

What I Learned @ MIT

by Philip Chu

Table of contents

1 Publication Information.....	2
2 Work Alone.....	2
3 Keep it Simple.....	3
4 Keep It Working.....	4
5 Steady Wins the Race.....	4
6 Don't Work from Scratch.....	5
7 Read the Documentation.....	5
8 Remember Presentation.....	5
9 Keep Your Standards High.....	6
10 Smart People Are a Dime a Dozen.....	6
11 Theory and Practice.....	7
12 Don't Trust Politicians.....	7

1. Publication Information

Copyright ©2005-2009 by Philip Chu All rights reserved.



When I get together with my old college pals, we typically regress and complain bitterly about the lack of women (this was back in the eighties), the concrete surroundings and lousy weather, how we missed out on the fun we're sure everyone else had at other colleges, and finally how they didn't teach us really practical things. Mostly we complain about the women. But as far as practical knowledge, looking back, I can see there were plenty of unintentionally imparted lessons about real life in the engineering world.

2. Work Alone

My software engineering course, 6.170 (MIT classes are known by number rather than name), consisted of successively more complex projects, starting with small modules with well-defined interfaces and culminating in three-person projects where we would partition the application into modules assigned among us and then put everything together for a demo

(and our final grade).

Our project that year was a Reversi game - my teammates worked on the user interface display and keyboard input, while I worked on the actual game engine. My teammates alternated between sending me pointless talk messages in the lab (for you youngsters out there, the forerunner of instant messaging), complaining about my absence when I wasn't in the lab, and then covered their own asses when our program displayed an obviously incorrect board the demo ("See, the user interface is working, his game engine must be the problem").

One could draw an important methodology lesson:

- Integrate early and often.

But I think there are some overriding lessons in this case:

- Academics don't know anything about software engineering. Setting aside the faith in the design-partition-integrate process (isn't that cute!), using a research language only used at MIT on an obsolete operating system (TOPS-20) on slow machines in an overcrowded lab isn't a recipe for productive software engineering. Although that's probably how we ended up with the current state of the air traffic control system.
- Don't work with anyone else if you don't need to. I could have done the entire project myself in less time (and I did, for my bachelor's thesis - a Reversi program running on an experimental multiprocessor machine).

3. Keep it Simple

As the final project of my electronics lab course, 6.111, my partner (who went by the moniker Psi) and I wanted to build a computer. My partner wanted to build a Lisp machine - the Lisp machines of the time involved a lot of memory and software, typically selling for \$100,000, so I persuaded him to scale our ambitions down to a Forth machine.

But then our teaching assistant looked at our design proposal and felt it wasn't complex enough and required us to double our chip count. After sleepless nights burning programmable logic arrays, finding/returning/replacing faulty chips, we had a wiry mess of lights that occasionally blinked out a correct computation.

The engineering lessons learned:

- Watch out for long wires - the digital abstraction doesn't always hold when inductance gets in the way.

The real lessons:

- Establish a good relationship with the suppliers. The old men staffing the lab parts

department would give the best, i.e. working, parts to the pretty girls ("she has better parts", one of them replied when asked why he gave atypically good service and components to one of the female students), and recycle the burnt out chips among everyone else.

- Keep the design as simple as possible. Adding complexity just for the sake of it is just risk maximization - not something you want to do in real world engineering projects, unless you're graded on complexity and not a working result.

4. Keep It Working

I did build a functioning computer in the undergraduate computer architecture course (possibly my favorite), 6.115. The course progressed through the construction of a computer from scratch - and I mean really from scratch, building the processor and clock, wiring the data pathways, and writing microcode.

The end of the project featured optional entry in a performance contest - if your computer is the fastest, you get to keep it (worth a couple hundred bucks). The hardware guys could add optimizations like caching - I restricted myself to the many obvious optimizations available in the microcode. But even with software changes, there was a long turnaround interval in getting them burned on the PAL's. In the end, I was unable to enter because my changes were buggy and I didn't have time to debug-fix-burn-test.

My hardware engineer friends might say I should have left the software alone and worked with hardware, but the fact is, only a few people entered the contest with working kits, and only the winner had achieved a significant performance improvement. So the lesson learned is:

- Keep your system in a working state. If I had made one minor change per PAL burn instead of trying to make every optimization I could think of, I would have entered the contest with a handful of optimizations and at least placed second.

5. Steady Wins the Race

I was a crunch-time student (which, ironically, has prepared me for common industry worst practices). It seems in hazy retrospect that I stayed up every night cramming. The result was academic probation my second year followed by mediocre grade points the rest of the way.

I noticed the students who had perfect grade points weren't the whiz-bang smartest - they were the ones who kept regular schedules and maintained discipline, closing their dorm room doors to study, eating regular meals and going to sleep at midnight.

- So I wouldn't go so far as to say slow and steady wins the race, but steady is important.

6. Don't Work from Scratch

I tried to do all my homework ("problem sets", they're called at MIT) by myself. Mistake. There's a time-honored tradition of, shall we say, collaborative work among MIT undergrads, assisted by class "bibles" containing assignments and exams material, meticulously assembled and lovingly passed down over the years. When the professors, clued in or not, see their students handling the class material easily, they increase the workload (hence the on-campus popularity of the phrase, "drinking from a fire hose") until the curve is back down to where they like it.

- Trying to learn everything from scratch is a loser's game - take advantage of available literature.

