Validating Software Requirements: Behavior-Driven Design and User Stories

CS 169 Spring 2012

David Patterson

Outline

- Introduction to BDD and User Stories (§4.1)
- SMART User Stories (§4.2)
- Introducing and Running Cucumber and Capybara (§4.3-§4.4)
- Lo-Fi UI Sketches and Storyboards (§4.5)
- Explicit vs. Implicit and Imperative vs. Declarative Scenarios (§4.7)
- Fallacies & Pitfalls, BDD Pros & Cons (§4.8-§4.9)

Introduction to Behavior-Driven Design and User Stories

(ELLS §4.1)

David Patterson

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

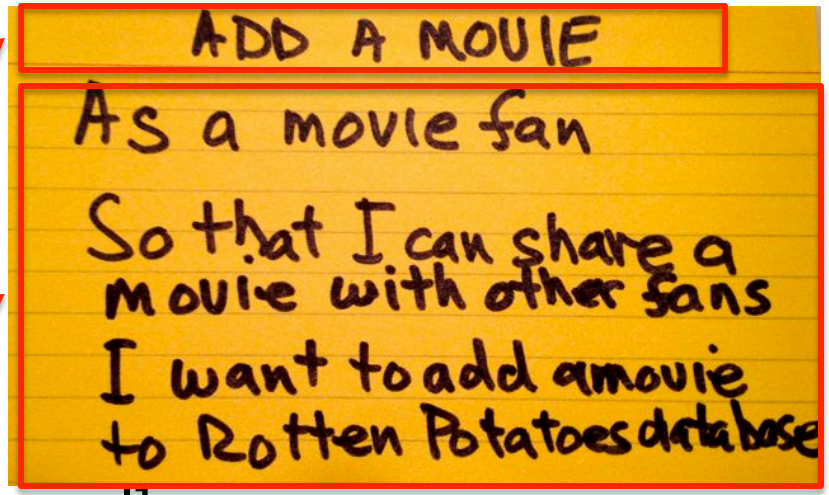
- Work closely, continuously with stakeholders to develop requirements, tests
  - Users, customers, developers, maintenance programmers, operators, project managers, …
- Maintain a working prototype while deploying new features every **iteration**
  - Typically every 1 or 2 weeks
  - Instead of 5 major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

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 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 can have 100s of user stories
- *Backlog*: a collection of User Stories not yet completed
  - (We'll see Backlog again with Pivotal Tracker)
- Prioritize so that most valuable items are highest
- Organize them so that 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 behavior, not app implementation
- ☐ BDD means you don't need to write acceptance tests
- ☐ 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 User Stories

(ELLS §4.2)

David Patterson

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 should be testable
  - Implies that some 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 / Timeboxed

- If can't deliver a whole feature in one iteration, deliver subset of stories
  - Always aim for working code @ end of iteration
- Estimate what's achievable using *velocity*
  - Each *story* assigned an estimated # of *points* (usually small, eg 1 to 3) based on difficulty
  - Velocity == Points completed per iteration
  - Use measured velocity to plan future iterations & adjust estimates of points per story

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 of her 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

Introducing and Running Cucumber and Capybara

(ELLS §4.3-§4.4)

David Patterson

Cucumber: Big Idea

- Tests customer-understandable 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 create 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

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 green for pass

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

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

Lo-Fi UI Sketches and Storyboards

(ELLS §4.5)

