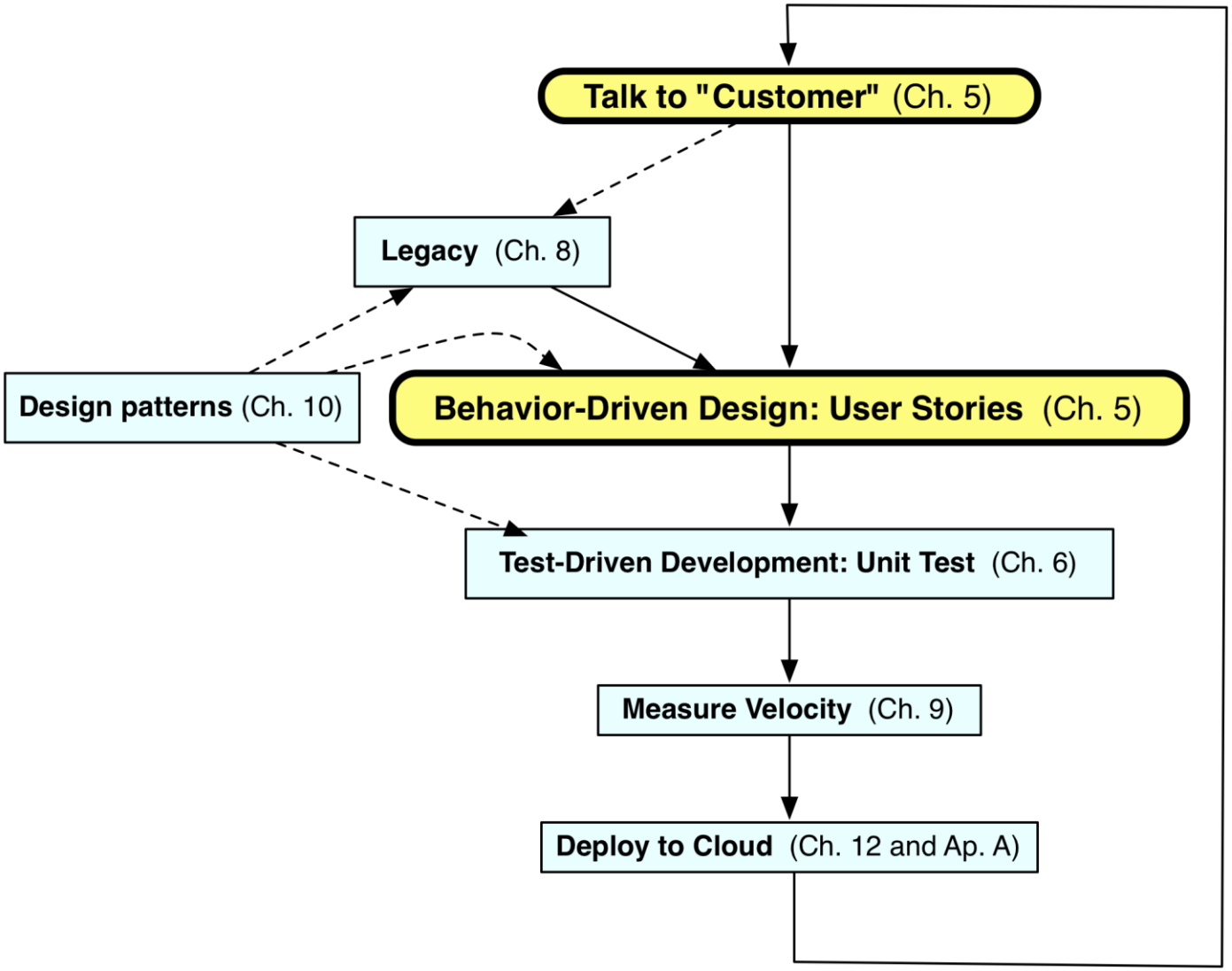
Why do SW Projects Fail?

