

# Learning software development — by developing software

Massachusetts Institute of Technology

Since at least the late 19th century, when John Dewey opened his experimental Laboratory School at the University of Chicago, experiential learning — learning by doing — has had strong proponents among educational theorists. In MIT's Department of Electrical Engineering and Computer Science (EECS), the influence of experiential-learning theory can be seen in several courses in which each student spends the entire semester working on a single programming project.

Even such project-based classes, however, miss aspects of the experience of commercial software development. "If you go to work at Microsoft, for example, you're going to be handed code with 30 years of history, and you have to be able to quickly get up to speed, navigate hundreds of thousands of lines of code and then build on top of it, often without access to the people who originally wrote it," says Ted Benson, a PhD student in EECS, who before coming to MIT worked for three years as a commercial software developer. "And then you need to make your contributions in a way where, 30 years later, other people can do the same."

This spring, Benson and his thesis advisor, professor of computer science and engineering David Karger, created a new course in which rather than developing small projects from scratch, students participate in large, ongoing, open-source-software development initiatives, mentored by industry professionals. And as is the case with much modern commercial software development, they collaborate online with geographically dispersed colleagues — in this case, their fellow students at some 15 universities around the world.

The MIT course, and the consortium of universities whose students participate in the development projects, grew out of the deliberations of a committee of leading computer science educators, convened by Facebook to evaluate university education in software engineering. Karger was on the committee, as was Jay Borenstein, a lecturer in computer science at Stanford University. Borenstein spearheaded the consortium's new educational initiative, working with members of the open-source-software community to identify promising projects and recruiting the industry mentors.

## Baptism of fire

Not only does working on real development projects impart practical skills that are difficult to acquire in a conventional classroom setting, Benson says, but it also engages the students in a way that readings and problem sets rarely do.

"There's this age-old question for any teacher, which is how do you motivate the students to really buy into what they're learning," Benson says. "They're fixing bugs and adding features to software that will touch millions of users. And when that's

your homework, it's completely different. I've had students give me high-fives when they come in to report that they've finished something."

One group of students, for instance, is helping repair a deep-rooted problem with the popular web-development framework Ruby on Rails. Frequently, tasks executed by commercial sites need to be processed as "transactions," meaning that either all the aspects of the task are executed or none are: You wouldn't want, say, a travel site charging you for one leg of a trip when it couldn't find a return flight. Ruby on Rails had a bug, however, that meant that sometimes, failed transactions left program code out of sync with the database. MIT students are helping fix it. Another group is helping to develop a monitoring tool for the open-source database application MongoDB, so that application users can tell what types of queries the database is receiving and which servers are processing them.

### Software studio

The design of the course — the Open Source Software Project Lab, or 6.S194 in MIT's course-numbering scheme — borrows elements from both the studio critiques typical of architecture courses and the residency model used in medical schools, Benson says. At the beginning of the semester, students were presented with the nine projects identified by Borenstein. On the basis of their personal preferences, they were sorted into five teams, with each assigned to a different project with a different mentor.

On Mondays, Benson lectures, often tailoring his subject matter to questions raised by the students' recent work. Every Wednesday, teams present their ongoing work to the rest of the class, explaining their approaches, inviting criticism and prompting general discussion of programming principles and philosophies.

Otherwise, the students work chiefly with each other, with their collaborators at the other schools, and with their mentors. Every week, each team meets with Benson for 20 minutes to describe the next stage of its project and report its progress on the previous stage. "They learn that a very valid thing to accomplish in a week is thoroughly understanding a particular technology and coming up with a plan for how you might use it," Benson says. "Sometimes learning is their assignment."

When students have completed work on a particular section of code, they log it into the open-source project's online code repository, where Benson can, if he chooses, review it to see if it accords with the weekly progress report. Sometimes, indeed, he has found that it doesn't — but in an unexpected way. "I have had to go to some students and say, 'Give yourself credit for this!'" Benson says. "You did far more work than I would have expected you to this week."

### Scaling up

That kind of individual attention, Benson acknowledges, is possible mainly because the MIT class, in its inaugural session, is intentionally small — only 11 students. One of the questions that he and Karger spend a lot of time discussing is how to preserve its advantages while increasing its size. One possibility is to assign the

students to more homogeneous projects — to have them all, for instance, work on different aspects of a single open-source application, such as Mozilla’s Firefox.

Another possibility is to introduce tiers of instruction, where some of the regular evaluation and feedback is provided by upperclassmen who have already taken the course. A peer mentorship program, Benson says, could be modeled on MIT’s celebrated Undergraduate Research Opportunity Program, in which undergraduates perform original research for either course credit or stipends. Or Benson and Karger might use some other recruitment mechanism altogether. “Some of the students seem to enjoy this so much that I wouldn’t be surprised if they would volunteer to do it,” Benson says.

Aaron Patterson, a senior software architect at AT&T and one of the student mentors, says that he would definitely participate in the course again, but that he could use some more help. “It’s a lot of work,” he says. “Next time, I would try to involve more people as mentors or reduce the number of students I have.”

Patterson doesn’t believe that programs like the Open Source Software Project Lab will supplant the conventional computer science curriculum, but he does think that they complement it. “The textbook background is important for long-term development, but I don’t think textbooks prepare you for how to apply those techniques to real-world software,” he says. “Techniques I learned in school were extremely helpful, but didn’t prepare me for dealing with — frankly — bad code from the real world.”

And with the Open Source Software Project Lab, he says, the students have more to show for their work than a stack of graded problem sets. “Overall, the students are making extremely valuable contributions,” he says. “The work being accomplished via the students I’m working with is greater than I could accomplish on my own.”

**Source URL (retrieved on 09/20/2014 - 9:17pm):**

<http://www.ecnmag.com/news/2013/04/learning-software-development-%E2%80%94-developing-software>