# Myths about MOOCs & Agile

## Retooling & scaling up an Introductory Software Engineering course

Armando Fox & David Patterson
University of California, Berkeley

CSEE&T 2013, San Francisco, CA

# Context

- UC Berkeley undergraduate Computer Science (not Software Engineering) degree program
- Intro. Software Engineering upper-division course
  - "restricted elective" (n-out-of-k)
  - fulfills design component (open-ended team project)
  - 15-week semester
- As instructors, not always clear which standards document should provide guidance (SE 2004, ACM/IEEE CS 2013, SWEBOK)

# The Problem

- Berkeley's SW Eng course had mixed reputation
  - Students: "we are learning *about* methodologies, but not *applying* them in relevant projects"
  - Instructors: students don't practice what we teach them
  - Employers*: students can write code, but lack basic and important software skills, especially:

1. Dealing with legacy code**
2. Working in team with nontechnical customer
3. Automated testing

** Large companies: Google, Microsoft, Amazon Web Services, VMware, eBay , Salesforce. Small companies: GitHub, Heroku, Pivotal Labs, SauceLabs

* Unanimously #1 among 6 large software companies we asked

# The Constraints

- Typical ugrad: ≤12 hrs/week per course
  - 15 week course = 3 weeks of fulltime work
- Need high productivity tools so nontrivial apps can be completed in 1 semester
- Future of exciting SW = "client + cloud" apps
- Rails on cloud has *best testing & code-grooming tools*

?

# This talk

- How did retooling to Agile+SaaS affect the course & students?
- If successful, can course be *scaled up* (teach more students) and *scaled out* (used flexibly at other institutions)?
- Does course meet new Software Engineering curriculum guidelines? (*cs2013.org*)

# This talk

- How did retooling to Agile+SaaS affect the course & students?
- If successful, can course be *scaled up* (teach more students) and *scaled out* (used flexibly at other institutions)?
- Does course meet new Software Engineering curriculum guidelines? (*cs2013.org*)

# Response: revised course

- Teach fundamental SW Engineering skills using productive Rails SaaS framework
- Learn by doing: methodologies → tools
- Uses & teaches Cloud Computing
- Small-team, Agile dev (ideal for classroom)
- Real customers
- Emphasizes testing

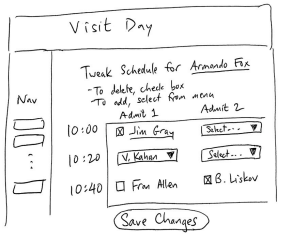
# UC Berkeley upper-division Intro to Software Engineering

saas-class.org

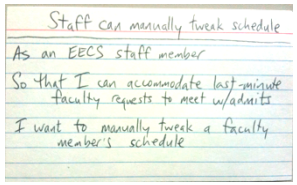
| Week# and Topics (3 lecture-hours + 1 section-hour per week) | 1-pizza team project |
|---|---|
| 1. Intro to SaaS, Agile vs. "Plan & Document" centric approaches | |
| 2. Pair programming, Scrum, Ruby intro, TDD intro | Form teams |
| 3. BDD intro, user stories, lo-fi mockups, velocity, SaaS architecture | Pick project/customer |
| 4. Model-View-Controller, Rails intro, ActiveRecord design pattern | Customer meeting 1 |
| 5. Unit & functional testing, mocks & stubs, fixtures, test coverage | Customer meeting 2 |
| 6. DRYing out code, Associations, advanced Rails features, RESTful service-oriented architecture | Review lo-fi mockups with customer |
| 7. Project management, design reviews, version control for small teams | Iter. 1 |
| 8. Legacy code: exploring codebase, characterization tests, metrics, code smells, refactoring | |
| 9. JavaScript intro, event-driven programming, JSON & AJAX | Iter 2 |
| 10. SOLID OO design principles, design patterns | |
| 11. Continuous integration/deployment, performance & availability, upgrades & feature flags, optimization, security/data integrity | Iter. 3 |
| 12-14. Optional extra topics, guest speakers | Iter. 4 |

# 2-week Agile/XP Iteration

Talk to customer




Lo-fi UI mockup



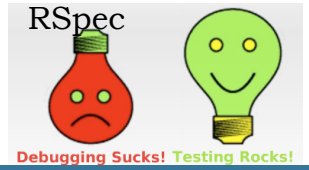

User stories & scenarios

Legacy Code

Design patterns

Cucumber

Behavior-driven Design / user stories




Test-first dev. (unit/funct.)

