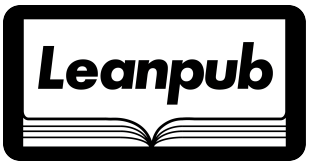
How to Do What You Love & Earn What You're Worth as a Programmer

Last published on 2012-02-29



This is a Leanpub book, for sale at:

http://leanpub.com/dowhatyoulove

Leanpub helps authors to self-publish in-progress ebooks. We call this idea Lean Publishing. To learn more about Lean Publishing, go to: http://leanpub.com/manifesto

To learn more about Leanpub, go to: http://leanpub.com

Contents

Preface

The chapters of this book originally appeared as online essays and blog posts. You can still read their original versions online, for free, at http://weblog.raganwald.com[1] and other places. I decided to publish these essays as an e-book as well as online. This format doesn't replace the original online essays, it's a way to present these essays in a more coherent whole that's easier to read consecutively. I hope you like it.

–Reginald "Raganwald" Braithwaite[2], Toronto, Valentine's Day 2012

copyright notice

cover photo

The cover photograph[3] is Copyright 2008 Alex Indigo[4]. It was taken the weekend of May 24-25th, 2008 in Lion's Head, Ontario.

[1] http://weblog.raganwald.com
[2] http://reginald.braythwayt.com
[3] http://www.flickr.com/photos/alexindigo/2572468190/
[4] http://www.alexindigo.com

About The Author

When he's not shipping Ruby, Javascript and Java applications scaling out to millions of users, Reg "Raganwald" Braithwaite has authored libraries[5] for Javascript and Ruby programming such as Katy, JQuery Combinators, YouAreDaChef, andand, and others.

He writes about programming on his "Homoiconic[6]" un-blog as well as general-purpose ruminations on his posterous space[7]. He is also known for authoring the popular raganwald programming blog[8] from 2005-2008.

contact

Twitter: @raganwald
Email: raganwald@gmail.com

Reginald "Raganwald" Braithwaite

[5] http://github.com/raganwald
[6] http://github.com/raganwald/homoiconic
[7] http://raganwald.posterous.com
[8] http://weblog.raganwald.com

Making Career Choices

Choosing where to work, what to work on, and with whom to work isn't just the most important question, in some ways it's the *only* question. Here are a few of my essays touching on that topic from over the years.

Why You Need a Degree to Work For BigCo

Hello applicant:

I'm very sorry, but I don't have good news about the job opening here at BigCo. We really liked meeting you, but the bottom line is that you don't have a degree from the right kind of University.

——What's that you say? You have learned an awful lot about writing software during the four years that you weren't carting textbooks around the campus? My dear applicant, thanks for pointing out the obvious[9].

I can read, you know. I was impressed by the list of projects you have completed. I even took an extra thirty seconds to Google your name and enjoyed the screen shots and links on your home page.

But nevertheless, University teaches you things above and beyond how to write code. And that's what we need here at BigCo.

——I beg your pardon? You think that you have learned just as much about communication, teamwork, and project management from shipping software in small teams as you would have learned completing coursework?

Please don't take this personally, but I need a moment to chuckle. Ok, I'm done. University isn't about communication, teamwork, or project management. If you happen to learn those things, that's a bonus. I was thinking of something else.

——Look, I admire your enthusiasm, but if you'd let me do a little of the talking I could tell you what we need. But since you bring it up, no I wasn't thinking of any of that Computer Science stuff.

Just in case you're thinking of referring any of your friends to BigCo, please let them know that if they learned why S, K, I[10], and Y[11] are the most important letters in the alphabet, they need not apply. Ever.

To paraphrase Eric Beck, "At either end of the educational spectrum there lies a hacker class." And we are not interested in hackers, even great hackers[12]. We need those middle of the spectrum folks who are going to live in the suburbs, commute to our offices, and do a decent job for a fair wage week after week, year after year.

Quite honestly, the very fact that you passed on University tells us something disturbing about you. Quite obviously you aren't stupid. And you knew that people like us would have a problem with your lack of education. But you believed in your heart of hearts that you could make up for this with excellence.

But you know what? That same attitude might have you think "It'll look bad if I quit this job in less than five years, but I'll make up for it." That kind of attitude makes you a little fearless. And while

[9] http://www.everything2.com/index.pl?node=obvious

[10] http://en.wikipedia.org/wiki/SKI_combinator_calculus

[11] http://weblog.raganwald.com/2007/02/but-y-would-i-want-to-do-thing-like.html

[12] http://www.paulgraham.com/gh.html

we try our best to build a decent working environment, we like our people to be just a little afraid of leaving the nice security blanket we give them.

This may come as a surprise to you, but we're looking for people who are looking for us. Of course we know that the educational component of University is a waste. We wouldn't have it any other way.

Like hazing rituals and wearing dark suits to work in August, attending a certain kind of University is a statement that you want to belong, that you know there is no practical purpose to the exercise but that you are prepared to make the sacrifice just to fit in. And you, dear applicant, would not fit in.

Let me stress this point about what kind of University. We aren't talking about some aerie faerie place where you build robots[13] or spend your free time writing business plans[14]. Those places exist to skim the cream off the top so we can hire a plain glass of 1% milk.

As a matter of fact, the kind of University we like discourages you from dreaming about the future and keeps your feet firmly planted in the ground. For example, our favourite institutes of higher learning send you to work for companies like ours on work terms. This provides us with cheap labour and has the pleasant side-effect of discouraging the more creative undergraduates from wasting everybody's time by coming to work for us.

——Look, I really have to go, and I don't want this call to end on a down note. There are lots of happy people in this world, and most have never even heard of BigCo, much less come to work here. So please consider this a redirection instead of a rejection. I know that's trite, but it's no less true just because it has a memorable rhyming form.

It's not you, it's us. The plain fact is, you wouldn't be happy here. So buck up, look around, and see if you can get yourself into something a little more early stage. Consider starting your own company.

Because quite honestly[15]? I'd read your business plan any day. Your résumé would look better on top of a funding proposal than under a cover letter.

Good luck out there.

[13] http://www.media.mit.edu/cogmac/

[14] http://www.google.com/corporate/history.html

[15] http://paulgraham.com/hiring.html

No Disrespect

> It's like if you walked into a painting class, told everyone that learning to paint was too hard, and then gave everyone a camera and told them that photography was the same thing as painting, only a lot easier to do. No disrespect towards photographers intended.

—Who did Kill the Software Engineer?[16]

I hear an argument that business programming isn't hard. It does not require understanding combinatorial logic and fixed points. There is no value in knowing how to program in OCaml (or why that might be a good idea). Business programming is practical, and business programmers need practical training to do a practical job.

Ok, I'll bite. Really. As Keith would say, I'll climb down from my former position on this. Fine. So here's my question to you, Mr. "An undergraduate Computer Science program ought to be as easy as business programming:"

Why don't businesses advertise for vocational college graduates?

You are describing a vocational job to me. The rote application of practical principles is nothing more and nothing less. How is what you're describing any different than a job as an accounts receivable clerk or a dental technician? Or a land surveyor? Or a architectural draftsperson?

> If your comp-sci course is easy then you're either a genius or you're wasting your money. The odds on that are not in your favour.

—Chris Cummer, Two Thoughts on Computer Science Courses[17]

These are all vocational jobs with a well-established and perfectly functional system for training clerks, technicians, surveyors, and draftspersons. Well? Why won't businesses come clean with everyone and say that's what they want?

Let me tell you the cold, hard, truth. You aren't going to like this, but I ask you to believe me when I say that I am telling you this for your own good:

There is a culture of pretending business programming is more than it is. Some of you calling for more Java in University may take false hope that I am on your side. You may think that the people arguing for Scheme, Haskell, and OCaml are elitists. Wrong. They do not have a problem. You are the one with a problem because you don't want to tell all your friends you have a job as a clerk.

