Modules 7 and 8

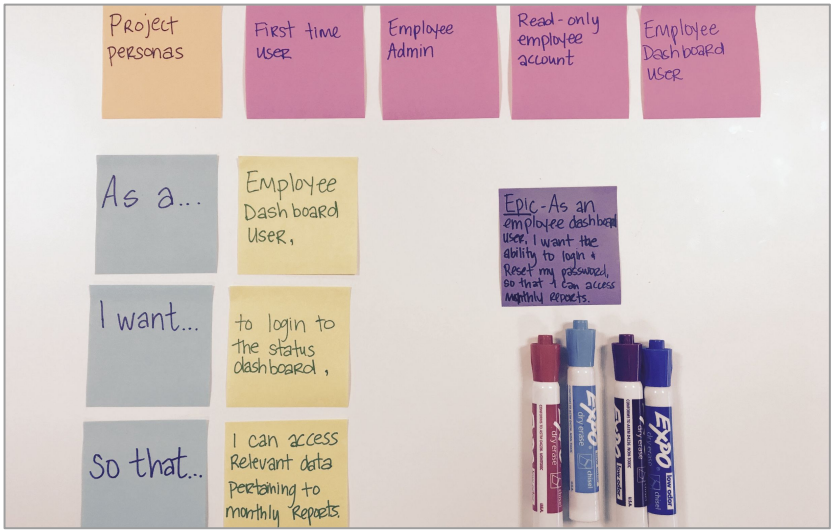
Topics covered:

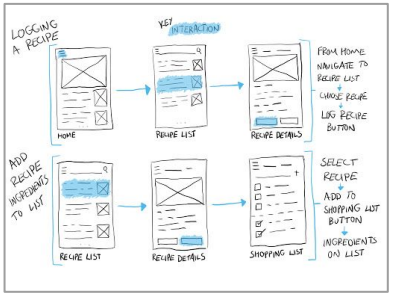
- Recap Module 7: Behavior Driven Design
- Practice questions for Module 7
- Introduction to Module 8: Test Driven Development

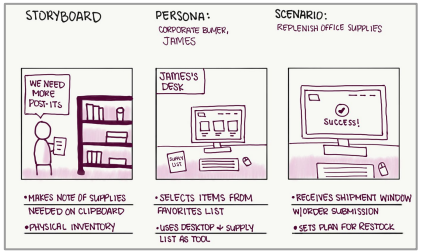
Module 7: Recap and Practice Problems

Module 7 Recap [Important Pointers for Quiz Revision & Cheat Sheet]

- **BDD** asks questions about the <u>behavior (validation)</u> of an application before and during development so that the stakeholders are less likely to miscommunicate.
- BDD's way of documenting **functional requirements** include user stories, low-fidelity wireframes and storyboarding.
  - **<u>User stories</u>** capture app behavior, user needs, and expected usage of the app. They follow the Connextra Format (Who is the stakeholder? What is their goal? What is their action?). They must be SMART.
  - Low-fidelity wireframes are low cost ways to explore the instantaneous state of user interface for a user story.
  - Storyboards capture the interaction between different pages depending on what the user does.
- How is BDD tested? Using **Cucumber**, a stylized, restricted form of English to describe a set of user stories, or scenarios, that collectively describe a feature story.







Module 7 Recap [Important Pointers for Quiz Revision & Cheat Sheet]

- A **feature** is the new app behavior that we want to implement, typically embodied in a user story. **Scenarios** are the different ways a feature can be exercised (*Happy* and *Sad* paths). The steps of a Cucumber scenario use following the keywords:
    - *Given* to describe the current state
    - *When* to identify user actions
    - *Then* to describe the intended consequences of those actions.
    - *And* to suggest the continuation of any of the above three listed keywords

