

Viewpoint

Crossing the Software Education Chasm

An Agile approach that exploits cloud computing.

VIA A REMARKABLE alignment of technologies, the future of software has been revolutionized in a way that also makes it *easier* to teach.

Cloud computing and the shift in the software industry^a toward Software as a Service^b (SaaS) using Agile development has led to tools and techniques that are a much better match to the classroom than earlier software development methods. In addition, modern programming frameworks for Agile development like Ruby on Rails demonstrate that big ideas in programming languages can deliver productivity through extensive software reuse. We can leverage such productivity to allow students to experience the whole software life cycle repeatedly within a single college course, which addresses many criticisms from industry about software education. By using free-trial online services, students can develop and deploy their SaaS apps in the cloud without using (overloaded) campus resources. Our enthusiasm for Agile, SaaS, and cloud computing is not based just on improving students' employ-

ability; rather, we are excited that they can learn and practice software engineering fundamentals, and that they use them even after graduation. Our 50,000 online students are a testimony to the popularity of the material and the scalability of our SaaS "courseware."

The Complaints and the Defense

While new college graduates are good at coding and debugging,¹ employers complain about other missing skills that are equally important. A standard faculty reaction to such criticisms is that we are trying to teach principles that prepare students for their whole careers; we are not trade schools that teach the latest superficial fads.

To understand the complaints in more depth, we spoke to representatives from a half-dozen leading software companies. We were struck by the unanimity of the number-one request from each company: that students learn how to enhance sparsely documented legacy code. In priority order, other requests were making testing a first-class citizen, working with non-technical customers, performing design reviews, and working in teams. We agree that the social skills needed to work effectively with nontechnical customers, work well in teams, and perform effective design reviews are helpful for the students' whole careers—the question is how to fit them

Turning software concepts into Rails development tools.

SWEBOK Concept	Agile Version	Rails Tool
Software Requirements	User Stories, Behavior-Driven Design (BDD)*	Cucumber (http://cukes.info)
Project Management	Velocity, Version Control	Pivotal Tracker (http://www.pivotaltracker.com/), GitHub (https://github.com/)
Software Verification and Testing	Test-driven development (TDD)	RSpec (unit/functional testing; http://rspec.info/), SimpleCov (coverage measurement; http://rubygems.org/gems/simplecov)
Software Maintenance	Refactoring to control complexity	metric-fu (captures metrics of code complexity e.g., cyclomatic and ABC; http://metric-fu.rubyforge.org/), pingdom (monitors 99%ile response times to deployed app from around the world; http://pingdom.com/). IDEs such as Aptana (http://aptana.com/) and RubyMine (http://www.jetbrains.com/ruby/) include refactoring support, method name autocompletion, visualizing dependencies among classes, and so on.
Software Lifecycle	Iterations	See Figure 3

*BDD is a variation of TDD that suggests writing higher-level acceptance or integration tests first before writing code.

a Virtually every shrink-wrap program is offered as a service, including PC standard-bearers like Office (see Office 365; <http://www.microsoft.com/en-ca/office365/online-software.aspx>) and TurboTax (see TurboTax Online; <http://turbotax.intuit.com/personal-taxes/online/compare.jsp>).

b Instead of binaries that must be installed on a local computer, SaaS delivers software and associated data as a service over the Internet, often via a thin program on client devices such as a browser.

into a course. Similarly, no one questions the value of emphasizing testing—the question is how to get students to take it seriously.

Instructors respond that even if we agreed with the recommendations, time constrains how much one class can cover. A typical undergraduate workload of four courses per term and a 50-hour workweek gives students about 12 hours per week per course, including lectures, labs, exams, and so forth. This works out to approximately 120 hours per quarter to 180 hours per semester, or just three to four weeks for a full-time developer!

