CODING DOJO

Practice Makes... Better

While I was preparing this meeting, a couple of things happened that have influenced the direction that I wanted to take this little talk.

1. The RSpec Beta Book from Prag Prog came out with their 3rd edition and it included a chapter on “the case for BDD”.

2. The company I was working for ran across some hard times that brought into stark reality the problems with not testing.

With these two things at the forefront of my mind, I thought I’d start off today with my own version of "the case for BDD", and how I see it applying to the real client-services world, which will lead us to starting the dojo.

HI

- My name is Dave Giunta
- I am a Graphic Designer turned Web Designer turned Ruby and Rails Developer
- In a career that spans so many job titles, I’ve seen all levels of the client / dev process
    - initial client meetings
    - design phases
    - development
    - management
- Having designed and built systems that start out great, but end up as a tangled mess, I began to look at testing and agile development.
- I've found that learning how to test is a frustrating experience:
    - makes me feel unsure I’m doing it right
    - makes me feel like I’m losing lots of productivity
    - Client projects can be a nightmare for a testing newb to test... because...
    - Clients don’t care enough about testing to pay for the extra time it takes for an inexperienced tester to test

CLIENTS

What we think about clients:
- Don’t know what they want
- Change their mind all the time
- Features are bullet points in 30 page RFPs
- Market-speak and buzzwords
- Want me-too features instead of business value

DEVELOPERS

Clients think:
- We speak in code and they don't get it
- We are terrible estimators—everything is possible and easy
- We're inflexible. When client's needs change, devs don't.

Loner:
 - more than one way to do something, but we prefer ours
 - why can't everyone code like me

Hero:
 - Build lots of functionality overnight
 - Quantity of code vs. Quality of code is an issue

Inconsistency of devs:
 - Most Ruby devs are newbs. Finding definitive experts is difficult.
 - PHP devs write Ruby like PHP. Java devs write Ruby like Java. etc.

#%$@ !

CLIENT FAIL - Managers quote hours based on their interpretation of the bad client RFP.

DEV FAIL - Developers start building alone, making design decisions based on management's assumptions of the feature.

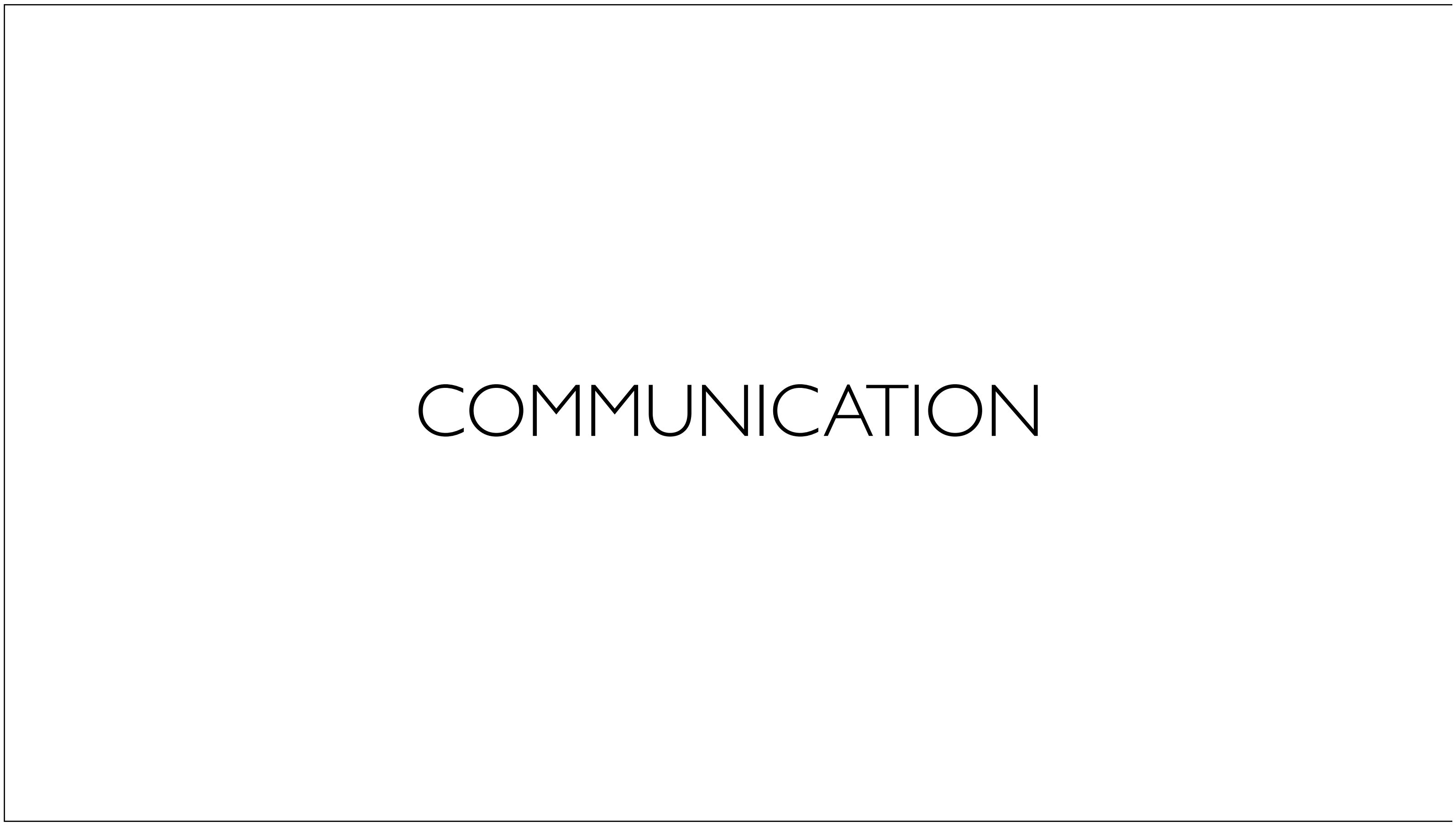
CLIENT FAIL - Client reviews after lots of work and decides devs didn't hit the mark.

