ATDD WORKSHOP

Hands On Experience

Presented by: Joseph Ours

Agenda

Should be Done Prior to Session

- Install Ruby via RailsInstaller
- Install Cucumber, Watir, TestGen,
- Ensure you have Chrome installed!

ATDD Automation Overview

- Key Things to Know

Drive a Browser

- Watir Basics
- Launch a browser
- Interact with Controls
- Complete a test

Leverage Cucumber

- Cucumber Basics
- Create a Feature, Scenario
- Create Outlines

Key Things to Know

Automation is hard

- Tests are slow
- Hard to write through UI
- Need to follow OO and developer practices
- Tends to be brittle
- Doesn't scale well
- It WILL fail!

But Automation is becoming more and more necessary

- Increasing regression collateral with the advent of Agile
- Can be a time saver for tedious tasks
- Is great for binary checking (pass/fail)

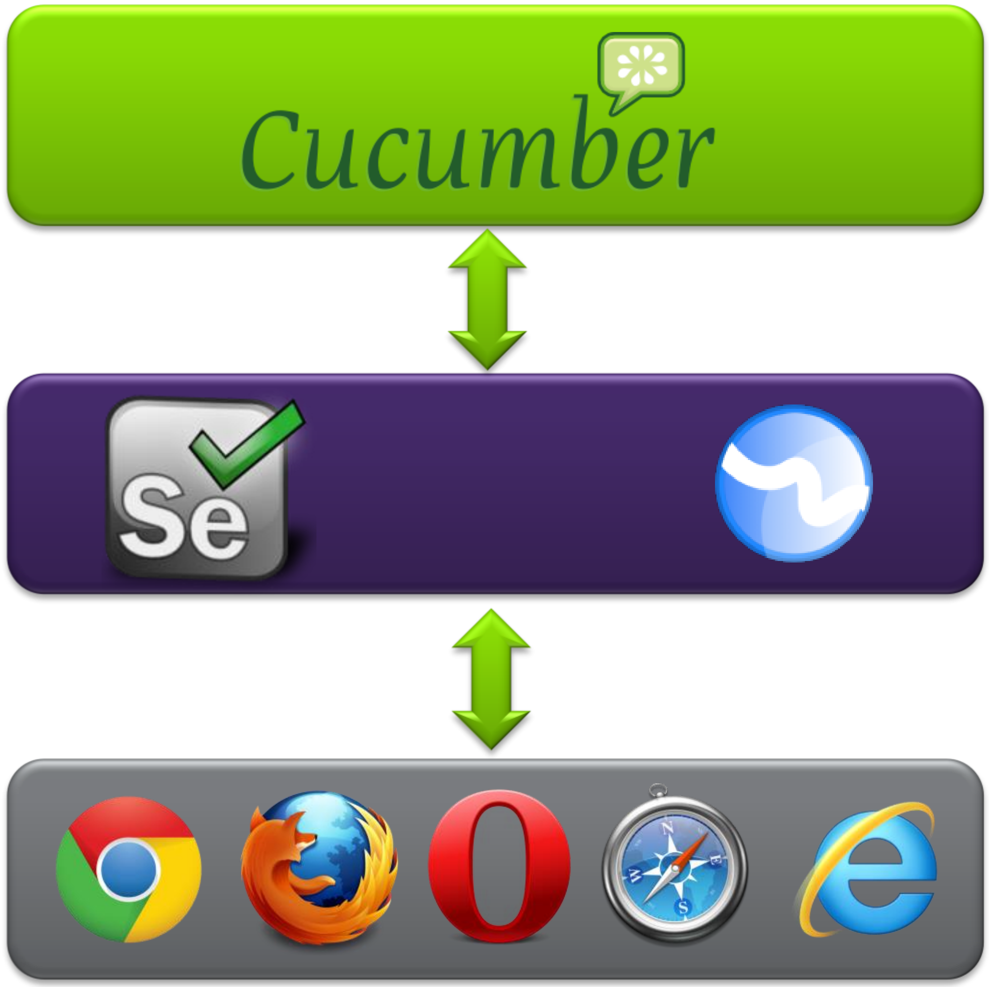
Not everything that can, should be automated

- 'nuf said

MY rules of thumb

- Run it 7 times or don't automate it
- No more than 40% of your tests ever can or should be automated