- Each scenario step is matched to a **step definition** using regular expressions.
- A typical step definition uses the browser simulator **Capybara** to simulate a user's actions corresponding to that step, or interrogates the app to check (or *expect* - read documentation to learn more) if the desired consequences of the user's actions have occurred.

```
Feature: Write comments
  As a blog reader
  I want to be able to write a comment
  So that I tell the author my opinion / feedback

  Background: Login with author user
    Given that the author adds a new post with title "Post to comment"


  Scenario: Leave a comment with all info filled in
    Given that I select the post
    When I add a new comment with name, email  and  body
    Then I will see the comment on the blog


  Scenario: Leave a comment with name field not filled in
    Given that I select the post
    When I add a new comment with email and body
    Then I will see the message "ERROR: please fill the required fields (name, email)."


  Scenario:  Leave a comment with email field not filled in
    Given that I select the post
    When I add a new comment with name and body
    Then I will see the message "ERROR: please fill the required fields (name, email)."


  Scenario: Leave a comment without body
    Given that I select the post
    When I add a new comment with name and email
    Then I will see an empty comment on the blog
```

```
# now
Given /^(?:|that )an account with username "([\S]+)" and password "([\S]+)"
exists$/ do |user|, |pass|
              User.create(:username => user, :password => pass)

end
```

Module 7 Recap [Important Pointers for Quiz Revision & Cheat Sheet]

- **Other helpful syntax:**

```
Given the following users exist:
  | name   | email              | twitter         |
  | Aslak  | aslak@cucumber.io  | @aslak_hellesoy |
  | Julien | julien@cucumber.io | @jbpros         |
  | Matt   | matt@cucumber.io   | @mattwynne      |
```

```
Given /^the following users exist:$/ do |table|
        table.hashes.each do |acct|
                User.create(. . .)
        end
end
```

```
page.all(:css, 'a#person_123')
page.all(:xpath, './/a[@id="person_123"]')
```

```
Given /^I have unsuccessfully made two login attempts$/ do
        steps %Q{
                When I make an unsuccessful login attempt
                And I make an unsuccessful login attempt
        }
end
```

```
# Handy for things like preserving credentials
and service results

Given /^some step$/ do
        @state = . . .
end

Given /^some other step$/ do
        expect(@state).to be_valid
end
```

- **Productivity:** teams should be able to manage each iteration and predict how long they will take to implement new features
  - **How?** The team can assign points to rate difficulty of user stories and tracks the team's velocity (average points per iteration)
  - Using the following **tools/techniques**:
    - Planning Poker: Provides a service that helps team members **simultaneously "vote" on the difficulty of a story** and then discuss discrepancies, are a quick and practical way to estimate points while diffusing knowledge of the project throughout the team. Use persuasion/discussion or average points to reach a consensus
    - Pivotal Tracker: Provides a service that helps prioritize and keep track of user stories and their status throughout the software development pipeline (assigned, in progress, in review, deployed), calculates velocity, and predicts time spent during an iteration based on the team's sprint history.

Practice Problems

Question 1 & 2

1. **Discuss whether the following user story is SMART. Why or why not?**

    - “The UI should be intuitive.”
    - “The login UI should be so intuitive that 80% of customers can log-in within twenty seconds.”
    - “As an e-commerce website user, so that I can go to cart and checkout my purchase seamlessly, the pre-purchase login UI should be so intuitive that 80% of customers can log-in within twenty seconds.”

2. **Discuss one advantage and a disadvantage of using user stories and BDD.**

Question 1 & 2

1. **Discuss whether the following user story is SMART. Why or why not?**

   - "The UI should be intuitive."
   - "The login UI should be so intuitive that 80% of customers can log-in within twenty seconds."
   - "As an e-commerce website user, so that I can go to cart and checkout my purchase seamlessly, the pre-purchase login UI should be so intuitive that 80% of customers can log-in within twenty seconds."

2. **Discuss one advantage and a disadvantage of using user stories and BDD.**
   - Advantages:
     i. Low cost
     ii. Encourages trial and error / Easy to make changes
     iii. Easy to involve non-technical customers
   - Disadvantages
     i. Availability of all stakeholders to do behavior driven testing
     ii. Communication overhead: A team of developers dedicated to customer interaction / communication => Ask how much time is actually saved, if at all?
     iii. Time overhead: Creating and maintaining the feature files and scenarios (costly for smaller projects)
     iv. Unsuitable for P&D development processes as this requires flexibility and doesn't run on pre-set requirements
     v. Data-driven testing is not straightforward / application depends on external data feed / unknown behavior (stub the Internet)

Q3

Which of the following elements, if any, is NOT an element that a user story should capture?