[16] http://www.ekinoderm.com/wordpress/?p=27

[17] http://www.postal-code.com/binarycode/2008/01/21/two-thoughts-on-computer-science-courses/

So you want to be a clerk

You do a clerk's job, you settle for a clerk's working conditions and wages, but you take solace in the thought that you are somehow more than a clerk, because *you have a university degree* and the dental technician who cleans your teeth doesn't.

Only everyone knows it's a sham, especially the hiring manager who puts "University degree required" in the job advertisement. He wants to hire a clerk, someone who will work long hours doing as they're told in a top-down, hierarchal command structure. Does that job sound like there is any Science involved? Of course not, everyone knows that, it's why the industry is trying to weed all of the Science out of a Computer Science degree.

And you're falling for it, hook, line, and sinker.

The university just wants your money, they're selling you a gown and mortar for the low, low price of $5,836 a year (books, room, board, and other expenses not included). Business just wants to let you have your little fantasy of being a cut above the X-Ray Technician who does the same, repetitive thing all day working with a big flat screen and data that goes on a hard drive ($45,950 median salary).

> The sooner you figure out the game, the sooner you can start playing instead of being played.

So we dumb down the curriculum. Who loses? Please don't tell me the University loses its reputation. They don't care, their budget is based on how many people like you they can con into getting a piece of paper that isn't really any better than a vocational college certificate. If they need any more prestige, they'll take your fees and give them to someone who will write a paper.

You didn't write any papers, did you? The one thing that is really, really useful for getting ahead in business is learning to write, learning to speak, learning to persuade. If a Computer Science degree was really meant to give you an advantage in a business environment, it would involve a lot more writing English than writing Java. You would know that if you went on to get an MBA.

Anyways, the university is doing just fine, thank you for helping to pay for a fancy new building. How about businesses? Are they losing because they're paying university degree wages for clerks? No, because the secret of business is that the market works out the right price unless you can monopolize supply.

And you, my friend, are not monopolizing supply. Did your university have a strict cap on the number of undergraduates entering the CompSci program? No? Then your degree has absolutely zero leverage. Wait, I could be wrong. Do businesses refuse to hire workers with foreign credentials? They do hire workers with foreign credentials? Well, there goes that component of value.

Let me tell you, and I speak as a hiring manager. We lie to you. We write in articles and books and on blogs how much we value a degree. Joel Spolsky argues that there ought to be good degrees. But I will bet anyone $100 that he will hire a high school graduate if that graduate is smart and can get things done.

Beyond your actual ability to write programs, a degree is only as meaningful as its scarcity. If degrees are easy to get, they mean squat. Sure, when I'm hiring I might choose to toss all the no-experience resumés without degrees. I'm still left with a pile of two hundred people to interview. Do you think I'm paying any of you top dollar when I have 199 more people to see?

The reality is that your degree is only a pacifier, a way to make you feel good about yourself. *The industry is selling you the illusion of respect.*

I'm telling you this because the sooner you figure out the game, the sooner you can start playing instead of being played. If you really want to be more than a clerk, you can pay more attention to what is to be done and how much freedom you have to do it and less attention to whether there is a title or a degree involved.

You can be respected for your job. But it has nothing to do with whether we pretend a university degree is required for the position. Everybody knows it isn't: if it was, there would be no need to dumb down the program to suit the job. Respect comes from what you accomplish, and where programs are involved, it comes from writing programs, not from a title or a piece of a paper (nor from a book or a blog). Nobody on Earth can stop you from writing software and earning genuine respect.

So do yourself a favour. Don't let them play this game at your expense. Don't allow them to disrespect you so transparently.

You deserve more respect than that.

p.s. Look, there is nothing wrong with being a clerk as long as you not in denial about your job. There is nothing wrong with educational institutions helping to fulfill a demand for clerks. And although I don't think it is in a business's best interests to treat programmers as clerks, there is nothing wrong with a business making that choice.

That being said, there are institutions that offer much more than vocational training in their undergraduate programs. Here's why it's in your best interests—not business's best interests, but your best interests—to get a degree involving actual Computer Science, not vocational training: What good is a CS degree?[18] And furthermore, there are plenty of programming careers that are an alternative to clerking. I am not going to say better or worse, just different. You don't have to follow that road, no matter how many others are going that way.

The point of this post is that you need to have your eyes wide open when choosing what type of education you need and what type of career to pursue.

[18] http://enfranchisedmind.com/blog/2008/01/21/what-good-is-a-cs-degree

Interlude: You and Your Research

Richard Hamming on solving important problems[19]:

> In order to get at you individually, I must talk in the first person. I have to get you to drop modesty and say to yourself, `Yes, I would like to do first-class work.'' Our society frowns on people who set out to do really good work. You're not supposed to; luck is supposed to descend on you and you do great things by chance. Well, that's a kind of dumb thing to say. I say, why shouldn't you set out to do something significant. You don't have to tell other people, but shouldn't you say to yourself,` Yes, I would like to do something significant.''
>
> ...
>
> Over on the other side of the dining hall was a chemistry table. I had worked with one of the fellows, Dave McCall; furthermore he was courting our secretary at the time. I went over and said, `Do you mind if I join you?'' They can't say no, so I started eating with them for a while. And I started asking,` What are the important problems of your field?'' And after a week or so, `What important problems are you working on?'' And after some more time I came in one day and said,` If what you are doing is not important, and if you don't think it is going to lead to something important, why are you at Bell Labs working on it?'' I wasn't welcomed after that; I had to find somebody else to eat with! That was in the spring.

If what you are doing is not important, and if you don't think it is going to lead to something important, why are you working on it?

[19] http://www.cs.virginia.edu/%7Erobins/YouAndYourResearch.html

How to get your first job developing software

HumbleCoder[20] asked:

> I've seen a number of people recommend doing programming work for free in order to gain experience. This can involve either volunteer work for a local charity or school, or working as an intern for a for-profit corporation. I'm not sure if this actually works or not, so I am wondering if anyone out there can comment on this strategy for getting experience. Email me if you have used this technique to help you land a first job. I have been hiring developers for almost a decade, and one of the things I look for is a habit of writing free or nearly free software.

In my experience, good developers love to write code. In fact, the very best developers don't know how to "not write code". When they were in high school, they wrote MUDs in Basic (well, that's what they did in the 80s). When they were in college, they developed Perl and Tcl scripts that scanned Usenet for updates to threads of interest. Even if they hold a "lowly" summer job working in a computer store, they have a Linux system at home running Apache and mod_perl or PHP.

Even after they get paying jobs working 70 hours a week, good developers can't stop coding. For example, Phil Greenspun started Ars Digita, yet he wrote lots of free tools on his home server (I use two of them on my own weblog). Paul Graham ran Franz and Viaweb (now Yahoo! Stores), yet he is busy developing a new language, Arc. In my own case, I was busy as a consultant in the early nineties but couldn't help writing a PDA application and a dynamic web service "on the side".

When I am asked how an inexperienced person can get their foot in the door, I give two pieces of advice:

First, I tell them to **just start writing code**. If they don't know what to write, I don't suggest anything specific, I just tell them that they should pick stuff they enjoy, using tools they like. It isn't important to use the latest "industry standard" tools or work on something "commercial".

Second, I tell them to think in terms of a "portfolio" instead of a "résumé". Even if they don't take a physical portfolio to job interviews, their résumé or cover letter should include an appendix with samples of their work. If an inexperienced person wants to stand out from the crowd, screen shots, links, and code snippets will appeal to a technical hiring manager.

Is any of this guaranteed? No. Some companies hire HR people that are not qualified to tell the difference between a talented candidate who compiles and builds their own development environment at home and a mediocre college graduate that didn't expand their own horizons.

I personally wouldn't want to work for a company that gives significant authority over its intellectual property to such an HR person, but not everyone is as picky as I am :-)

This was written before the rise of Github. I now think that this advice is twice as good as when I first wrote this piece.

[20] http://www.humblecoder.com/

I heard you twice the first time

Dear programmer:

Most people have at least one manager. They may not have eight managers[21], but they have at least one. The way the software business is usually structured, most programmers have two managers.

