

# Engineering Software as a Service An Agile Software Approach

ACM Webinar

David Patterson  
University of California, Berkeley  
*May 8, 2013*



Association for  
Computing Machinery

Advancing Computing as a Science & Profession

© 2013 Armando Fox & David Patterson

Licensed under [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](#)



# ACM Learning Center

<http://learning.acm.org>

- 1,300+ trusted technical **books and videos** by leading publishers including O'Reilly, Morgan Kaufmann, others
- **Online courses** with assessments and certification-track mentoring, member discounts on tuition at partner institutions
- **Learning Webinars** on big topics (Cloud Computing/Mobile Development, Cybersecurity, Big Data, Recommender Systems, SaaS)
- **ACM Tech Packs** on big current computing topics: Annotated Bibliographies compiled by subject experts
- Popular video tutorials/keynotes from ACM Digital Library, A.M. Turing Centenary talks/panels
- Podcasts with industry leaders/award winners

# Talk Back

- Use the **Facebook** widget in the bottom panel to share this presentation with friends and colleagues
- Use **Twitter** widget to Tweet your favorite quotes from today's presentation with hashtag **#ACMWebinarPatterson**
- Submit questions and comments via Twitter to **@acmeducation** – we're reading them!

# Outline

- Software as a Service (SaaS)
- Development process: Waterfall v. Agile
- Book and Online Course
- Pair Programming and Scrum
- Points, Velocity, & Pivotal Tracker
- Lo-Fi UI Sketches & Storyboards
- Behavior-Driven Development & User Stories
- Test-Driven Design, Cucumber, & RSpec
- Summary

# Software as a Service (SaaS)



*(Engineering Software as a Service § 1.2)*

# Software as a Service: SaaS

- Traditional SW: binary code installed and runs wholly on client device
- SaaS delivers SW & data as service over Internet via thin program (e.g., browser) running on client device
  - Search, social networking, video
- Now also SaaS version of traditional SW
  - E.g., Microsoft Office 365, TurboTax Online
- SaaS is revolutionizing software industry



# 6 Reasons for SaaS

1. No install worries about HW capability, OS
2. No worries about data loss (at remote site)
3. Easy for groups to interact with same data
4. If data is large or changed frequently, simpler to keep 1 copy at central site
5. 1 copy of SW, controlled HW environment  
=> no compatibility hassles for developers
6. 1 copy => simplifies upgrades for developers *and* no user upgrade requests

# Development processes: Waterfall vs. Agile



*(Engineering Software as a Service  
§ 1.8- § 1.9)*



# “Plan-and-Document” Processes: Waterfall

- Waterfall “**lifecycle**” or development process
  - Requirements analysis and specification
- 1. Architectural design
- 2. Implementation and Integration
- 3. Verification
- 4. Operation and Maintenance
- Complete one phase before start next one
  - Why? Earlier catch bug, cheaper it is
  - Extensive documentation/phase for new people

# How well does Waterfall work?

- Works well for SW with specs that won't change: NASA spacecraft, aircraft control, ...
- Often when customer sees it, wants changes
- Often after built first one, developers learn right way they should have built it



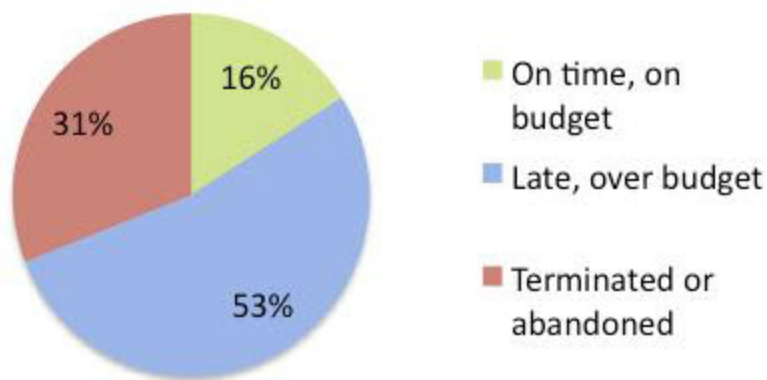
# (Many) Other Processes

- Spiral
  - Add prototypes with 6 to 24 month iterations
  - Rather than document all requirements 1st, develop requirement documents across the iterations of prototype
- Rational Unified Process (RUP)
  - More closely allied to business issues
  - Iteration in each phase: Inception, Elaboration, Construction, Transition

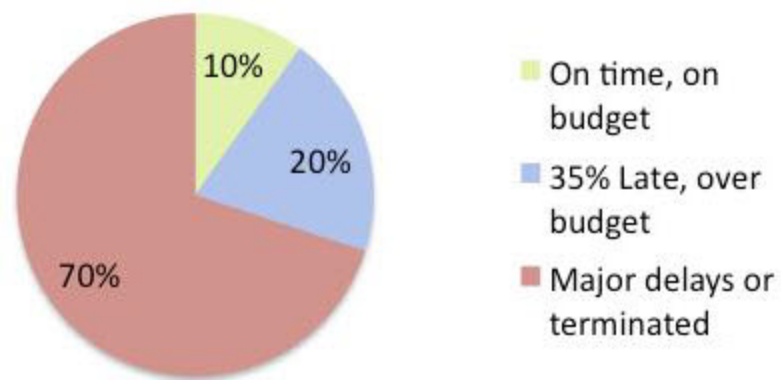


# State of SW Development

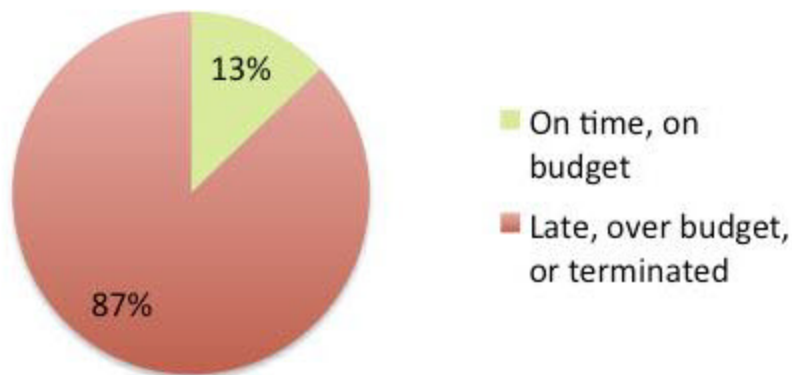
**Software Projects (Johnson 1995)**



**Software Projects (Jones 2004)**



**Software Projects (Taylor 2000)**



# Agile Manifesto, 2001

“We are uncovering better ways of developing SW by doing it and helping others do it. Through this work we have come to value

- Individuals and interactions over processes & tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.”