PIVOTAL TRACKER

Measure Velocity

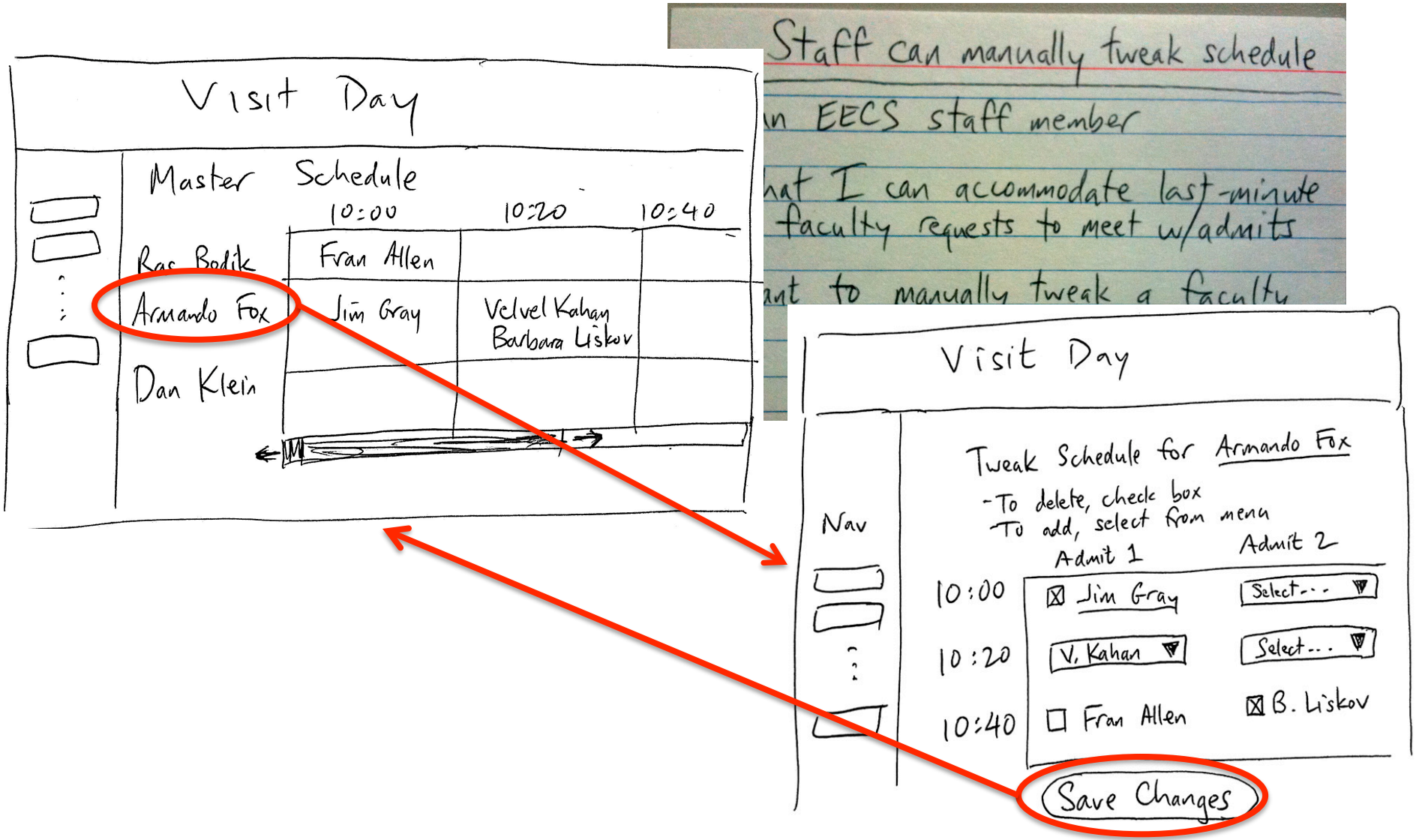
heroku

Deploy

# Methodologies …become Tools

| | |
|---|---|
| • Software arch., design pat-terns, coding practices | • Ruby & Rails |
| • Test-first development, unit testing | • RSpec |
| • Behavior-driven design, integration testing | • Cucumber |
| • Agile, iteration-based project management | • Pivotal Tracker |
| • Version management & collaboration skills | • Git & Github |
| • SaaS technologies, deployment & operations | • Cloud computing: EC2, Heroku |