The first manager is often called a team leader. Programmers report directly to the team leader and work with her on a daily basis. Team leaders direct the work of programmers, mentor them, and evaluate their contribution.

Team leaders do not have true 'hire/fire' authority. They also usually don't have final say over things like deadlines and requirements (although this may not be the most effective way to do things, that's how it usually works). In other industries, team leaders often have titles like 'supervisor,' reflecting the fact that they direct the work but have little tangible authority.

Programmers also usually have a true manager, often the team leader's direct manager. This manager has some real authority, usually including the authority to hire, fire, set pay or bonus, and negotiate what can be done by when.

Most programmers have no trouble with this setup. They work well with their team leader, have a reasonable (but not awestruck) respect for their team leader's manager, and in the fullness of time their contributions are recognized in the form of a promotion to team leader or something more technical like 'architect' (whatever that means).

However… From time to time a programmer chafes. He smarts at the setup. He agitates.

In my experience, there are several reasons for this. Many programmers don't respect their team leader's technical competence. Some programmers combine an adversarial personality with a short term horizon, and they have trouble reporting to someone who can't impose immediate consequences or rewards. Some feel unfairly treated because they are not the anointed team leader and feel the need to express this feeling at every opportunity.

Obviously, the programmer, the team leader, and the team as a whole suffers when one of the members of the team begins to express their frustration with the situation. I'd like to share the manager's perspective. What do you think is running through my mind when the feces begin to flow uphill and land in my in box?

The first thing is, obviously the programmer is sending me a message that he feels the setup is wrong. He ought to be the team leader, or we shouldn't have a team leader, or maybe we should have a team and a team leader but he's an exception and ought to report to me, or we should have a separate technical structure and he should report to the Exalted Grand Visionary Poobah of Architecture Astronautics[22], or some other scheme.

Well, my door is open and I want to listen, and it's ok to tell me that. But do me a favour, okay? Tell me directly and tell me once.

[21] http://www.imdb.com/title/tt0151804/

[22] http://www.joelonsoftware.com/articles/fog0000000018.html

Quite often people walk around with this dysfunctional idea that there's a black and white, right and wrong for everything. When things aren't exactly so, they say things "ought to" be another way. By strange coincidence, the way things "ought to be" happens to be something that favours them. Go figure.

Anyways, I have found that this "ought to" mentality invariably walks in tandem with a belief that everyone should know how things ought to be. This means, in practice, that the "ought to" believer shouldn't have to explain how things ought to be, the rest of us should just get it.

Please, please, please don't let this pathology take over your career. If you have an issue, the very best thing to do is pretend that I'm a smart guy, that I get it, that I understand and support you, and that perhaps I've been a little busy with a ton of other things that need my attention. So just walk up to me and tell me flat out what you want and why you want it.

Don't drop hints. Don't undermine the team leader. I may not act like I get why you shouldn't have a team leader, but I totally get that something's wrong when you constantly come to me for decisions and information that you should be getting from the team leader. Sure, I'm pissed off at the team leader for not keeping you happy.

And if that's the reaction you wanted, congratulations. But don't think that somehow you're smelling like a rose. After all, everyone else seems to be getting along ok. Am I supposed to believe that you are the sole voice of reason in the building?

I also get that something's wrong when you undermine the team leader by somehow forgetting to do or being to busy to do a zillion things she asks you to do, like the administrivia of project management. But you know I don't get? I don't get the reports and statistics and numbers I need to do my job, which is convincing the people that write the cheques to keep the money flowing.

You know, I'm somewhat aware of the team leader's strengths and weaknesses. And thanks for double underlining the weaknesses in red pencil. And maybe she won't have my job one day, or my boss' job. Maybe I ought to find someone else to lead your team.

But you know what? I may be busy. I may have other plans for her and for you. I may have other priorities. That doesn't mean I will or won't act on your initiative. But it does mean that dropping hint after hint after hint very quickly goes from irritating to full blown obstruction to success.

Right now, I need software. Are your actions helping us ship higher quality software, with less risk, in less time? So…

Do yourself a favour. Come to see me. Talk to me. Make sure I understand by speaking plainly. And then listen to what I have to say. Be sure you understand. And then stop dropping hints. One way or the other, let's ship software.

Yours very truly,

Reginald Braithwaite

post scriptum

I've run into this several times in my career. From my experience, once it reaches the point of sabotaging the man in the middle, both parties become very tarnished, very quickly.

I'll share one story where I was the hapless man in the middle. I was managing the development activities for one company, where I had several team leaders reporting to me. The company founder also had an R&D team cooking up some revolutionary software.

This team had been whiling away the years on this project, and to all appearances they had an unlimited budget and schedule. The project required a very deep domain knowledge, and the team tended to hire people with high domain knowledge and little or even no experience writing software, much less developing products.

The founder was exceptionally interested in this project and loved to brainstorm with the team. One day I got the news: he wanted me to manage this team, to teach them how to ship software. He saw their brains and my experience as a match made in heaven. The team's leader was told to stop reporting to the founder and start reporting to me.

Well, managing software development isn't that hard, is it? I asked the team leader for a list of what he thought the team could accomplish, by when. I simultaneously asked the company's business development folks for a list of what we needed to have in order to close deals. I proposed that we put the two lists together and create a development schedule.

Apparently this was not to the R&D team leader's liking. He questioned whether someone lacking a doctoral degree in his field could understand the software, much less participate in its creation.

He flat out refused to participate in any management exercise I cared to propose, letting me know that the software would be ready when his team shipped it, and that it would contain whatever features his team deemed necessary to include, and that the team was not going to commit to those features in advance, as research was still underway.

And all the while, he continued to deal directly with the company founder, who remained involved as a 'product manager' while we pretended that I was the actual manager.

Needless to say, that didn't last very long. The company founder realized that the team leader was forcing his hand and reluctantly took action.

The single most important thing you must do to improve your programming career

Yes, I said improve your programming career, not your programming prowess. And by career, I mean all of it in any form, whether your career is working for BigCo, launching a start up, consulting, whatever.

Let's assume you're already as smart as Steve Wozniak, the über-engineer. Do you want to hack part-time for the rest of your life or do you want to change the world? To change the world, you must create the new, but you must also show people why it matters. And that's why…

The single most important thing you must do to improve your programming career is improve your ability to communicate.

To program, you must elicit ideas from other people and share your ideas with them. Sharing ideas comes in many forms: explaining how you did something. Suggesting a new practice for the team. Demonstrating how something works. Convincing everyone to switch programming languages. Persuading a brilliant engineer to join your team. Persuading your manager to get out of the way and let you do your thing.

Advancing your career is entirely about communicating. Getting a job. Turning down a job. Asking for a promotion. Turning down a promotion. Getting onto a good team. Politely extricating yourself from a good team. Persuading a brilliant engineer to co-found a company. Helping a brilliant engineer understand why co-founding a company isn't the right thing to do. Asking for funding. Turning down funding. Getting clients. Turning down clients.

It's all communication. Now you know, I have a background in Sales and Marketing. So I worry that I am holding a Golden Hammer and suggesting to you that your career is all about nailing things. But here is the short and obvious truth: Human beings are all about physical communication. It's a huge, huge part of what makes us human, this talking and listening jibber-jabbering thing.

Frankly, there is no substitute for actual speech with visible body language. We do not react to the written word the same way that we react to a speech where we can see the speaker. So let's skip right past learning to write well. It's phenomenally important, but learning to speak well trumps everything. Where I work, we talk about technical things all the time, and the remote members of the team switch out of Campfire into iChat or Skype to hear each other's voices and see each other's faces at least once a day.

If you take just one thing from this post, let it be this: To improve your programming career, the single most important thing you must do is improve your ability to communicate your ideas face to face.

What to do right now

Would you like to improve? I have deliberately avoided suggesting what specifically you might want to improve: My Latin teacher said we crave the three Ps: Power, Prestige, and Pecuniam. But the

nuänces of aspiration are many. You have ideas you want us to understand. You have people you want to associate with. You have things you want to build and you need to harness other people's efforts to build them.