# Extreme Programming (XP)

Takes common sense principles to extreme:

- If short iterations are good, make them as short as possible: days vs. months
- If simplicity is good, always do the simplest thing that could possibly work
- If testing is good, test all the time. Write the test code before you write the code to test
- If code reviews are good, review code continuously, by programming in pairs, 2 programmers to a computer

# Agile lifecycle

- Embraces change as a fact of life: continuous improvement vs. phases
- Developers continuously refine working but incomplete prototype until customers happy, with customer feedback on each **Iteration** (every ~2 weeks)
- Agile emphasizes **Test-Driven Development (TDD)** to reduce mistakes, written down **User Stories** to validate customer requirements, **Velocity** to measure progress



# Implications of SaaS

- Rapid Improvement Opportunity => Rapid Improvement Obligation
  - Alta Vista used to be most popular search engine
  - MySpace used to be most popular social networking site
- Need software development process that embraces can evolve software rapidly
- => Agile software development

# Massive Open Online Course

- Berkeley has decided to make MOOCs tuition-free and non-credit
  - Search for CS169.1x and CS169.2x
- 7-10 minute "lecturelets"
- Self-check questions
- Online quizzes and homework assignments that are *machine graded*
- Discussion forums monitored by TAs
- Synchronous deadlines

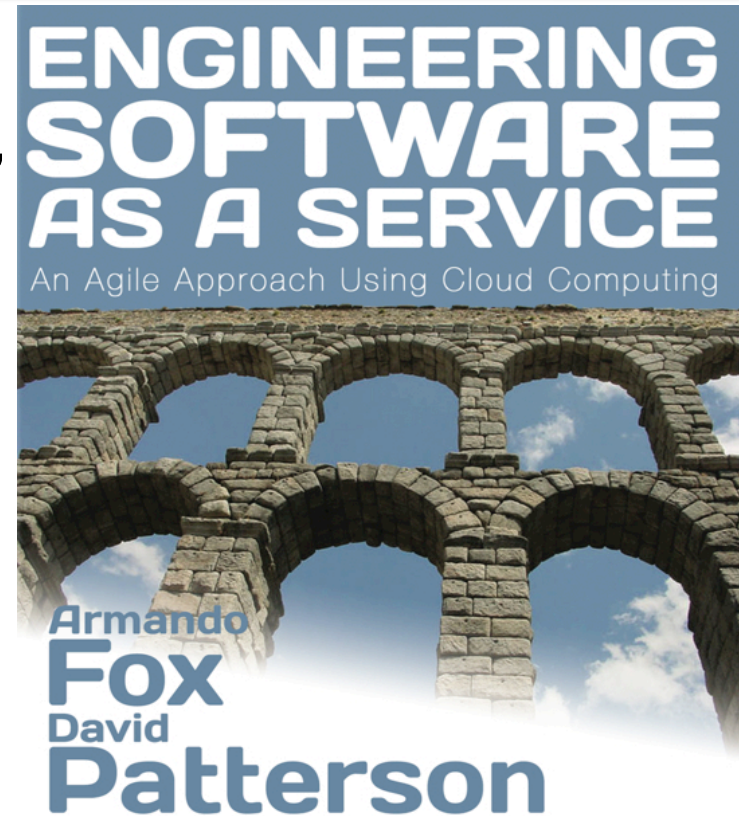


Part I covers Software as a Service (SaaS)

- SaaS Architecture: Model-View-Controller, 3-Tier architecture, & RESTful APIs
- Introduction to Ruby on Rails framework
- Introduction to JavaScript, jQuery, AJAX

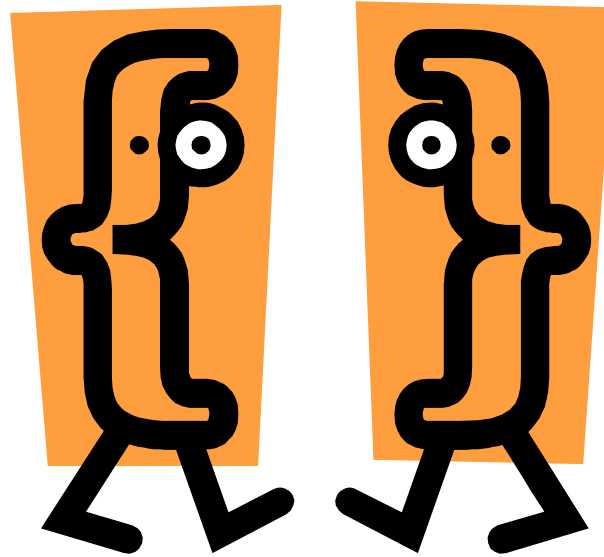
Part II covers Agile development

- Eliciting requirements via User Stories
- BDD to convert User Stories into acceptance tests using Cucumber
- TDD to transform acceptance tests into unit tests using RSpec
- Schedule via Velocity & Pivotal Tracker
- Organizing SaaS teams using Scrum and Pair Programming
- Incorporating SaaS Design Patterns



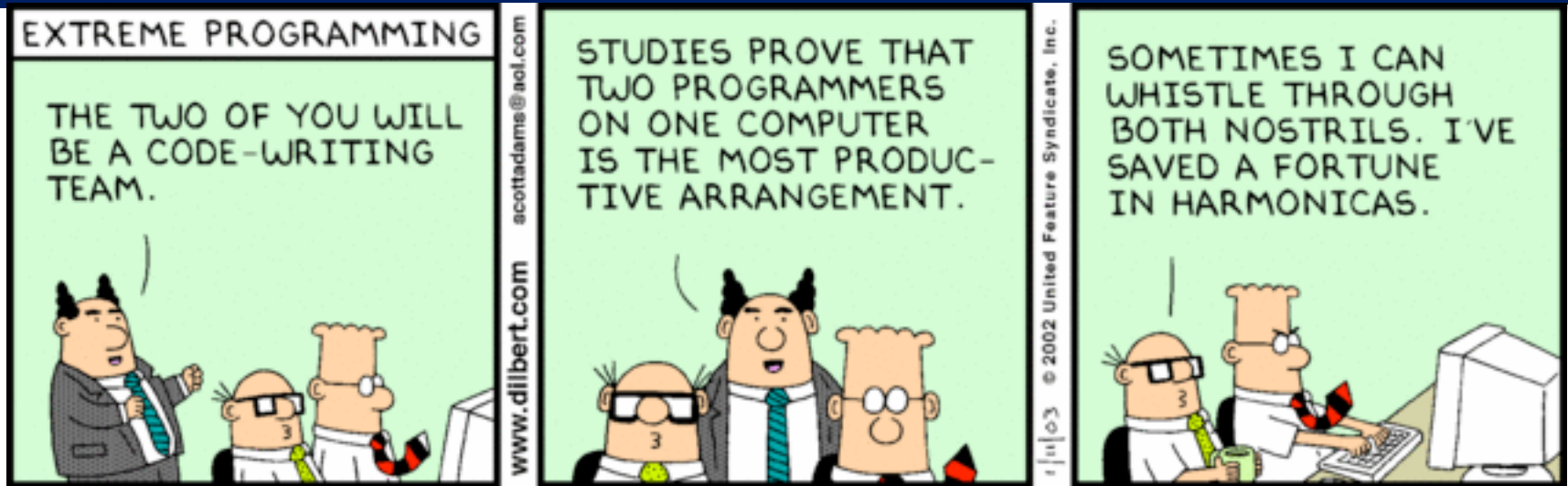
<http://saasbook.info>

# Pair Programming



*(Engineering Software as a Service § 10.2)*

# Pair Programming



- Goal: improve software quality, reduce time to completion by having 2 people develop the same code
- Non-goal: reduce total cost (programmer-hours) of developing code



# Pair Programming



- Sit side-by-side facing screens together,
  - Not personal computer; many for any pair to use
  - To avoid distractions, no email reader, browser

