

Joel on Software

Advice for Computer Science College Students

by Joel Spolsky

Sunday, January 02, 2005

Despite the fact that it was only a year or two ago that I was blubbering about how rich Windows GUI clients were the wave of the future, college students nonetheless do occasionally email me asking for career advice, and since it's recruiting season, I thought I'd write up my standard advice which they can read, laugh at, and ignore.

Most college students, fortunately, are brash enough never to bother asking their elders for advice, which, in the field of computer science, is a good thing, because their elders are apt to say goofy, antediluvian things like "the demand for keypunch operators will exceed 100,000,000 by the year 2010" and "lisp careers are really very hot right now."

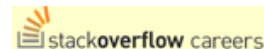
I, too, have no idea what I'm talking about when I give advice to college students. I'm so hopelessly out of date that I can't really figure out AIM and still use (horrors!) this quaint old thing called "email" which was popular in the days when music came on flat round plates called "CDs."

So you'd be better off ignoring what I'm saying here and instead building some kind of [online software thing](#) that lets other students find people to go out on dates with.



[File a CV](#) and let the great jobs come to you!

Wanted: [Google: Unix System/Applications Administrator, Google.com at Google Inc.](#) (Dublin, Ireland / London, United Kingdom / Zurich, Switzerland). See this and other great job listings on [the jobs page](#).



Nevertheless.



If you enjoy programming computers, count your blessings: you are in a very fortunate minority of people who can make a great living doing work they enjoy. Most people aren't so lucky. The very idea that you can "love your job" is a modern concept. Work is supposed to be something unpleasant you do to get money to do the things you actually like doing, when you're 65 and can finally retire, if you can afford it, and if you're not too old and infirm to do those things, and if those things don't require reliable knees, good eyes, and the ability to walk twenty feet without being out of breath, etc.

What was I talking about? Oh yeah. Advice.

Without further ado, then, here are Joel's Seven Pieces of Free Advice for Computer Science College Students (worth what you paid for them):

1. Learn how to write before graduating.
2. Learn C before graduating.
3. Learn microeconomics before graduating.
4. Don't blow off non-CS classes just because they're boring.
5. Take programming-intensive courses.
6. Stop worrying about all the jobs going to India.
7. No matter what you do, get a good summer internship.

Now for the explanations, unless you're gullible enough to do all that stuff just because I tell you to, in which case add: 8. Seek professional help for that self-esteem thing.

Learn how to write before graduating.

Would Linux have succeeded if Linus Torvalds hadn't [evangelized](#) it? As brilliant a hacker as he is, it was Linus's ability to convey his ideas in written English via email and mailing lists that made Linux attract a worldwide brigade of volunteers.

Have you heard of the latest fad, Extreme Programming? Well, without getting into what I think about XP, the reason you've heard of it is because it is being promoted by people who are very gifted writers and speakers.

Even on the small scale, when you look at any programming organization, the programmers with the most power and influence are the ones who can write and speak in English clearly, convincingly, and comfortably. Also it helps to be tall, but you can't do anything about that.

The difference between a tolerable programmer and a great programmer is not how many programming languages they know, and it's not [whether they prefer Python or Java](#). It's whether they can communicate



their ideas. By persuading other people, they get leverage.



By writing clear comments and technical specs, they let other programmers understand their code, which means other programmers can use and work with their code instead of [rewriting it](#). Absent this, their code is worthless. By writing clear technical documentation for end users, they allow people to figure out what their code is supposed to do, which is the only way those users can see the value in their code. There's a lot of wonderful, useful code buried on sourceforge somewhere that nobody uses because it was created by programmers who don't write very well (or don't write at all), and so nobody knows what they've done and their brilliant code languishes.

I won't hire a programmer unless they can write, and write well, in English. If you can write, wherever you get hired, you'll soon find that you're getting asked to write the specifications and that means you're already leveraging your influence and getting noticed by management.

Most colleges designate certain classes as "writing intensive," meaning, you have to write an awful lot to pass them. Look for those classes and take them! Seek out classes in any field that have weekly or daily written assignments.