Here is what to do today to shorten the line between where you are and where you need to be: *Volunteer to give a presentation that you can entirely control, and invest the time and effort to do the best possible job.*

And that's why it matters: Thinking deeply about your ideas from a new perspective engages your brain and makes you smarter.

Now, you may already be giving presentations. Maybe it's part of your job already. But I am still recommending you add one to your workload, one that you entirely control. This is crucial: if you don't control the subject, the content, the audience, everything, it may not be the very best way for you to improve.

I recommend you find a venue that is begging for presentations—local programming users groups are ideal—and sign up to give a talk. Pick a topic. If you don't have a burning desire to shout something across the roof tops, you can always fall back on giving an "Experience Report." That's where you discuss something you did or tried to do and how it worked out.

Or go with one of the ideas I suggested for blog posts[23]: "What I learned from Language X that makes me a better programmer when I use Language Y," "Something surprising that you probably wouldn't guess about Language X from reading blog posts," or "My personal transformation about Idea X."

By the way: If you already have a certain degree of comfort with presenting to an audience and you really want to ignite your career, try doing a "Fish out of Water" presentation: Present to a group with whom you have very little in common. For example, give a presentation about how to work with programmers to a business networking breakfast club. Learning to present ideas to people who are not just like you will make you a superstar.

Here is what to do today to shorten the line between where you are and where you need to be: Volunteer to give a presentation that you can entirely control, and invest the time and effort to do the best possible job.

Now put together a presentation deck for your talk. Do not give a talk without a deck, you will lose valuable experience giving presentations and this is about your experience. Furthermore, presentations are tougher than speeches because the presentation medium gives you many, many opportunities to screw it up (like having what you say and what's on the screen be the same thing).

Then give the talk. And then give another one. And keep going. And never stop, because presenting well is a skill that helps you from where you are now right up to when you are presenting the numbers to the Board of Directors or presenting a new product to a hall full of cheering customers or, or, or anything and everything.

See you there!

[23] http://weblog.raganwald.com/2007/10/three-blog-posts-id-love-to-read-and.html

Interviews (and Job Hunting)

I'd like to think that if you do the right things and get to know the right people and apply for a job at the right company, the rest will take care of itself. Unfortunately, it doesn't work that way. Our industry doesn't interview very well, and we programmers often don't show our best sides even if the interviewer is doing a good job.

I try not to take it personally: Interviewing is a skill, and every moment a senior programmer spends learning to be a good interviewer is a moment they don't spend writing code. And if we hand the whole process off to the "professionals" in HR, the cure is worse than the disease.

Over the years I've written a few essays about interviewing software developers. Most of them are written for the interviewer, but honestly they're just as important for the interviewee. There's no trick to my suggestions, nothing to hide. "Preparing" for one of these interviews consists of loving what you do and putting in the time to become good at it.

But a little preparation for the interview will never hurt. Start by reading what your interviewer has already read.

The Mother of All Interview Questions

> Does your process involve invention? If so, please describe the three most recent things you have invented and why it was necessary to invent something new.

That's it. As an interviewer, you can ask this question of a candidate. As a candidate, you can ask this question of the team you are considering joining.

The results are really easy to interpret. Does your team invent new things in the course of its work? If so, what are you to make of a candidate who doesn't invent new things and believes that software should be built by integrating tried and true existing components? Such a candidate is not wrong, but are they a good fit?

Even if you invent and the candidate invents, discussing their inventions drives directly to the core of what kind of developer they are. Are their inventions part of the core value proposition of the software they develop? Are their inventions focused on developer productivity but invisible to users and stakeholders? Do they prefer novelty to reliability?

Likewise as a candidate, what are you to make of a team that invents when they could integrate? Do you want to be the newest member of "The Knights Who Say NIH[24]?" Do you want to wrestle with somebody's home-brewed framework that implements an ad hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp[25]?

Software Development is a tremendously large subject. No one question, not even this one, is going to get to the heart of everything you need to know about a person or team's character. But deep at the heart of what we do is the difficult question of when and why one creates something new instead of using what already exists.

Types of brains

Some teams go out of their way to avoid invention. If faced with a requirement that seems to demand nerd sniping[26], they will renegotiate the requirements, educating stakeholders on the non-trivial risks that wait for the project like icebergs awaited the Titanic.

Other teams embrace invention for competitive advantage. They will use an existing plug-in or gem for user authentication, which they consider a solved problem, but will happily create their own code—and possibly open sourcing it as a gem or other package—for something central to the value proposition of the application they are developing.

Likewise, some developers strongly prefer the conservative approach, avoiding invention. Such developers are not laggards, we are not talking of COBOL programmers. Instead, we are speaking of developers who have discovered that tremendous value can be created by managing development

[24] https://secure.wikimedia.org/wikipedia/en/wiki/Not_Invented_Here

[25] https://secure.wikimedia.org/wikipedia/en/wiki/Greenspun%27s_Tenth_Rule

[26] http://xkcd.com/356/

successfully, and especially by avoiding catastrophes. Such developers are often motivated by results: They like to deliver working software, and see invention as a fickle mistress that can delay and derail delivery far more often than she assists it.

Others are eager to find a place to create value though differentiation. Such developers have realized that if you never invent, your code cannot be better than anybody else's. This might be fine for some internal HR workflow management application, but when building anything that is part of their employer's core value proposition, you cannot add great, disruptive value without being prepared to take the road less traveled somewhere in the code base.

I will not suggest that one or the other approach is the best way forward. I think that for each person, there is a natural affinity to invention or a natural dislike of the risks and time sinks involved. Some people feel deeply rewarded giving birth to something new after a long, arduous period of research, trial and error. These developers can often be insanely smart.

Other developers feel deeply rewarded when they can travel in a direct line from conception to execution to delivery, with as little uncertainty along the way as possible. These "results-oriented" developers want to get things done. In theory, we all want to work with people who are smart and get things done[27]. But in practice, what we want are people who are smart in a way that complements our own approach to solving problems and who get things done in a way that helps us rather than hinders our own efforts.

What to do now

Despite my admiration for both types of developer and both types of team, I am keenly aware that culture clashes can and will cause stress and unhappiness on any project, which is why I value this question so highly.

Even if you aren't looking for a new place to ply your trade, I suggest that if you haven't thought about your personal bias with respect to invention before, you take some time to work out how you really feel about inventing new things. It's important to come to grips with your true feelings, not what you think other people want to hear or what would get upvotes on Hacker News.

Likewise, if you aren't interviewing candidates it's still important to truly understand your team's dynamic so you can understand how you fit. if your team has a bias toward invention while you are conservative, that doesn't mean you should quit, but perhaps you should think of yourself as the team's conscience and look for ways to steer it away from invention for the sake of invention and towards invention for the sake of competitive advantage.

This question has no right answer. But it may suggest the right way forward for you.

[27] http://www.amazon.com/gp/product/1590598385/ref=as_li_ss_tl?ie=UTF8&tag=raganwald001-20&linkCode=as2&camp=217145&creative=399369&creativeASIN=1590598385

My favourite interview question

Some years back, I was interviewed by an interesting start-up in Palo Alto. I was really tempted to take their offer, mostly because:

1. One of the interviewers was an avid Ultimate player;
2. Stanford. In walking distance. (And the regular Stanford Friz-ball variant game);
3. They asked me a question which has become my favourite interview question to ask and to answer.

The question they asked was: **How might you design a program that moderates or referees people playing Monopoly[28] on the Internet?**

Of course there's a lot to be learned on both sides when an interviewee is asked to design something and explain their design. Both parties learn a lot about each other's communication styles and approach.

It works both ways: for example, if I were sketching out a design and the interviewers repeatedly interrupted me to discuss my UML notation, I could infer certain things about their culture.