# Pair Programming

- **Driver** enters code and thinks tactically about how to complete the current task, explaining thoughts while typing
- **Observer** reviews each line of code as typed in, and acts as safety net for the driver
- **Observer** thinking strategically about future problems, makes suggestions to driver
- Should be lots of talking
- Pair alternates roles



# Pair Programming Evaluation

- PP **quicker** when task complexity is low
- PP yields **higher quality** when high
- More effort than solo programmers?
- Also transfers knowledge between pair
  - programming idioms, tool tricks, company processes, latest technologies, ...
  - Some teams purposely swap *partners* per task  
=> eventually everyone is paired (“promiscuous pairing”)

# Team Organization



*(Engineering SaaS § 10.1)*

# Scrum: Team Organization



- “2 Pizza” team size (4 to 9 people)
- Name inspired by frequent short meetings
  - 15 minutes every day at same place and time
  - To learn more: *Agile Software Development with Scrum* by Schwaber & Beedle

# Daily Scrum Agenda



- Answers 3 questions at “daily scrums”:
  1. What have you done since yesterday?
  2. What are you planning to do today?
  3. Are there any impediments or stumbling blocks?

# Scrum roles

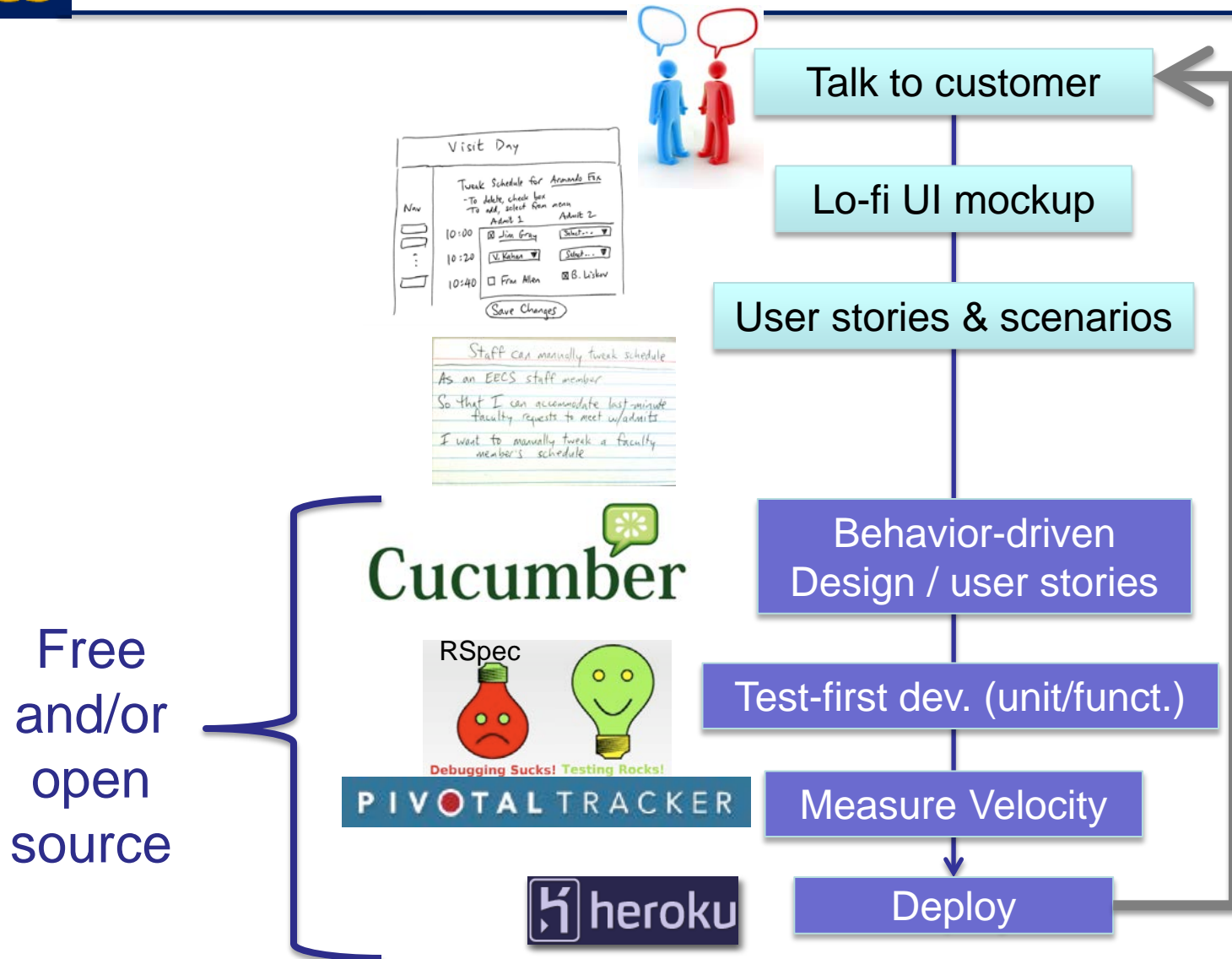
- **Team**: 2-pizza size team that delivers SW
- **ScrumMaster**: team member who acts as buffer between the Team and external distractions, keeps team focused on task at hand, enforces team rules, removes impediments that prevent team from making progress
- **Product Owner**: A team member (not the ScrumMaster) who represents the voice of the customer and prioritizes user stories

# Scrum Summary

- Basically, self-organizing small team with daily short standup meetings
- Work in “sprints” of 2-4 weeks
- Suggest member rotate through roles (especially Product Owner) each iteration