A Classroom Opportunity: Agile Development

The Agile Manifesto signaled a paradigm shift for many software applications. This approach embraces change as a fact of life; small teams of developers continuously refine a working but incomplete prototype until the customer is happy with the result, with the customer offering feedback with every iteration. Agile emphasizes *Test-Driven Development*^c (TDD) to reduce mistakes, which addresses industry's request to make testing a first-class citizen; *user stories*^d to reach agreement and validate customer requirements, which addresses working with non-technical customers; and *velocity*^e to measure progress. The Agile software philosophy is to make new versions available every two weeks. The assumption is basically continuous code refactoring over its lifetime, which develops skills that can also work with legacy code. Clearly, small teams and multiple iterations of incomplete prototypes could match the classroom.

Note that we do not tell students that Agile is the only way to develop software; indeed, we explain Agile is inappropriate for safety-critical apps, for example. We believe that new programming methodologies develop and become popular in response

c In TDD you first write a failing test case that defines a new feature, and then write code to pass that test.

d A user story is a few nontechnical sentences that captures a feature the customer wants to include in the app.

e Velocity is calculated by estimating units of work per user story and then counting how many units are completed.

The only hope for addressing all the concerns from industry within the course time constraints is to use a highly productive programming framework.

to new opportunities, so we tell students to expect to learn new methodologies and frameworks in the future. Our experience is that once students learn the classic steps of software development and have a positive experience in using them via Agile, they will use these key software engineering principles in other projects no matter which methodology is used (see Figure 5).

A Classroom Target for the Post-PC Era: "I Do and I Understand"

To motivate students, it is helpful to use a platform that allows them to create compelling apps. In this emerging post-PC era, mobile applications for smartphones and tablets and Software as a Service (SaaS) for cloud computing are both compelling. (50,000 students are evidence that SaaS is indeed compelling.) As you can teach the principles with either target, given the time constraints mentioned earlier, why not pick the platform that has the most productive tools? Our view is that the only hope for addressing all the concerns from industry within one course is to use a highly productive programming framework.

Our experience is that the Rails ecosystem has by far the best tools to support test-driven development, behavior-driven design, and Agile processes, many of which are made possible by intellectually deep Ruby language features such as closures, higher-order functions, functional idioms, and

Calendar of Events

May 15–17

Computing Frontiers Conference, Caligari, Italy, Sponsored: SIGMICRO, Contact: John Feo, Email: john.feo@pnl.gov, Phone: 509-375-3768

May 15–16

Workshop on Software and Compilers for Embedded Systems, Sankt Goar, Germany, Contact: Henk Corporaal, Email: h.corporaal@tue.nl

May 16–21

The 9th Annual Conference on Theory and Applications on Models and Computation, Beijing, China, Contact: Li Angsheng, Email: angsheng@ios.ac.cn

May 21–24

6th International Conference on Pervasive Computing Technologies for Healthcare, San Diego, CA, Contact: Dr. Rosa I. Arriaga, Email: arriagea@cc.gatech.edu

May 21–25

Shape Modeling International, College Station, TX, Contact: Ergun Akleman, Email: ergun.akleman@gmail.com, Phone: 979-845-6599

May 21–25

The 2012 International Conference on Collaboration Technologies and Systems, Denver, CO, Contact: Geoffrey Fox, Email: gcfexchange@gmail.com

May 22–25

International Working Conference on Advanced Visual Interfaces, Capri Islands (Naples), Italy, Contact: Tortora Genevieve, Email: tortora@unisa.it

May 23–25

Euro-American Conference on Telematics and Information Systems, Valencia, Spain, Contact: Samper J. Javier, Email: jjsamper@uv.es

metaprogramming. The accompanying table shows critical topics from the SWEBOK (software engineering body of knowledge),⁵ the Agile technique for that topic, and the Rails tool that implements that technique. Because these tools are lightweight, seamlessly integrated with Rails, and require virtually

no installation or configuration—some are delivered as SaaS—students quickly begin learning important techniques directly by doing them. We agree with Confucius: “I hear and I forget. I see and I remember. I do and I understand.” Students see and use tools that we can explain and check, rather than just hear

lectures about a methodology and then forget to use them in their projects.

