Introduction to Behavior-Driven Design and User Stories

SELT, Fall 2019

Why do SW Projects Fail?

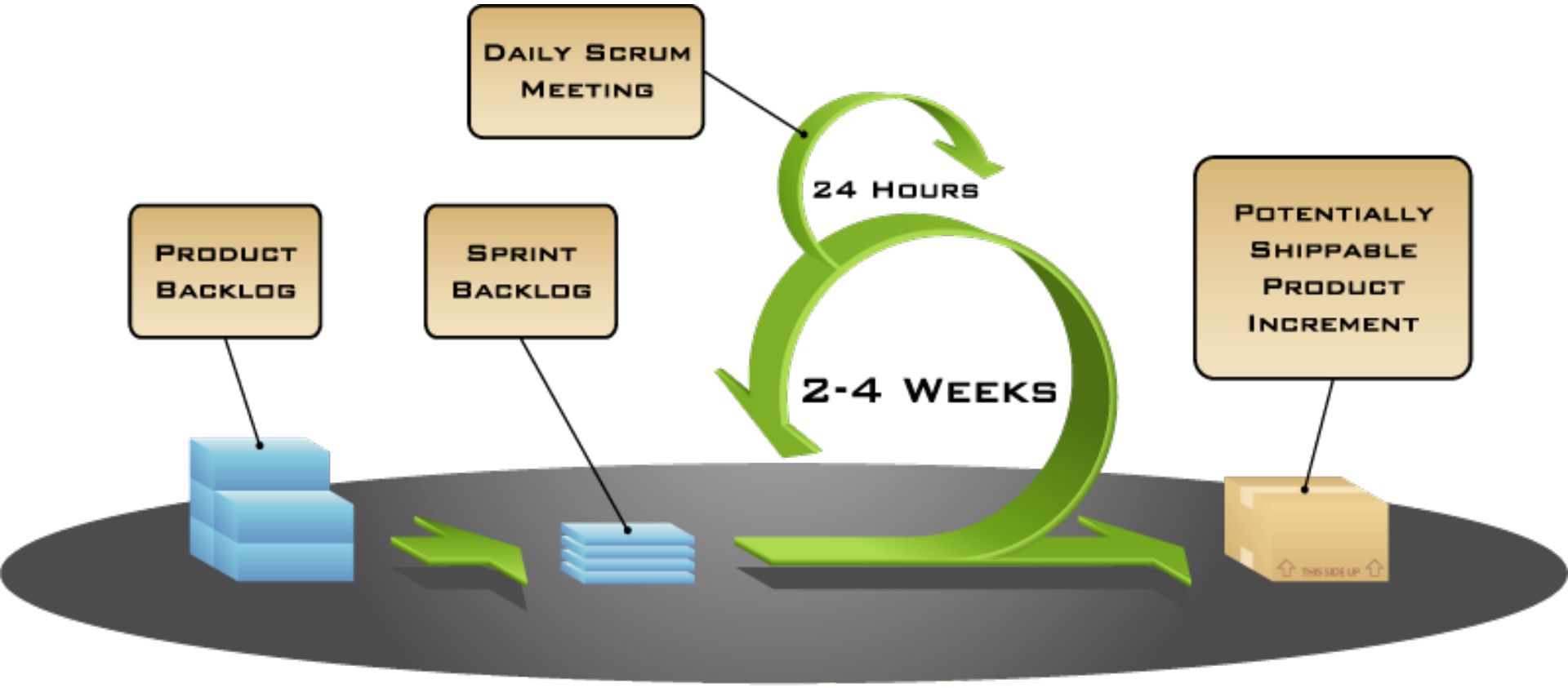
- Don't meet customers' needs
- Or, are buggy
- Or, are late
- Or, are over budget

- Or, are 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 2 to 4 weeks
  - As opposed to months-long stages or iterations for BDUF
- Create acceptance tests up front for each iteration

An Agile Development Model



Behavior-Driven Design (BDD)