David Patterson

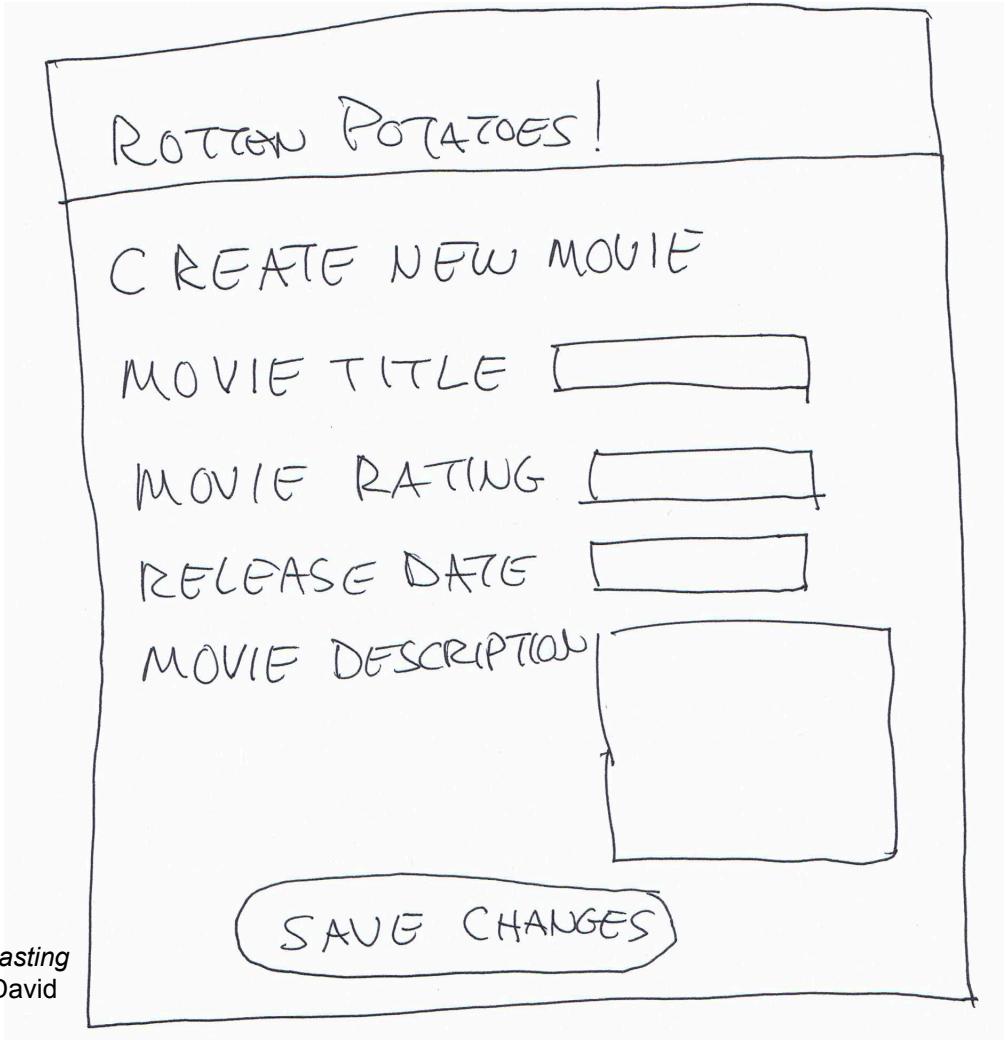
SaaS User Interface Design

- SaaS app often faces users

⇒User stories need User Interface (UI)

- Want all stakeholders involved in UI design
  - Don't want UI rejected after lots of work!
- Need UI equivalent of 3x5 cards
- Sketches: pen and paper drawings or "Lo-Fi UI"