- Don't do what customers want
- Or projects are late
- Or over budget
- Or hard to maintain and evolve
- Or all of the above
- Inspired Agile Lifecycle

Agile Lifecycle

- Work closely, continuously with stakeholders to develop requirements, tests
  - Users, customers, developers, maintenance programmers, operators, project managers, …
- Maintain working prototype while deploying new features every **iteration**
  - Typically every 1 or 2 weeks
  - Instead of 5 major phases, each months long
- Check with stakeholders on what's next, to validate building right thing (vs. verify)

Agile Iteration

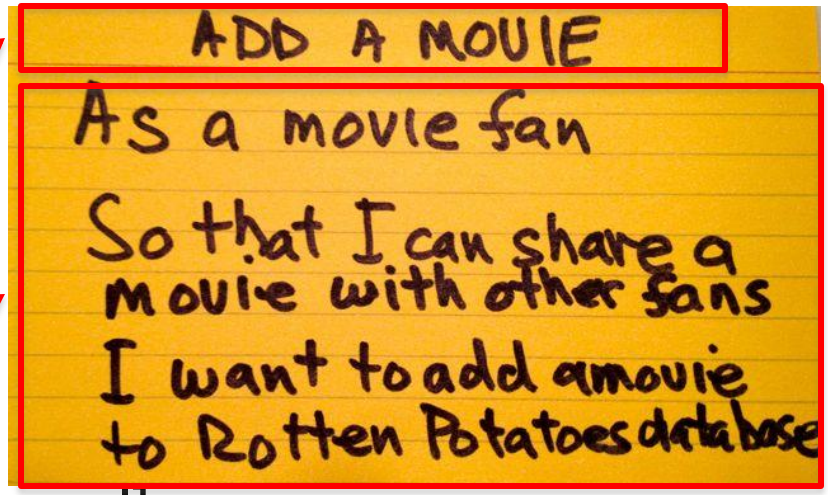


Behavior-Driven Design (BDD)

- 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 or TDD (next chapter) 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 stories must be 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

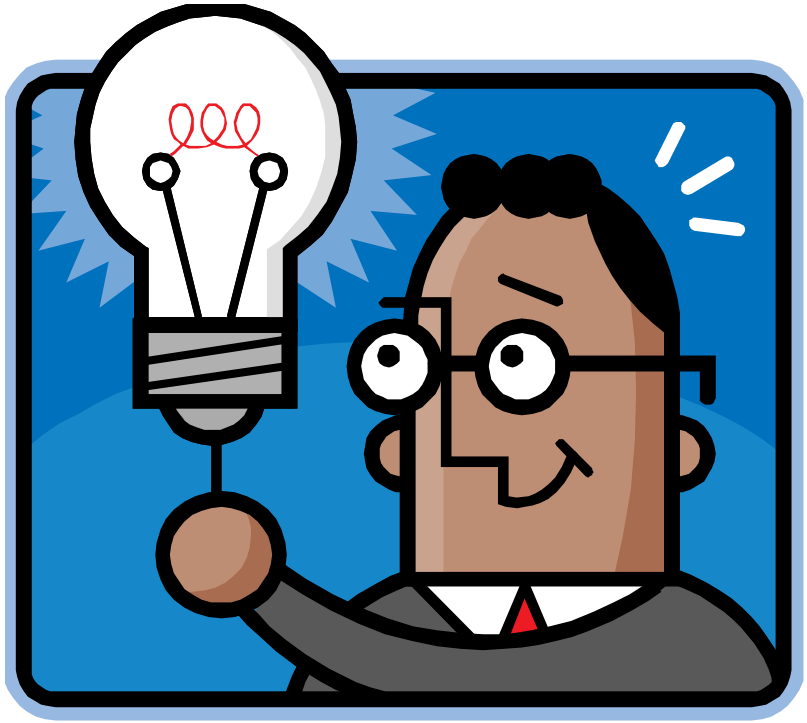
- Real systems have 100s of user stories
- *Backlog*: User Stories not yet completed
  - (We'll see Backlog again with Pivotal Tracker)
- Prioritize so most valuable items highest
- Organize so they match SW releases over time

Which expression statement regarding BDD and user stories is FALSE?