Start a journal or weblog. The more you write, the easier it will be, and the easier it is to write, the more you'll write, in a virtuous circle.

Learn C before graduating

Part two: C. Notice I didn't say C++. Although C is becoming increasingly rare, it is still the lingua franca of working programmers. It is the language they use to communicate with one another, and, more importantly, it is much closer to the machine than "modern" languages that you'll be taught in college like ML, Java, Python, whatever trendy junk they teach these days. You need to spend at least a semester getting close to the machine or [you'll never be able to create efficient code in higher level languages](#). You'll never be able to work on compilers and operating systems, which are some of the best programming jobs around. You'll never be trusted to create architectures for large scale projects. I don't care how much you know about continuations and closures and exception handling: if you can't explain why **while (*s++ = *t++);** copies a string, or if that isn't the most natural thing in the world to you, well, you're programming based on superstition, as far as I'm concerned: a medical doctor who doesn't know basic anatomy, passing out prescriptions based on what the pharma sales babe said would work.

Learn microeconomics before graduating

Super quick review if you haven't taken any economics courses: econ is one of those fields



that starts off with a bang, with many [useful theories and facts](#) that make sense, can be proven in the field,



etc., and then [it's all downhill from there](#). The useful bang at the beginning is microeconomics, which is the foundation for literally every theory in business that matters. After that things start to deteriorate: you get into Macroeconomics (feel free to skip this if you want) with its interesting theories about things like the relationship of interest rates to unemployment which, er, seem to be disproven more often than they are proven, and after that it just gets worse and worse and a lot of econ majors switch out to Physics, which gets them better Wall Street jobs, anyway. But make sure you take Microeconomics, because you have to know about supply and demand, you have to know about competitive advantage, and you have to understand NPVs and discounting and marginal utility before you'll have any idea why business works the way it does.

Why should CS majors learn econ? Because a programmer who understands the fundamentals of business is going to be a more valuable programmer, to a business, than a programmer who doesn't. That's all there is to it. I can't tell you how many times I've been frustrated by programmers with crazy ideas that make sense in code but don't make sense in capitalism. If you understand this stuff, you're a more valuable programmer, and you'll get rewarded for it, for reasons which you'll also learn in micro.

Don't blow off non-CS classes just because they're boring.

Blowing off your non-CS courses is a great way to get a lower GPA.

Never underestimate how big a deal your GPA is. Lots and lots of recruiters and hiring managers, myself included, go straight to the GPA when they scan a resume, and we're not going to apologize for it. Why? Because the GPA, more than any other one number, reflects the sum of what dozens of professors over a long period of time in many different situations think about your work. SAT scores? Ha! That's one test over a few hours. The GPA reflects hundreds of papers and midterms and classroom participations over four years. Yeah, it's got its problems. There has been grade inflation over the years. Nothing about your GPA says whether you got that GPA taking easy classes in home economics at Podunk Community College or taking graduate level Quantum Mechanics at Caltech. Eventually, after I screen out all the 2.5 GPAs from Podunk Community, I'm going to ask for transcripts and recommendations. And then I'm going to look for *consistently* high grades, not just high grades in computer science.

Why should I, as an employer looking for software developers, care about what grade you got in European History? After all, history is *boring*. Oh, so, you're saying I should hire you because you don't work very hard when the work is boring? Well, there's boring stuff in programming, too. Every job has its boring moments. And I don't want to hire people that only want to do the fun stuff.

I took this course in college called Cultural Anthropology because I figured, what the heck, I need to learn *something* about anthropology, and this looked like an interesting survey course.