# Example: Behavior-driven Design from Lo-fi Mockup



# Reaching agreement with customer via User Stories

```
Feature: staff can add admit to meeting with open slot
  As an EECS staff member
  So that I can accommodate last-minute requests
  I want to manually tweak a faculty member's schedule

Scenario: add an admit to a meeting with an open slot
  Given "Velvel Kahan" is available at 10:20
  When I select "Velvel Kahan" from the menu for the 10:20
       meeting with "Armando Fox"
  And I press "Save Changes"
  Then I should be on the master meetings page
  And I should see "Velvel Kahan added to 10:20AM meeting."
  And "Armando Fox" should have a meeting with "Velvel
       Kahan" at 10:20

Scenario: remove admit from meeting
```

*...etc.*


# From user stories to acceptance tests

- Runs "natural language" user stories as integration tests
- Each *scenario* describes one user story
  - *Given* steps: setup preconditions
  - *When* steps: take actions, using built-in browser simulator or *Selenium*
  - *Then* steps: assertions to check post-conditions
- *Step definitions* match story steps to code
- Quantify correctness and coverage


# Measuring & Estimating Progress

- Assign 1-3 *points* to each story in advance
  - 1= straightforward stories (1-2 hours)
  - 2 = medium stories (~1/2 day)
  - 3 = complex (~1-1.5 days)
  - >3 = you don't really know, so subdivide it
- Teams assign value: vote & discuss discrepancies
- Velocity = average number of points/iteration
  - How many stories will team finish during this iteration?
  - How long will it take to complete a set of features?
  - Students graded on improving ability to estimate

# PivotalTracker.com

PIVOTAL TRACKER

Tracker has some new updates - read more!

Welcome, David Patterson

PROJECTS ▾ DASHBOARD REPORTS PROFILE ACCOUNT HELP SIGN OUT

## Visit Day Meeting Scheduler

velocity 10

CURRENT BACKLOG ICEBOX DONE MORE ▾ PROJECT ▾ STORIES ▾

**DONE** displaying last 12 iterations (show all)

| | | |
|---|---|---|
| 41 | 3 Oct | Pts: 0 |
| 42 | 10 Oct | Pts: 0 |
| 43 | 17 Oct | Pts: 0 |
| 44 | 24 Oct | Pts: 8 |
| 45 | 31 Oct | Pts: 2 |
| 46 | 7 Nov | Pts: 1 |
| 47 | 14 Nov | Pts: 8 |
| 48 | 21 Nov | Pts: 15 |
| 49 | 28 Nov | Pts: 0 |
| 50 | 5 Dec | Pts: 0 |
| 51 | 12 Dec | Pts: 0 |
| 52 | 19 Dec | Pts: 0 |

**CURRENT**

• 53 | 26 Dec - Current — Pts: 0 of 8

- Replace relevant parts of view layer with spine.js (VC) — Finish
- Standardize "data field" module interface (VC) — Finish
- Administrator can invite people to sign up as Visitors. — Start

**ICEBOX** Select All

- algorithm Algorithm should try fitting an n-1 (down to 1) slot meeting when an n-slot meeting cannot be arranged (BM) — Start
- Can quickly view (modal window) ranking/availability from any of the schedule views. (VC) — Start
- Faculty can cap total number of admits. (VC) — Start
- Faculty can cap number of meetings. (VC) — Start
- Staff can impose global cap on number of meetings per admit. (VC) — Start
- Standardize the Room field. — Start
- Staff can add comments to individual/multiple schedules. — Start
- Can save and restore schedules. — Start
- Admit menus on Tweak Sched page should only show available admits — Start
- Meeting schedule generation can run as background process instead of as a page request — Start
- Admits/Faculty with unsatisfied rankings due to nonattendance are notified via automatic comment. — Start
- Algorithm should prefer to pair admits with the same number of slots (avoid awkward pairings). — Start
- Algorithm should prefer to spread meetings rather than pack them. — Start
- Only top-ranked rankings should be allowed to be flagged as mandatory. — Start
- Default Room is automatically filled with office (requires new LDAP application). — Start