Technology Stack



UI Automation Architecture

Driver

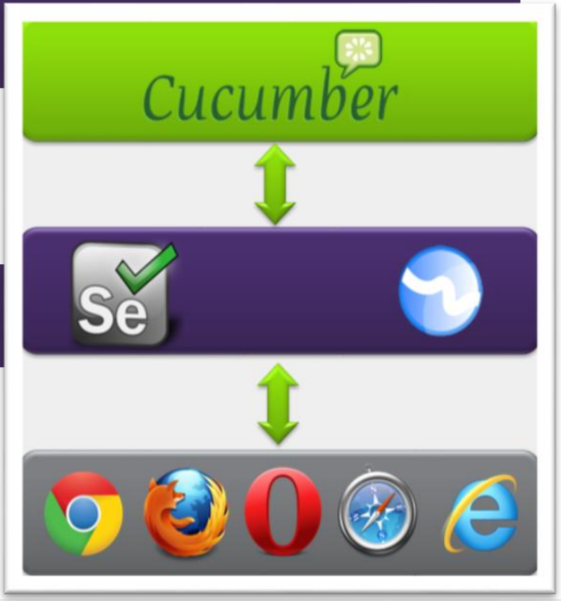
- Interacts with the browser (Selenium, Watin, Watir, etc... [Mobile Ones?])
- Talks to application/browser
- Even commercial tools have drivers
- Does not how to do testing [run/start a test, do comparisons, do reports do asserts, etc....]