A) A task that a particular stakeholder wants to accomplish
B) The role of a stakeholder
C) The business reason why the task is important to the stakeholder
D) The approximate effort (points) required to code the functionality
E) All of the above should be captured by a user story

Q3

Which of the following elements, if any, is NOT an element that a user story should capture?

A) A task that a particular stakeholder wants to accomplish
B) The role of a stakeholder
C) The business reason why the task is important to the stakeholder
D) The approximate effort (points) required to code the functionality
E) All of the above should be captured by a user story

Q4

Which of the following is true about user stories? (i) they should describe how the application is expected to be used (ii) they should have business value (iii) they do not need to be testable (iv) they should be implemented across multiple iterations of the Agile lifecycle

A. i only

B. i and ii

C. i and iv

D. i, iii, and iv

Q4

Which of the following is true about user stories? (i) they should describe how the application is expected to be used (ii) they should have business value (iii) they do not need to be testable (iv) they should be implemented across multiple iterations of the Agile lifecycle

A. i only

B. i and ii

C. i and iv

D. i, iii, and iv

Q5

Which statements are TRUE regarding imperative vs. declarative scenarios?

A) Imperative scenarios are OK if the details of the scenario are the focus of the test
B) Declarative scenarios should always be preferred over imperative scenarios
C) Imperative scenarios are necessary when the customer is nontechnical
D) Declarative scenarios can often re-use steps from imperative scenarios

Q5

Which statements are TRUE regarding imperative vs. declarative scenarios?

A) Imperative scenarios are OK if the details of the scenario are the focus of the test
B) Declarative scenarios should always be preferred over imperative scenarios
C) Imperative scenarios are necessary when the customer is nontechnical
D) Declarative scenarios can often re-use steps from imperative scenarios

Q6

Which of the following is true about implicit requirements and explicit requirements?

A) You cannot write user stories for both explicit and implicit requirements
B) Implicit requirements tend to be more concise, while explicit requirements tend to be more verbose
C) Implicit requirements are the logical consequence of explicit requirements, and typically correspond to integration tests
D) Explicit requirements are usually defined with imperative scenarios and implicit requirements are usually defined with declarative scena

Q6

Which of the following is true about implicit requirements and explicit requirements?

A) You cannot write user stories for both explicit and implicit requirements

B) Implicit requirements tend to be more concise, while explicit requirements tend to be more verbose

C) Implicit requirements are the logical consequence of explicit requirements, and typically correspond to integration tests

D) Explicit requirements are usually defined with imperative scenarios and implicit requirements are usually defined with declarative scena

The goal of Behavior-Driven Design (BDD) is:

(i) to verify that the application meets the specification
(ii) to validate that the design does what the customer wants
(iii) to help the customer understand the use of the application
(iv) to ask questions about the behavior of an application before and during development

A. i and ii
B. i, ii, and iii
C. i, ii, and iv
D. i, ii, iii, and iv

The goal of Behavior-Driven Design (BDD) is:

(i) to verify that the application meets the specification
(ii) to validate that the design does what the customer wants
(iii) to help the customer understand the use of the application
(iv) to ask questions about the behavior of an application before and during development

A. i and ii
B. i, ii, and iii
C. i, ii, and iv
D. i, ii, iii, and iv

Q8

For the Cucumber step Given RottenPotatoes contains a movie "Rambo" with rating "PG" which of the following COULD be true about its step definition?

A) It could set up this precondition by calling Movie.create
B) It could set up this precondition by calling a sequence of steps corresponding to how an end user would add this movie manually (go to New Movie page, submit form, etc.)
C) It would need at least 2 regular-expression capture groups
D) If the step appears in a Background: section, the step definition will be called only once even if there are multiple scenarios after the Background section

Q8

For the Cucumber step Given RottenPotatoes contains a movie "Rambo" with rating "PG" which of the following COULD be true about its step definition?