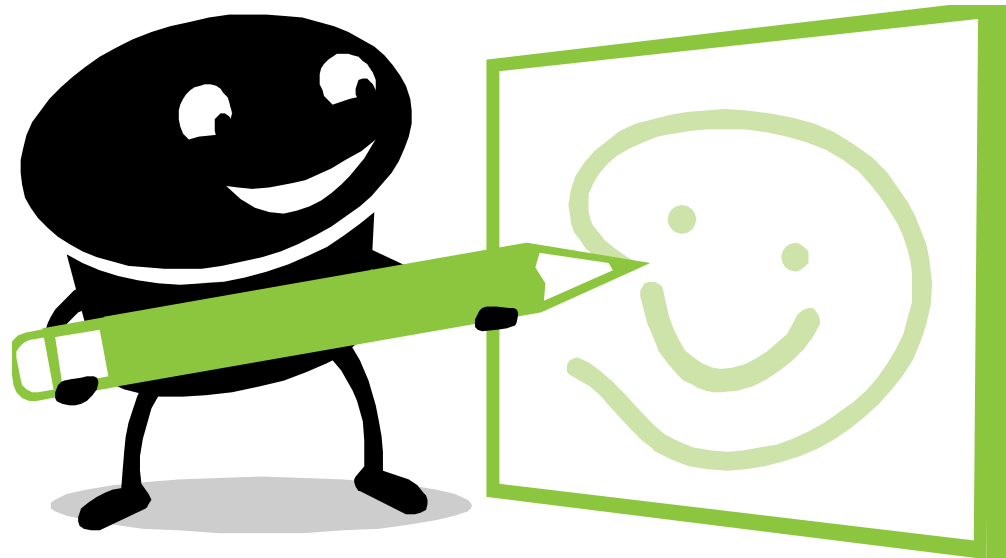
# 2-week Agile/XP Iteration





# Lo-Fi UI Sketches and Storyboards

*(Engineering SaaS § 7.7)*

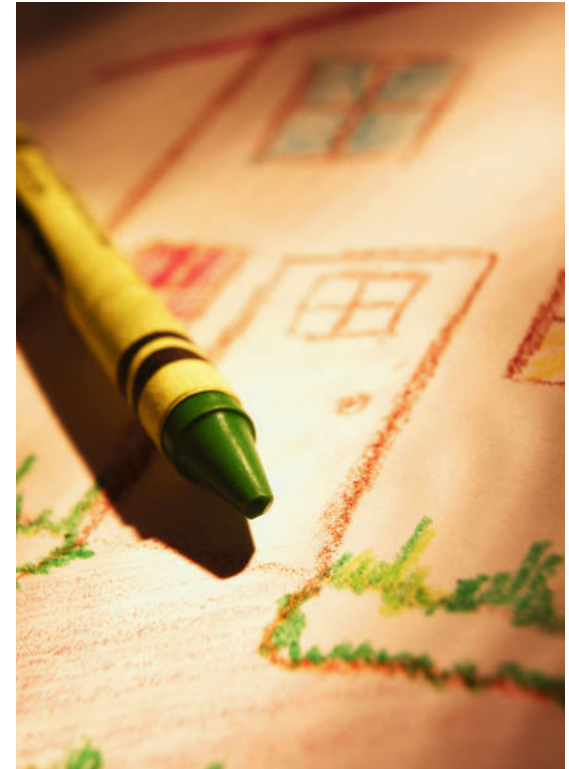


© 2012 David Patterson & David Patterson  
Licensed under [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](https://creativecommons.org/licenses/by-nc-sa/3.0/)



# SaaS User Interface Design

- SaaS apps often faces users  
⇒ User stories need User Interface (UI)
- Want *all* stakeholders involved in UI design
  - Don't want UI rejected!
- Need UI equivalent of 3x5 cards
- **Sketches**: pen and paper drawings or “**Lo-Fi UI**”



# Lo-Fi UI Example

ROTTEN POTATOES!

CREATE NEW MOVIE

MOVIE TITLE

MOVIE RATING

RELEASE DATE

MOVIE DESCRIPTION

SAVE CHANGES

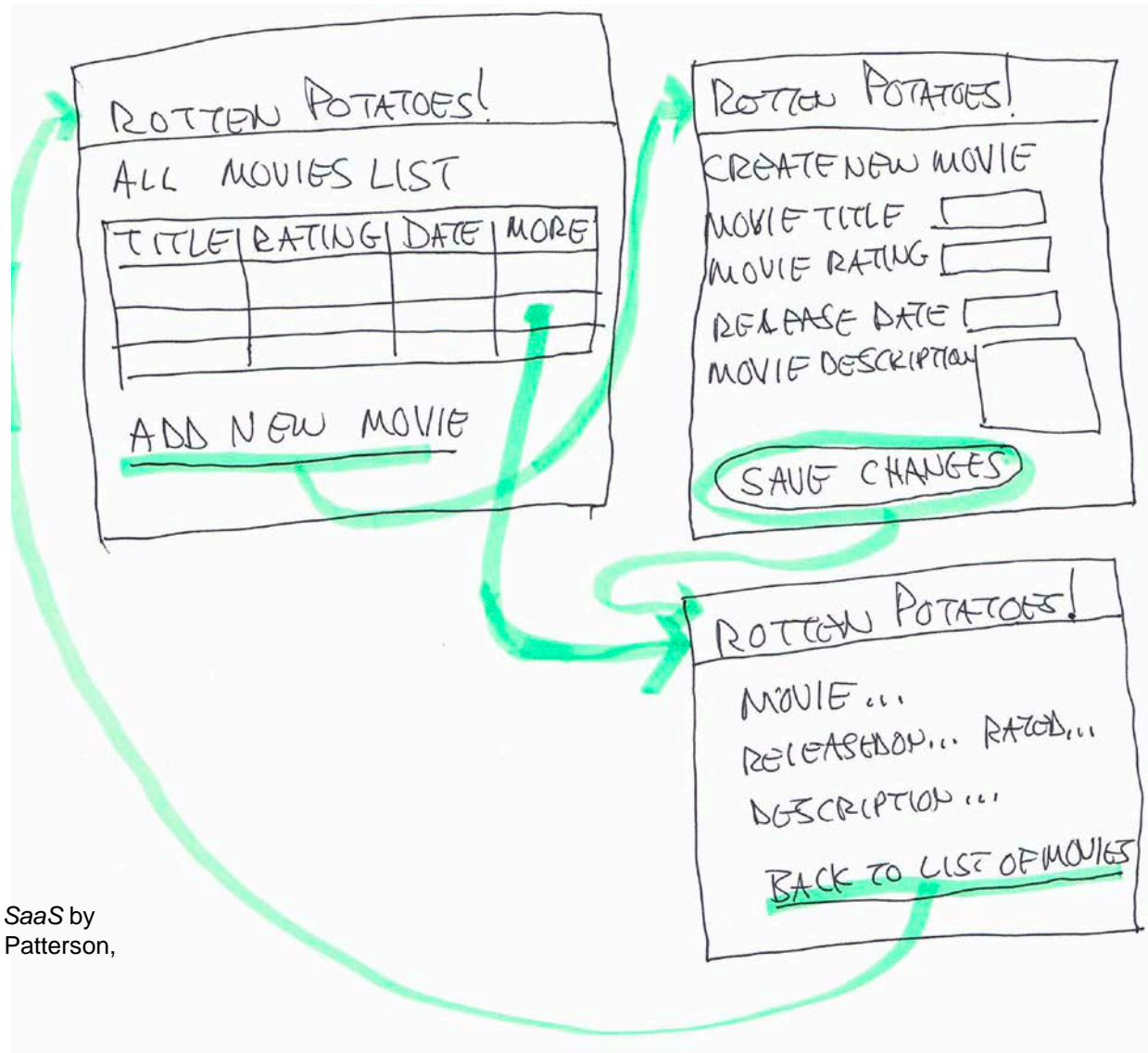
(Figure 4.3, *Engineering SaaS* by Armando Fox and David Patterson, Alpha edition, 2012.)

# Storyboards

- Need to show how UI changes based on user actions
- HCI => “storyboards”
- Like scenes in a movie
- But not linear



# Example Storyboard



(Figure 4.4, *Engineering SaaS* by Armando Fox and David Patterson, Alpha edition, 2012.)