Lo-Fi UI Example

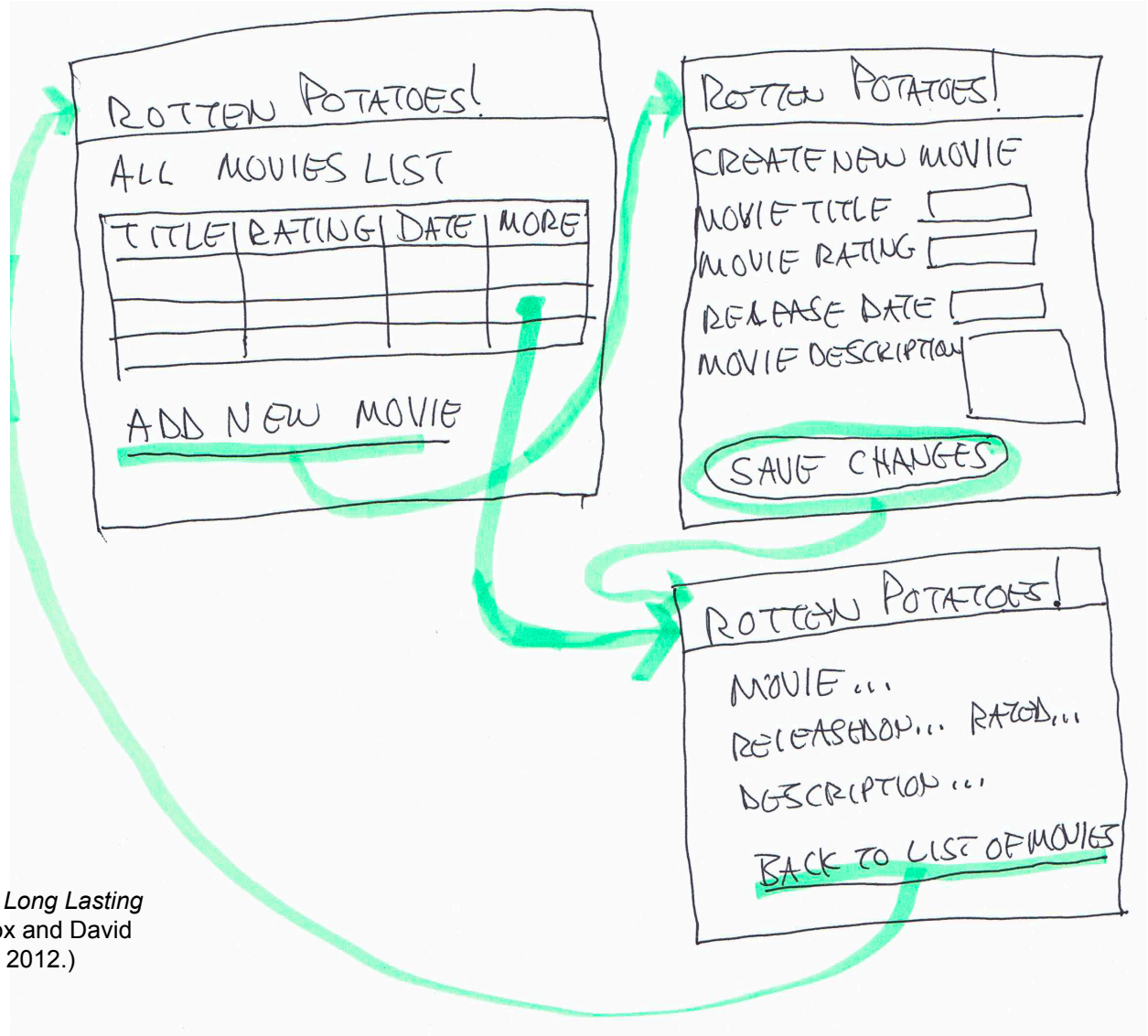


(Figure 4.3, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Alpha edition, 2012.)

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 program
  - More likely to be happy with ultimate UI
- Next steps: CSS 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, the Lo-Fi approach is more likely to lead to a UI that customers like