For example, the Cucumber tool automates turning customer-understandable user stories into acceptance tests. Figure 1 is an example feature for a cash register application and one “happy path” user story (called a scenario in Cucumber) for that feature (see http://en.wikipedia.org/wiki/Cucumber_%28software%29). Note that this format is easy for the nontechnical customer to understand and help develop, which is a founding principle of Agile and addresses a key criticism from industry. Cucumber uses regular expressions to match user stories to the testing harness. Figure 2 is the key section of the Cucumber and Ruby code that automates the acceptance test by matching regular expressions.

Such tools not only make it easy for students to do what they hear in lecture, but also simplify grading of student effort from a time-intensive subjective evaluation by reading code to a low-effort objective evaluation by measuring it. SimpleCov measures test coverage, *saikuro* measures cyclomatic complexity of the code, *flog* measures code assignment-branch-condition complexity, *reek* comments on code quality by highlighting “code smells,”^f Cucumber shows the number of user stories completed, and Pivotal Tracker records weekly progress and can point out problems in balance of effort by members of teams. Indeed, these tools make it plausible for the online course to have automatically gradable assignments with some teeth in them.

Compared to Java and its frameworks, Rails programmers have found factors of three to five reductions in number of lines of code, which is one indication of productivity.^{6,8} Rails also helps with a criticism of Agile in that TDD and rapid iteration can lead to poor software architecture. We do teach design patterns (see Figure 3). Indeed, the Rails framework follows the Model View Controller (MVC)^g design pattern

f Code smells highlight to sections of code that may be hard to read, maintain, or evolve.

g MVC is a design pattern that separates input logic, business logic, and user interface logic yet provides a loose coupling between them.

Figure 1. An example feature for a cash register application and one “happy path” user story for that feature.

```
Feature: Division
  In order to avoid silly mistakes
  Cashiers must be able to calculate a fraction

Scenario: Regular numbers
  Given I have entered 3 into the calculator
  And I have entered 2 into the calculator
  When I press divide
  Then the result should be 1.5 on the screen
```

Figure 2. The key section of the Cucumber and Ruby code that automates the acceptance test for the user story in Figure 1 by matching regular expressions.

```
Given /I have entered (\d+) into the calculator/ do |n|
  @calc.push n.to_i
end

When /I press (\w+)/ do |op|
  @result = @calc.send op
end

Then /the result should be (.*?) on the screen/ do |result|
  @result.should == result.to_f
end
```

Figure 3. One Agile iteration in our course.

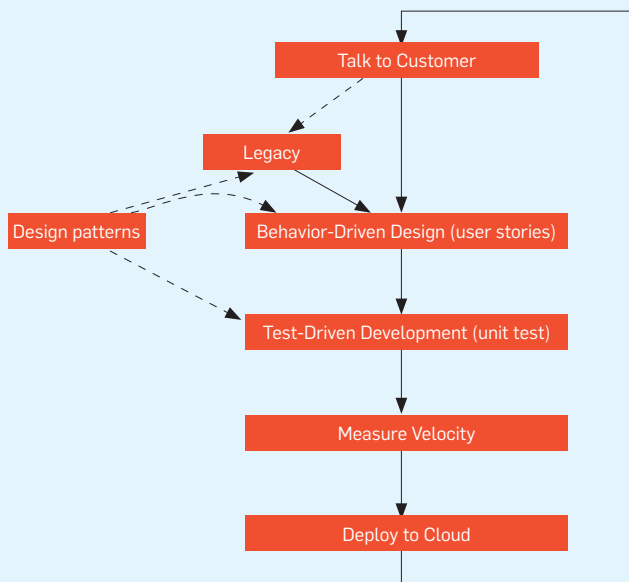


Figure 4. Survey results of software experience for former Berkeley students now in industry. (The waterfall software development process is characterized by much of the design being done in advance of coding, completing each phase before going on to the next one. The spiral model combines features of a prototyping model with the waterfall model and is intended for large projects.)



to simplify development of the classic three-tiered applications of cloud computing. In addition, because of cloud computing, deploying their projects in the same horizontally scalable environment used by professional developers is instant, free for small projects, and requires neither software installation nor joining a developer program. In particular, it frees the course from instructional computers, which are often antiquated, overloaded, or both.