This isn't to say you shouldn't learn the material - if you're going to apply the knowledge, you have to learn it. A fraternity member in one of my classes turned in a photocopy for his homework. Now, that's just too much.

7. Read the Documentation

My woeful track record on hardware projects over three years was compounded by my inability to use the ultra-cool and completely mystifying Tektronix oscilloscopes that populated the labs. I would twiddle the knobs helplessly until a TA would come by and click-click all of a sudden I had a nice waveform on the screen. When I wrapped up the final project of my final lab in my final year, I happened to look behind the instrument and realized each of them had a manual.

- RTFM

8. Remember Presentation

When I was there, the undergraduate MIT body boasted a three-to-one male-female ratio, a marked improvement over previous years (and vastly superior to Caltech's nine-to-one ratio, which scared me away from that school). I could count on my male friends to ditch me any minute they had a chance to talk to a female classmate. But on Friday nights they gazed turned to the buses from area all-female schools disgorging perfumed, coiffed high-heeled undergrads looking for MIT frat parties, while MIT women in jeans and t-shirts looked on in irritation.

- When there's competition, packaging counts.

9. Keep Your Standards High

MIT does try to attract a diverse student body - at a party hosted by the chairman of MIT, a classmate of mine hailing from Michigan asked a representative from the admissions office if MIT gave preference to applicants from underrepresented regions. The rep explained that MIT gave consideration to the educational obstacles Midwesterners might encounter.

Notwithstanding that politically-correct response (or else the admissions officer really thought I studied by candlelight after putting the hogs back in the pen during my high school years in Iowa) you can't get into MIT, and you definitely can't get out, unless you can get through the math and science courses. And MIT doesn't have catch-up courses for varsity athletes or "physics for poets" that even the Ivy League schools have. You have to pass all the core courses that everyone else has to, and that includes the swim test. And you don't get in by being pretty - MIT doesn't require a photo with your application, unlike Stanford (and, oddly enough, CalTech - maybe it's a California thing).

Furthermore, as an MIT undergrad, you get very bright peers, famous researchers as your instructors (and some of them are excellent instructors, too), access to cutting-edge facilities, and opportunities to participate in research. Besides the brand-name degree, you come out of that institution with high standards.

I came to that realization soon after graduating and moving to Texas. The sales guy who sold me my first car at Toyota of Irving asked me if MIT was in Minnesota. Slightly more informed was my coworker at Texas Instruments who stated upon meeting me that he would have gone to MIT if he had known about it. (I thought that was the dumbest thing I'd ever heard, but it was my first day) And a few years later, I was astounded by the level of whining I encountered as a teaching assistant at Johns Hopkins for Computer Literacy 101 (the name says it all). At MIT I listened to bragadaccio about consecutive all-nighters and a lot of persistent negotiation for grades, but the elite students of JHU were upset at me for not announcing the exact questions to expect on their exams.

- Even when your expectations are low (and to be pragmatic and realistic, they often have to be), maintain high standards.

10. Smart People Are a Dime a Dozen

At MIT, everyone is bright, and some are super-bright. Some super-bright people are total jerks, and some are the nicest you'll ever meet.

I had one famous computer scientist as an instructor who bore down on a student for not knowing how to solve a digital signal processing problem - after she protested that she didn't know the complex algebra required, he said, "come to me after class, and I'll teach you

complex algebra." And then, after class, he told her, "I don't have time."

In contrast, I had an equally famed materials science instructor who set aside office hours to meet with any of his dozens of students who needed help on their homework, even though he had plenty of teaching assistants.

- So when I join a company and hear how smart the people there are - I'm not impressed. Unless they're also good people to work with. That's a lot harder to find.

11. Theory and Practice

Despite boasting some of the brightest minds and a top-ranked business school, MIT is hardly a flawlessly-run institution. How many other industries can get away with increasing prices faster than the rate of inflation? Any illusion I had that MIT was a smoothly-running machine was shattered when, as a party-time system administrator, I had to run a purchase order by four consecutive desks in the purchasing office, where the form was literally rubber-stamped each time, only to return across campus to my supervisor, who looked at it and said, "this isn't right".

- You can have smart people, lots of money and the latest technology, but your execution can still suck.

And like any other bureaucracy, MIT seems to run smoothly most of the time, but when something goes wrong, fixing it is like arguing with a rock. I didn't get paid the entire summer I worked at the MIT Microcomputer Center because my manager repeatedly submitted my monthly timecard late (and lied about it), and that gets you shrugs instead of paychecks from the payroll office (I got paid after complaining to the Information Services director that I had tuition to pay, and my boss got a nice referral to a cushy job at another high-profile university) When I later worked in one of the MIT labs, it was Fedex who had a similar problem - they refused to accept deliveries from us for a while because MIT's account was in arrears.

- Top-notch institutions attract the best and the brightest, but that doesn't mean they don't also attract the worst.

My favorite encounter with mindless MIT bureaucracy came after graduation. The student loan department sent me a letter saying they didn't have my address. (I suppose I should have called them and complained I didn't have their phone number). The humor of the situation was diminished by the late fees they charged me.

12. Don't Trust Politicians

Before graduation, some of my classmates in the student government went around asking for donations to our class gift, a scholarship fund. They neglected to mention that only members of the student government or varsity sports were eligible. Members of the unique and probably more useful organizations like the Lecture Series Committee (they arranged popular weekly showings of modern and classic movies and hosted such distinguished speakers as William Shatner), were left out.

- Even at MIT, nerds get screwed. Or, as they say in New England, scrod.