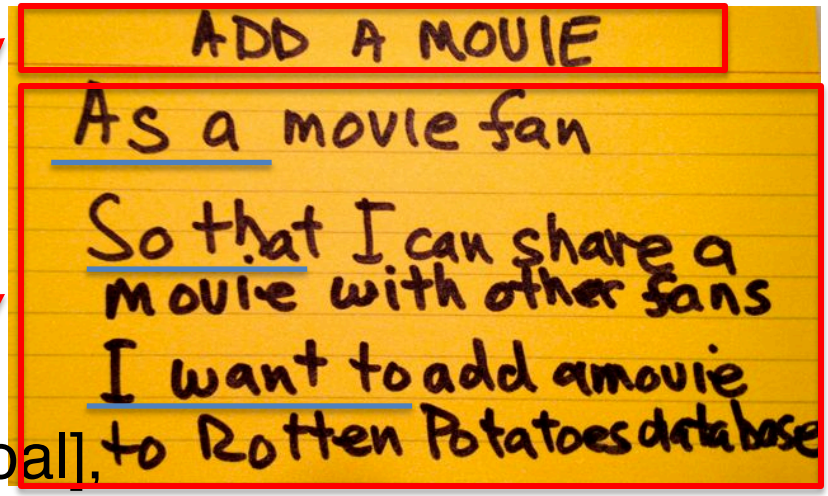
- BDD asks questions about behavior of app *before and during development* to reduce miscommunication
- Requirements written down as *user stories*
  - Lightweight descriptions of how app used
- Acceptance tests created before features are implemented
- BDD concentrates on *behavior* of app vs. *implementation* of app
  - Test Driven Design or TDD (later) tests implementation

Validation and Verification

- Validation: Are we building the right system?
  - Requirements specification
  - Acceptance testing
- Verification: Are we building the system right?
  - Unit testing
  - Integration testing

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

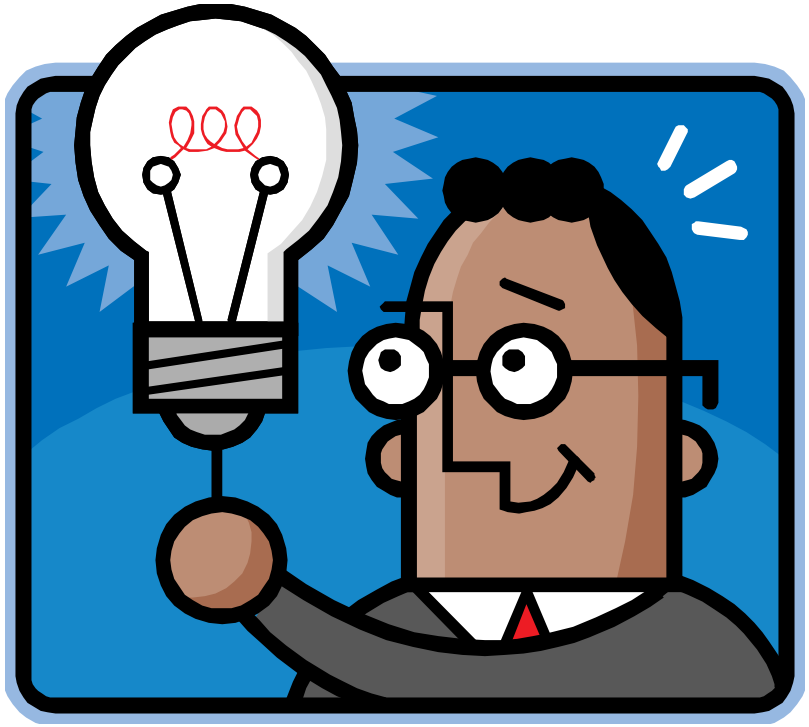
- *See which of my friends are going to a show*
  - As a theatergoer
  - So that I can enjoy the show with my friends
  - I want to see which of my Facebook friends are attending a given show
- *Display patron's Facebook friends*
  - As a box office manager
  - So that I can induce a patron to buy a 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

SMART User 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 one iteration
- If can't deliver feature in one iteration, 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 / iteratior
  - 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 users/customers

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: From User Stories to Acceptance tests