DEV FAIL - Developers quickly change course in order to stay on target for launch. This creates regressions. And so, bug-fixing ensues!

DEV FAIL - Developers try valiantly to be heroes! Which leads to quick-fixes, which lead to more bugs, and an even more unstable system.

CLIENT FAIL - Client changes their mind about the feature again.

RESULT - Everyone is frustrated. Nobody's happy. What could we do to make this situation better?



COMMUNICATION

Wouldn’t it be great if:
- clients handed us features that were actual descriptions of what should happen instead of bullet points.
- we could execute these descriptions against our running system.
- no matter how developers write their code, they specify what it does in English first.

What might we end up with:
- client expectations that can actually be met.
- quotes that are slightly more accurate.
- Developers that have English documentation of what their code is doing.
- (hopefully) agility!
- (hopefully) happier clients and developers.

Cucumber:
- focuses on client communication through feature stories
- tries to ensure business value in every feature
- aligns client’s expectations with developer goals
- tests the outside of your app to the inside

RSpec:
- focuses on inter-team communication

test the inside of your app’s objects

PRACTICE

Testing is hard. Practice helps.

When you practice:
- Write tests that express the reality of what you're building, in as simple and declarative language as possible.
- Try to make your code and tests easy for the next person in line to understand.
- The process is more important than the outcome.
- Start small, and build to larger, harder problems.

Practice with your team:
- Smooth out experience levels across the team
- Creates a common approach to problems across the whole team.

CODING DOJO

- A pair at the keys at a time. One person writes specs, the other writes implementation
- Both the tester and the implementor should describe what they're doing / thinking aloud.
- Baby steps! The smallest amount of code to make the test pass as possible
- The group can offer suggestions for refactorings on a green bar.
  The pair can request silence on a red bar.
- Every 5 minutes, the tester should swap out with a new person,
  and the implementor should become the tester

RULES OF BOWLING

- There are 10 frames in a game. Each one provides two opportunities for a player to knock down 10 pins.
- Frame is marked "strike" if the player hits 10 pins on the first roll. To score a strike, you add the next two rolls to the current frame's pins (10).
  Example: Rolls = 10, 1, 1, Frame 1 score is 12, Frame 2 score is 2.
- Frame is marked "spare" if the sum of the player's 2 rolls is 10. To score a spare, you add the next 1 roll to the current frame's pins (10).
  Example: Rolls = 5, 5, 1, 0, Frame 1 score is 11, Frame 2 score is 1.
- A spare on the tenth frame allows 1 bonus roll. A strike on the tenth frame allows 2 bonus rolls.

For the purposes of this dojo, we don't care about individual frame score (ie - the code will never be expected to know at any given time what the score of Frame 5 is)

BASICS

Open up Code window to show basics

project setup and layout

rspec
- relationships (person.should have(2).parents)
- predicates (person.should be_awesome => person.awesome?)
- properties (person.full_name.should == 'Dave Giunta')

red green refactor

RED / GREEN / REFACTOR

Cucumber

Write out feature with client

See pending steps

🟡

Make steps real

See step fail

🔴

➡

RSpec

Write spec

See spec fail

🔴

Write code

See spec pass

🟢

Refactor code & spec

🔄

➡

Cucumber

See step pass

🟢

Refactor code, spec & steps

🔄

Cucumber Red Failure should trigger a reaction to go into RSpec
RSpec Green Passing Test should trigger a reaction to refactor
Refactored and Green Passing code should trigger a reaction to go back up to Cucumber
Cucumber Green passing should trigger a reaction to refactor
Refactored and Green Passing code should trigger a reaction to start over again.