- ☐ BDD is designed to help with validation (build the right thing) in addition to verification
- ☐ BDD should test app implementation
- ☐ User stories in BDD play same role as design requirements in Big Design Up Front
- ☐ This is a valid User Story: "Search TMDb
  I want to search TMDb
  As a movie fan
  So that I can more easily find info"

SMART stories

- **S**pecific
- **M**easurable
- **A**chievable
  (ideally, implement in 1 iteration)
- **R**elevant
  ("the 5 why's")
- **T**imeboxed
  (know when to give up)



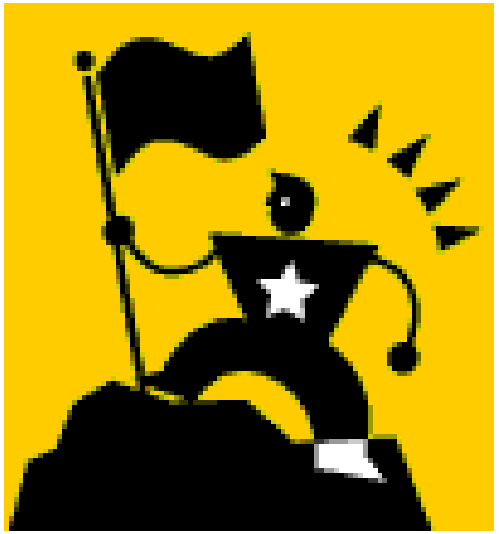
Specific & Measurable

- Each scenario testable
  - Implies known good input and expected results exist
- Anti-example: "UI should be user-friendly"
- Example: Given/When/Then.

1. *Given* some specific starting condition(s),
2. *When* I do X,
3. *Then* one or more specific thing(s) should happen

Achievable

- Complete in 1 iteration
- If can't deliver feature in 1iteration, deliver subset of stories
  - Always aim for working code @ end of iteration



Timeboxed

- Estimate what's achievable using *velocity*
  - Each story assigned *points* (1-3) based on difficulty
  - Velocity = Points completed / iteration
  - Use measured velocity to plan future iterations & adjust points per story

- Pivotal Tracker (later) tracks velocity

Relevant: “business value”

- Ask “Why?” recursively until discover business value, or kill the story:
  - Protect revenue
  - Increase revenue
  - Manage cost
  - Increase brand value
  - Making the product remarkable
  - Providing more value to your customers

http://wiki.github.com/aslakhellesoy/cucumber has a good example

Stories are SMART— but features should be relevant

- Specific & Measurable: can I test it?
- Achievable? / Timeboxed?
- Relevant? use the "5 whys"
- *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 Facebook friends are going to a given show

Which feature below is LEAST SMART?

- ☐ User can search for a movie by title
- ☐ Rotten Potatoes should have good response time
- ☐ When adding a movie, 99% of Add Movie pages should appear within 3 seconds
- ☐ As a customer, I want to see the top 10 movies sold, listed by price, so that I can buy the cheapest ones first

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

≥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: Ruby 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 to 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)

Capybara

- Need tool to act like user that pretends to be user follow scenarios of user story
- Capybara simulates browser
  - Can interact with app to receive pages
  - Parse the HTML
  - Submit forms as a user would
- Cannot handle JavaScript
  - Other tool (Webdriver) can handle JS, but it runs a lot slower, won't need yet

Demo

- Add feature to cover existing functionality
  - Note: This example is doing it in wrong order – should write tests first
  - Just done for pedagogic reasons
- (Or can look at screencast:

  http://vimeo.com/34754747)

Which is FALSE about Cucumber and Capybara?

- ☐ Cucumber and Capybara can perform acceptance and integration tests
- ☐ A Feature has ≥1 User Stories, which are composed typically of 3 to 8 Steps
- ☐ Steps use Given for current state, When for action, and Then for consequences of action
- ☐ Cucumber matches step definitions to scenario steps using regexes, and Capybara pretends to be user that interacts with SaaS app accordingly