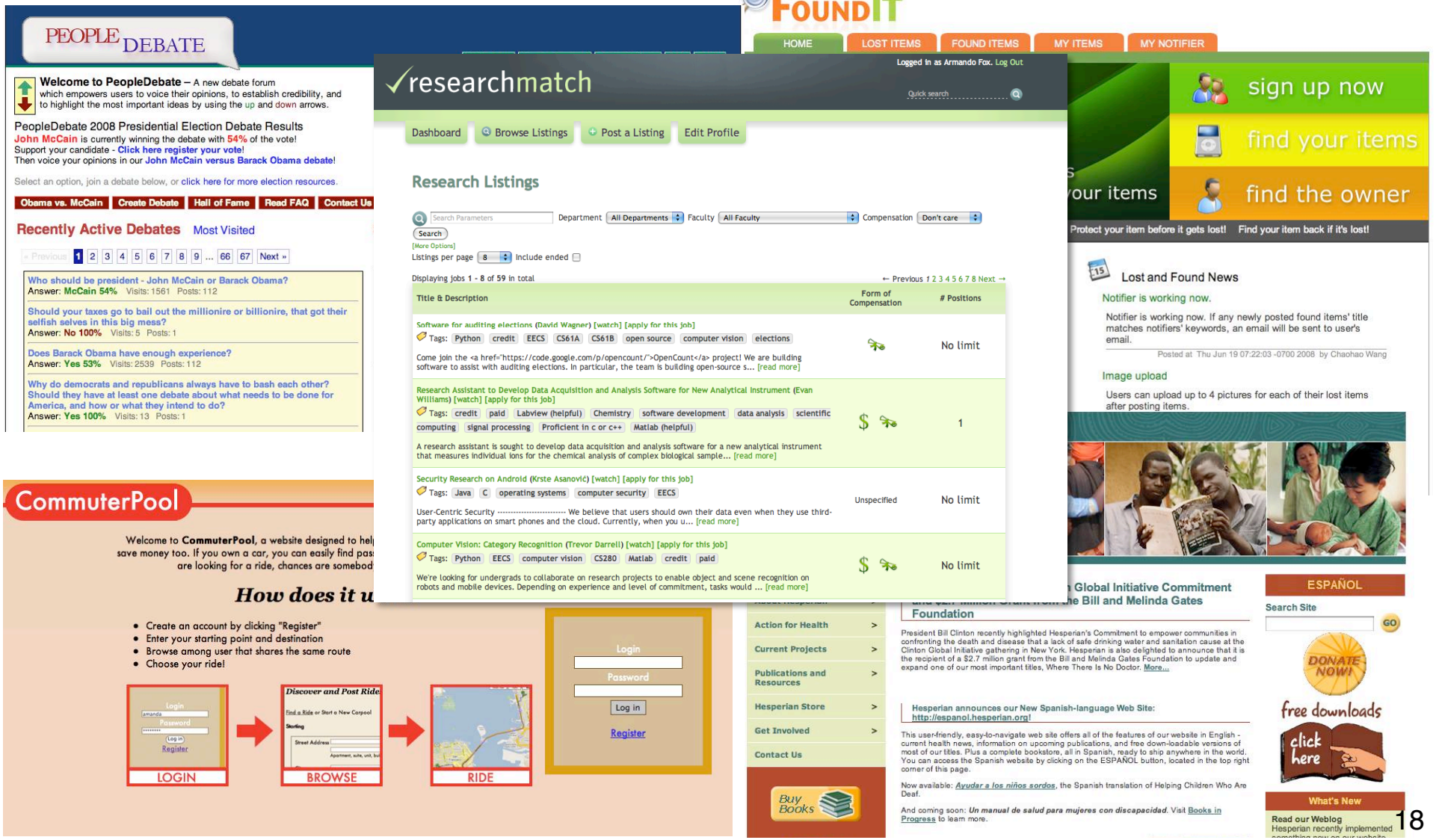

# Methodologies ➔ Tools

- Students can more easily follow our advice (methodologies)
- Instructors can more easily grade
- Per-iteration progress can be quantified
- Students get feedback on how realistic their estimates are
- All these tools are free, some are hosted

# Results/Observations

- Course popularity: 35 – 50 – 75 – 110 – 165 – 225 (F'13 est.)
- Customer feedback (F'12)
  - 92% customers "happy" or "thrilled"
  - 48% customers tried to hire students to continue work
  - 67% students intend to maintain app regardless
- Students appear to engage in process!
  - Stories became more uniform in complexity & size in later iterations
  - Projects varied in code *quantity* but rarely *quality*
- 60% students believe we should do everything possible to enroll more students to course