- 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

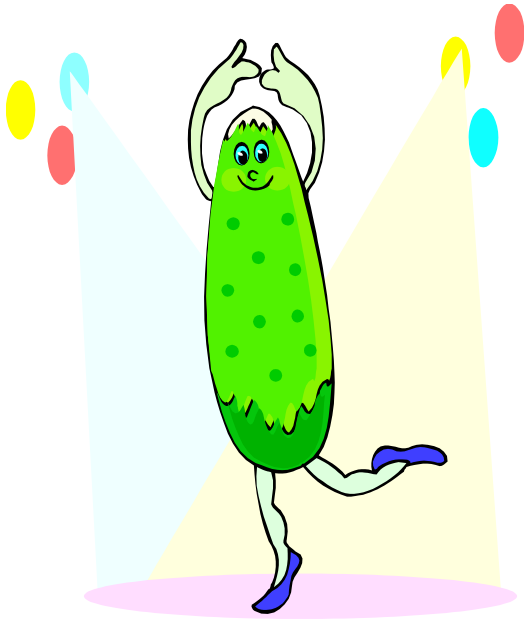
- User stories kept in one set of files: steps
- Separate set of files has Ruby code that tests steps: step definitions
- Step definitions are like method definitions, steps of scenarios are like method calls
- How match steps with step definitions?
- ***Regexes to match the English phrases in steps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

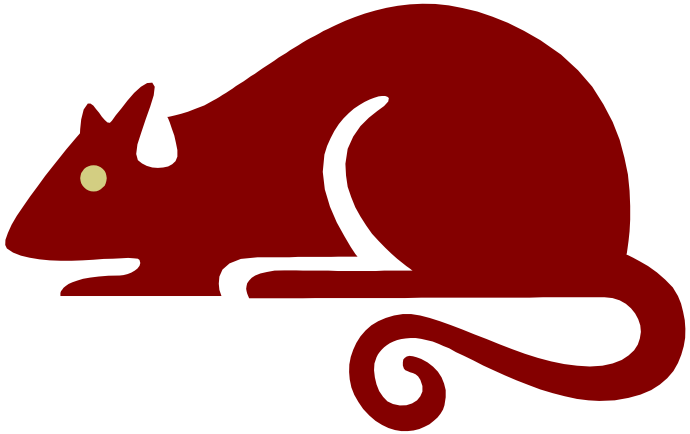
Red-Yellow-Green Analysis

- 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

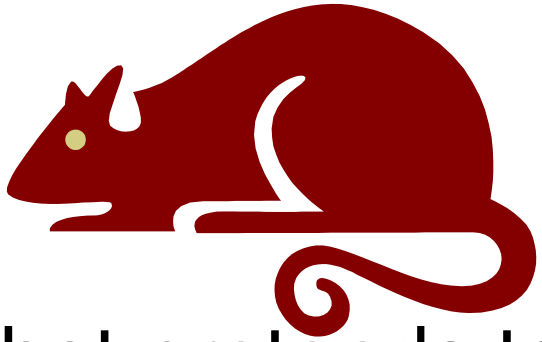
  (Hence green vegetable for name of tool)

Running Cucumber and Introducing Capybara





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 tools (Webdriver, Jasmine) can handle JS—we'll discuss them later in the semester.

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
- Look at screencast 7.7.1 in text)

  (or http://vimeo.com/34754747)

We are going to redo this example using **Declarative Scenarios**

"Imperative Scenario Used in Screencast 7.7.1"

Feature: User can manually add movie

Scenario: Add a movie
  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"

A "better" way: Declarative Scenarios

Feature: Allow RottenPotatoes user to add a new movie

Scenario: Add a new movie (Declarative)
When I have added a movie with title "Men in Black" and rating "PG-13"
And I am on the RottenPotatoes home page
Then I should see a movie list entry with title "Men in Black" and rating "PG-13"

For an explanation of the advantages of Declarative Scenarios over Imperative Scenarios and why Cucumber is abandoning support for Imperative Scenarios, read the article by Aslak Hellesoy at:
http://aslakhellesoy.com/post/11055981222/the-training-wheels-came-off

Step Definitions for the Declarative Add a new Movie scenario

```
When /^I have added a movie with title "(.*?)" and rating "(.*?)"$/ do |title, rating|
  visit new_movie_path
  fill_in 'Title', :with => title  select rating, :from => 'Rating'
  click_button 'Save Changes'
end

Then /^I should see a movie list entry with title "(.*?)" and rating "(.*?)"$/ do |title, rating|
 result=false
  all("tr").each do |tr|
    if tr.has_content?(title) && tr.has_content?(rating)
      result = true
      break
    end
 end
 expect (result).to be_truthy
end
```

Now lets do a "View Movie Details" feature

Feature: view the "More about" page for a movie

Scenario: Click on the "More about " link for a movie to view additional info
  Given I have added a movie with title "Argo" and rating "R"
  When I have visited the Details about "Argo" page
  Then I should see "Details about Argo"

