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.

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

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

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

That's what these ideas say to us.

This phenomenon is one of the most important things you can understand about startups. [1] You'd expect big startup ideas to be attractive, but actually they tend to repel you. And that has a bunch of consequences. It means these ideas are invisible to most people who try to think of startup ideas, because their subconscious filters them out. Even the most ambitious people are probably best off approaching them obliquely.

1. A New Search 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) 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. [2] If you made something no better than GMail, but fast, that alone would let you start to pull users away from GMail.

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

3. Replace 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. [3]

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

6. Bring Back Moore's 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 any more. Intel can no longer give us faster CPUs, just more of them.

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

It would be great if a startup could give us something of the old Moore's Law back, by writing software that could make a large number of CPUs look to the developer like one very fast CPU. There are several ways to approach this problem. The most ambitious is to try to do it automatically: to write a compiler that will parallelize our code for us. There's a name for this compiler, *the sufficiently smart compiler,* and it is a byword for impossibility. But is it really impossible? Is there no configuration of the bits in memory of a present day computer that is this compiler? If you really think so, you should try to prove it, because that would be an interesting result. And if it's not impossible but simply very hard, it might be worth trying to write it. The expected value would be high even if the chance of succeeding was low.

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

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

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

optimization.

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

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

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

7. Ongoing 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 todo-list software. That sounds harmless. People can notice you've replaced email when it's a *fait accompli*. [4]

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

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

bigger your ambition, the longer it's going to take, and the further you project into the future, the more likely you'll get it wrong.

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

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

Notes

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

[2] This sentence originally read "GMail is painfully slow." Thanks to Paul Buchheit for the correction.

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

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

Thanks to Sam Altman, Trevor Blackwell, Paul Buchheit, Patrick Collison, Aaron Iba, Jessica Livingston, Robert Morris, Harj Taggar and Garry Tan for reading drafts of this.