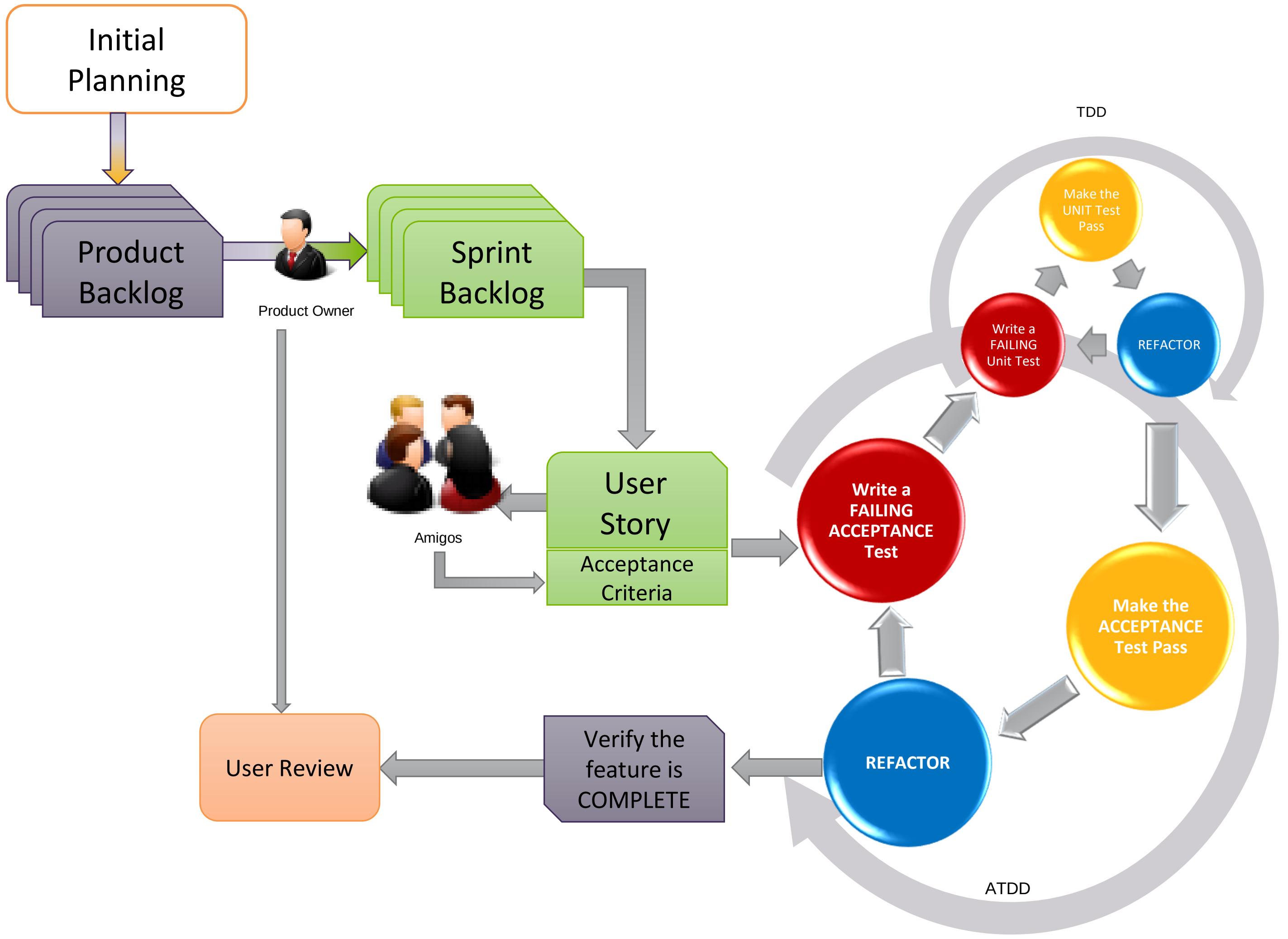
Test Framework

- Executes test - communicates with driver
- RSpec, JUnit, F#'s is Canopy (Over Selenium)