Now let's complete the step definition(s) for this scenario

```
When /^I have visited the Details about "(.*?)" page$/ do |title|
  visit movies_path
  click_on "More about #{title}"
end
```

Let's do one more: Edit Movie Feature

Feature: edit an existing movie

Scenario: Edit a movie from the RottenPotatoes app
  Given I have added a movie with title "Ted" and rating "G"
   And I have visited the Details about "Ted" page
  When I have edited the movie "Ted" to change the rating to "R"
  And I am on the RottenPotatoes home page
  Then I should see a movie list entry with title "Ted" and rating "R"

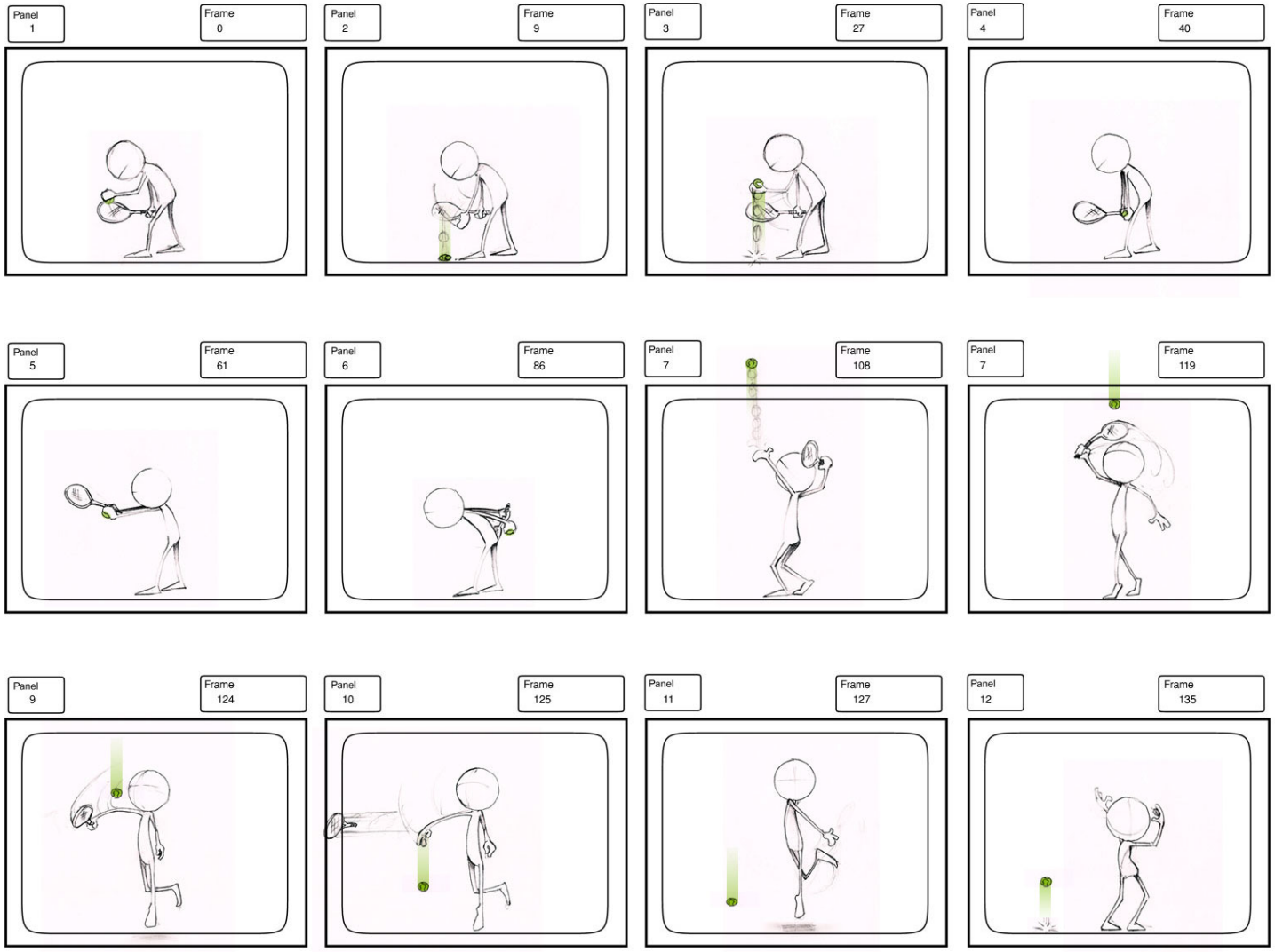
Step definition(s) for Edit Movie Scenario

```
When /^I have edited the movie "(.*?)" to change the rating to "(.*?)"$/ do |movie, rating|
  click_on "Edit"
  select rating, :from => 'Rating'
  click_button 'Update Movie Info'
end
```