One criticism of the choice of Ruby is its inefficiency compared to languages like Java or C++. Since hardware has improved approximately 1,000X in cost-performance since Java was announced in 1995 and 1,000,000X since C++ was unveiled in 1979,⁷ the efficiency of low-level code matters in fewer places today than it used to. We think using the improved cost-performance to increase programmer productivity makes sense in general, but especially so in the classroom. Note that for cloud computing, horizontal scalability can trump single-node performance; deploying as SaaS on the cloud in this course lets us teach (and test) what makes an app scalable across many servers, which is not covered elsewhere in our curriculum. Once again, without using the cloud to teach the class, we could not

offer students the chance to experiment with scalability.

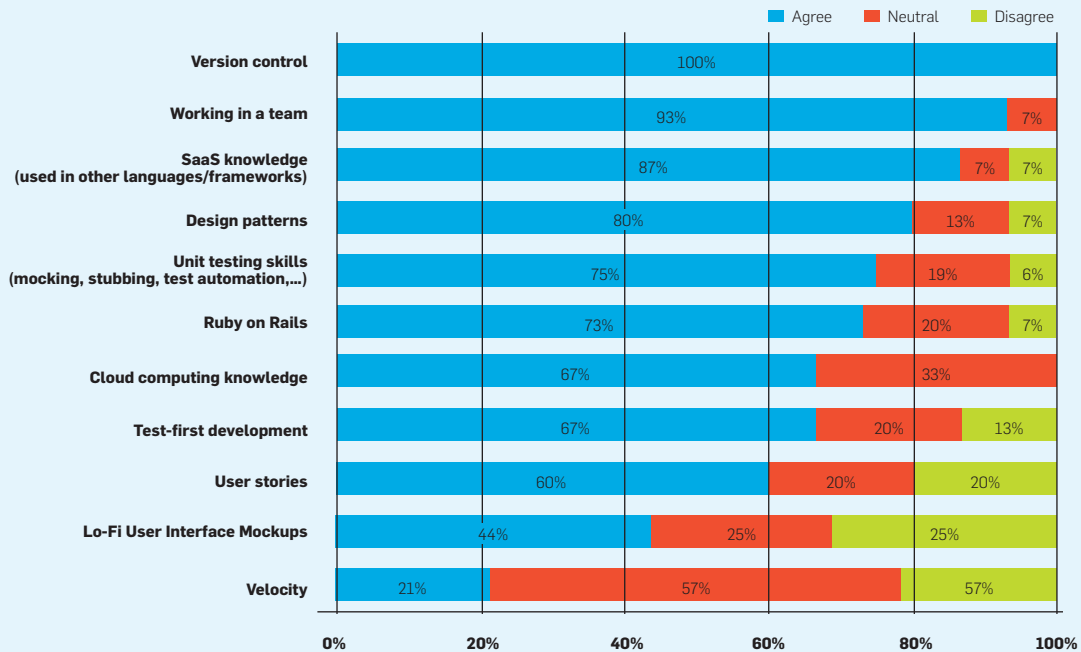
An Example Course

We use this approach to teach a software course, which currently has more than 100 juniors and seniors, at UC Berkeley. We also are offering the first part as a massive open online course (MOOC) to 50,000 online students, most of whom work in the IT industry and graduated from college five to 10 years ago.³ (The MOOC tests the scalability of the tools and automatic grading of programming assignments.) Students follow steps shown in Figure 3, which turns the concepts and tools listed in the table into a logical process. Additional tools facilitate tying together the steps of this process; for example, Autotest monitors the source code tree in the background, automatically rerunning tests and user stories via RSpec/Cucumber in response to even minor code changes, giving immediate feedback if something breaks. Following the Agile philosophy, they deploy on Amazon Web Services (AWS) (via Heroku) multiple times during the course. The teaching staff evaluates iterations by interacting with the deployed app on AWS and by using Pivotal Tracker to check velocity and stories remaining to be implemented.