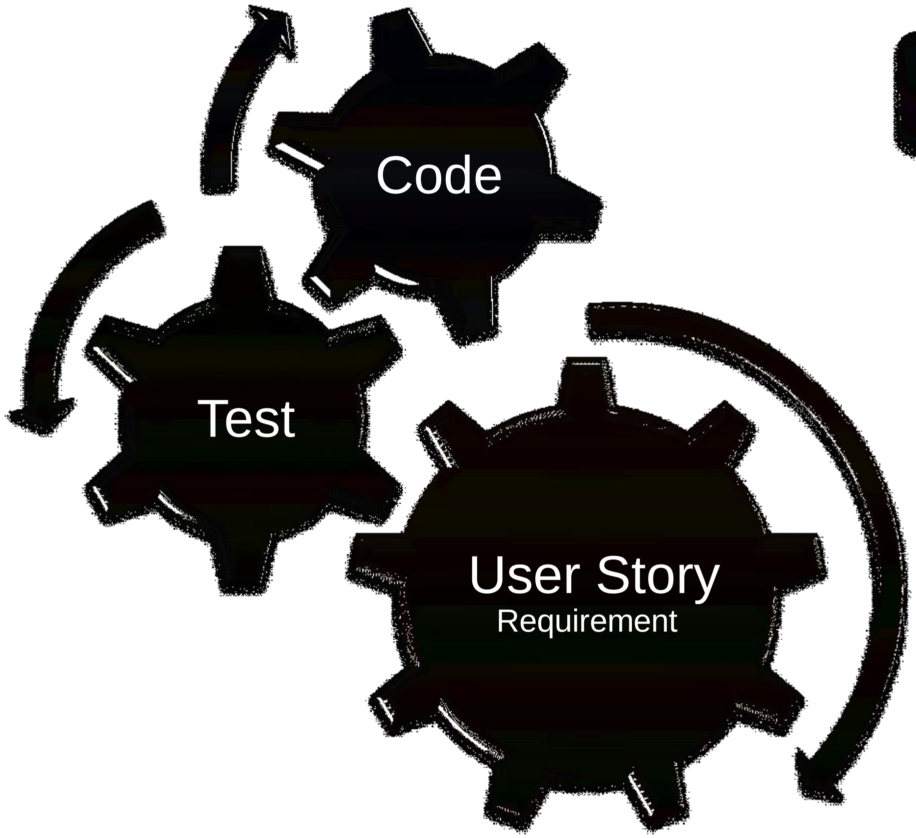
Automation Framework

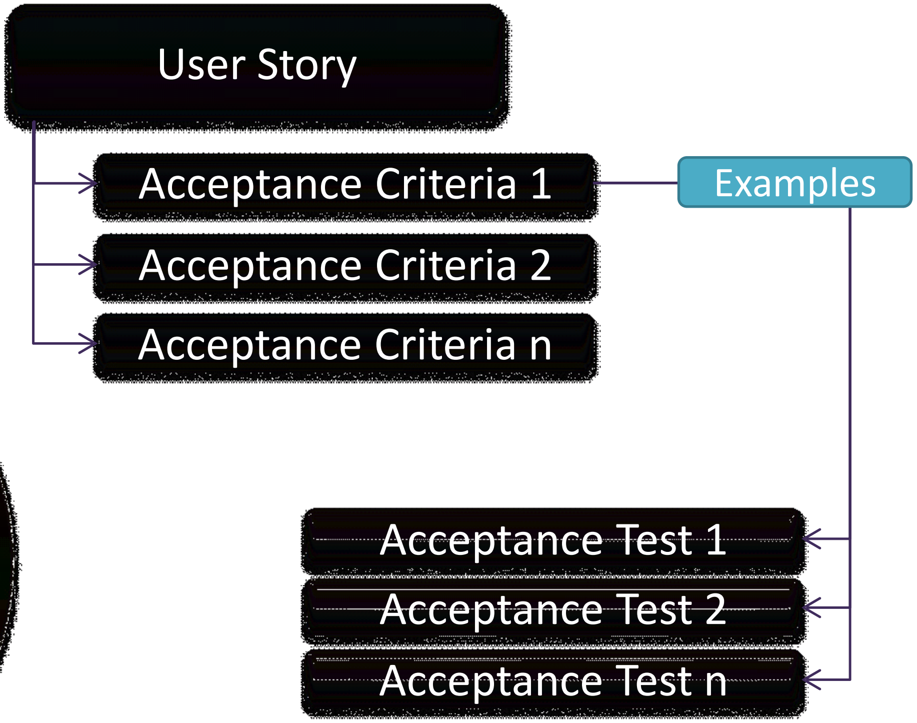
- Bridge gap between English and Test Framework
- Cucumber, Fit/FitNesse
- Allows people to communicate with tests