Enhancing Rotten Potatoes Again

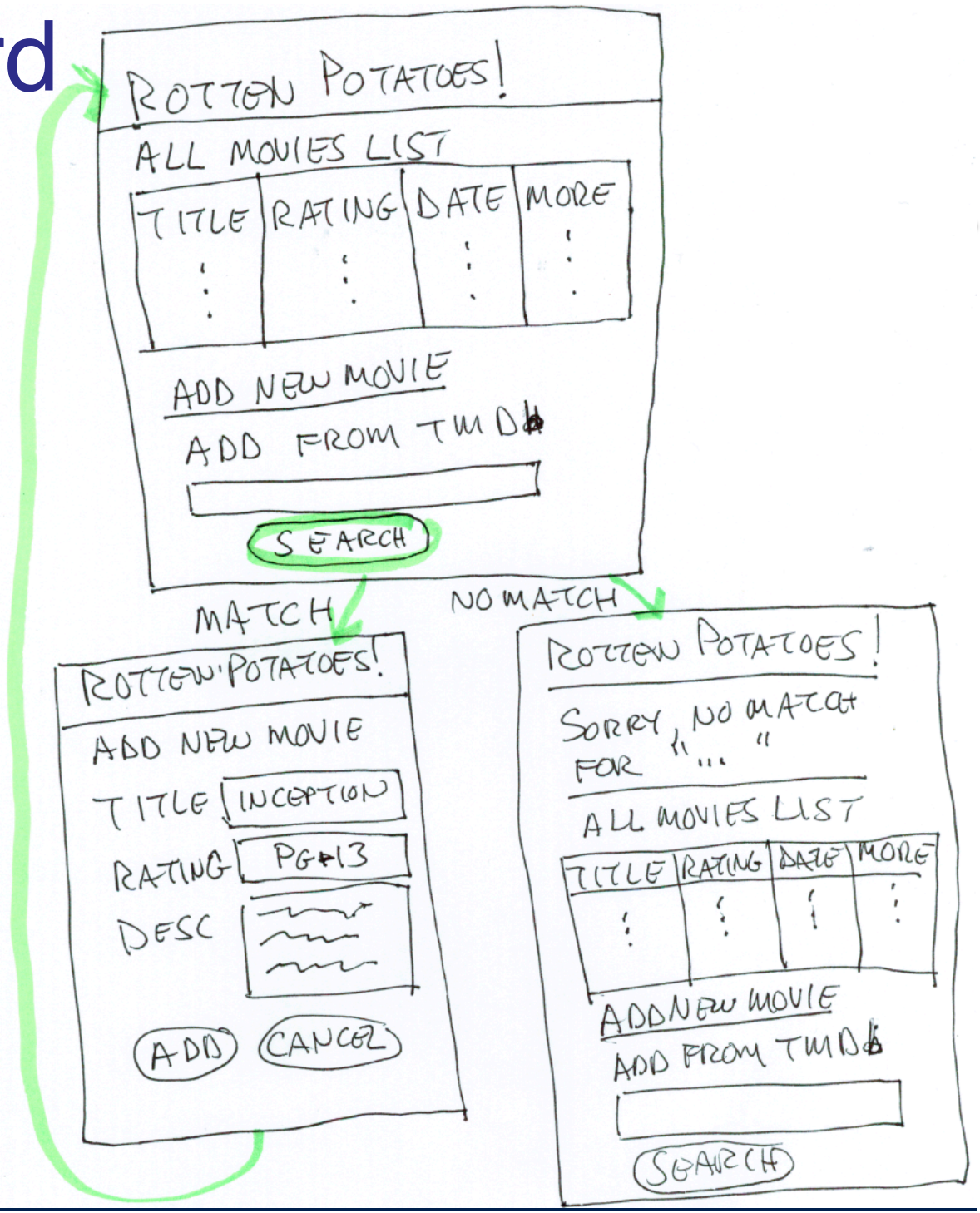
(ELLS §4.6)

David Patterson

Integrated with The Movie Database (TMDb)

- New Feature: Populate from TMDb, versus enter information by hand
- Need to add ability to search TMDb from Rotten Potatoes home page
- Need LoFi UI and Storyboard

Storyboard TMDb



Search TMDb User Story
(Figure 4.6 in ELLS)

Feature: User can add movie by searching in The Movie Database (TMDb)

As a movie fan

So that I can add new movies without manual tedium

I want to add movies by looking up their details in TMDb

Scenario: Try to add nonexistent movie (sad path)

Given I am on the RottenPotatoes home page

Then I should see "Search TMDb for a movie"

When I fill in "Search Terms" with "Movie That Does Not Exist"

And I press "Search TMDb"

Then I should be on the RottenPotatoes home page

And I should see "'Movie That Does Not Exist' was not found in TMDb."