# Behavior-Driven Design and User Stories (*Engineering SaaS § 7.7*)



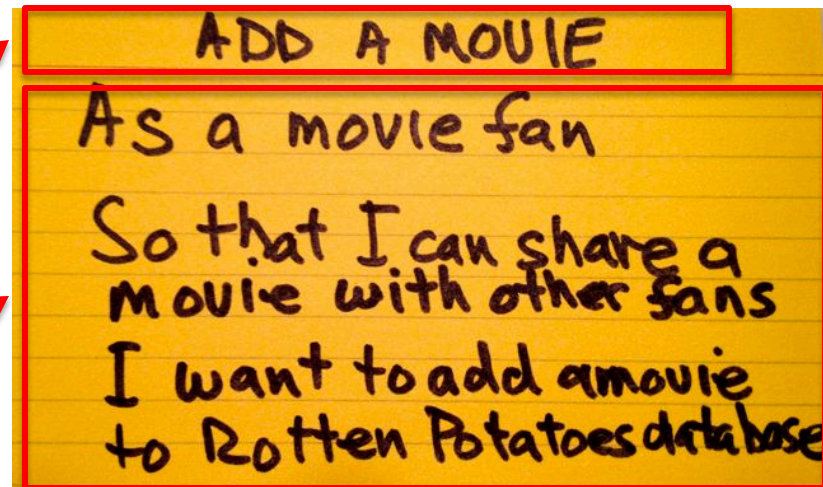
# Behavior-Driven Design (BDD)

- BDD asks questions about behavior of app *before and during development* to reduce miscommunication
- Requirements written down as *user stories*
  - Lightweight descriptions of how app used
- BDD concentrates on *behavior* of app vs. *implementation* of app
  - Test Driven Design tests implementation



# User Stories

- 1-3 sentences in everyday language
  - Fits on 3" x 5" index card
  - Written by/with customer
- “Connextra” format:
  - Feature name
  - **As a** [kind of stakeholder],  
**So that** [I can achieve some goal],  
**I want to** [do some task]
  - 3 phrases must be there, can be in any order
- Idea: user story can be formulated as *acceptance test before* code is written



# Why 3x5 Cards?

- (from User Interface community)
- Nonthreatening => all stakeholders participate in brainstorming
- Easy to rearrange => all stakeholders participate in prioritization
- Since short, easy to change during development
  - As often get new insights during development

# Different stakeholders may describe behavior differently

- *See which of my friends are going to a show*
  - As a theatergoer
  - So that I can enjoy the show with my friends
  - I want to see which of my Facebook friends are attending a given show

---

- *Show patron's Facebook friends*
  - As a box office manager
  - So that I can induce a patron to buy a ticket
  - I want to show her which of her Facebook friends are going to a given show

# Product Backlog

- Real systems have 100s of user stories
- *Backlog*: User Stories not yet completed
  - (We'll see Backlog again with Pivotal Tracker)
- Prioritize so most valuable items highest
- Organize so they match SW releases over time

# Points, Velocity, and Pivotal Tracker



*(Engineering SaaS § 7.3)*

# Measuring Productivity

- A measure of team productivity:  
calculate avg number of stories per week
- Some stories are much harder than others
- Rate each user story in advance on a simple integer scale: 1 for straightforward stories, 2 for medium stories, and 3 for very complex stories
- **Velocity**: average number of points per week

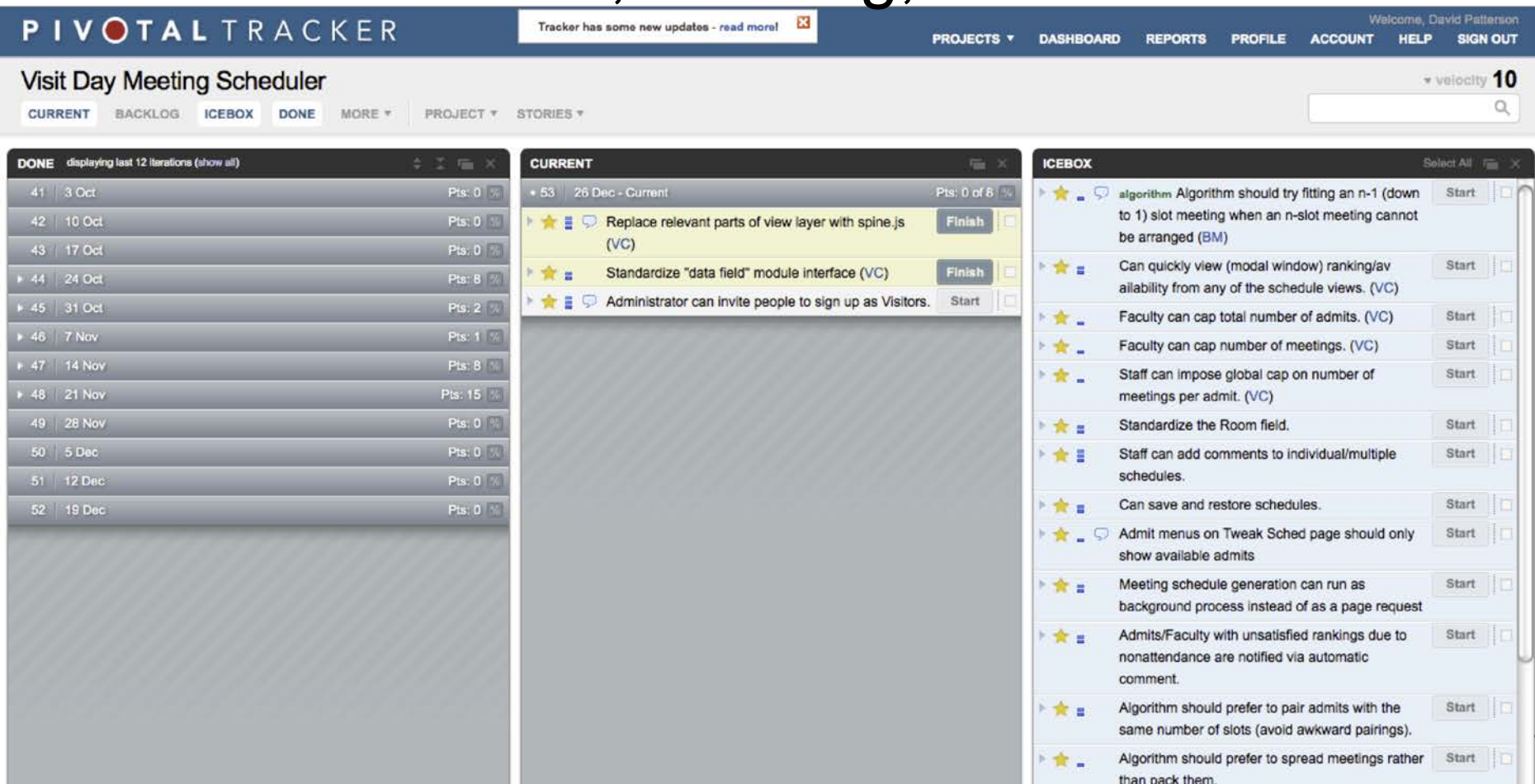
# More on Points

- Once get experience, Fibonacci scale is commonly used: 1, 2, 3, 5, 8
  - Each new number is sum of previous 2
  - At Pivotal Lab, 8 is extremely rare
- Teams assign value: vote by holding up fingers simultaneously, take average
  - If a big disagreement (2 and 5), discuss more
- $\geq 5 \Rightarrow$  user story should be broken up into simpler stories so that the backlog never has anything that's too demanding



# Pivotal Tracker

- Calculates velocity for team, manages user stories: Current, Backlog, Icebox



**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)

Iteration	Date	Points	Percentage
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](#)