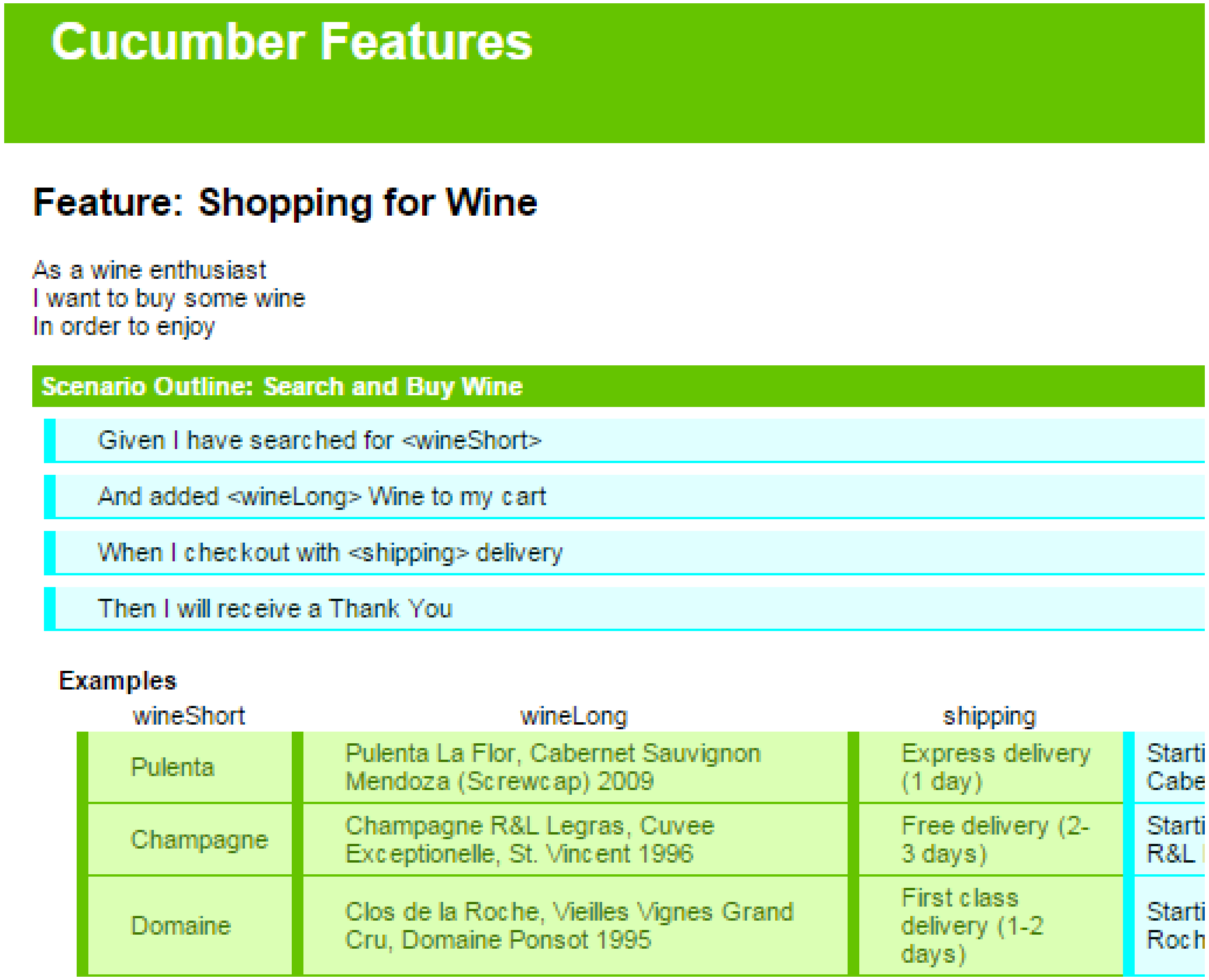



Initial Planning
Product Backlog
Product Owner
Sprint Backlog
Amigos
User Story
Acceptance Criteria
TDD
Make the UNIT Test Pass
Write a FAILING Unit Test
REFACTOR
Write a FAILING ACCEPTANCE Test
Make the ACCEPTANCE Test Pass
REFACTOR
ATDD
Verify the feature is COMPLETE
User Review

ATDD INTERACTIONS







Cucumber Features

Feature: Shopping for Wine

As a wine enthusiast
I want to buy some wine
In order to enjoy

Scenario Outline: Search and Buy Wine

Given I have searched for <wineShort>

And added <wineLong> Wine to my cart

When I checkout with <shipping> delivery

Then I will receive a Thank You

Examples

wineShort	wineLong	shipping	
Pulenta	Pulenta La Flor, Cabernet Sauvignon Mendoza (Screwcap) 2009	Express delivery (1 day)	Starti Cabe
Champagne	Champagne R&L Legras, Cuvee Exceptionelle, St. Vincent 1996	Free delivery (2-3 days)	Starti R&L
Domaine	Clos de la Roche, Vieilles Vignes Grand Cru, Domaine Ponsot 1995	First class delivery (1-2 days)	Starti Roch

Let’s Get Started

Driving a Browser

Setting Up Our First Testing Project

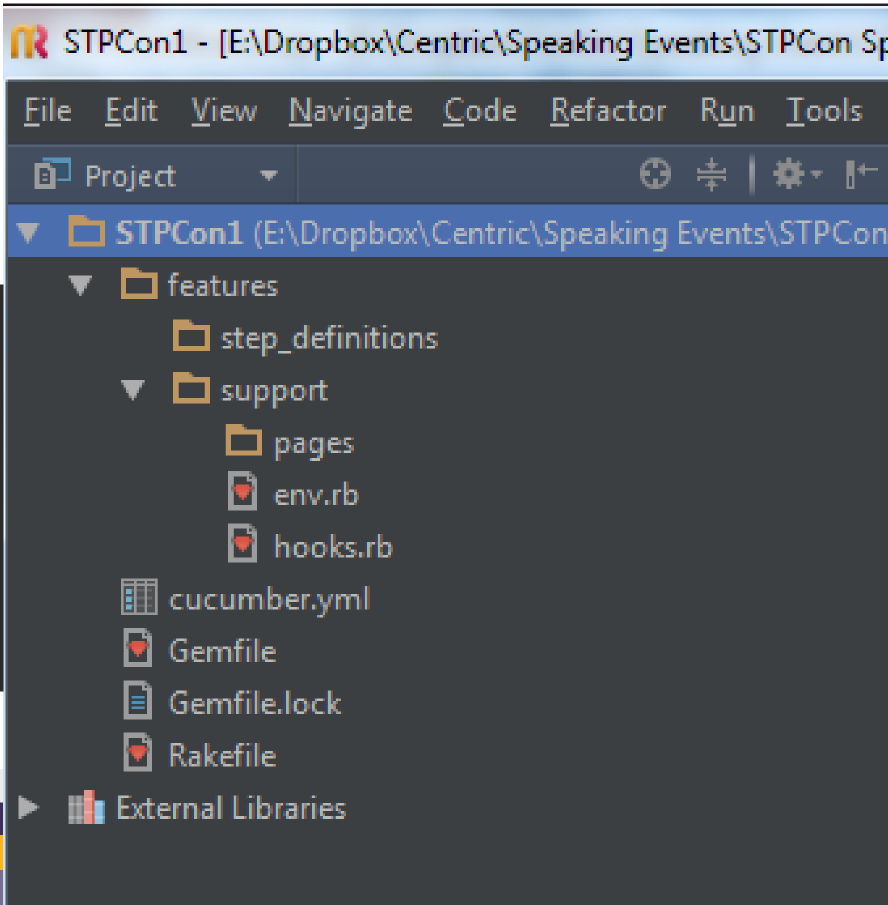
Leverage a Plug-in “TestGen” to create a project for us!