Haml for Search TMDb page
(Figure 4.7 in ELLS)

```
-# add to end of app/views/movies/index.html.haml:

%h1 Search TMDb for a movie

= form_tag :action => 'search_tmdb' do

  %label{:for => 'search_terms'} Search Terms
  = text_field_tag 'search_terms'
  = submit_tag 'Search TMDb'
```

http://pastebin/18yYBVbC

Haml expansion last 2 lines

- Haml

  = text_field_tag 'search_terms'

  = submit_tag 'Search TMDb’

- Turns into

<label for='search_terms'>Search Terms</label>

<input id="search_terms" name="search_terms"
type="text" />

- `for` attribute of `label` tag matches `id` attribute of `input` tag, from text_field_tag helper

Cucumber?

- If try Cucumber, it fails
- MoviesController#search_tmdb is controller action that should receive form, yet doesn't exist in movies_controller.rb
- Should use Test Driven Development (next Chapter) to implement method search_tmdb
- Instead, to let us finish sad path, add fake controller method that always fails

Fake Controller Method: Will Fail Finding Movie (Figure 4.8)

```
# add to movies_controller.rb, anywhere inside
#  'class MoviesController < ApplicationController':


def search_tmdb
  # hardwired to simulate failure
  flash[:warning] = "'#{params[:search_terms]}' was
   not found in TMDb."
  redirect_to movies_path
end
```

http:/pastebin/smwxv70i

Trigger Fake Controller when form is POSTed (Figure 4.8)

```
# add to routes.rb, just before or just after 'resources :movies' :

# Route that posts 'Search TMDb' form
post '/movies/search_tmdb'
```

http://pastebin/FrfkF6pd

- Try Cucumber now

Happy Path of TMDb

- Find an existing movie, should return to Rotten Potatoes home page
- But some steps same on sad path and happy path
- How make it DRY?
- `Background` means steps performed before *each* scenario

TMDb with 2 Scenarios

http://pastebin/icQGrYCV

Feature: User can add movie by searching for it in The Movie Database (TMDb)

As a movie fan

So that I can add new movies without manual tedium

I want to add movies by looking up their details in TMDb

Background: Start from the Search form on the home page

Given I am on the RottenPotatoes home page

Then I should see "Search TMDb for a movie"

Scenario: Try to add nonexistent movie (sad path)

When I fill in "Search Terms" with "Movie That Does Not Exist"

And I press "Search TMDb"

Then I should be on the RottenPotatoes home page

And I should see "'Movie That Does Not Exist' was not found in TMDb."

Scenario: Try to add existing movie

When I fill in "Search Terms" with "Inception"

And I press "Search TMDb"

Then I should be on the RottenPotatoes home page

And I should see "Inception"

And I should see "PG-13"

Summary

- New feature => UI for feature, write new step definitions, even write new methods before Cucumber can color steps green
- Usually do happy paths first
- Background lets us DRY out scenarios of same feature
- BDD/Cucumber test behavior; TDD/RSpec in next chapter is how write methods to make all scenarios pass

Evaluate this statement: You need to implement all the code being tested before Cucumber will say that the test passes

- ☐ True
- ☐ False

Explicit vs. Implicit and Imperative vs. Declarative Scenarios

(ELLS §4.7)

David Patterson

Explicit vs. Implicit Scenarios

- Explicit requirements usually part of acceptance tests => likely explicit user stories and scenarios
- Implicit requirements are logical consequence of explicit requirements, typically integration testing
  - Movies listed in chronological order or alphabetical order?