Those kinds of issues apply to any reasonably sized design problem. Anything larger than, say, "write a procedure that reverses a string in place." But let's look at three of the characteristics of the game of Monopoly that I find attractive for this exercise:

- Monopoly is poorly defined;
- Monopoly requires more than simplistic object design, and;
- Monopoly is too big to be designed in one session.

First, Monopoly is poorly defined. Chess, for example, is rigorous. The official rules of Monopoly[29] are silent on some critical questions and vague on others. A tournament-playing aficionado will realize this immediately, but it's easy to guide a candidate to this realization by asking some of the well-known FAQs[30].

This 'problem' makes it a great interview question. It drives a lot of valuable interaction between the candidate and the interviewers. You have to ask questions, make assumptions, and know when to stop gathering requirements and start driving the design.

There have been some blog posts pointing out that even trivial requirements can have many hidden implications[31]. A determined and finicky developer can drive questions for the length of the

[28] http://en.wikipedia.org/wiki/Monopoly_%28game%29
[29] http://richard_wilding.tripod.com/monorules.htm
[30] http://www.google.ca/search?q=monopoly+faq
[31] http://exold.com/article/stupid-interview-questions

interview. I think Monopoly's missing requirements actually improve its suitability as an interview question.

If the candidate uses up all of the interview time trying to obtain perfect requirements, we have a problem. In the software development I do, the requirements are never perfect. I don't demand that a candidate try to create an agile, iterative process on the spot, but I look for someone who knows when to say "close enough, let's move forward."

Another good way to move forward for both interviewer and candidate is to say, "ok, we've covered the most important requirements. Let's make a bunch of assumptions and document them. In a real-world situation we could obtain feedback on the assumptions after presenting an initial approach."

After all, who's to say that a programmer, designer, or architect is always 100% beholden to others for requirements?

The second reason I like this problem is that Monopoly requires more than just a simplistic object design. What I'm about to say will be blindingly obvious to the Enterprise crowd (sorry, not you over there with the Greasemonkey[32] script that translates your web email to Klingon). The rules must be considered as carefully as the entities. Enterprise developers have known this for years: that's why you see rules engines, table-driven designs, and visual workflow editors in many Enterprise applications.

Now let's talk about 'object-oriented programming' for a second. 99% of the stuff you read discusses modelling real-world physical objects. Things. Nouns. It is a Kingdom of Nouns[33]. Most candidates start every design by dutifully listing all of the nouns they can think of and then they spend the rest of the time available thinking about piling them into phyla, hierarchies of "IS-A" and "A-KIND-OF."

This approach, which I will call "noun and verb," is so limited I'll dare to call it brain damaged[34]. In fun. But seriously, competent software developers spend much more time on relationships, responsibilities, and constraints than they do on trying to prematurely optimize[35] reuse through inheritance.

Now let's ask a question about Monopoly (and Enterprise software). *Where do the rules live*? In a noun-oriented design, the rules are smooshed and smeared across the design, because every single object is responsible for knowing everything about everything that it can 'do'. All the verbs are glued to the nouns as methods.

Let's take a look at a simple rules question. **If a player owns Baltic Avenue, can she add a house to it?**

Well, there's a bunch of stuff about whether she can afford it and whether there is a house available in the bank. Where does that live? In the bank object? And there is a bunch of stuff about whether it is either the player's turn or between turns. Where does that live?

[32] http://greasemonkey.mozdev.org/

[33] http://steve-yegge.blogspot.com/2006/03/execution-in-kingdom-of-nouns.html

[34] http://www.folklore.org/StoryView.py?project=Macintosh&story=A_Mac_For_Mick.txt

[35] http://www.petefreitag.com/item/509.cfm

And there is a bunch of stuff about whether the property already has four houses. Where does that live? Somewhere in the property hierarchy? Sounds reasonable. Now what about mortgaged property? If Baltic is mortgaged, the answer is no. That's easy. But what if Mediterranean Avenue is mortgaged? And what if, for example, Baltic has one house but Mediterranean has none? Where does that logic live? Both of these last two questions involve knowing something about the other properties of a colour group.

Now you can debate which verbs belong to which nouns, but here is an opportunity to step back a bit and consider the larger implications of maintaining such a 'classical' OO design.

Consider a 'noun and verb' design. First, an easy question. How well does the design document the actual game of Monopoly? If someone were to read the source code, do you think they could learn how to play the actual game?

Is this a stupid question[36]? I think it strikes at the root of sustainable software development. In real world applications, software lives for a long time and teams change over that time. Programmers are often asked to maintain software with insufficient training in its intended use.

OO programs can be brilliant communicators in some designs ("here's everything I need to know about a Money Transfer") and terrible in others. I think this is part of the appeal and effectiveness) of the Domain Specific Language[37] approach. It does a better job of communicating its intentions than a simple OO design.

I'm not saying that a DSL is right for Monopoly. Or wrong. Just that it's valuable to consider this issue when discussing an approach. There is a lot of opportunity for a candidate and the interviewers to learn about each other's thinking that goes way beyond coding if you spend a few moments discussing the importance (or irrelevance) of a design that communicates its intentions. And I think that makes Monopoly a good design subject.

Another issue about the rules. In software development, requirements often change. What kinds of requirements changes are easy with the design? What kinds are difficult?

I suspect it's easy to change the prices and names of properties, but very difficult to change the fundamental rules. Well, it's probably easy to introduce the popular 'lottery' variation where fines are paid into a pool and anyone landing on Free Parking' scoops it up. But some of the more esoteric variations would require making lots of changes to many different classes.

The 'noun and verb' design is tightly coupled, and distributes the rules across the design. If you really want a mess, consider that popular computer versions of Monopoly allow you to pick and choose rules variations for each game.

The key to making this easy is to find another way to represent the rules. I generally provide hints along these lines by asking the candidate how they plan to implement the Chance and Community Chest cards. What is the relationship between the cards, special squares like "Luxury Tax," and introducing variant rules?

[36] http://www.quotationspage.com/quote/27543.html

[37] http://jamis.jamisbuck.org/articles/2006/04/20/writing-domain-specific-languages

> (You can probably save yourself a lot of interview time. Instead of all this hoopla, ask the candidate to describe when they have actually used the Strategy[38], Visitor[39], and Command[40] patterns outside of a framework.)

The bottom line is that designing a Monopoly game requires giving a lot of thought to representing the rules. There are a lot of ways to do that. I've hinted at two, and you I've heard of a lot of other interesting approaches. It's a great chance to really go beyond OOP 101.

Now to close as rapidly as possible with what I consider an essential characteristic of a good design problem, and one that applies here. **Monopoly is too large to solve in one interview**. This is related to the vague requirements characteristic I mentioned above.

The reason this is important is that it forces the designer to pick and choose what elements of the design to solve in limited time. Some candidates will fail outright because their problem solving style is to consider all of the implications before starting. It's easy to spend hours on a problem like Monopoly. But will you get up and apply the marker to the white board?

> Does a candidate start with the trivia? Who cares how to represent the colour of a property? CSS? A getter in each object? Yawn. I'll prod him to move along.

Perhaps a candidate spends all of her time working in an area of the problem she knows well. Some candidates can spend forever engineering a Monopoly Online site up to Google scale while hand-waving over the business logic of making the game work. Or gathering requirements for players joining games and leader boards. Or basking in the glow of whatever interactive technology is hot ("I'd build it with Baackbone, no wait, Ember, and Node, yeah, Node. With Redis!")

I'm not saying that's bad. Or good. But I will say that given a problem too large to solve in the time allotted, there's fantastic value in evaluating what's important. And that cuts both ways. If you think that some of the OO design choices are critical but the interviewers hand wave it and want to see how you plan to handle 10,000 requests per minute, that's revealing.

Summary

I like asking (and being asked) about Monopoly because it provides fertile ground for discussing issues that are critical to judging both competence and cultural fit, like:

- Communication style and skills;
- Deep OO design philosophy;
- Prioritization and time management, and;
- Handling ambiguity.