Interesting? Not even close! I had to read these incredibly monotonous books about Indians in the Brazilian rain forest and Trobriand Islanders, who, with all due respect,



are not very interesting to me. At some point, the class was so incredibly wearisome that I longed for something more exciting, like *watching grass grow*. I had completely lost interest in the subject matter. Completely, and thoroughly. My *eyes teared* I was so tired of the endless discussions of piling up yams. I don't know why the Trobriand Islanders spend so much time piling up yams, I can't remember any more, it's incredibly boring, but It Was Going To Be On The Midterm, so I plowed through it. I eventually decided that Cultural Anthropology was going to be my Boredom Gauntlet: my personal obstacle course of tedium. If I could get an A in a class where the tests required me to learn all about potlatch blankets, I could handle anything, no matter how boring. The next time I accidentally get stuck in Lincoln Center sitting through all 18 hours of Wagner's Ring Cycle, I could thank my studies of the Kwakiutl for making it seem pleasant by comparison.

I got an A. And if I could do it, you can do it.

Take programming-intensive courses.

I remember the exact moment I vowed never to go to graduate school.

It was in a course on [Dynamic Logic](#), taught by the dynamic Lenore Zuck at Yale, one of the brightest of an array of very bright CS faculty.

Now, my murky recollections are not going to do proper credit to this field, but let me muddle through anyway. The idea of Formal Logic is that you prove things are true because other things are true. For example thanks to Formal Logic, "Everyone who gets good grades will get hired" plus "Johnny got good grades" allows you to discover the new true fact, "Johnny will get hired." It's all very quaint and it only takes ten seconds for a deconstructionist to totally tear apart everything useful in Formal Logic so you're left with something fun, but useless.

Now, *dynamic* logic is the same thing, with the addition of time. For example, "*after* you turn the light on, you can see your shoes" plus "The light went on in the past" implies "you can see your shoes."

Dynamic Logic is appealing to brilliant theoreticians like Professor Zuck because it holds up the hope that you might be able to *formally* prove things about computer programs, which could be very useful, if, for example, you could formally prove that the Mars Rover's flash

card wouldn't overflow and cause itself to be rebooted again and again all day long when it's supposed to be driving around the red planet looking for Marvin the Martian.

So in the first day of that class, Dr. Zuck filled up two entire whiteboards and quite a lot of the wall *next to* the whiteboards proving that if you have a light switch, and the light was off, and you flip the switch, the light will then be on.

The proof was insanely complicated, and very error-prone. It was harder to prove that the *proof* was correct than to convince yourself of the fact that switching a light switch turns on the light. Indeed the multiple whiteboards of proof included many skipped steps, skipped because they were too tedious to go into formally. Many steps were reached using the long-cherished method of Proof by Induction, others by Proof by Reductio ad Absurdum, and still others using Proof by Graduate Student.

For our homework, we had to prove the converse: *if* the light was off, *and* it's on now, prove that you flipped it.

I tried, I really did.

I spent hours in the library trying.

After a couple of hours I found a mistake in Dr. Zuck's original proof which I was trying to emulate. Probably I copied it down wrong, but it made me realize something: if it takes three hours of filling up blackboards to prove something trivial, allowing hundreds of opportunities for mistakes to slip in, this mechanism would never be able to prove things that are *interesting*.

Not that that matters to dynamic logicians: they're not in it for *useful*, they're in it for tenure.

I dropped the class and vowed never to go to graduate school in Computer Science.

The moral of the story is that computer science is not the same as software development. If you're really really lucky, your school might have a decent software development curriculum, although, they might not, because elite schools think that teaching practical skills is better left to the technical-vocational institutes and the prison rehabilitation programs. You can learn mere *programming* anywhere. We are Yale University, and we Mold Future World Leaders. You think your \$160,000 tuition entitles you to learn about *while loops*? What do you think this is, some fly-by-night *Java seminar* at the Airport Marriott? Pshaw.

The trouble is, we don't really have professional schools in software development, so if you want to be a programmer, you probably majored in Computer Science. Which is a fine subject to major in, but it's a *different subject* than software development.

If you're lucky, though, you can find lots of programming-intensive courses in the CS department, just like you can find lots of courses in

the History department where you'll write enough to learn how to write. And those are the best classes to take. If you love programming, don't feel bad if you don't understand the point of those courses in lambda calculus or linear algebra where you never touch a computer. Look for the 400-level courses with Practicum in the name. This is an attempt to hide a useful (*shudder*) course from the Liberal Artsy Fartsy Administration by dolling it up with a Latin name.