# Success stories with Bay Area nonprofits



# This talk

- How did retooling to Agile+SaaS affect the course & students?
- If successful, can course be *scaled up* (teach more students) and *scaled out* (used flexibly at other institutions)?
- Does course meet new Software Engineering curriculum guidelines? (*cs2013.org*)

# What's a MOOC?

- Video lectures
- Self-check questions
- Online quizzes and homework assignments that are *machine graded*
- Discussion forums monitored by TAs
- Synchronous deadlines
- Berkeley has decided to make MOOCs tuition-free and non-credit

# Adapting for a MOOC

- Nontrivial autograders for programming assignments (open source)
- Adapting lectures to 7-10 min segment + peer learning/self assessment question
  - 7-10 min segment + peer learning question
  - 8-10 hrs/week ugrad to convert & format videos
- No design project in MOOC!
- Same HWs, quizzes, deadlines
- Offered 3 times on Coursera, 3 times on EdX, plus new "part II" now on EdX

# Autograding Strategies

| Assignment type | Grading strategy |
|---|---|
| Write code | · RSpec (correctness)<br>·[soon] `reek/flay` (code style)<br>·[soon] `CodeClimate.org` (metrics) |
| Write test cases (unit, functional, or user stories) | · Mutation testing (Amman & Offutt): app with inserted bugs should cause some tests to fail |
| Enhance legacy SaaS app (deploy on Heroku) | · Remote (cloud-based) integration test using `Mechanize`<br>· C0, happy path, sad paths coverage |
| Interactive short-answer/multiple-choice | · Our tools emit both printed & online-format (XML) quizzes<br>· [soon] open-ended short-essay |

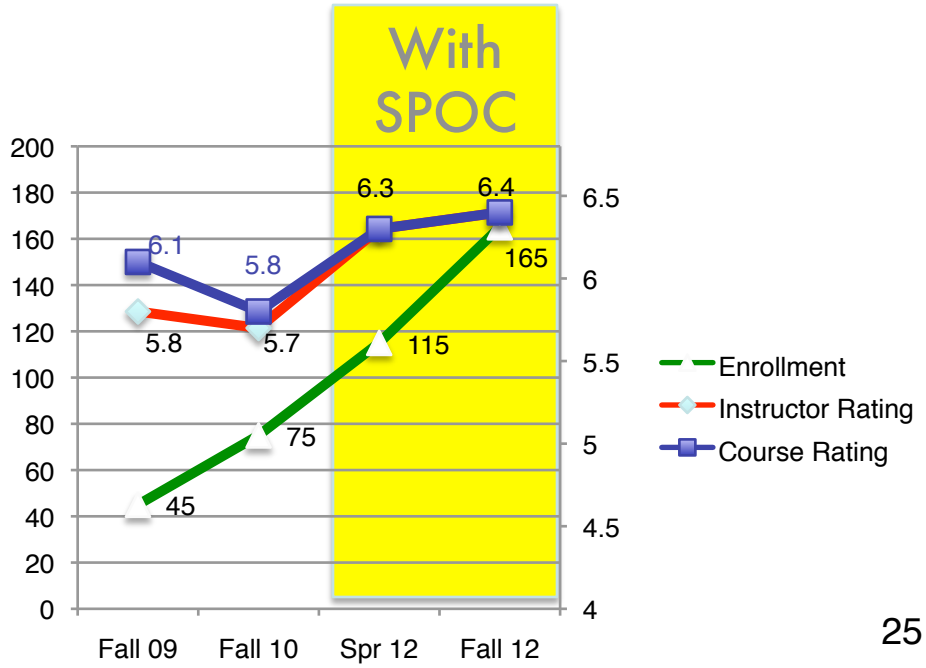
# What role can MOOCs play in software education?

Myth :
Universities will use MOOCs to save money by firing faculty & TAs, sacrificing education quality.