Imperative vs. Declarative Scenarios

- Imperative: Initial user stories usually have lots of steps, specifying logical sequence that gets to desired result
- Declarative: many fewer steps
- Example Feature: movies should appear in alphabetical order, not added order
- Example Scenario: view movie list after adding 2 movies

Example Imperative Scenario

- 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 "Zorro"
- And I select "PG" from "Rating"
- And I press "Save Changes"
- Then I should be 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 "Apocalypse Now"
- And I select "R" from "Rating"
- And I press "Save Changes"
- Then I should be on the RottenPotatoes home page
- And I should see "Apocalypse Now" before "Zorro"

Only step specifying behavior;
Rest are implementation

Example Declarative Scenario

- Given I have added "Zorro" with rating "PG-13"
- And I have added "Apocalypse Now" with rating "R"
- Then I should see "Apocalypse Now" before "Zorro" on the Rotten Potatoes home page

Declarative Scenario Needs New Step Definitions

```
Given /I have added "(.*)"
  with rating "(.*)"/ do |
  title, rating|
  Given %Q{I am on the
  Create New Movie page}
  When  %Q{I fill in
  "Title" with "#{title}"}
  And   %Q{I select "#
  {rating}" from "Rating"}
  And   %Q{I press "Save
  Changes"}
end
```

```
Then /I should see "(.*)"
   before "(.*)" on (.*)/ do
   |string1, string2, path|
   Given %Q{I am on #{path}}
   regexp = Regexp.new ".*#
   {string1}.*#{string2}"
   page.body.should =~
   regexp
end
```

- As app evolves, reuse steps from first few imperative scenarios to create more concise and descriptive declarative scenarios
- Declarative scenarios focus attention on feature being described and tested vs. steps needed to set up test

Which is TRUE about implicit vs. explicit and declarative vs. imperative scenarios?

- ☐ As you get more experience with user stories, you will write many more declarative scenarios
- ☐ Explicit scenarios usually capture integration tests
- ☐ Declarative scenarios usually capture implementation as well as behavior
- ☐ Explicit requirements are usually defined with imperative scenarios and implicit requirements are usually defined with declarative scenarios

Fallacies & Pitfalls, BDD Pros & Cons, End of Chapter 4

(ELLS §4.8-§4.9)

David Patterson

Pitfalls

- Adding cool features that do not make the product more successful
  - Customers reject what programmers liked
  - User stories help prioritize, reduce wasted effort