Stop worrying about all the jobs going to India.

Well, OK, first of all, if you're already *in* India, you never really had to worry about this, so don't even *start* worrying about all the jobs going to India. They're wonderful jobs, enjoy them in good health.

But I keep hearing that enrollment in CS departments is dropping perilously, and one reason I hear for it is "students are afraid to go into a field where all the jobs are going to India." That's so wrong for so many reasons. First, trying to choose a career based on a current business fad is foolish. Second, programming is incredibly good training for all kinds of fabulously interesting jobs, such as business process engineering, even if every single programming job does go to India and China. Third, and trust me on this, there's still an incredible shortage of the really good programmers, here *and* in India. Yes, there are a bunch of out of work IT people making a lot of noise about how long they've been out of work, but you know what? At the risk of pissing them off, really good programmers *do* have jobs. Fourth, you got any better ideas? What are you going to do, major in History? Then you'll have no choice but to go to law school. And there's one thing I do know: 99% of working lawyers *hate* their jobs, hate every waking minute of it, and they're working 90 hour weeks, too. Like I said: if you love to program computers, count your blessings: you are in a very fortunate minority of people who can make a great living doing work they love.

Anyway, I don't think students really think about this. The drop in CS enrollment is merely a resumption of historically normal levels after a big bubble in enrollment caused by dotcom mania. That bubble consisted of people who didn't really like programming but thought the sexy high paid jobs and the chances to IPO at age 24 were to be found in the CS department. Those people, thankfully, are long gone.

No matter what you do, get a good summer internship.

Smart recruiters know that the people who love programming wrote a database for their dentist in 8th grade, and taught at computer camp for three summers before college, and built the content management system for the campus newspaper, and had summer internships at software companies. That's what they're looking for on your resume.

If you enjoy programming, the biggest mistake you can make is to take any kind of job--summer, part time, or otherwise--that is not a



programming job. I know, every other 19-year-old wants to work in the mall folding shirts, but you have a skill that is incredibly valuable even when you're 19, and it's foolish to waste it folding shirts. By the time you graduate, you really should have a resume that lists a whole bunch of programming jobs. The A&F graduates are going to be working at Enterprise Rent-a-Car "helping people with their rental needs." (Except for Tom Welling. He plays Superman on TV.)



To make your life really easy, and to underscore just how completely self-serving this whole essay is, my company, Fog Creek Software, has [summer internships in software development](#) that look *great* on resumes. "You will most likely learn more about software coding, development, and business with Fog Creek Software than any other internship out there," says Ben, one of the interns from last summer, and not entirely because I sent a goon out to his dorm room to get him to say that. The application deadline is February 1st. Get on it.

If you follow my advice, you, too, may end up selling stock in Microsoft way too soon, turning down jobs at Google because you want your own office with a door, and other stupid life decisions, but they won't be my fault. I told you not to listen to me.

Next: [Colo Expansion Version 2.0](#)

Want to know more? You're reading [Joel on Software](#), stuffed with years and years of completely raving mad articles about software development, managing software teams, designing user interfaces, running successful software companies, and rubber duckies.

About the author. I'm [Joel Spolsky](#), founder of [Fog Creek Software](#), a New York company that proves that you can treat programmers well and still be highly profitable. Programmers get private offices, free lunch, and work 40 hours a week. Customers only pay for software if they're delighted. We make [FogBugz](#), an enlightened project management system designed to help great teams develop brilliant software, [Kiln](#), which provides distributed version control and code reviews, and [Fog Creek Copilot](#), which makes remote desktop access easy. I'm also the co-founder of [Stack Overflow](#).

© 2000-2010 Joel Spolsky
joel@joelonsoftware.com

Have you been wondering about Distributed Version Control? I wrote a tutorial for my favorite DVCS, Mercurial. It's called [Hg Init](#)—check it out!

