Disarmingly Forthright MSCS Advice*

Nick Black (nick.black@gatech.edu)

November 21, 2009

1 Abstract

As a second-year CSMS student at GT (CSBS '05, from this same beloved Institute) and a presumptuous cad, I thought fit to provide some advice "from the trenches" as it were, sans hindsight or indeed even a 4.0.

2 If you'll only take away two things...

1. Read the damn man pages.[1]
2. Check your damn return values.[2]

90% of life can be handled with these two maxims, and the rest is mainly just filling out forms, getting brakes fixed, and spendin' all that coder money. Right?

3 Let's Get Down to Brass Tacks

No one's forcing you to be here. Possession of a GT MSCS is pretty much winning the life lottery. Some people win the state lottery, and yet somehow a decade later they're freshly broke and no less ignorant. These people are loathed, as is right and proper; hate's what makes America strong. So long as you effect reasonable choices from here on out, and provide work befitting your talent, you've got it made for all reasonable definitions of the expression.

This comes with a grim responsibility: don't make the most of it, and you'll be roundly despised, most of all by yourself. That doesn't mean finishing with 3 great recs and a publication record, or even finishing at all[3], but it does mean constantly learning and always trying hard. Cliché and trite, sure, but clichés can mean something entirely new after a year at GT.

[1] If you don't know what a 'man page' is, find out ASAP. Quick, like a bunny! *Schnell!*

[2] If you don't know what a 'return value' is... you all know what a return value is, right?

[3] One of the finest people I know left his BSCS, three years in, to be a chef. Next year, he'll be Cornell's first Food Science doctorate, and he's on NPR all the time. Epic win.

4 You're a CS (or infosec, or bioinformatics, or CSE) MS student. Act it.

- Join the ACM and IEEE. I can't stress this enough. Read the *Communications* at a minimum, and keep abreast of the *Journal*. If you've never done the ACM Programming Competition, you get a year of graduate student eligibility — it's a ton of fun, a great resume entry, and hones your skills like nothing else (also, you meet good people).
- Don't embarrass yourself. Bad passwords, unencrypted authentication channels, and files left world-accessible may well see your data stolen, modified, or trojanized. There are thousands of undergraduates here, a substantial portion both frightfully skillful and utterly mad.
- If you don't have at least 100 semi-frequent, provocative/informative RSS feeds you're checking a few times daily, you're not learning enough.
- Read the classics. Browse the New Hacker Dictionary (I got an entry in as an undergraduate). Gaily overuse words like 'grok', 'batch', '*-state' and 'contextswitch', and sound the occasional barbaric **W00T** over the roofs of the world. Take at least a stroll through APIUE[4], SICP[5] and TAoCP[6].

5 Software

- I strongly encourage you to run an open, POSIX-style operating system during at least your MS. If you need some Windows or OSX closed-source crutchery at first, use any of the advanced virtualization solutions on Linux or FreeBSD — "learning KVM" ought be something you can eagerly tackle in a day or two. No one who learns UNIX well ever leaves her, and it's immediate street cred. You'll also be far out ahead of the game in Systems and Networking classes, and anything involving server administration.
- This goes without saying, but learn and run Vim or Emacs, or both.[7]
- All documents produced from here out ought be either UTF-8 plaintext or LaTeX output. My first reaction to non-LaTeX documents is to deflate; it's very unlikely that anything really interesting lurks within. Putting your valuable results into anything else would be like building a time machine out of a 1986 Dodge Caravan — it just ain't done!

[4] Really, everything W. Richard Stevens wrote is canonical.
[5] Available online: `http://mitpress.mit.edu/sicp/`
[6] Find Knuth's mistakes: `http://www-cs-faculty.stanford.edu/~knuth/boss.html`
[7] I'm a vim man myself (over 500 lines of `.vimrc`!), but I do love some Emacs pgsml mode.