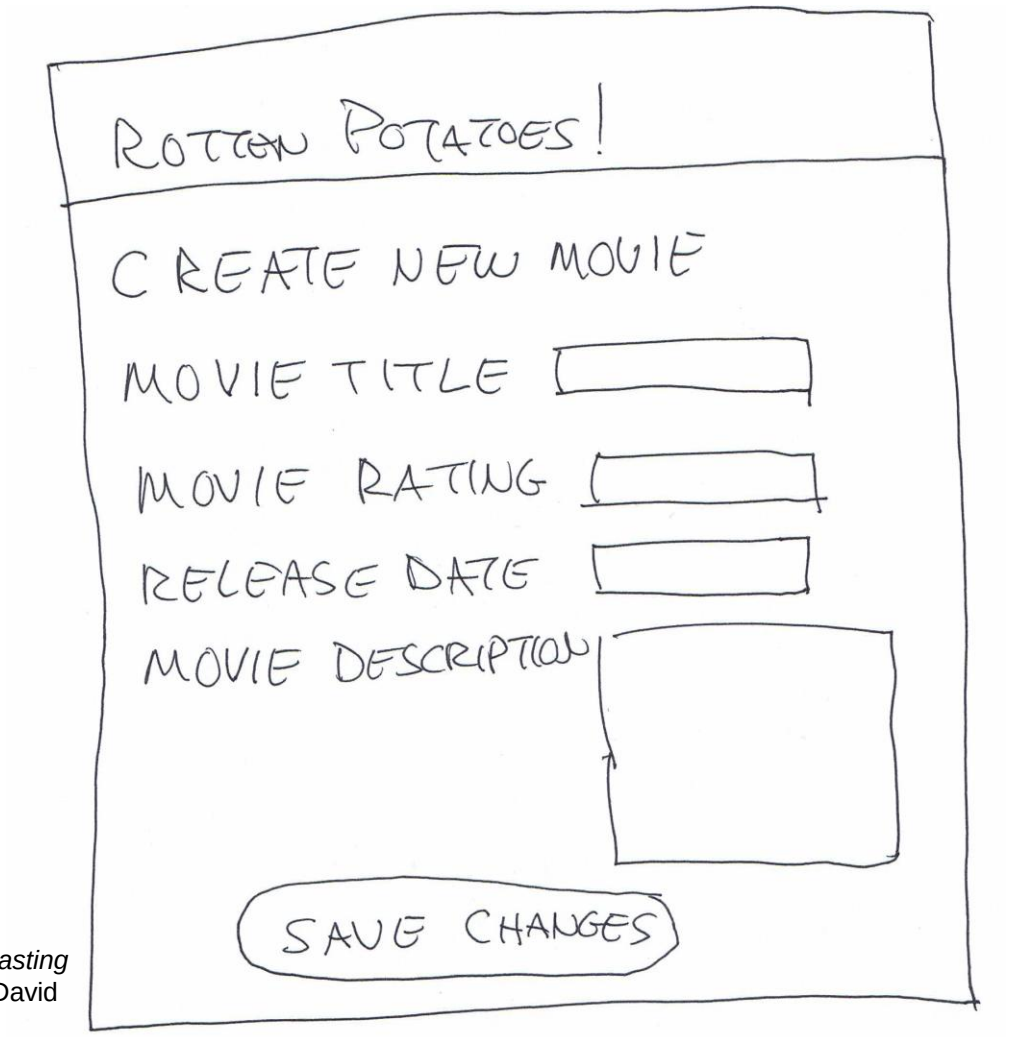
SaaS User Interface Design

- 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

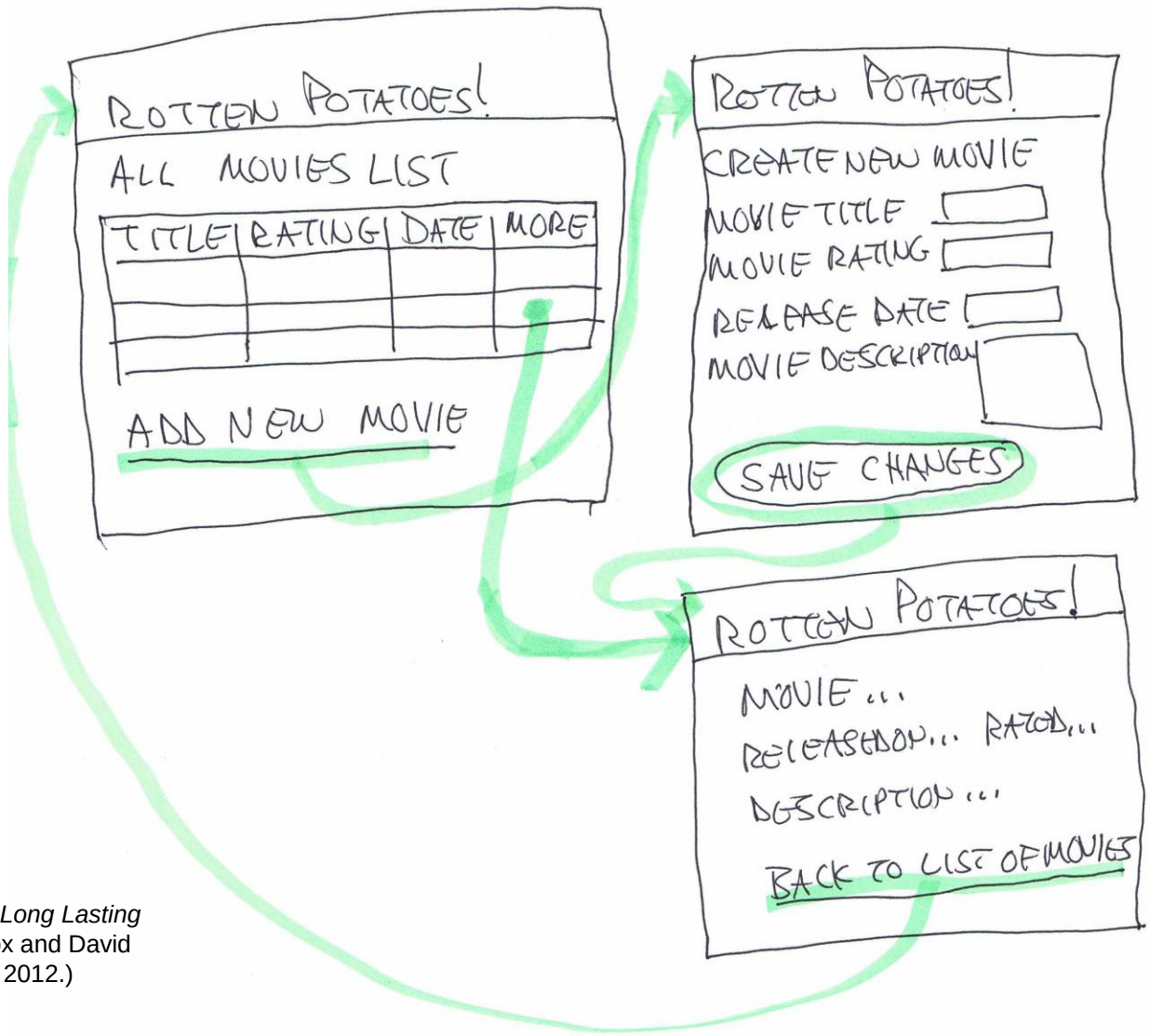


(Figure 4.3, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Alpha edition, 2012.)

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 Long Lasting Software* by Armando Fox and David Patterson, Alpha edition, 2012.)

Lo-Fi to HTML

- Tedious to do sketches and storyboards, but easier than producing HTML!
  - Also less intimidating to nontechnical stakeholders => More likely to suggest changes to UI if not code behind it
  - More likely to be happy with ultimate UI
- Next steps: CSS (Cascading Style Sheets) and Haml
  - Make it pretty *after* it works

Which is FALSE about Lo-Fi UI?

- ☐ Like 3x5 cards, sketches and storyboards are more likely to involve all stakeholders vs. code
- ☐ The purpose of the Lo-Fi UI approach is to debug the UI before you program it
- ☐ SaaS apps usually have a user interfaces associated with the user stories
- ☐ While it takes more time than building a prototype UI in CSS and Haml, the Lo-Fi approach is more likely to lead to a UI that customers like

And in Conclusion

- Debugging: Read, Ask, Search, Post
- Rails Pitfalls: Too much code in Controller, Some extra code in View
- Agile – prototypes, iterate with customer
- BDD – Design of app before implementation
- User Story – all stakeholders write what features want on 3x5 cards
- Cucumber – magically turns 3x5 card user stories into acceptance tests for app