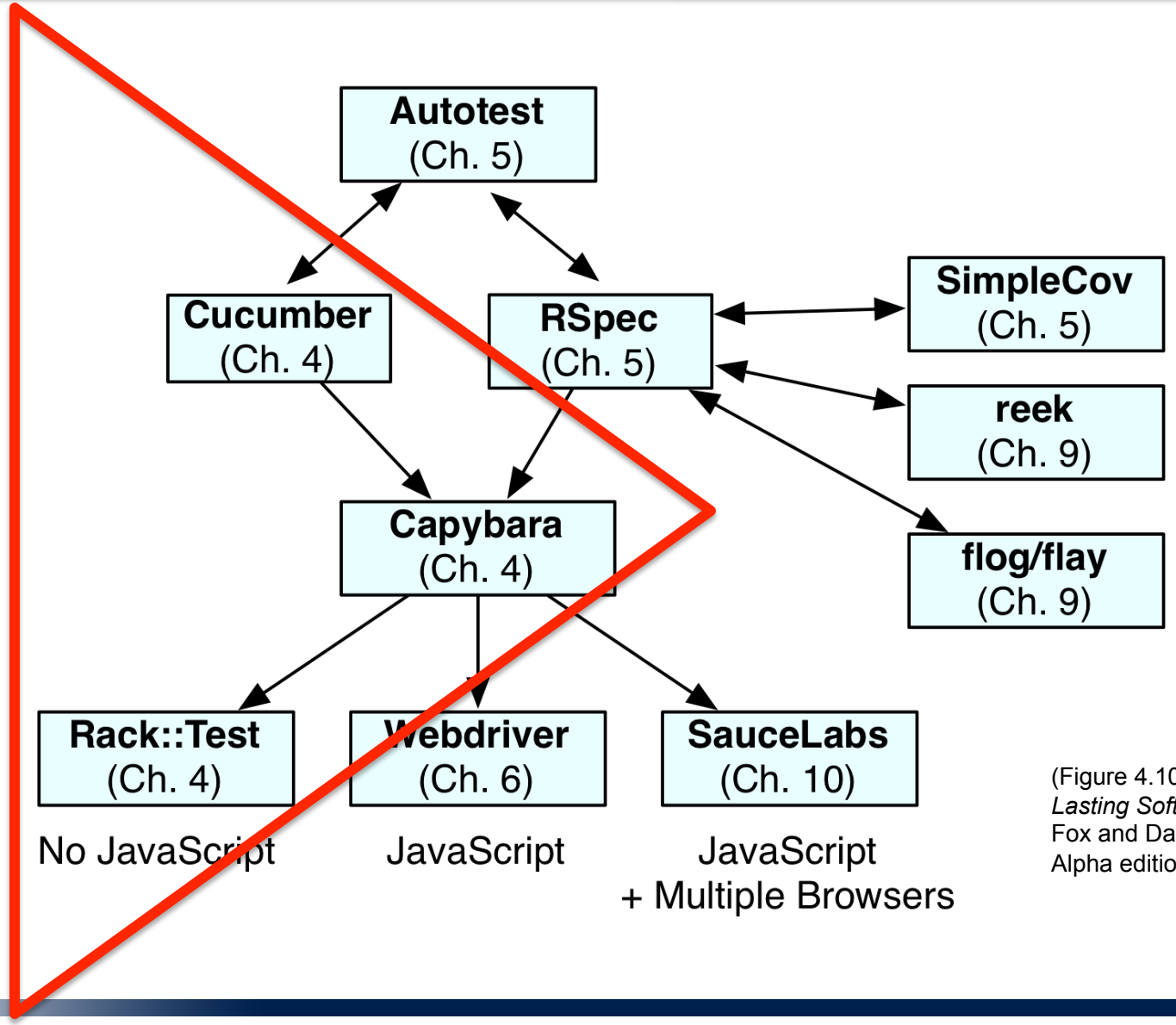
Pitfalls

- Trying to predict what code you need before need it
  - BDD: write tests *before* you write code you need, then write code needed to pass the tests
  - No need to predict, wasting development

Pitfalls

- Careless use of negative expectations
  - Beware of overusing “Then I should not see….”
  - Can’t tell if output is what want, only that it is not what you want
  - Many, many outputs are incorrect
  - Include positives to check results “Then I should see …”

Testing Tools in Book



(Figure 4.10, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Alpha edition, 2012.)

Which statement is FALSE about Lo-FI UI and BDD?

- ☐ The purpose of the Lo-Fi UI approach is to debug the UI before you program it
- ☐ A BDD downside is requiring continuous contact with customers, which may not be possible
- ☐ A BDD downside is that it may lead to a poor software architecture, since focus is on behavior
- ☐ None are false; all three above are true

Pros and Cons of BDD

- Pro: BDD/user stories - common language for all stakeholders, including nontechnical
  - 3x5 cards
  - LoFi UI sketches and storyboards
- Pro: Write tests before coding
  - Validation by testing vs. debugging
- Con: Difficult to have continuous contact with customer?
- Con: Leads to bad software architecture?
  - Will cover patterns, refactoring 2nd half of course

BDD

(Figure 4.11, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Alpha edition, 2012.)

- 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