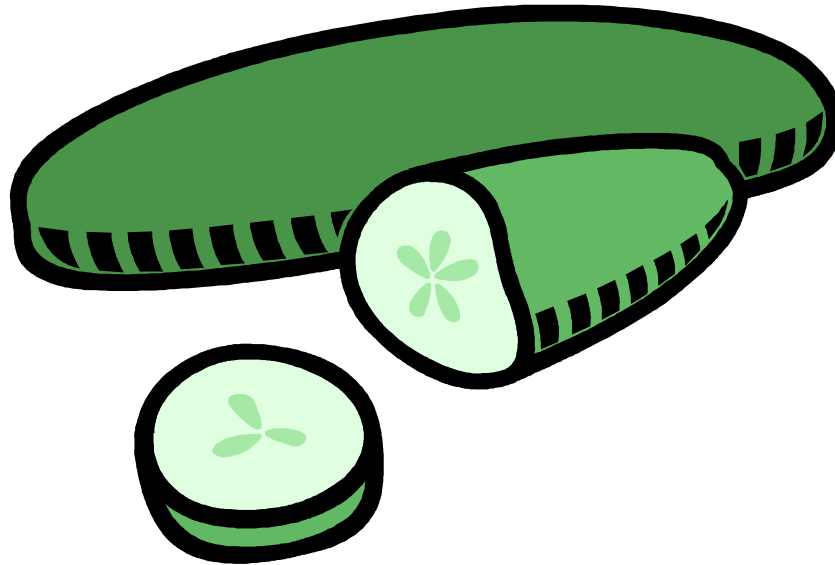
# Pivotal Tracker

- Prioritize user stories by where place them in Current, Backlog, Icebox
- Can add logical Release points, so can figure out when a Release will really happen
  - Remaining points/Velocity

# Tracker Roles

- Developers don't decide when user stories completed
  - Pushes Deliver button, which sends to “Product Owner”
- Product Owner tries out the user story and then either hits
  - Accept, which marks user story as done, or
  - Reject, which marks story as needing to be Restarted by developer

# Cucumber



*(Engineering SaaS § 7.5- § 7.6)*

# Cucumber: Big Idea

- Tests from customer-friendly user stories
  - Acceptance: ensure satisfied customer
  - Integration: ensure interfaces between modules consistent assumptions, communicate correctly.
- Cucumber meets halfway between customer and developer
  - User stories don't look like code, so clear to customer and can be used to reach agreement
  - Also aren't completely freeform, so can connect to real tests

# Example User Story

Feature: User can manually add movie 1 Feature

Scenario: Add a movie  $\geq 1$  Scenarios / Feature

Given I am on the RottenPotatoes home page  
When I follow "Add new movie"  
Then I should be on the Create New Movie page  
When I fill in "Title" with "Men In Black"  
And I select "PG-13" from "Rating"  
And I press "Save Changes"  
Then I should be on the RottenPotatoes home page  
And I should see "Men In Black"

3 to 8 Steps / Scenario

# Cucumber User Story, Feature, and Steps

- **User story:** refers to a single **feature**
- **Feature:** 1 or more **scenarios** that show different ways a feature is used
  - Keywords `Feature` and `Scenario` identify the respective components
- **Scenario:** 3 to 8 **steps** that describe scenario
- **Step definitions:** Code that tests steps
  - Usually many steps per step definition



# 5 Step Keywords

1. **Given** steps represent the state of the world before an event: preconditions
2. **When** steps represent the event (e.g., push a button)
3. **Then** steps represent the expected outcomes; check if its true
4. / 5. **And** and **But** extend the previous step

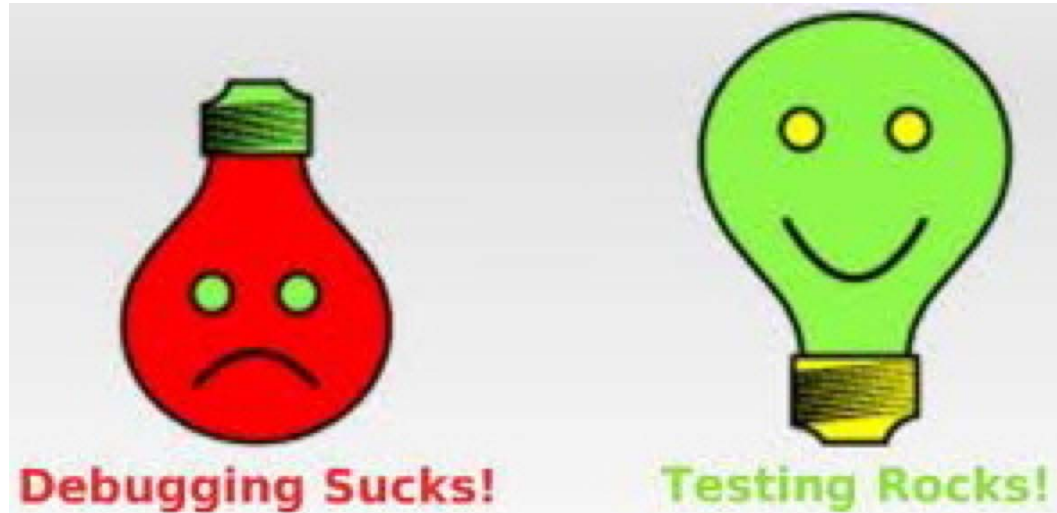
# Steps, Step Definitions, and Regular Expressions

- User stories kept in one set of files: **steps**
- Separate set of files has Ruby code that tests steps: **step definitions**
- Step definitions are like method definitions, steps of scenarios are like method calls
- How match steps with step definitions?
- **Regexes** match the English phrases in steps of scenarios to step definitions
  - `Given /^(?:|{ }I )am on (.+)\$/`
  - "I am on the Rotten Potatoes home page"

# Red-Yellow-Green Analysis

- Cucumber colors steps
- Green for passing
- Yellow for not yet implemented
- Red for failing  
(then following steps are Blue)
- Goal: Make all steps green for pass  
(Hence green vegetable for name of tool)
- Demo: <http://vimeo.com/34754747>

# TDD and RSpec



*(Engineering SaaS § 5.2)*

# Test-First development

- Think about one thing the code *should* do
- Capture that thought in a test, which fails
- Write the simplest possible code that lets the test pass
- Refactor: DRY out commonality w/other tests
- Continue with next thing code should do

**Red – Green – Refactor**

***Aim for “always have working code”***

# RSpec, a Domain-Specific Language for testing

- DSL: small programming language that simplifies one task at expense of generality
  - examples so far: migrations, regexes, SQL
- RSpec tests are called *specs*, and inhabit **spec** directory

