Behavioral Driven Development (BDD)
(Engineering Software as a Service § 7)

Philip Ritchey
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
• How does Agile try to avoid failure?
year = ORIGINYEAR; /* = 1980 */

while (days > 365)
{
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
}
Agile Lifecycle Review

• 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)
BDD + TDD: The Big Picture

• Behavior-Driven Design (BDD)
  – develop user stories (*the features you wish you had*) to describe how app will work
  – via **Cucumber**, user stories become **acceptance tests** and **integration tests**

• Test-Driven Development (TDD)
  – *step definitions* for a 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*
Introduction to Behavior-Driven Design and User Stories

(Engineering Software as a Service § 7.1)
Behavior-Driven Design (BDD)

- BDD asks questions about behavior of app before and during development to reduce miscommunication
  - Validation vs. Verification
- 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 (future segments) 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
  - Often get new insights during development
Creating User Stories

• How do you know if you have a good user story vs. bad user story?
  – Right size?
  – Not too hard?
  – Is worthwhile?
SMART User Stories
(Engineering Software as a Service § 7.3)
SMART Stories

- **Specific**
- **Measurable**
- **Achievable**
  (ideally, implement in 1 iteration)
- **Relevant**
  ("the 5 why’s")
- **Timeboxed**
  (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 1 iteration, deliver subset of stories
  – Always aim for working code @ end of iteration
• If <1 story per iteration, need to improve point estimation per story
Relevant: “Business Value”

- 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
5 Whys to Find Relevance

• *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

1. Why?
2. Why?
3. Why?
4. Why?
5. Why?
Timeboxed

- Stop story when exceed time budget
  - Give up or divide into smaller stories or reschedule what is left undone
- To avoid underestimating length of project
- Pivotal Tracker tracks velocity, helps avoid underestimate
Which feature below is LEAST SMART?

1. User can search for a movie by title

2. *Rotten Potatoes should have good response time*

3. When adding a movie, 99% of Add Movie pages should appear within 3 seconds

4. As a customer, I want to see the top 10 movies sold, listed by price, so that I can buy the cheapest ones first
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?

1. BDD is designed to help with validation (build the right thing) in addition to verification

2. **BDD should test app implementation**

3. User stories in BDD play same role as design requirements in Plan-and-Document

4. 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”
Lo-Fi UI Sketches and Storyboards

(Engineering Software as a Service § 7.4)
Building Successful UI

- SaaS app often faces users
  ⇒ User stories need User Interface (UI)
- How to get customer to participate in UI design so is happy when complete?
  - Avoid WISBNWIW* UI
  - UI version of 3x5 cards?
- How to show UI interactivity without building a prototype?

*What-I-Said-But-Not-What-I-Want
SaaS User Interface Design

- **UI Sketches**: pen and paper drawings or “Lo-Fi UI”
Lo-Fi UI Example

(Figure 7.3, Engineering Long Lasting Software by Armando Fox and David Patterson, 1st edition, 2014.)
Storyboards

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

(Figure 7.4, Engineering Long Lasting Software by Armando Fox and David Patterson, 1st edition, 2014.)
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/Erb
  – 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 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
Productivity and Tools

• Don’t we want to avoid major planning effort in Agile? If so, how to estimate time without a plan?

• Can User Stories be used to measure progress on project?

• What should a tool do to help measure progress for Agile?
Points, Velocity, and Pivotal Tracker

*(Engineering Software as a Service § 7.2)*
Measuring Productivity

• A measure of team productivity: calculate avg. no. stories / week?
  – But some stories much harder than others
• Rate each user story in advance on a simple integer scale
  – 1 for straightforward, 2 for medium, 3 for very complex
• **Velocity**: avg. number of points / 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 Labs, 8 is extremely rare
• Teams assign value: vote by holding up fingers simultaneously, take average
  – If a big disagreement (2 and 5), discuss more
More on Points

• $\geq 5 \implies$ divide user story into simpler stories
  – backlog not too demanding
• Doesn’t matter if velocity is 5 or 10 points per iteration
  – As long as team is consistent
• Idea is to improve self-evaluation and suggest number of iterations for feature set
Pivotal Tracker

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

• Prioritize user stories by where place them in Current, Backlog, Icebox panels
• When completed, move to Done panel
• 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 Finish button, which sends to “Product Owner” (as in Scrum team organization)

• Product Owner tries out the user story and then either hits “Accept” or “Reject”
  – Accept, which marks user story as done, or
  – Reject, which marks story as needing to be Restarted by developer
Pivotal Tracker: Features vs. Chores

• **Features**
  – User stories that provide verifiable business value to customer
    • “Add agree box to checkout page”
  – Worth points & therefore must be estimated

• **Chores**
  – User Stories that are necessary, but provide no direct, obvious value to customer
    • “Find out why test suite is so slow”
  – No points
Which expression statement regarding Points, Velocity, and Tracker is TRUE?

1. When comparing two teams, the one with the higher velocity is more productive

2. When you don’t know how to approach a given user story, just give it 3 points

3. With Tracker, developers pick the user stories and mark as Accepted when done

4. Tracker helps prioritize and keep track of user stories and their status, calculates velocity, and predicts software development time
GitHub Issues & Pivotal Tracker

- Bug tracking vs Project management

Alternatives

- Trello [https://trello.com/](https://trello.com/)
  - It’s Free!
  - Simple, clean UI

- Waffle [https://waffle.io/](https://waffle.io/)
User Stories

=> Acceptance Tests?