6 Programming

Some of you will never be programmers, and think of it as something you need "get through" — you're here planning to manage people or product, or start a company, or add side skills to an existing career. I guess this doesn't really apply to you, unless you want to be able to identify good programmers[8] *and know why things work. The rest of you, read on...*

- If you haven't yet, internalize that the vast majority of code you'll read is laughably broken. This is true for open source (except for a few projects), textbooks (except for a few revered texts), industry (except for a very few coworkers), and especially webfora. It's true for yourself — if you aren't, at any given time, scandalized by code you wrote five or even three years ago, you're not learning anywhere near enough. You got away with this because undergraduate projects are thrown away upon completion, and standards in the industry are appallingly low. Apply Sturgeon's Law!

- As a corollary, seek out, study, and bookmark good code. When you see a new technique, reason it out. If you dig it, go apply it to some existing or new code of your own — and seek further gems. Gold in code, just like ore, comes in localized seams. You either got it, or you don't.[9]

- You must learn to program axiomatically — that is, you must take each element of the system, language, and toolchain, and learn it throughout. You needn't memorize every detail right away, but you do need to know what kind of things to watch for and what capabilities exist. This will, of course, improve any given project you're working on. More importantly, you'll begin to see how things connect and existing design patterns.

- In addition to (at least) a scriptingish language (Python, Ruby), a functional language (Haskell, ML, F♯), and a data retrieval language (SQL, XPath), learn C *well*. It's small enough to easily do so, remains a *lingua franca*, allows direct analysis of caching effects and other architectural aspects, and as of 2009 provides the native bindings for every operating system that matters. Java and perl lie atop the rubbish heap of programming language history. Eschew them. Learn the shell and GNU readline.

- Keep all your projects in source control systems like `git` or `svn`.

7 Cheating

If you see people cheating, set them on fire. Working with people who obviously cheated is the most unpleasant work experience I've ever had.

[8] I advise Paul Graham's essay, "Great Hackers": `http://www.paulgraham.com/gh.html`

[9] Some of the best code I can recommend includes: the core Linux kernel, OpenVPN, OProfile, OpenSSH, and the NPTL threading implementation from GNU libc. I recommend OpenSSL, the Linux PATA or TTY implementations, the SVR4 shared memory API, or the Berkeley Sockets resolver(3) API for examples of what *not* to do.

8 Classes

- Be constantly shaping your project or thesis. **Do** a project or thesis.
- Project specs will contain ambiguities, contradictions and outright nonsense. Watch the class newsgroup and webfora diligently, before, as and after you work. Don't be afraid to post requests for clarification (this is preferable to mailing TA's).
- Understand that professors will regularly have only a hazy idea of what's going on in a project, and class is never the place to bring projects up.
- Whining about grading, assignment dates, assignment difficulty etc elicits contempt. Someone else will always be the one to bitch, and if not, you're the only one bitching. There is almost always a curve; just calm down, do the best anyone can do with the class, and be fine.
- PhD students are singularly-driven creatures, and take classes for their own, mysterious reasons. Working with one in a group will mean phone calls and emails, pretty much constantly, until the assignment is done. This can be both useful and maddening. In a pinch, they also generally know what's going on at any given time.
- Every prof begins the semester by dividing it up into project periods. Every project class has 1–4 projects. Pigeonhole principle that against a fulltime load, and realize you'll almost certainly have work coming due in clusters. This is why you're given weeks for what seem weeklong tasks.
- Nothing seems to piss off professors more than totally bailing out on an assignment. This seems worse to do late than early.
- Similarly, masterful or atrocious final exam performances are pivotal.
- You'd better be very, very good if you skip 20% of a class.
- Corollary: Night owl? Think twice about classes before 10:00.
- Barring extraordinary personal efforts, Systems and Networking classes are the most programming-intensive classes, while Theory and statalicious AI offerings are the most mentally demanding. They're similarly the most valuable on a general-purpose programming resume.
- Theory classes beginning with a '7' are probably best avoided unless one's a theory student. '6' indicates "standard grad class" while '8' suggests "experimental/seminar", but '7' somehow got stuck with "*lasciate ogne speranza, voi ch'entrate.*"
- You're useless after 40 hours awake. We live in an age of cognitive enhancement; perhaps their most baneful effect is fostering a delusion of competence long after the brain's shut down. Get regular sleep.

—Atlanta, 2009