A typical faculty reaction to the request for students to deal with poorly documented legacy code is that it should not exist if students are taught good practices. However, despite decades of well-meaning instructors expounding on the importance of properly factored and highly documented code, our industry colleagues assure us the legacy code problem will continue to persist. Thus, on this point we quote Shimon Peres: *“If a problem has no solution, it may not be a problem, but a fact—not to be solved, but to be coped with over time.”* Hence, if we can find principles that teach how to cope with legacy code, they *would* be appropriate for the classroom since it is a long-lasting challenge.

To learn to deal with legacy apps, our students learn and enhance a large, popular, well-written (but poorly documented) open source app written in Rails. We use Typo, a blogging framework containing 30,000 lines of Ruby and 15,000 lines of JavaScript, which suggests that it has the functionality of a Java program of 100,000 to 150,000 lines of code. We choose a Rails app rather than a Java app to maintain the programmer productivity we need to fit within our time budget. Feathers² argues that the first step is to write tests for legacy code to ensure understanding before modification, so *enhancing*

Figure 5. Ranked results from survey of former Berkeley students on whether course topics listed in the table and Figure 3 were useful in their industrial projects. These earlier versions of the course did not offer enhancing legacy code, design reviews, and working with nontechnical customers.



legacy code fits surprisingly well with the test-driven development of Agile.

To learn to communicate with nontechnical customers, for the Berkeley course we asked for projects from nonprofit organizations. The projects are typically less than 3,000 lines of code, with typically two to three times more code for testing than for the app. Teams of four or five students meet with customers on each Agile iteration for feedback on the current prototype and to prioritize the next features to add. (Moreover, we encourage students to connect these applications to Facebook or Twitter, which gives students the chance to deal with users as well as nontechnical customers within a single course.) Teams members do design reviews for each other as part of a bi-weekly class laboratory section, as Agile favors frequent code check-ins and design reviews. (Online students do not do projects.)

Using the Course Content Afterward

Figure 4 shows the survey results of Berkeley students from two earlier course offerings. Just 22 of the 47 respondents had graduated, and just 19 had done significant software projects.

The percentages indicate the results of their 26 software projects. We were surprised that Agile software development was so popular (68%) and that the cloud was such a popular platform (50%). Given that no language was used in more than 22% of the projects, our alumni must be using Agile in projects that use languages other than Ruby or Python. All the class teams had four or five students, which directly matches the average team size from the survey.

Once again, Agile development and Rails were not selected because we expected them to dominate students' professional careers upon graduation; we use them to take advantage of their productivity so we can fit several critical ideas into a single college course in the hope they will use them later no matter what methodology, programming language, and framework.

Figure 5 shows the students' ranking of the topics the table and Figure 3 in terms of usefulness in their industrial projects. Most students agreed that the top nine topics in the course were useful in their jobs. Once again, we were pleased to see that these ideas were still being used, even in industrial projects that did not rely on Agile or on Rails. The two lower ranked top-

ics were Lo-Fi User Interface Mockups, which makes sense since few developers work on the UI of a user-facing project, and Velocity, as progress can be measured in other ways in industry.

Although a small sample and not a conclusive user study, our survey offers at least anecdotal evidence that students of this course *do* continue to use successful software development techniques in later software projects of all kinds.