A) It could set up this precondition by calling Movie.create

B) It could set up this precondition by calling a sequence of steps corresponding to how an end user would add this movie manually (go to New Movie page, submit form, etc.)

C) It would need at least 2 regular-expression capture groups

D) If the step appears in a Background: section, the step definition will be called only once even if there are multiple scenarios after the Background section

Q9

In terms of how Cucumber executes scenarios (features), what is the difference between Given, When, and Then steps?

A. Only When steps (representing actions to be taken) can modify application state or have side effects, for example via POST requests.
B. You must specify at least one Given step before any When or Then steps.
C. There is no difference in execution; Given, When, and Then are aliases for the same method

Q9

In terms of how Cucumber executes scenarios (features), what is the difference between Given, When, and Then steps?

A. Only When steps (representing actions to be taken) can modify application state or have side effects, for example via POST requests.
B. You must specify at least one Given step before any When or Then steps.
C. There is no difference in execution; Given, When, and Then are aliases for the same method

Q10

What is the role of the Background steps?

A. It makes the user story DRYer by factoring out common steps across all scenarios of one feature for this app.
B. It makes the user story DRYer by factoring out common steps across all scenarios for this app.
C. It is a 'holding area' for steps that have not yet been implemented.
D. It sets the environment for this app.

Q10

What is the role of the Background steps?

A. It makes the user story DRYer by factoring out common steps across all scenarios of one feature for this app.

B. It makes the user story DRYer by factoring out common steps across all scenarios for this app.

C. It is a 'holding area' for steps that have not yet been implemented.

D. It sets the environment for this app.

Q11

For the Cucumber step Given RottenPotatoes contains a movie "Rambo" with rating "PG" which of the following COULD be the first line of its step definition?

A.
```
Given /RottenPotatoes contains a movie (".*") with rating (".*")/ do
|movie,rating|
```

B.
```
 Then /RottenPotatoes contains a movie (".*") with rating (".*")/ do
|movie,rating|
```

C.
```
 When /RottenPotatoes contains a movie (".*") with rating (".*")/ do
|movie,rating|
```

D.
```
 Then /RottenPotatoes contains a movie (".*")(with rating (".*"))?/ do
|movie,rating|
```

E.
```
Then /RottenPotatoes contains a movie (".*")(with rating (".*"))?/ do
|movie,has_rating,rating|
```

Q11

For the Cucumber step Given RottenPotatoes contains a movie "Rambo" with rating "PG" which of the following COULD be the first line of its step definition?

A.
```
Given /RottenPotatoes contains a movie (".*") with rating (".*")/ do
|movie,rating|
```

B.
```
 Then /RottenPotatoes contains a movie (".*") with rating (".*")/ do
|movie,rating|
```

C.
```
 When /RottenPotatoes contains a movie (".*") with rating (".*")/ do
|movie,rating|
```

D.
```
 Then /RottenPotatoes contains a movie (".*")(with rating (".*"))?/ do
|movie,rating|
```

E.
```
Then /RottenPotatoes contains a movie (".*")(with rating (".*"))?/ do
|movie,has_rating,rating|
```

Q12

For the Cucumber step Given RottenPotatoes contains a movie "Rambo" with rating "PG" which of the following COULD be true about its step definition?

A. It could set up this precondition by calling Movie.create
B. It could set up this precondition by calling a sequence of steps corresponding to how an end user would add this movie manually (go to New Movie page, submit form, etc.)
C. It would need at least 2 regular-expression capture groups
D. If the step appears in a Background: section, the step definition will be called only once even if there are multiple scenarios after the Background sectio

Q12

For the Cucumber step Given RottenPotatoes contains a movie "Rambo" with rating "PG" which of the following COULD be true about its step definition?

