Crossroads
The ACM Student Magazine

Este artículo tambíen está en Español.

The Future of Programming: An Interview with Paul Graham

by ??

Paul Graham was co-founder of Viaweb, the first ASP; discovered the algorithm that inspired the current generation of spam filters, is co-founder of Y Combinator, a new seed venture firm, started the Spam Conference and the Startup School, is working on a new Lisp dialect called Arc, wrote two books on Lisp and a book of essays called Hackers & Painters, and is writing a new book about startups. He has a PhD in CS from Harvard and studied painting at RISD and the Accademia in Florence.

In the following interview Graham discusses the future of programming, outsourcing, and Y Combinator.

Where do you see programming as a discipline in five, ten, or twenty years?

I think in the future programmers will increasingly use dynamic languages. You already see this now: everyone seems to be migrating to Ruby, which is more or less Lisp minus macros. And Perl 6, from n what I've heard, seems to be even more Lisplike. It's

even going to have continuations.

Another trend I expect to see a lot of is Web-based applications. Microsoft managed to keep a lid on these for a surprisingly long time, by controlling the browser and making sure it couldn't do much. But now the genie is out of the bottle, and it's not going back in.

I don't think even now Microsoft realizes the danger they're in. They're worrying about Google. And they should. But they should worry even more about thousands of twenty year old hackers writing Ajax applications. Desktop software is going to become increasingly irrelevant.

What has your experience developing a new programming language, Arc, been like?

Interrupted. I haven't spent much time on it lately. Part of the problem is that I decided on an overambitious way of doing it. I'm going back to McCarthy's original axiomatic approach. The defining feature of Lisp, in his 1960 paper, was that it could be written in itself. The language spec wasn't a bunch of words. It was code.

Of course as soon as his grad students got hold of this theoretical construct and turned it into an actual programming language, that plan came to a halt. It had to, with the hardware available then. But with the much faster hardware we have now, you could have working code as the entire language spec.

I hope to get back to work on Arc soon. One of the reasons Y Combinator operates in 3-month cycles is that it leaves me some time to work on other stuff. (The other is that it's actually the right way to do seed investing.)

What is starting a startup incubator like?

Y Combinator is not really an incubator. Incubators interfere a lot in the startups they fund, even to the point of making you work in their building (which is where the name "incubator" comes from). I think the reason we get called an incubator is that we fund startups at the very beginning, and till now the only companies doing that have been incubators. Really, we're a new kind of thing, but because there's only one of us, there's no name for it.

Several things have surprised me about it. The biggest surprise is that it worked, or seems to be working so far. We had no idea what would happen if we just gave smart hackers some money and let them work on whatever they wanted. Fortunately the first batch turned out really well.

Another surprise is how much work it was. I'd hoped it would be a part-time job, but it hasn't been so far.

I'm also surprised at how fun it's been. I really like the founders. Many of them have become personal friends. And most of their startups are working on interesting, novel stuff. There's a new startup boom happening now, so there's a feeling of excitement around the Web generally, but it's especially concentrated when you have eight startups founded at the same time by young guys who all (now) know one another.

Why did you start Y Combinator?

Originally it started almost by accident. I gave a talk at Harvard about how to start a startup. In it I said that would-be founders should get their initial funding from individual rich people called "angels," and that the best angels were people who'd made their money in technology. And then, worried that I'd be deluged with business plans, I added: "but not me." I was kind of joking, but not entirely.

Afterward I felt bad about this. So I figured out a way to give seed money to startups without being deluged with pitches. We would start a company to do it, and tell people to send the pitches to the company. Of course I end up reading them in the end, but it gets concentrated into a couple weekends a year.

So the original motivation for Y Combinator was to avoid work, but as so often happens, I got sucked into it and I'm constantly coming up with new schemes that require me to do more work. Like the Startup School we just organized this October.