- Open COMMAND Prompt to your desired project location
- Type `testgen project LETSTEST --pageobject-driver=watir-webdriver`
- This will create our folder structure

- Open env.rb and replace with

```ruby
require 'rspec'
require 'rubygems'
require 'watir'

Before do |scenario|
  @browser = Watir::Browser.new :chrome
end
After do |scenario|
  @browser.close #closes the browser instance
end
```


STPCon1 - [E:\Dropbox\Centric\Speaking Events\STPCon Sp
File Edit View Navigate Code Refactor Run Tools
Project

```
STPCon1 (E:\Dropbox\Centric\Speaking Events\STPCon
  features
    step_definitions
    support
      pages
      env.rb
      hooks.rb
  cucumber.yml
  Gemfile
  Gemfile.lock
  Rakefile
External Libraries
```

General Syntax

```
object.element(:attribute=>’value').action ‘VALUE’
```

```
Examples
@browser.link(:text=>'Add to Basket').click
@browser.select_list(:class=>’mylist').select '2‘
```

Typical Commands

Browser Commands

```
# start new driver session
b = Watir::Browser.new :firefox
b = Watir::Browser.new :chrome
b = Watir::Browser.new :ie

# goto url
b.goto "http://www.letstest.com"

# refresh
b.refresh

# close
b.quit
```

TextBox Interactions

```
# enter value
b.text_field(:id => "text").set “Lets Test"

# get value
b.text_field(:id => "text").value

# clear
b.text_field(:id => "text").clear
```

Listbox Interactions

```
# select from list text
b.select_list(:id => "list").select "var"

# select using value
b.select_list(:id => "list").select_value "var2"

# value is selected?
b.select_list(:id => "list").selected?("var2")

# get value
puts b.select_list(:id => "list").value

# get all items
b.select_list(:id => "list").options.each do |i|
  puts "#{i.text}"
end
```

Image Interactions

```
is image loaded?
b.image(:src => "img.gif").loaded?

# height
b.image(:src => "img.gif").height

# width
b.image(:src => "img.gif").width

# click
b.image(:src => "img.gif").click

# click 1st image
b.images[0].click
```

Typical Commands

Button Interactions

```
# is enabled?
b.button(:id => "btn").enabled?

# button's text
b.button(:id => "btn").text

# click
b.button(:id => "btn").click
```

Checkbox Interactions

```
# check
b.checkbox(:id => "btn").set
b.checkbox(:id => "btn").set(true)

# uncheck
b.checkbox(:id => "btn").clear
b.checkbox(:id => "btn").set(false)

# is checked?
b.checkbox(:id => "btn").set?
```

Radio Interactions

```
# select value
b.radio(:id => "radio").set

# is var selected?
b.radio(:id => "radio").set?
```

DIV Interactions

```
# get text
b.div(:class => "body").text
# get text of 2nd div when it appears
b.divs[1].when_present.text
```

Table Interactions

```
# row 1, col 1
b.table(:id => "table")[0][0].text

# row 1, col 2 (alternate)
b.table(:id => "table").tr{0}.cell{1}.text

# row 2 - entire text
puts b.table(:id => "table")[1].text

# click row #4
puts b.table(:id => "table")[3].click

# get column count
b.table(:id => "table").row.cells.length

# row count
b.table(:id => "table").row_count
b.table(:id => "table").rows.length
```