[38] http://c2.com/cgi/wiki?StrategyPattern

[39] http://c2.com/cgi/wiki?VisitorPattern

[40] http://c2.com/cgi/wiki?CommandPattern

Update: A cautionary note

Some people have objected to this question because it may be a trivia challenge: how well does the candidate know the game of Monopoly? In the comments I mentioned that the question really only applies to candidates that already know the game. If someone doesn't know the game, ask another question.

Others have suggested it may be a form of "Guess the design I'm thinking of" where there is exactly one right answer, the design the interviewer has already come up with. I think this is a legitimate concern.

It really applies to any interview question. For example: "name your three strengths" or "tell me about a challenge you faced in your last position and how you overcame the obstacles." If an interviewer is looking for something specific, they can and will twist any interview question into a variation of "guess the answer I'm thinking of."

I won't say that Monopoly is somehow better than other questions in this regard, nor is it worse. If you have an interview based on reviewing a candidate's past accomplishments, don't you think there are some interviewers that will discount those that don't fit their personal criteria for merit?

> I once had an interview where the interviewer spent forty minutes talking to me about my education, and something like fifteen talking about my fifteen years (at that time) of actual career experience. Bias in an interviewer is orthogonal to the choice of interviewing technique.

What do these two cautions have in common? Only that the most important thing for a successful interview is that the interviewer be genuinely seeking out a view of the candidate's strengths and weaknesses. All I can really say about this question is that if (and it's a big if) the interviewer is sincerely attempting to evaluate a candidate's technical ability, and if the candidate has some familiarity with the game, then the session can be very productive.

Interlude: Idiocracy

> Never hire an engineer on the basis of questions any idiot could answer.
>
> If you ignore the above maxim, the result of your negligence may one day wind up becoming your boss.

—Based on an intelligent comment from bairespace[41]

> If you take a job with a company that asks you questions any idiot could answer, sooner or later they're going to put you on a project with some idiot who answered them.

—Giles Bowkett, Programmer Interviews: Two Warning Signs[42]

[41] http://programming.reddit.com/info/xrex/comments/cxs0l

[42] http://gilesbowkett.blogspot.com/2007/05/programmer-interviews-two-warning-signs.html

Take control of your interview

I'm helping some colleagues interview programmers, and for once I'd like to offer some suggestions to the interviewees and not the interviewers. This post is about using the interview to maximize your chance of landing the right job.

Have an objective

First, you need to walk in with an objective. No, not "*To secure a strategically progressive position leveraging your forward-going architectural vision and hands-on experience to advance the division's mission*," but a simple, tactical objective for the interview.

Here's what I suggest. **Think of three things you want the interviewer to know about you that you think they are unlikely to find out if they ask all the questions**.

The important ideas are that (a) you want the interviewer to know about each of the three things, and that (b) the interviewer is unlikely to ask about all three if you don't exercise some control over the interview.

Let's start by ruling a few things out. First, don't bother with how many years of technical experience you have with the company's tools and platforms. If they are using in-house Common Lisp macros compiled to C and then distributed on a grid with MapReduce, I guarantee that they will ask whether you have any Lisp or distributed programming experience all by themselves. (You can still take advantage of your experience, I'll show you how below.)

Second, rule out anything that doesn't really help sell them on you. Yeah, yeah, hockey is competitive and it takes a special kind of focus to be a goalie. I get that, but it really only sells when the interviewer is also a hockey player. My suggestion is put that kind of thing at the bottom of your resumé and let them ask about it if they care.

The three things

The three things you want to take into the interview should be stuff that matters to them but is hard for them to ask about. Stuff that doesn't pop out of your resumé. I don't have a pat formula for generating these items that I can share, but here are a couple of ideas to get you started.

Remember the technology buzz-phrases we rejected ("five years of JEE" "Common Lisp")? Think about the difference between yourself, presumably an expert in these areas, and someone who has only worked on one project with the same technology. You both touched all the same tools and code, but there's something special about you, your extra experience means something. What is it?

Joel Spolsky[43] and Peter Norvig[44] both suggest that you cannot pick up a new programming language or platform and become proficient overnight. That being said, if you tell me that you have five years

[43] http://www.joelonsoftware.com/articles/lordpalmerston.html

[44] http://www.norvig.com/21-days.html

of experience, I have no idea whether you have five years of experience or whether you simply did the same thing over and over again. What is it that you learned that makes you special? What secrets to you possess that can't be listed on your resumé?

For each person, there will be different answers. One person might say that their "secret sauce" is that they have learned the ins and outs of the platform, they know what works and what doesn't, they know how to work around the shortcomings. Another might emphasize the non-technical skills. I would personally be impressed with anyone who said that what makes them special is that they are very, very accurate when they estimate tasks and projects.

This leads naturally to another area I would mine for nuggets: Soft or non-technical skills.

99% of the dreck[45] you read about interviewing programmers is about finding out whether they can actually[46] write programs. Rightly or wrongly, most interviewers focus on the hard skills. But if they never get around to discovering your ability to juggle priorities, or write effective technical documentation, or analyze requirements, how will they know that you are far and away superior to the other five people they will interview this week?

You have to tell them, that's how they'll find out.

Tell them about it

Walk into the interview with your three things. Now, the interview is a game. Presuming you don't get thrown out or you don't walk out before it ends, you win the game if the interviewer discovered all three of your things. If you like games as a metaphor, call them **goals**.

- If the interviewer asks you about one of your things, that's an easy goal. Take it.
- If the interviewer asks you if there's anything you'd like to mention about yourself, that's an easy goal. Talk about one of your things.
- If the interviewer asks you about something related to one of your things, answer the interviewer's question and then 'coat tail' your thing onto the end. For example:

Interviewer: It says here you worked on JProbe, that's a Java tool, right?

Interviewee: Absolutely. JProbe is a suite of tools for JEE Server-Side development (answers the interviewer's question). Under my management, we released three consecutive versions on schedule. I'm really hoping that there's an opportunity to apply my focus on hitting plan to this team (adds on).

Watch politicians "staying on message." Now matter what question they are asked, they spit out their own pat statements. You don't want to be that plastic, but you have to take responsibility

[45] http://weblog.raganwald.com/2006/06/my-favourite-interview-question.html

[46] http://weblog.raganwald.com/2006/07/hiring-juggler_02.html

for a a successful interview. And you know what? If the interview ends with your best features undisclosed, the company loses as well.

One more time with the coat tail:

Interviewer: What's your experience with Ruby?

Interviewee: Well, I was the lead developer on the Certitude project. We built that with Rails and we included a fairly heavy dose of Ruby idioms, including a domain-specific language for pattern-matching and lots of dynamic meta-programming (answers the interviewer's question). One of the things I discovered on that project was the importance of a bomb-proof quality control process when you have such a powerful language. I customized our continuous integration server to track changed files, tests, bugs in a unified report interface so we could monitor the most troublesome modules. It really saved our bacon late in the project when we had to really tighten up our risk management to ship on time (the add on, emphasizing the process).

If all else fails

Perhaps you didn't get a good opportunity to mention your three things and the interview is winding down to a close. Don't try a desperation coat tail where you try to stick two completely unrelated things together. Instead, try using a question to introduce one of your things indirectly.

Some examples:

- "Could you describe the product management process for me?" leads to an oportunity to talk about your product management experience or your experience working with product managers.
- "What tools do you use for source code management, builds, and bug tracking?" is a great opportunity to talk about your experience with the full software development life cycle, not just the coding.
- "How do you plan and track ship dates?" turns the conversation towards how well you estimate dates, manage priorities, and your track record of shipping code on time.

Opportunities to ask questions should not be wasted on trivia like the company's dental plan or whether they prefer cubicles to private offices (actually, neither are trivial, but they are not important until you have given the interviewer every reason to hire you).

My suggestion is to only ask questions about areas where you have something positive to contribute if the question leads to a spirited discussion. And the best way to make that happen is to plan in advance.