**Reality**: MOOCs can instead save money by improving throughput and *increasing* education quality.

# Classroom + MOOC = SPOC (Small Private Online Course)

- Accommodate increased demand (now admit juniors, vs. turning away graduating seniors)
- Autograders improve TA leverage, fulfill student request for more practice➔stronger design projects
- Course ratings up despite larger size
- ~800 instructors passed MOOC; 8 now using our SPOC & book
- F'13: >200 students

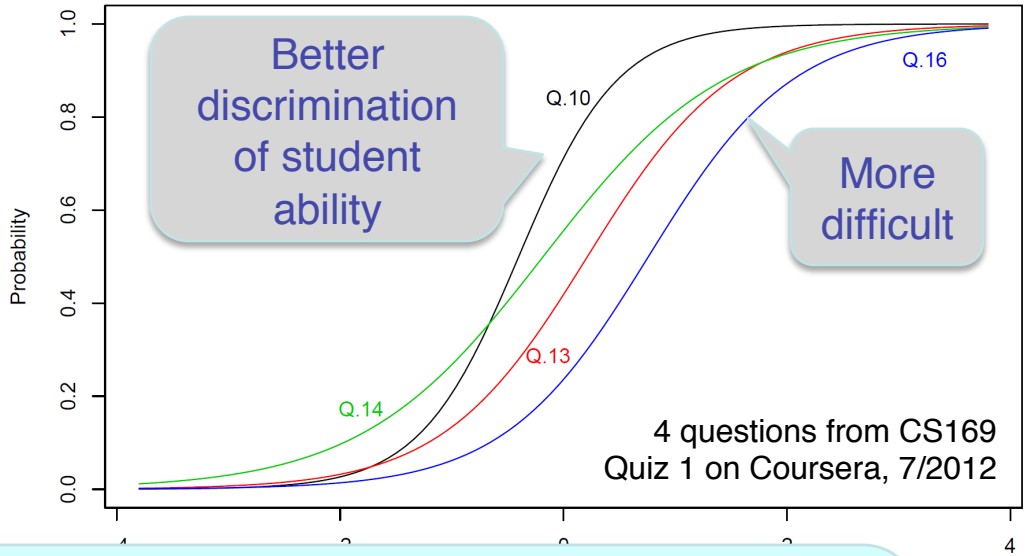


Myth:
MOOCs distract faculty from focusing on improving their on-campus teaching.

**Reality**: MOOCs can help to improve on-campus courses.

# Scale can accelerate education innovation

- **Item response theory**
  Predicts probability that a student of a given ability will answer a given question correctly
- Do
- Ca




**Large # of students reduces standard error of question difficulty & discrimination model by 3x-10x.**

* Frederic M. Lord, *Statistical Theories of Mental Test Scores* (1968) and *Applications of Item Response Theory to Practical Testing Problems* (1980)

# This talk

- How did retooling to Agile+SaaS affect the course & students?
- If successful, can course be *scaled up* (teach more students) and *scaled out* (used flexibly at other institutions)?
- Does course meet new Software Engineering curriculum guidelines? (*cs2013.org*)

# ACM/IEEE 2013 SW Engineering Curriculum

"In general, students learn best at the application level much of the material… by participating in a project. Such projects should require students to work on a team to develop a software system through as much of its lifecycle as is possible. Much of software engineering is devoted to effective communication among team members and stakeholders. …

*While organizing and running effective projects within the academic framework can be challenging, the best way to learn to apply software engineering theory and knowledge is in the practical environment of a project.*"

from Iron Man draft 1.0, cs2013.org

# Checklist: "Yes"➔plan & document, "No"➔agile*

| | |
|---|---|
| 1 | Is a specification required? |
| 2 | Are customers unavailable? |
| 3 | Is the system to be built large? |
| 4 | Is the system to be built complex (e.g., real time)? |
| 5 | Will it have a long product lifetime? |
| 6 | Are you using poor software tools? |
| 7 | Is the project team geographically distributed? |
| 8 | Is team part of a documentation-oriented culture? |
| 9 | Does the team have poor programming skills? |
| 10 | Is the system to be built subject to regulation? |

- For class project, Agile seems appropriate unless building safety-critical system or using bad tools

* R. Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed., McGraw-Hill, 2010

# Is new curriculum standard "Agile-friendly"?

- "agile" appears only twice in 50K+ words document
- Only 2 topics use Agile terminology
- Zero learning outcomes described in Agile terms

*If not, what should instructors do?*

- Follow outcomes, ignore advice to do projects?
- Follow outcomes, ignore advice to do Agile project?
- Ignore outcomes, follow advice to do Agile project?