`rails generate rspec:install` creates structure

`app/models/*.rb`

`spec/models/*_spec.rb`

`app/controllers/  
*_controller.rb`

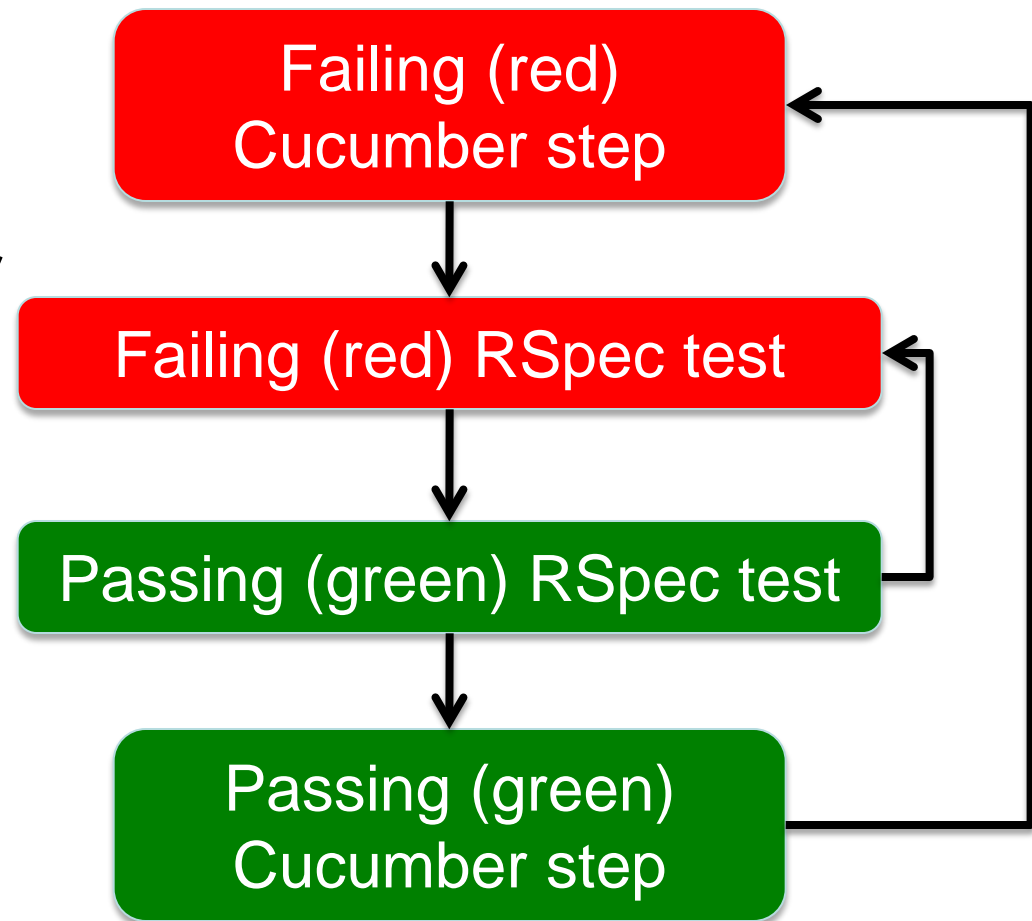
`spec/controllers/  
*_controller_spec.rb`

`app/views/**/*.html.html`

`(use Cucumber!)`

# Cucumber & RSpec

- Cucumber describes *behavior* via features & scenarios (*behavior driven design*)
- RSpec tests individual modules that contribute to those behaviors (*test driven development*)





# BDD+TDD: The Big Picture

- Behavior-driven design (BDD)
  - develop user stories to describe features
  - via Cucumber, user stories become *acceptance tests* and *integration tests*
- Test-driven development (TDD)
  - *step definitions* for new story, may require new code to be written
  - TDD says: write unit & functional tests for that code *first*, **before** the code itself
  - that is: write tests for *the code you wish you had*

# BDD Summary

---

- Doesn't feel natural at first
- Rails tools make it easier to follow BDD
- Once learned BDD and had success at it, no turning back
  - 2/3 Alumni said BDD/TDD useful in industry

# TDD vs. Conventional debugging

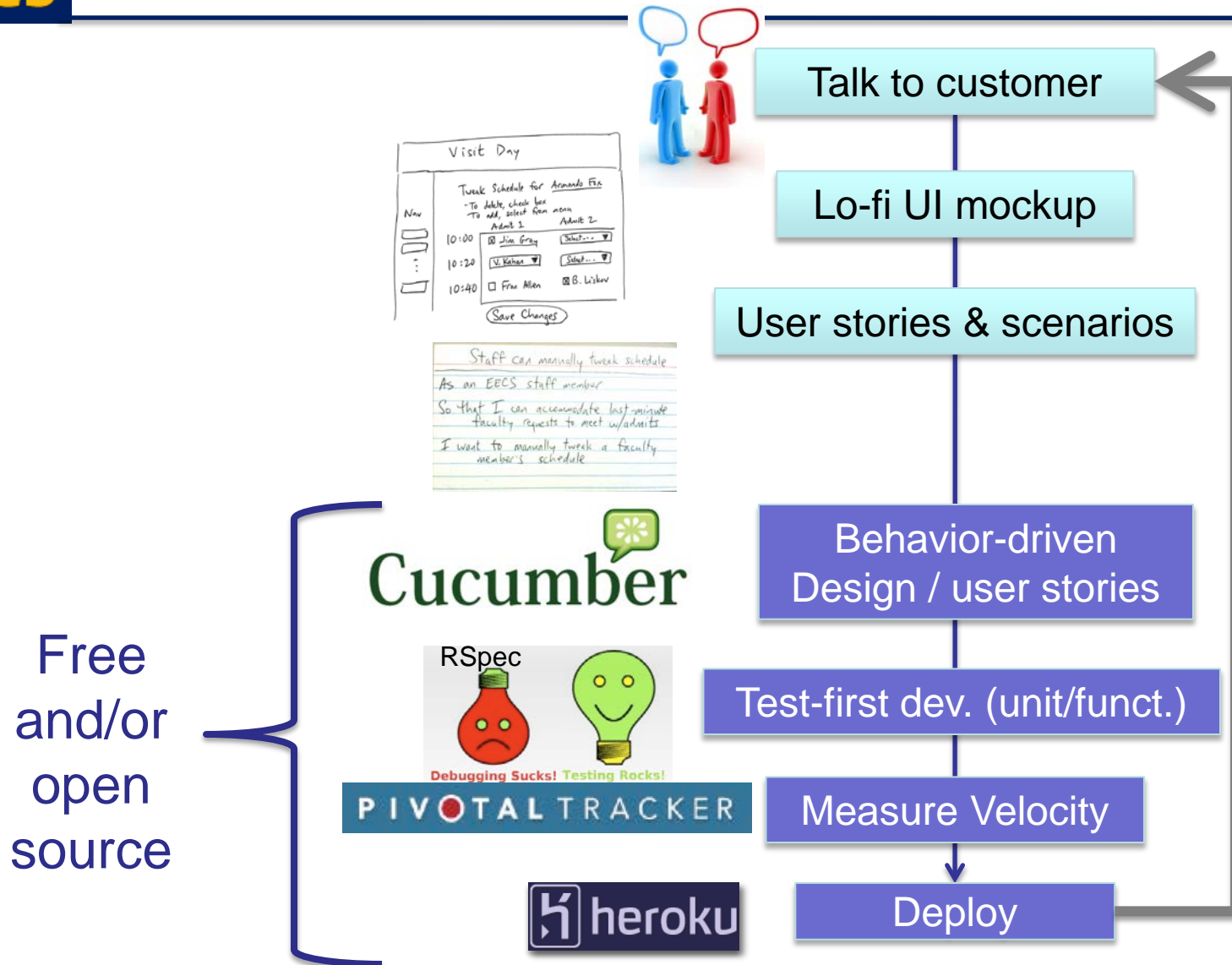
Conventional	TDD
Write 10s of lines, run, hit bug: break out debugger	Write a few lines, with test first; know immediately if broken
Insert printf's to print variables while running repeatedly	Test short pieces of code using expectations
Stop in debugger, tweak/set variables to control code path	Use mocks and stubs to control code path
Dammit, I thought for sure I fixed it, now have to do this all again	Re-run test automatically