You have three things you want to say, and you should walk into the interview with three questions you can ask that will turn the conversation towards your agenda.

Back to square one

So here's how to get started:

1. List your three things you want the interviewer to know about you that you think they are unlikely to find out if they ask all the questions;
2. Write a one-paragraph description of for each thing that you could use to 'coat tail' onto another question;
3. Think of a question you could ask for each thing that would naturally lead to a discussion.

Good luck.

Are you thinking of working for a start up?

Let's say you want to work for a start up. You aren't one of the founders: you're coming in after the company has raised some money: it looks like it's serious about success.

You know that the start up route is a work hard, work harder environment. You're going to push yourself to the end of your rope and then push yourself harder. You know the deal: work 60, 70, 80, or even 100+ hours a week in exchange for working with great people on something you believe in.

If the personal rewards matter more than any monetary consideration, you can stop reading right here. I wish you well, and nothing would please me more than to hear that the years you spent in a start up were the most thrilling years of your life to date. Go for it!

But *what if you want a little more than the vibe*? Everyone's heard about the riches available to employees with stock options when their company goes public or gets bought for big money. Should you consider the working for a venture backed company to make money? Is it worth it?

I'm going to give you my perspective, based on widely disseminated anecdotes about working for start up companies in the technology sector. After you've read this post, you should be able to come up with a basic figure of how much money you need to make on your options to justify the long hours, hard work, and lower pay of a start up company.

> Let's say you're a senior kind of person and you're offered options worth 1% of the company. Not so fast! As the company grows, it takes in more investment. With every round of investment, the existing stock holders get diluted. You need to figure out how much of the company you'll have when the company is worth M$100.

First, a disclaimer: *I absolutely positively cannot value your stock options.* Even if your employer is wildly successful, your ability to cash in depends entirely on the whims of your Board of Directors. If they want to screw you out of your option money, they can and will do so. If you're an employee, the only reasons to let you cash in are to keep you from quitting and to hold you up as an example so they can motivate other people to work for them.

(Some of the time, those two reasons are good enough. This is especially true if the company has been backed by a reputable VC firm. The VC business depends on "deal flow": a steady stream of entrepreneurs walking into the offices and offering the VCs new businesses to own. It hurts their deal flow when word gets around that they don't share the wealth.)

So how much will you get? And is it worth the work? As I said, I don't know how much you'll get. I can give you a few tools for making a Wild Assed Guess.

Raganwald's Scientific Wild-Assed Guesstimation Methodology

First, you have to convert options into your interest in the company. Don't waste your time figuring out "if the company goes public at $100 a share, and each option has a strike price of $1, I get $99

per option, times 10,000 options is…" That doesn't work. The thing to do is to figure out *how much of the company you have a right to buy.*

Let's say you're a senior kind of person and you're offered options worth 1% of the company. That sounds like a lot. If your company is purchased for M$100, your options are worth a million dollars give or take. (Just so you know, lots of software companies are sold for between one and four times revenues. So unless your company is really, really hot, your company isn't going to be worth M$100 unless you get revenues up to M$25 at the very, very least.)

Okay, 1% is worth a million dollars. Not so fast! As the company grows, it takes in more investment. With every round of investment, the existing stock holders get diluted. That means the 1% you have now (I'm being generous: you typically have the right to buy 1% provided you don't quit or get fired over the next four years) will actually be worth less than 1% of the company when you cash in.

So you ask the President about the dilution, and she laughs and says "yes, but that means we'll be on track and on our way to riches, so it's worth it." Okay, but you can't have it both ways: you can't say 1% of M$100 is worth a million dollars: you need to figure out how much of the company you'll have when the company is worth M$100.

Here's an easy way to figure out what your 1% is worth: pretend you're a VC. They just invested in the company, right? So… what was the post-money valuation? If the company shares this with you, they're telling you how much the VCs thought the company was worth when they made their investment.

Let's say you get a rough estimate of the post-money valuation. I'm going to take a really early stage company, a company that just raised M$4 on a post-money valuation of M$12. In other words, the investors just paid M$4 in exchange for 33% of the company.

Guess what? Some people with a reasonable idea of what start up companies are worth just said that 100% of the company is worth exactly M$12. So what do you think 1% of the company is worth? That's right, 1% of M$12 is worth $120,000. No more, no less.

Another SWAG

Is that worth it to you? Maybe you want a second opinion. Or maybe you aren't told the post money valuation. Okay, here's the other way to calculate what your options are worth. For this, you need to figure out your share after all the dilution. Basically, you need to estimate how many times the company will raise more money and how much stock they'll give investors to do it.

You're going to have to do some research on comparables to get a good estimate. I'll take the example I've used above and hypothesize a company that's just raised M$4 on a post money of M$12. This is their second round: the founders started with credit cards and sweat, then management and an "angel" investor put another M$1 to really get things going.

I would expect a company like this to raise at least two more rounds of capital if they're going to do something big. They'll probably give away another 50% of the company to do so. So the 1% you have today is going to be worth .5% when the company scores a big win. What's that worth?

> A VC is having a good year when one in five investments hits a home run. If you honestly can do a better job of picking winners, drop your day job and go into venture capital. You'll make way more money investing in start ups than working for them!

Well, you can always say ".5% of M$100 is half a million dollars." And that sounds great, much better than $120,000. Where did the $380,000 come from? Well, the first calculation measured what investors think of the company. That takes risk into account. The second calculation doesn't take risk into account. What's the chance that the company will be a big success?

Making book

I know, I know. *It's a sure thing.* If you didn't think so, you wouldn't go to work there. And the President is awfully confident. She ought to be, she's put her career and possibly her retirement savings on the line. But don't forget, everyone is that confident, and most VC investments tank. By most, I don't mean 51%. I mean *80% or more go right down the toilet.*

The bottom line is, you ought to ignore your gut and the pedigree of the management team. The VCs take that into account when they invest, and the best they can do is about 20%. Yup, a VC is having a good year when one in five investments hits a home run.

If you honestly can do a better job of picking winners, drop your day job and go into venture capital. You'll make way more money investing in start ups than working for them!

So back to 20%. That's the number I'd pick when figuring out the odds of a spectacular success. Oh, I don't mean that there's an 80% chance of mass layoffs and/or bankruptcy. Many VC backed start ups eke out a living and make a little money. But the option are worthless unless the company does really, really well and can go public or get bought for major money. And that only seems to happen about 20% of the time.

So… What does that do to the $500,000 we get if the company is sold for M$100? You guessed it, that means that the value of those options is really only about $100,000. Not $120,000 by this calculation, but close enough. And it should be: the investors use pretty much the same reasoning when they decide on the post money valuation.

So is that a lot of money or not?

Okay, final calculation: what's $120,000 worth to you? I mean, is $120,000 a good return on your investment?

To figure that out, you need to know how much you're investing by working for this company. You need to calculate your opportunity cost, the money you could have made elsewhere. Let's start with your salary: if you could make more money somewhere else, calculate how much salary you forgo.

For example, let's say you can get another $5,000 a year working somewhere else with little or no risk. Over the four years your options need to vest, that's $20,000 you've just invested to get $120,000 when everything works out. Okay, not bad.

But wait: how hard will you work? If the other job is only 45 hours a week and the start up is 60 hours a week, you're investing 15 hours a week. What's that worth? You can take the other job and work out the hourly rate, but I prefer to calculate what I could make by the hour consulting. If you can work 60 or more hours in a start up, you can work a night job programming for other people.

Let's say it's $30 an hour. $30 by 15 hours is obviously $450 a week. You're probably going to work 200 weeks over four years, so you're investing another $90,000 over four years. Add that to the $20,000 and you've just put $110,000 into the company to get $120,000 out.

For me, that's breaking even. It's not terrible, but it's no short cut to riches. If you can get a substantially better stake in the company, you're on track to do much better. But if you're such a hot shot, you can probably do very well seeking out a job or contract that pays well. So you probably end up in the same place.