CUCUMBER BASICS

becoming_awesome.feature

```
Feature: doing something awesome
  In order to be awesome
  As a non-awesome person
  I want to help someone out

  Scenario: Helping Old Ladies
    Given I am not awesome
    When I help an “old lady”
    Then they should be “happy”
      And I should be awesome
```

awesome_steps.rb

```
Given /^I (am|am not) awesome$/ do |boolean|
  @me = Person.new
  @me.awesome = boolean == ‘am’
end

When /^I help an “(.*)”$/ do |person|
  person.gsub!(/ /, ‘_’)
  @person = Person.new(:type => person.to_sym)
  @me.help(@person)
end

Then /^they should be “(.*)”$/ do |emotion|
  @person.status.should == emotion.to_sym
end

Then /^I should be awesome$/ do
  @me.should be_awesome
end
```

- Plain text input through Ruby bridge

Story structure:

In order to <business value>
As a <role>
I want to <execute feature>

Scenario structure:

Given <context>
When <action>
Then <outcome>
And takes continues the previous statement

RSPEC BASICS

game_spec.rb

```
describe Person do
  before(:each) do
    @person = Person.new
  end

  context “when first created” do
    it “should not be awesome” do
      @person.should_not be_awesome
    end
  end
end
```

Expectations

Equality:

```
  @person.name.should == “Dave”
```

Predicates (? boolean methods):

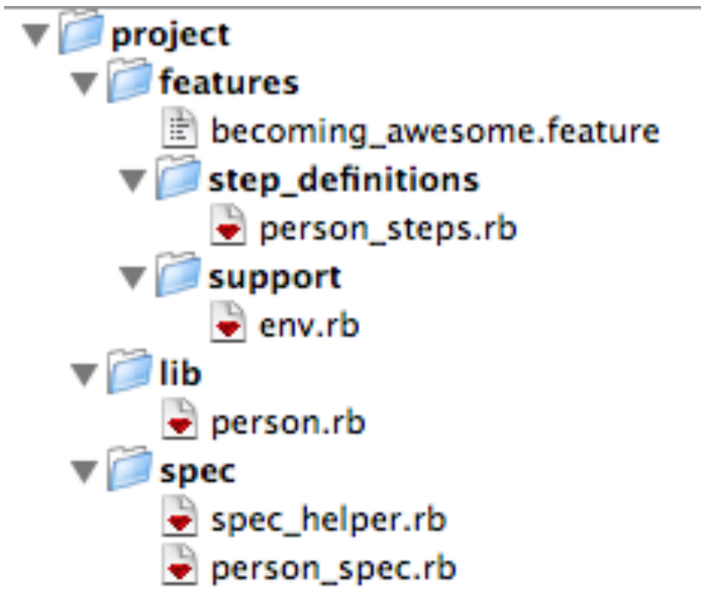
```
  @person.should be_awesome
    => @person.awesome? == true
```

Collections:

```
  @person.should have(0).siblings
    => @person.siblings.length == 0
```

“describe” and “context” are grouping mechanisms
“it” wraps a specification with the english language of what it should do
“before(:each or :all)” and “after(:each or :all)” set up shared context for any group (describe / context)

PROJECT STRUCTURE



```
project
  features
    becoming_awesome.feature
    step_definitions
      person_steps.rb
    support
      env.rb
  lib
    person.rb
  spec
    spec_helper.rb
    person_spec.rb
```

features/ - holds everything about cucumber
features/*.feature - a cucumber plain-text feature file
features/step_definitions/ - holds any step definition files that are required to run the feature files
features/step_definitions/*_steps.rb - step definition file
features/support/ - holds any environment setup files or custom matchers that will be used in the cucumber features
features/support/env.rb - this file, at the very least, needs to require all the code in the lib directory that will be used in the step definition files

lib/ - where your actual code will go

spec/ - holds everything about rspec. This directory should mirror the file and directory structure of the lib/ directory
spec/spec_helper.rb - the helper file that will get included in all of the _spec.rb files and require any code in the lib/ directory
spec/*_spec.rb - an rspec spec file containing tests about the code

THANKS!

Dave Giunta
dgiunta@mac.com
dgiunta on twitter and github

Coding Dojo Files:
http://github.com/dgiunta/chicagoruby.org-4-18-coding-dojo/