- Lesson 1: TDD uses same skills & techniques as conventional debugging—but more productive
- Lesson 2: writing tests *before* code takes more time up-front, but often less time overall

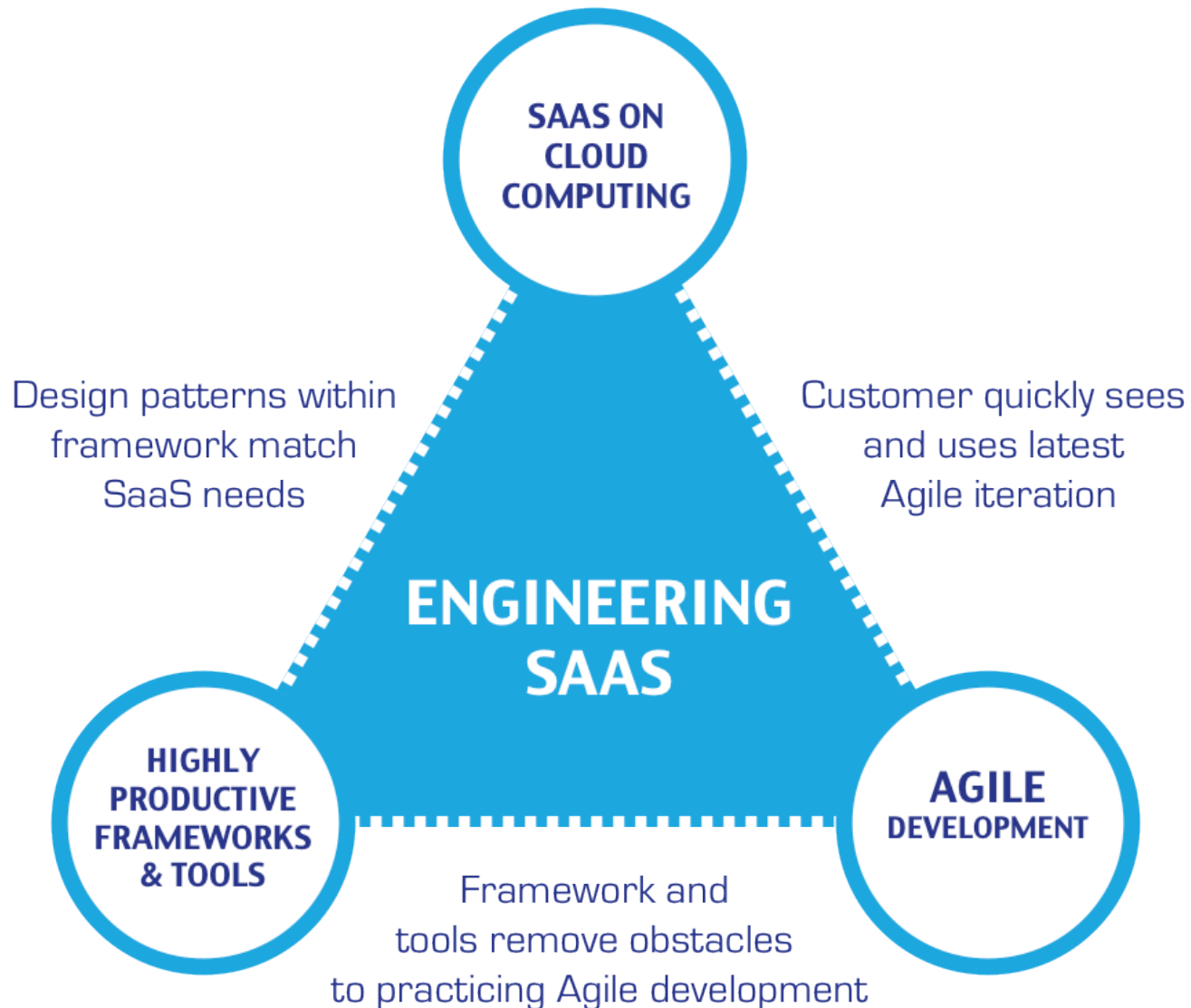
# Outline

- Software as a Service (SaaS)
- Development process: Waterfall v. Agile
- Book and Online Course
- Pair Programming and Scrum
- Points, Velocity, & Pivotal Tracker
- Lo-Fi UI Sketches & Storyboards
- Behavior-Driven Development & User Stories
- Test-Driven Design, Cucumber, & RSpec
- Summary

# 2-week Agile/XP Iteration



# Summary: Engineering SaaS is More Than Programming



# Reactions from Colleagues

“It is a pleasure to see a student text that emphasizes the production of real useful software.”

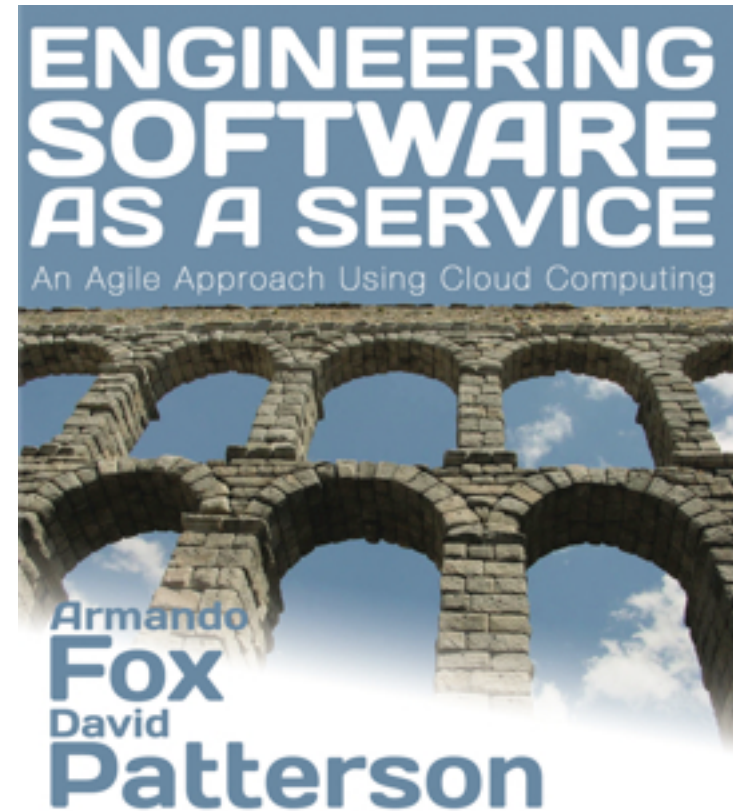
*Frederick P. Brooks, Turing Award winner and author of The Mythical Man-Month*

“I'd be far more likely to prefer graduates of this program than any other I've seen.”

*Brad Green, Engineering Manager, Google Inc.*

“I recommend this unique book and course to anyone who wants to develop or improve their SaaS programming skills.”

*Thomas Siebel, founder of Siebel CRM Systems*



<http://saasbook.info>



# ACM: The Learning Continues...

---

- Questions about this webcast?  
[learning@acm.org](mailto:learning@acm.org)
- ACM Learning Webinars:  
<http://learning.acm.org/multimedia.cfm>
- ACM Learning Center:  
<http://learning.acm.org>