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

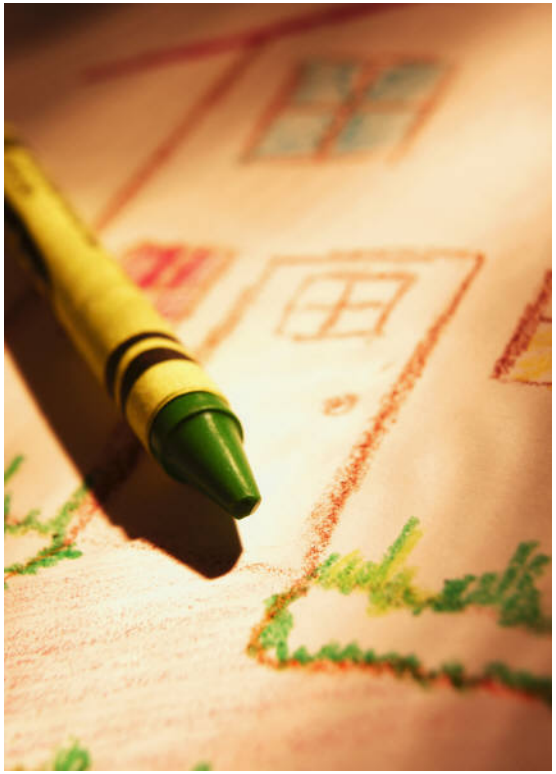


SaaS User Interface Design

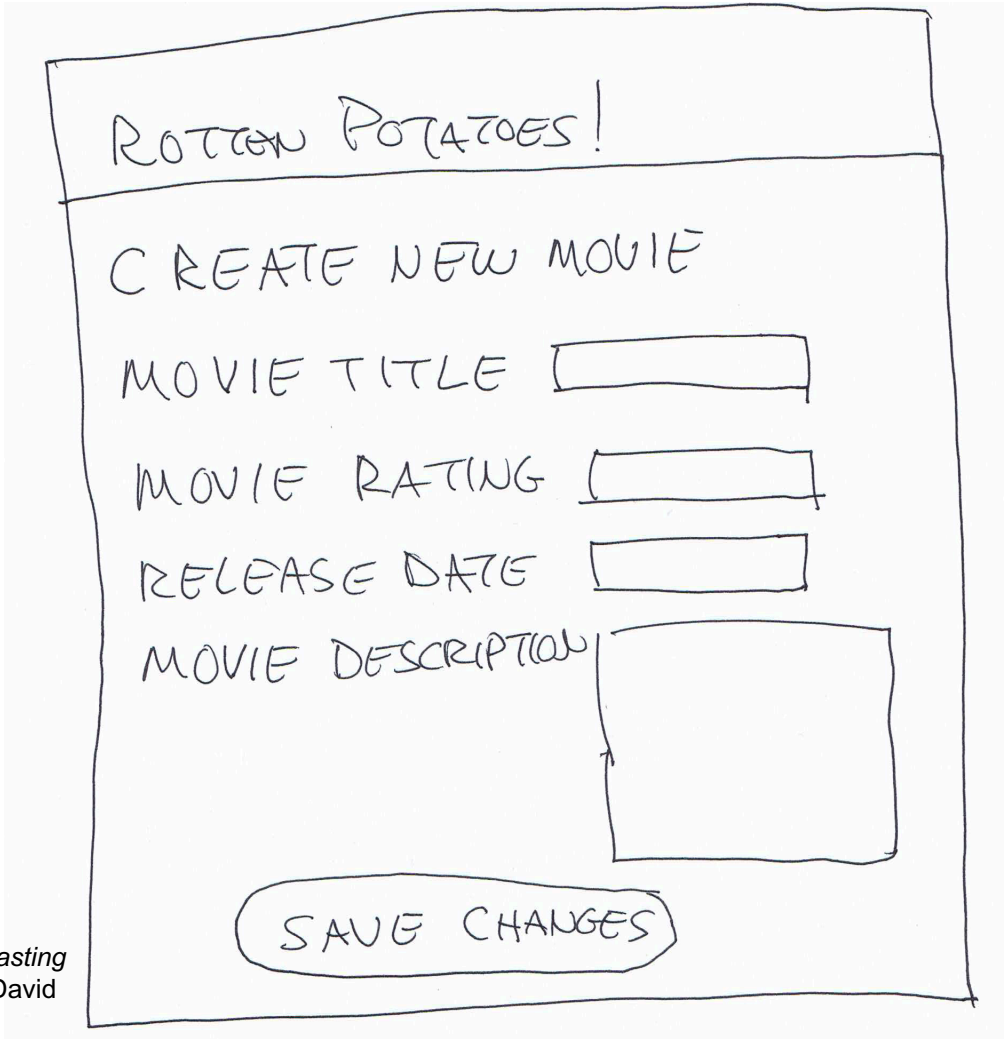
- SaaS apps often face users (e.g. web apps)

