difference between a class project and a real startup? If we could answer that question it would be useful not just to would-be startup founders but to students in general, because we'd be a long way toward explaining the mystery of the so-called real world.

There seem to be two big things missing in class projects: (1) an iterative definition of a real problem and (2) intensity.

The first is probably unavoidable. Class projects will inevitably solve fake problems. For one thing, real problems are rare and valuable. If a professor wanted to have students solve real problems, he'd face the same paradox as someone trying to give an example of whatever "paradigm" might succeed the Standard Model of physics. There may well be something that does, but if you could think of an example you'd be entitled to the Nobel Prize. Similarly, good new problems are not to be had for the asking.

In technology the difficulty is compounded by the fact that real startups tend to discover the problem they're solving by a process of evolution. Someone has an idea for something; they build it; and in doing so (and probably only by doing so) they realize the problem they should be solving is another one. Even if the professor let you change your project description on the fly, there isn't time enough to do that in a college class, or a market to supply evolutionary pressures. So class projects are mostly about implementation, which is the least of your problems in a startup.

It's not just that in a startup you work on the idea as well as implementation. The very implementation is different. Its main purpose is to refine the idea. Often the only value of most of the stuff you build in the first six months is that it proves your initial idea was mistaken. And that's extremely valuable. If

you're free of a misconception that everyone else still shares, you're in a powerful position. But you're not thinking that way about a class project. Proving your initial plan was mistaken would just get you a bad grade. Instead of building stuff to throw away, you tend to want every line of code to go toward that final goal of showing you did a lot of work.

That leads to our second difference: the way class projects are measured. Professors will tend to judge you by the distance between the starting point and where you are now. If someone has achieved a lot, they should get a good grade. But customers will judge you from the other direction: the distance remaining between where you are now and the features they need. The market doesn't give a shit how hard you worked. Users just want your software to do what they need, and you get a zero otherwise. That is one of the most distinctive differences between school and the real world: there is no reward for putting in a good effort. In fact, the whole concept of a "good effort" is a fake idea adults invented to encourage kids. It is not found in nature.

Such lies seem to be helpful to kids. But unfortunately when you graduate they don't give you a list of all the lies they told you during your education. You have to get them beaten out of you by contact with the real world. And this is why so many jobs want work experience. I couldn't understand that when I was in college. I knew how to program. In fact, I could tell I knew how to program better than most people doing it for a living. So what was this mysterious "work experience" and why did I need it?

Now I know what it is, and part of the confusion is grammatical. Describing it as "work experience" implies it's like experience operating a certain kind of machine, or using a certain programming language. But really what work experience refers to is not some specific expertise, but the elimination of certain habits left over from childhood.

One of the defining qualities of kids is that they flake. When you're a kid and you face some hard test, you can cry and say "I can't" and they won't make you do it. Of course, no one can make you do anything in the grownup world either. What they do instead

is fire you. And when motivated by that
you find you can do a lot more than you realized. So one of the
things employers expect from someone with "work experience" is the
elimination of the flake reflex—the ability to get things done,
with no excuses.

The other thing you get from work experience is an understanding
of what work is, and in particular, how intrinsically horrible it
is. Fundamentally the equation is a brutal one: you have to spend
most of your waking hours doing stuff someone else wants, or starve.
There are a few places where the work is so interesting that this
is concealed, because what other people want done happens to coincide
with what you want to work on. But you only have to imagine what
would happen if they diverged to see the underlying reality.

It's not so much that adults lie to kids about this as never explain
it. They never explain what the deal is with money. You know from
an early age that you'll have some sort of job, because everyone
asks what you're going to "be" when you grow up. What they
don't tell you is that as a kid you're sitting on the shoulders of
someone else who's treading water, and that starting working means
you get thrown into the water on your own, and have to start treading
water yourself or sink. "Being" something is incidental; the
immediate problem is not to drown.

The relationship between work and money tends to dawn on you only
gradually. At least it did for me. One's first thought tends to
be simply "This sucks. I'm in debt. Plus I have to get up on monday
and go to work." Gradually you realize that these two things are
as tightly connected as only a market can make them.

So the most important advantage 24 year old founders have over 20
year old founders is that they know what they're trying to avoid.
To the average undergrad the idea of getting rich translates into
buying Ferraris, or being admired. To someone who has learned from
experience about the relationship between money and work, it
translates to something way more important: it means you get to opt
out of the brutal equation that governs the lives of 99.9% of people.
Getting rich means you can stop treading water.

Someone who gets this will work much harder at making a startup succeed—with the proverbial energy of a drowning man, in fact. But understanding the relationship between money and work also changes the way you work. You don't get money just for working, but for doing things other people want. Someone who's figured that out will automatically focus more on the user. And that cures the other half of the class-project syndrome. After you've been working for a while, you yourself tend to measure what you've done the same way the market does.

Of course, you don't have to spend years working to learn this stuff. If you're sufficiently perceptive you can grasp these things while you're still in school. Sam Altman did. He must have, because Loopt is no class project. And as his example suggests, this can be valuable knowledge. At a minimum, if you get this stuff, you already have most of what you gain from the "work experience" employers consider so desirable. But of course if you really get it, you can use this information in a way that's more valuable to you than that.

Now

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

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

startups there's a unique problem: by definition the founders of successful startups don't need to write books to make money. As a result most books on the subject end up being written by people who don't understand it.

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

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

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

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

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

Notes

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

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

[3]
Most employee agreements

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

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

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

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

Comment on this essay.

|