# ACM/IEEE "Iron Man" draft 1.0 of SDF & SE curriculum guidelines

- Types of learning outcomes (116 outcomes total)
  - Core-Tier 1: must cover 100% (13)
  - Core-Tier 2: must cover 80% (50)
  - Electives (53)
- Depth of coverage for each outcome
  - Familiarity: "what do you know about this?" (53)
  - Usage: "what do you know how to do?" (58)
  - Competence: "why would you do that?" (5)

# Example outcomes

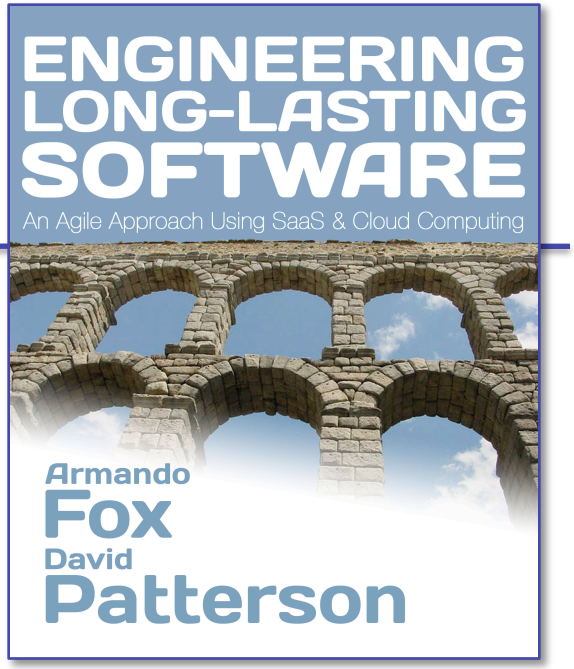
- Identify common coding errors that lead to insecure programs (e.g., buffer overflows, memory leaks, malicious code) and apply strategies for avoiding such errors. [Usage] [Core-Tier 1]
- Describe different categories of risk in software systems. [Familiarity] [Core Tier 2]
- Use a common, non-formal method to model and specify (in the form of a requirements specification document) the requirements for a medium-size software system [Usage] [Elective]

# Is CS 2013 "Agile-friendly"?

- Some topics can be "mapped" to Agile equivalents
  - User stories ➔ requirements elicitation
  - Stories + mockups + customer meeting notes/interviews ➔requirements documentation
  - Cucumber scenarios ➔integration/system testing
- Some Plan & Document processes can be covered in project management
  - Planning & estimation; code reviews
- Beta edition of textbook revised to expand "Plan & Document perspective" while focusing on Agile

# Our results: 100% CT1, 94% CT2

| | Section Title | Core-Tier1 Number | Core-Tier1 % Covered | Core-Tier2 Number | Core-Tier2 % Covered | Electives Number | Electives % Covered | Total |
|---|---|---|---|---|---|---|---|---|
| 1 | Software Processes | 5 | 100% | 2 | 100% | 7 | 43% | 14 |
| | Software Project | | | | | | 94% | |
| 2 | Management | 0 | -- | 9 | 100% | 16 | | 25 |
| 3 | Tools and Environments | 0 | -- | 4 | 100% | 0 | -- | 4 |
| 4 | Requirements Engineering | 3 | 100% | 3 | 100% | 5 | 80% | 11 |
| 5 | Software Design | 5 | 100% | 9 | 78% | 6 | 67% | 20 |
| 6 | Software Construction | 0 | -- | 7 | 86% | 3 | 67% | 10 |
| | Software Verification | | | | | | 43% | |
| 7 | Validation | 0 | -- | 7 | 86% | 7 | | 14 |
| 8 | Software Evolution | 0 | -- | 6 | 100% | 0 | -- | 6 |
| 9 | Formal Methods | 0 | -- | 0 | -- | 5 | 60% | 5 |
| 10 | Software Reliability | 0 | -- | 3 | 67% | 4 | 25% | 7 |
| | TOTAL | 13 | | 50 | | 53 | | 116 |




- Details in downloadable Instructors Manual at beta.saasbook.info
- Exemplar online & handout at ICSE 2013 (*Strawberry Canyon LLC*)

beta.saasbook.info/icse2013

$10 discount

# Summary

- Agile-focused courses *can* fulfill CS 2013 curriculum guidelines for SE
  - More Agile presence in curriculum would be nice
- MOOCs & SPOCs augment book, increase instructor leverage, reuse good materials
- Looking for additional beta testing
  - SPOC for use in your classroom
  - Inexpensive book/ebook that matches SPOC & fulfills CS 2013 if used according to our schema

beta.saasbook.info/icse2013

$10 discount


# Thanks!

Acknowledgments: David Patterson, staff of UC Berkeley CS 169, support staff for EdX CS 169.1x/169.2x