⇒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

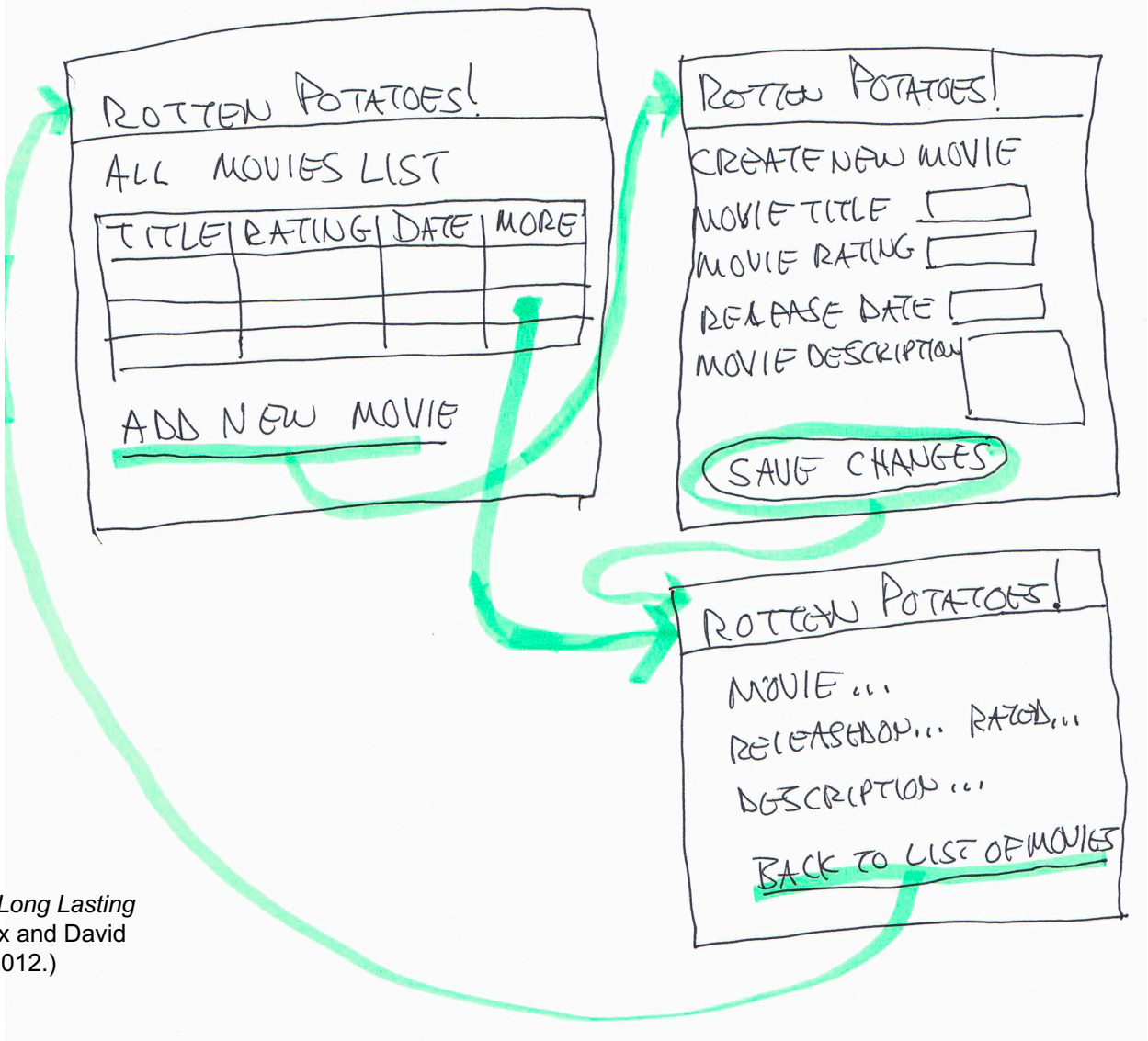


(Figure 5.3, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Beta edition, 2012.)