A. It could set up this precondition by calling Movie.create

B. It could set up this precondition by calling a sequence of steps corresponding to how an end user would add this movie manually (go to New Movie page, submit form, etc.)

C. It would need at least 2 regular-expression capture groups

D. If the step appears in a Background: section, the step definition will be called only once even if there are multiple scenarios after the Background sectio

Q13: True or False

- You need to implement all the code being tested before Cucumber will say the test passes
- A sad path Cucumber scenario can pass without having the code written need to make a happy path pass
- Cucumber matches step definitions to scenario steps using regular expressions, and Capybara pretends to be a user that interacts with the SaaS application according to these scenario
- The purpose of the Lo-Fi UI and storyboards is to debug the UI before you program it.

Q13: True or False

- You need to implement all the code being tested before Cucumber will say the test passes
- A sad path Cucumber scenario can pass without having the code written need to make a happy path pass
- Cucumber matches step definitions to scenario steps using regular expressions, and Capybara pretends to be a user that interacts with the SaaS application according to these scenario
- The purpose of the Lo-Fi UI and storyboards is to debug the UI before you program it.

Q13: True or False

- You need to implement all the code being tested before Cucumber will say the test passes
- **A sad path Cucumber scenario can pass without having the code written need to make a happy path pass**
- Cucumber matches step definitions to scenario steps using regular expressions, and Capybara pretends to be a user that interacts with the SaaS application according to these scenario
- The purpose of the Lo-Fi UI and storyboards is to debug the UI before you program it.

Q13: True or False

- You need to implement all the code being tested before Cucumber will say the test passes
- A sad path Cucumber scenario can pass without having the code written need to make a happy path pass
- Cucumber matches step definitions to scenario steps using regular expressions, and Capybara pretends to be a user that interacts with the SaaS application according to these scenario
- The purpose of the Lo-Fi UI and storyboards is to debug the UI before you program it.

Q13: True or False

- You need to implement all the code being tested before Cucumber will say the test passes
- A sad path Cucumber scenario can pass without having the code written need to make a happy path pass
- **Cucumber matches step definitions to scenario steps using regular expressions, and Capybara pretends to be a user that interacts with the SaaS application according to these scenario**
- The purpose of the Lo-Fi UI and storyboards is to debug the UI before you program it.

Q13: True or False

- You need to implement all the code being tested before Cucumber will say the test passes
- A sad path Cucumber scenario can pass without having the code written need to make a happy path pass
- Cucumber matches step definitions to scenario steps using regular expressions, and Capybara pretends to be a user that interacts with the SaaS application according to these scenario
- The purpose of the Lo-Fi UI and storyboards is to debug the UI before you program it.

Q13: True or False

- You need to implement all the code being tested before Cucumber will say the test passes
- A sad path Cucumber scenario can pass without having the code written need to make a happy path pass
- Cucumber matches step definitions to scenario steps using regular expressions, and Capybara pretends to be a user that interacts with the SaaS application according to these scenario
- **The purpose of the Lo-Fi UI and storyboards is to debug the UI before you program it.**

Q13: True or False

- You need to implement all the code being tested before Cucumber will say the test passes
- A sad path Cucumber scenario can pass without having the code written need to make a happy path pass
- Cucumber matches step definitions to scenario steps using regular expressions, and Capybara pretends to be a user that interacts with the SaaS application according to these scenario
- The purpose of the Lo-Fi UI and storyboards is to debug the UI before you program it.

Module 8: Test Driven Development

Introduction (More Next Week)

TDD: Test Driven Development

- Write tests before you write functional code. Starting from the acceptance and integration tests derived from User stories, write failing unit tests that test the nonexistent code you wish you had.
  - Write minimum code to pass the test
  - Code is always tested
  - Low chances of writing code that will be discarded
- An aspect of agile software development
- In agile development, unlike plan-and-document, the role of the Quality Assurance team (QA) team is not to implement all the tests.
- Testing is a shared responsibility of the whole team: developers write most of their own tests, with QA staff improving the testing environment and handling some kinds of testing that are hard to automate.
- Reasons for test-centric software development:
  - Software correctness
  - Readability
  - Maintainability

Q1

Which of the following are FALSE about testing in Agile and Plan-and-Document?