One of the startups we funded this summer was started by two guys who were in the audience at that original Harvard talk. And better still they're one of the more successful startups. Their site, Reddit, is so useful that almost everyone who was around Y Combinator this summer is now genuinely addicted to it, including me. It's the first site I look at every morning and the last I look at every night.

What advice can you give to aspiring entrepreneurs?

I've written a lot about this, so generally I'd advise reading the essays about startups on paulgraham.com. Especially "How to Start a Startup" and "Hiring is Obsolete."

The most important piece of advice is just: go do it. A lot of people in their early twenties are intimidated by the idea of starting a company and feel they're not ready. Actually they have a huge advantage they don't even know they have: they're not tied down.

If you don't have kids yet, you can (a) work long hours without feeling you're neglecting them, (b) live on nothing, (c) move anywhere, and (d) afford to fail. The last is the most important of all, because it means you can take risks, and risk and reward are always proportionate.

What is your position on outsourcing programming/tech jobs, and where will this lead the US?

I'm in favor of free trade in this as in everything else. If you can get a job done cheaper in another country, great. Protectionism almost always turns out to be a loss, even for the country that's supposedly being protected. It may benefit some small group within the country, but usually at the expense of everyone else.

In any case, I don't think outsourcing per se is much of a threat. I bet much of the time it's just a symptom of using a language that's not abstract enough. In effect you're using the programmers in India or wherever as human compilers.

The danger to the US is not the outsourcing of implementation, but that whole applications will get designed and implemented entirely overseas. But if other countries can develop software better than us, they deserve to win.

My guess is that they won't be able to, incidentally. You need a special environment to develop really novel technology. It's not just that you won't necessarily find this environment in India or China; you don't find it in 99% of the US either.

What motivates or inspires your work on a daily basis?

I keep having ideas for new things to do. It's almost pathological. Mostly bad ideas, of

course. But I have various tricks for filtering out those. (One of the best is asking friends.)

At any given time I'm in the grip of some scheme or other. These vary greatly in size. Some take a couple hours and others take years. The scheduling algorithm is totally random. I just work on whichever I feel like at the moment.

This may sound disorganized, but I've found that planning doesn't work well. It forces you to work on stuff you're not interested in, and then you do a bad job.

The main motivator is the schemes themselves. Once you have an idea, it would be a shame to waste it. But if there is an underlying goal, it's to make stuff that will last. That's one reason I avoid writing about politics. A lot of famous writers wasted years and years writing about controversies of their time that no one cares about now, because they were just cases of the star-bellied sneetches versus the plain-bellied ones.

What is your problem-solving approach or strategy?

That's a hard one to answer. I have a thousand and one tricks.

One thing I try to do is treat the world like math. Good mathematicians are good at visualizing problems. They can see how things must be. Actually writing down the steps must often be mere transcription, or at least, implementation.

I try to understand non-math things so well that I can rotate and rearrange them in my head like that -- so I can see how things must be, then just write it down.

For example, I try to understand history so well that I can run thought experiments in my head. How would a Roman legionary or a medieval Flemish merchant seem if we could bring one forward in a time machine? If someone like Hitler took over the US now, who would be the first recruits marching with him in the street, and who would resist? Was the European domination of the rest of the world inevitable, or due to one or two random events in Chinese politics? (Diamond wrote about the easy question. The real question is: why not China?)

What advice do you have for our readers to succeed in the current tech job market?

Are they sure they want jobs? Maybe some of them would prefer to start their own companies.

In either case the single most important thing is to work on one's own projects. When we hired hackers at our startup, this was practically the whole interview: what have you built on your own, outside of school or work?

We asked that partly to tell if someone was a real hacker, since anyone who likes to hack will invariably be working on schemes ofb their own. (Unless they've been working at a startup, which could well absorb 100% of ones's energy.)

A lot of employers have learned this test. Both Yahoo and Google seem hot to hire people who've made a name for themselves by creating admired open-source projects -- to say nothing of venture capitalists.

The other reason to work on your own projects is that that's the best way to learn. You learn by doing, and you'll work more energetically on something that interests you, and that you own.