Conclusion

Using Agile to develop SaaS apps via highly productive tools like Rails and deploying them using cloud computing cultivates good software practices and pleases many stakeholders:


- Students like it because they get the pride of accomplishment in shipping code that works and is used by people other than their instructors, plus they get experience that can help land internships or jobs.

- Faculty like it because students actually use what they hear in lecture, even after graduation, and they experience how big CS ideas genuinely improve productivity. Virtual machines reduces hassles for faculty plus the cloud allows for more interesting program-

ming assignments—which the testing and code evaluation tools of Rails can help grade—thereby allowing us to offer a MOOC with 50,000 students.

► Colleagues in industry like it because it addresses several of their concerns. An example is this quote from a Googler: *“I think what you’re doing in your class is amazing. I’d be far more likely to prefer graduates of this program than any other I’ve seen. As you know, we’re not a Ruby shop, but I feel this is a good choice for the class to be able to get real features done. Java or C++ would take forever.”*¹⁴

We received similar comments from colleagues at Amazon, eBay, and Microsoft, none of which are “Ruby shops.” As we expected, leading software companies prefer students learn important ideas rather than steer us to teach specific languages and tools used inside those companies.

We believe Agile+Cloud+Rails can turn a perceived weakness of the CS curriculum into a potential strength. If you are a potentially interested instructor, we would be happy to help you cross the long-standing chasm between what industry recommends and what academia offers. 

References

1. Begel, A. and Simon, B. Novice software developers, all over again. In *ICER '08: Proceedings of the 4th International Workshop on Computing Education Research* (Sept. 2008).
2. Feathers, M. *Working Effectively with Legacy Code*, Prentice Hall, 2004.
3. Fox, A. and Patterson, D. Software engineering for Software as a Service; <http://www.saas-class.org>, March 2012 and May 2012.
4. Green, B. Private Communication, 2011.
5. IEEE. Guide to the Software Engineering Body of Knowledge (SWEBOK), 2004.
6. Ji, F. and Sedano, T. Comparing extreme programming and waterfall project results. *Conference on Software Engineering Education and Training 2011* (2011).
7. Patterson, D.A. and Hennessy, J.L. *Computer Organization and Design: The Hardware/Software Interface*. Revised 4th Edition, Morgan Kaufmann Publishers, 2012.
8. Stella, L.F.F., Jarzabek, S. and Wadhwa, B. A comparative study of maintainability of Web applications on J2EE, .NET, and Ruby on Rails. *WSE 2008. 10th International Symposium on Web Site Evolution* (Oct. 3–4, 2008), 93–99.

Armando Fox (fox@cs.berkeley.edu) is an adjunct associate professor at UC Berkeley and a co-founder of the Berkeley RAD Lab.

David Patterson (patt@cs.berkeley.edu) is the E.H. and M.E. Pardee Chair of Computer Science at UC Berkeley and is a past president of ACM.

We thank our colleagues at Amazon Web Services, eBay, Facebook, GitHub, Google, Heroku, Microsoft, and Pivotal Labs for their feedback and suggestions on this Viewpoint, for their support and feedback during the development of the course, and for their donations of services to support the online course.

Copyright held by author.

ACM's Career & Job Center

Looking for your next IT job?

Need Career Advice?

Visit ACM's Career & Job Center at:

<http://jobs.acm.org>

Offering a host of career-enhancing benefits:

- A highly targeted focus on job opportunities in the computing industry
- Access to hundreds of corporate job postings
- Resume posting keeping you connected to the employment market while letting you maintain full control over your confidential information
- An advanced Job Alert system notifies you of new opportunities matching your criteria
- Career coaching and guidance from trained experts dedicated to your success
- A content library of the best career articles compiled from hundreds of sources, and much more!

The ACM Career & Job Center is the perfect place to begin searching for your next employment opportunity!

<http://jobs.acm.org>



Association for
Computing Machinery

Advancing Computing as a Science & Profession

Copyright of Communications of the ACM is the property of Association for Computing Machinery and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.