A. There is no manual testing in agile development.
B. In plan-and-document, testing is entirely done by the QA team.
C. In agile development, there is a QA team whose role is to reproduce customer reported bugs.
D. In plan-and-document, the development team, not the QA team, fixes bugs.

TDD FIRST Principle

- The FIRST principle in TDD lists a set of agreed upon features of a good test.
  - **Fast**: test cases should be easy and quick to run.
  - **Independent**: The order in which we run tests should not interfere with the test results.
  - **Repeatable**: Test behavior should not be influenced by external factors such as today's date.
  - **Self-checking**: Test should automatically report whether they failed/passed and should not rely on manual verification.
  - **Timely**: Test creation/update should not be postponed. Instead write/update tests at the same time you write/update code.

Q2

All of the following are general features of good tests. Which one is NOT referred to in the FIRST principle?

A. Tests should have short execution time.
B. We should be able to run tests in any order.
C. Tests should be easily readable.
D. Tests should not require manual verification.

Red-Green-Refactor

- **Red-Green-Refactor** refers to the recommended sequence of steps for writing test in TDD
  1. Write test for the behavior you expect the code to have.
  2. Red step: Run the test and verify that it fails since the behavior has not yet been implemented.
  3. Green step: Write the simplest possible code that achieves expected behavior and passes the test.
  4. Refactor step: Refactor/optimize your code and your test while ensuring that tests are passing.

Q3

In which stage of the Red-Green-Refactor practice do we first write functional code for the behavior under test?

A. Step 1
B. Red step
C. Green step
D. Refactor step

System under Testings

- System under testing (SUT) is a term that refers to the object being tested. The object could be a single method, group of methods, entire class etc.
- A test suite refers to a collection of test cases where each test case checks a specific behavior of a system under testing. It is a full set of tests, which usually includes unit, integration, and perhaps other types of tests.
    - Example: A unit test is a fine-grained test case for which the SUT is a single method.
- *More on SUTs next week*

- **In general, every test case follows the Arrange, Act, Assert (3A) structure. Knowledge of SUTs is fundamental to understanding the 3A testing structure.**

3A Test Case Structure

1. ***Arrange***: Set up necessary pre-conditions for the test case eg. dependency injection (ie setting up required variables).
2. ***Act***: Exercise the SUT.
3. ***Assert***: Verify the behavior matches expectation.

How to set up a 3A testing structure for this controller method?

```
class MoviesController < ApplicationController
	def review_movie
		search_string = params[:search]
		begin
			matches = Movie.find_in_tmdb(search_string)
			if matches.empty? # nothing was found
				redirect_to review_movie_path , :alert => "No
			elsif matches.length == 1
				@movie = matches [0]
				render 'review_movie'
			else # more than 1 match
				@movies = matches
				render 'select_movie'
			end

		end
		rescue Movie::ConnectionError => err

			redirect_to review_movie_path, :alert => "Error contacting
		TMDb: #{err.message}"

		end
	end
end
```

How to set up a 3A testing structure for this controller method?

```
class MoviesController < ApplicationController
	def review_movie
		search_string = params[:search]
		begin
			matches = Movie.find_in_tmdb(search_string)
			if matches.empty? # nothing was found
				redirect_to review_movie_path , :alert => "No
			elsif matches.length == 1
				@movie = matches [0]
				render 'review_movie'
			else # more than 1 match
				@movies = matches
				render 'select_movie'
			end

		end
		rescue Movie::ConnectionError => err

			redirect_to review_movie_path, :alert => "Error contacting
		TMDb: #{err.message}"

		end
	end
end
```