Storyboards

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

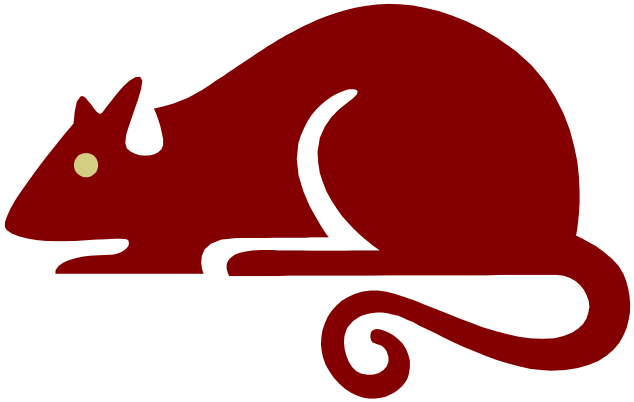
Example Storyboard

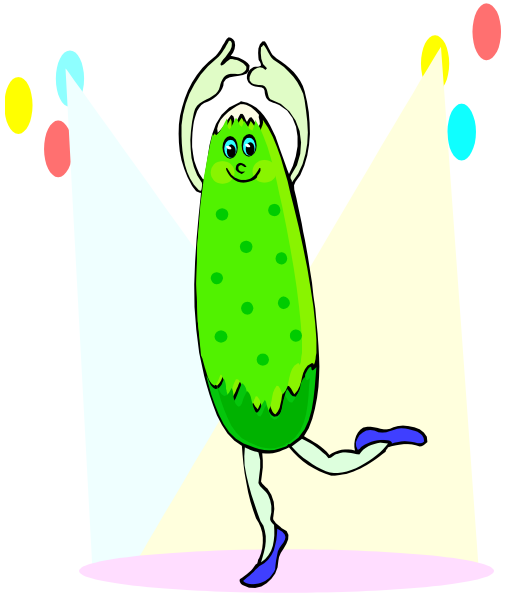


(Figure 5.4, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Beta edition, 2012.)