Coda

Of course, there are other ways to get rich in a start up. *The best is to be one of the founders rather than an employee.* In this capitalistic world of ours, entrepreneurs are higher on the food chain than employees. Below investors, but higher than employees.

Should you submit your resumé through a recruiter?

I have obtained just one job in my career through a recruiter. It worked out well[47] for me. If my experience can help you, great.[48] I happen to hire through recruiters from time to time, so I can also give you some perspective from the client end.

So: A recruiter calls you up and says he's representing a fanttasic company with a great working environment, excellent benefits, market-leading compensation, and perquisites (that's the long-winded way of saying "perks" out the yin-yang). Should you give him your resumé?

I think it's ok, *provided* that (a) you take great care to ensure that your resumé does not get submitted twice, *and* that you have no other opportunity to submit your resumé directly.

Double submission

> …one staffing company tried to tell me that a second submission would "cancel out" the first and the hiring company would simply reject me as a candidate.

This is 100% true. It's **absolutely bad** to have two recruiters submit your resumé to the same client. Speaking as a client, I round-file any resumé received through two recruiters. This is because I do not want to end up fighting with them over who should get paid. And they will fight.

The trouble is, the recruiter thinks that they only have upside to submitting your resumé, even if the client already has the resumé, or you've worked there in the past, or another recruiter has already submitted it. They will tell you all kinds of stories, such as "if you haven't submitted it in the last ninety days I can resubmit it for you." The recruiter would rather have a slim chance of a fee than no chance, and if your resumé gets tossed, well, better luck with the next company.

Never mind what the recruiter wants to do, follow these guidelines:

Always ask for the client name. Recruiters will often stall, claiming the client demands confidentiality. Simply say that you may already have contacts into the company and you wish to save them time and trouble. If they still refuse, ask them what else they have for you. **Never submit your resumé blindly**. Sadly, some recruiters will screw things up and submit you without permission. It's your loss when this happens. The best you can do is refuse to do any further business with them.

Go direct if you can

If you have had *any* contact with that company about getting a job, especially if the company has already seen your resumé, do not allow the recruiter to submit your resumé. Period. You can only

[47] http://www.quest.com/jprobe/

[48] This is about what works for me, not the great be-all and end-all of how you can live your life. Especially for this subject: we're talking about making money. I once took a year off work because I didn't think there was a good fit with the companies hiring at the time. I've been told that most people consider this a poor career choice.

lose, you can never gain. For example, I have your resumé. Now I see it from a recruiter. I could have hired you without paying a fee. Now your resumé has a big fat "+20% in cash" sign on it. It goes to the bottom of the pile. I don't want to argue with the recruiter if there's someone else who is "clean."

Of course, if you're the right person you get the job and I'll tough it out with the recruiter. But it's a lot of bother forwarding emails and stuff to prove I had you before the recruiter sent you in. I'm not going to bother unless you're exceptional[49]. And if you're exceptional, you were already at the top of the pile and you didn't need the recruiter to help you get an interview. Capiche?

The corollary is that you must never go around the recruiter if you first heard of the opportunity through them. If your friend Steve works there and he never mentioned they were hiring, let the recruiter run with the ball and earn their fee. If your friend Steve mentioned that they are always looking for good people but you never gave Steve your resumé, let the recruiter run with the ball and earn their fee.

If you gave Steve your resumé and for whatever reason you didn't get an interview, but the recruiter tells you they're hiring, politely say that you are already working with the client and ask them what else they have. Then bug Steve to get things rolling.

Summary

Obviously, it's better for you and your future employer f you can submit your resumé directly. And you know what? That's under your control. Are you attending all the relevant meetups in your area? Do you belong to the appropriate programming groups and associations?

It's up to **you** to go out there and make those connections that lead to friends mentioning that their company is hiring and that you should submit your resumé through them. So get out there.

There are some jobs that you cant network your way into, such as jobs requiring relocation, or stealth jobs, or jobs in industries you din't know needed programmers. For those, it's fine to work through a recruiter. Be sure to get the company's name and manage your affairs such that you aren't submitted twice.

[49] http://weblog.raganwald.com/2006/04/why-do-we-resist-idea-that-programming.html

Disclosing your compensation

A few words about disclosure. Google around. You will find that it is always a bad move to discuss your current or past compensation details with an employer. Guess what? That goes 2^n for recruiters. Do not discuss past or present compensation details with a recruiter. **Ever**.

As a client, I bug the recruiter to get that info so that I am armed with the power to underpay you when we negotiate. The recruiter gets to play the hard ass while I pretend to be Uncle Reg fighting to get you every dollar.

Ok, I'm not really that bad. But you know what? I'm human, I have a budget, and within a certain range of 'reasonable' for what you can do, it's my job to pay you closer to the bottom of the range and let you earn a raise. It's your job to get me to pay the top of the range and earn a raise anyways.

There're books[50] written on the subject of negotiating salary. They all say the same damn thing. If you want the Coles Notes, when the recruiter says "no details, no submit," you say "You win, no submit." If the recruiter wants to take themselves out of the picture, that's their business. Don't believe them when they say the client insists. I prefer to have the details, but if Peter Norvig wants to meet me and won't tell me what Google is paying him, I'm not arguing.

Now I said you don't disclose details. I think you want to mention a range. Here's exactly what I say when people ask me:

> Over the past five years, my compensation has ranged from X to Y dollars.

And when they press for more details, I ask them whether this is in the budget. If the answer is yes, let's submit the resume. And they will ask for more details. How much is base? How much bonus? How much last year? Does that include stock options? I blow off all additional details. If they can't submit my resume without additional details, that's their problem.

What the client *needs* is to know that you are neither too cheap (you must suck, or be ignorant, or both), nor too dear (motivated by money, or ignorant, or both). The range answers the question while giving very little away for the negotiation.

Here's how to calculate X and Y. Over the last five years, take the smallest base salary you've earned in any one year or at any one job. Include only the cash portion. This is X.

Now take the most you've earned in any one of the five years. Include the cash equivalent of perquisites like travel to trade shows or conferences, meals, drinks, everything (I drink two free cups of coffee a day. If I'm calculating Y for this year, I'm adding $2.20 a working day to Y).

That's your range. Notice how it works even if you've been at the same job for five years? Nice. And don't discuss this calculation with a recruiter, ever. Their job is to get you an interview. X and Y are all they need to get you the interview.

[50] http://www.amazon.com/dp/1580083102/ref=nosim/104-5191901-1508708?tag=raganwald001-20&linkCode=sb1&camp=212353&creative=380549

Speaking as a client, this is enough. I may want more before I make an offer, but this is enough to know whether I'm wasting your time bringing you in for an interview.

Okay, three tips. That's enough for now. **Good luck with your search**.

Afterword

Q: Why is it so much worse to tell a recruiter your compensation details?

A: Because recruiters will tell *everyone*. If you tell one prospective employer exactly what you made last year, they do not tell every other employer. They probably consider it advantageous to keep that information to themselves. You've damaged your ability to negotiate with them, but if things don't work out, you can start again with a fresh slate at the next interview.

But if you tell a recruiter, they put it on your file and every time they send your resume out they will send that information with it. Really nasty recruiters will use you to sell other candidates (*Ok, you like Kate but you want to see a few more people? Well, I can send you Reg, but he's asking 20% more. If I were you, I'd make an offer to Kate before someone else gets her*). If the recruiter has no idea about your compensation history, or only has a broad range, the recruiter will probably send your resume in.

But if they know your exact details, the shortest line between this situation and their fee is pushing Kate instead of you. Not only are you screwed with clients that want to make you an offer, you're screwed with clients that haven't even interviewed you yet! Speaking as a client, I might have liked you so much more that I would happily pay more. But it's hard to know that if I haven't met you, and it's hard for me to have the optimism that somehow you're going to be worth the extra 20% when I haven't seen you juggle[51].

Don't tell recruiters anything more than X and Y. And even then, **skip X and Y if you can**.

[51] http://weblog.raganwald.com/2006/07/hiring-juggler_02.html