Arrange

How to set up a 3A testing structure for this controller method?

```
class MoviesController < ApplicationController
	def review_movie
		search_string = params[:search]
		begin
			matches = Movie.find_in_tmdb(search_string)
			if matches.empty? # nothing was found
				redirect_to review_movie_path , :alert => "No
			elsif matches.length == 1
				@movie = matches [0]
				render 'review_movie'
			else # more than 1 match
				@movies = matches
				render 'select_movie'
			end

		end
		rescue Movie::ConnectionError => err

			redirect_to review_movie_path, :alert => "Error contacting
		TMDb: #{err.message}"

		end
	end
end
```

Arrange

Act

How to set up a 3A testing structure for this controller method?

```
class MoviesController < ApplicationController
	def review_movie
		search_string = params[:search]
		begin
			matches = Movie.find_in_tmdb(search_string)
			if matches.empty? # nothing was found
				redirect_to review_movie_path , :alert => "No
			elsif matches.length == 1
				@movie = matches [0]
				render 'review_movie'
			else # more than 1 match
				@movies = matches
				render 'select_movie'
			end

		end
		rescue Movie::ConnectionError => err

			redirect_to review_movie_path, :alert => "Error contacting
		TMDb: #{err.message}"

		end
	end
end
```

Arrange

Act

Assert

How to set up a 3A testing structure for this controller method?

```
class MoviesController < ApplicationController
        def review_movie
                search_string = params[:search]
                begin
                        matches = Movie.find_in_tmdb(search_string)
                        if matches.empty? # nothing was found
                                redirect_to review_movie_path , :alert => "No
                        elsif matches.length == 1
                                @movie = matches [0]
                                render 'review_movie'
                        else # more than 1 match
                                @movies = matches
                                render 'select_movie'
                        end

                end
                rescue Movie::ConnectionError => err

                        redirect_to review_movie_path, :alert => "Error contacting
                TMDb: #{err.message}"

                end
        end
end
```

4 cases to test in asserting phase. But it essentially depends on the an external data stream.

What should we do??

How to set up a 3A testing structure for this controller method?

```
class MoviesController < ApplicationController
       def review_movie
              search_string = params[:search]
              begin
                     matches = Movie.find_in_tmdb(search_string)
                     if matches.empty? # nothing was found
                            redirect_to review_movie_path , :alert => "No
                     elsif matches.length == 1
                            @movie = matches [0]
                            render 'review_movie'
                     else # more than 1 match
                            @movies = matches
                            render 'select_movie'
                     end

              end
              rescue Movie::ConnectionError => err

                     redirect_to review_movie_path, :alert => "Error contacting
              TMDb: #{err.message}"

              end
       end
end
```

4 cases to test in asserting phase. But it essentially depends on the an external data stream.

What should we do??

We have to find a way to control the behavior *of the call* to find_in_tmdb.

How do we do that?

How to set up a 3A testing structure for this controller method?

```
class MoviesController < ApplicationController
	def review_movie
		search_string = params[:search]
		begin
			matches = Movie.find_in_tmdb(search_string)
			if matches.empty? # nothing was found
				redirect_to review_movie_path , :alert => "No
			elsif matches.length == 1
				@movie = matches [0]
				render 'review_movie'
			else # more than 1 match
				@movies = matches
				render 'select_movie'
			end

		end
		rescue Movie::ConnectionError => err

			redirect_to review_movie_path, :alert => "Error contacting
		TMDb: #{err.message}"

		end
	end
end
```

4 cases to test in asserting phase. But it essentially depends on the an external data stream.

What should we do??

We have to find a way to control the behavior *of the call* to find_in_tmdb.

How do we do that?

Find out during next lecture!

RSpec

RSpec is a ruby testing framework.

- ***Describe*** block used to group together a suite of related test cases. We can have nested describe blocks. Think of this as a feature definition from Cucumber.
- ***Context*** block used to group together tests cases within a describe block that share the same state.
- ***It*** block used to specify a single test case. Think of this as a scenario from Cucumber.
- ***Before*** block is used to set up necessary pre-conditions. Think of this as the background block from Cucumber.

Next Week: More TDD

System under Testing, Seams & Doubles, Factories & Fixtures, Coverage, Integration and System Testing, More questions

Attendance

tinyurl.com/cs169a-disc-8-attendance

passcode: SMART