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



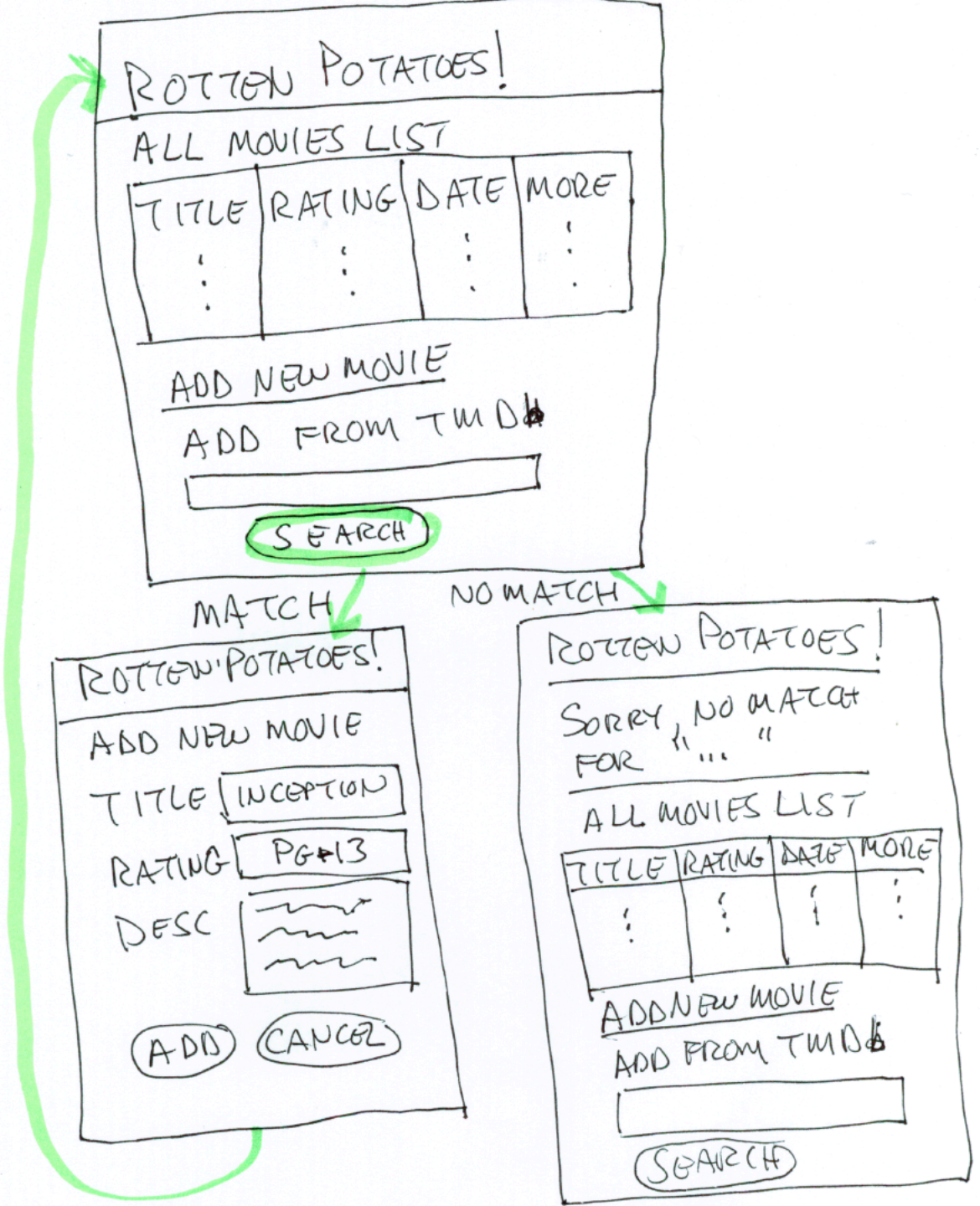


Enhancing Rotten Potatoes Again

Integration of RottenPotatoes with The Movie Database (TMDb.org)

- 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 integration



Search TMDb User Story
(Figure 7.7 in text)

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."

Search TMDb User Story

(Figure 7.7 in text)

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."

Search TMDb User Story
(Declarative Version)

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
When I have attempted to add a non-existent movie "SELT Unchained" from TMDb
Then I should see "SELT Unchained" was not found in TMDb."

Step Definitions for 'Search TMDb' Sad Path Scenario

```
When /^I have attempted to add a non\-existent movie "(.*?)" from TMDb$/ do |movie|
  fill_in 'Search Terms', :with => movie
  click_on 'Search TMDb'
 end

Then /^I should see "(.*?)" was not found in TMDb\."$/ do |movie|
  str = "\'#{movie}\' was not found in TMDb."
  expect(page).to have_content(str)
end
```

Haml for Search TMDb page
(Figure 7.8 in text)

```
-# 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'
```

Haml expansion last 3 lines

- Haml

```
%label{:for => 'search_terms'} Search Terms
 = 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?

- Cucumber will fail
- MoviesController#search_tmdb is controller action that should receive form, but it 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 7.9)

```
# 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 7.9)

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

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

- 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

Competing the feature

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

Background:
Given: I am on the RottenPotatoes home page

Scenario: Try to add nonexistent movie (sad path)

When I have attempted to add a non-existent movie "SELT Unchained" from TMDb
Then I should see "SELT Unchained" was not found in TMDb."

Scenario: Add a movie that exists in TMDB (happy path)

When I have added the movie "Life of Pi" from TMDb
Then I should see a movie list entry with title "Life of Pi" and rating "PG"

New Feature 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

Behavior Driven Design

- 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
  - BDD/TDD widely used in industry

Fallacies & Pitfalls, BDD Pros & Cons,

Pitfalls

- Customers who confuse mock-ups with completed features
  - May be difficult for nontechnical customers to distinguish a polished digital mock-up from a working feature
- Solution: LoFi UI on paper clearly *proposed* vs. implemented

Pitfalls

- Sketches without storyboards
  - Sketches are static
  - Interactions with SaaS app = sequence of actions over time
- "Animating" the Lo-Fi sketches helps prevent misunderstandings before turning stories are into tests and code
  - "OK, you clicked on that button, here's what you see; is that what you expected?"

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 …”

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