• Wouldn’t it be great to automatically map 3x5 card user stories into tests for user to decide if accept the app?
• How would you match the English text to test code?
• How could you run the tests without a human in the loop to perform the actions?
Introducing Cucumber & Capybara

(Engineering Software as a Service § 7.6)
Cucumber & RSpec

• Cucumber describes behavior via features & scenarios (behavior driven design)

• RSpec tests individual modules that contribute to those behaviors (test driven development)
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 are not code, so clear to customer and can be used to reach agreement
  – Also not completely freeform, so can connect to real tests
Example User Story

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"

1 Feature
≥1 Scenarios / Feature
3 to 8 Steps / Scenario
Cucumber User Story, Feature, and Steps

• **User story:** refers to single feature

• **Feature:** ≥1 scenarios that show different ways a feature is used
  – Keywords *Feature and Scenario* identify respective components
  – Kept in `.feature` files

• **Scenario:** 3 - 8 steps that describe scenario

• **Step definitions:** Ruby code to test steps
  – Kept in `X_steps.rb` files
5 Step Keywords

1. **Given** steps represent state of world before event: *preconditions*
2. **When** steps represent event
   - e.g., simulate user pushing a button
3. **Then** steps represent expected *postconditions*; check if true
4. / 5. **And & But** extend previous step
Steps => Step Definitions via Regular Expressions

- **Regexes match English phrases in steps of scenarios to step definitions!**
  
  - Given ` /^(?:|I )am on (.+)$/`
  - “I am on the Rotten Potatoes home page”

- Step definitions (Ruby code) likely use captured string
  - “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)
The BDD Cycle in Rails

WRITE A SCENARIO

RUN THE SCENARIO

FAIL

IMPLEMENT A STEP DEFINITION

REFLECT ON REFACTORING AND VALIDATE WITH STAKEHOLDERS

WRITE A FAILING SPEC

WRITE CODE TO MAKE SPEC PASS

REFACTOR

STEP PASSES

https://semaphoreci.com/community/tutorials/applying-bdd-to-ruby-on-rails-web-applications
More on “Cuke”

• Need to install Cucumber Gem
  – Just for test and development environment, not for production environment
• When Cucumber installed, it creates commonly used step definitions
• Need a test database to run app
• Then edit `.features` file to add features
Fake User to Try Scenarios?

• Need tool that pretends to be the user to follow scenarios of user stories
• Capybara simulates browser
  – Can interact with app to receive pages
  – Parse the HTML
  – Submit forms as a user would

https://github.com/jnicklas/capybara
Which is FALSE about Cucumber and Capybara?

1. Step definitions are in Ruby, and are similar to method calls, while steps are in English and are similar to method definitions

2. A Feature has one or more Scenarios, which are composed typically of 3 to 8 Steps

3. Steps use Given for current state, When for actions, and Then for consequences of actions

4. Cucumber matches step definitions to scenario steps using regexes, and Capybara pretends to be a user that interacts with the SaaS app accordingly
Enhancing Rotten Potatoes Again

(Engineering Software as a Service § 7.8)
Add a *Real* New Feature?

- What if we add something harder?
  - e.g., includes form to fill in
  - e.g., needs a User Interface
  - e.g., needs to add route to connect view to controller
  - e.g., includes both a happy path and a sad path
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

- Figure 7.6 of Engineering Software as a Service
**Search TMDb User Story**

(Fig. 7.7 ESAAS)

**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
(Fig. 7.8 ESAAS)

{-# 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
Try Cucumber?

- If try Cucumber, it fails
- Missing the route
- Also MoviesController#search_tmdb is controller action that should receive form, yet not in movies_controller.rb
- Should use Test Driven Development to implement method search_tmdb
- Instead, to finish sad path, add fake controller method that always fails
• Add feature to search for movie in TMDb
  – Note: This will be a sad path, in that won’t find it
  – Will use fake method (until future when implement it using TDD)
• (Or can look at screencast: http://vimeo.com/34754766)
# add to routes.rb, just before or just after 'resources :movies':

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

http://pastebin/FrfkF6pd
Fake Controller Method:
Will Fail Finding Movie (Fig. 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
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 to make it DRY?
• **Background** means steps performed before *each* scenario
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 (happy path)
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"

---

http://pastebin/icQGrYCV
Cucumber Summary

• New feature => UI for feature, write new step definitions, even write new methods before Cucumber can color steps green
• Both happy and sad paths (don’t forget the sad paths!)
• Background lets us DRY out scenarios of same feature
• BDD/Cucumber test behavior; TDD/Rspec later is how write methods to make all scenarios pass
And in Conclusion

• Cucumber – “magically” maps 3x5 card user stories onto acceptance tests and integration tests for the application
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 …”
BDD

Good & Bad

• User stories - common language for all stakeholders, including nontechnical
  – 3x5 cards
  – LoFi UI sketches and storyboards

• Write tests before coding
  – Validation by testing vs. debugging

• Difficult to have continuous contact with customer?

• Leads to bad software architecture?
  – Will cover design patterns, refactoring in future
BDD

• 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
Which statement is FALSE about Lo-Fi UI and BDD?

1. The purpose of the Lo-Fi UI approach is to debug the UI before you program it

2. A BDD downside is requiring continuous contact with customers, which may not be possible

3. A BDD downside is that it may lead to a poor software architecture, since focus is on behavior

4. None are false; all three above are true