Waits

```
# [wait_until_present]
b.button(:id => "btn").wait_until_present

# [when_present]
b.button(:id => "btn").when_present.click
b.button(:id => "btn").when_present(10).click

# [wait_while_present]
b.button(:value => "submit").click
b.button(:value => "submit").wait_while_present
```

General Tips

General Tips

```
# [exists?]
b.text_field(:id => "text").exists?

# [enabled?]
b.select_list(:id => "list").enabled?

# [present?]
b.element(:id => "e").present?

# [tag_name]
b.element(:id => "e").tag_name

# [screenshot]
b.screenshot.save "c:\\page.png"

# [to_subtype] # returns button
b.element(:id => "btn").to_subtype

# [index] click 2nd image on page
b.image(:index => 1).click

# [loops]
# get names of all text-fields
b.text_fields.each do |i|
  puts i.name
end

# get name of first text-field
puts b.text_fields[0].name

# get name of second text-field
puts b.text_fields[1].name
```

Driving a Browser

Goal!

- Launch Browser/Navigate to a page
- Search for a product
- Add it to the cart
- Start Checkout
- Enter Shipping Info
- Enter Payment Info
- Receive Confirmation

Layering BDD

What is Behavior-Driven Development (BDD)?

- Dan North’s response to issue encountered teaching TDD.
    - Where to start? What to test? How much to test? Tests are not tied to a business need.

- BDD is about implementing an application by describing its behavior from the perspective of its stakeholders.

- BDD focuses on obtaining a clear understanding of desired software behavior through discussion with stakeholders.

- A communication and collaboration framework for developers, QA, and non-technical or business participants in a software project.

- Consists of 3 components:
    - Test-Driven Development (TDD)
    - Domain-Driven Design (DDD)
    - Acceptance Test Driven Development(ATDD)

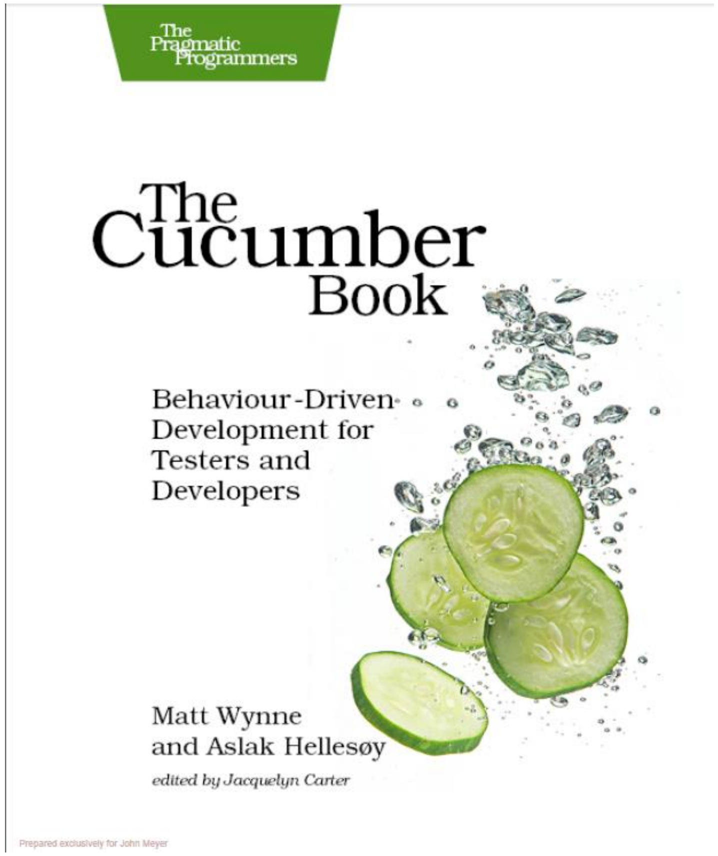
The Cucumber Framework

An automated testing framework created by Aslak Hellesoy in 2008.

Based on Behavior-Driven Development.

Gherkin Scripting uses "real-world examples to describe the desired behavior of the system we want to build"

This enables teams to "stay grounded in language and terminology that makes sense to our stakeholders: *we're speaking their language.*"



The Pragmatic Programmers

The Cucumber Book

Behaviour-Driven Development for Testers and Developers

Matt Wynne and Aslak Hellesøy

edited by Jacquelyn Carter

Prepared exclusively for John Meyer

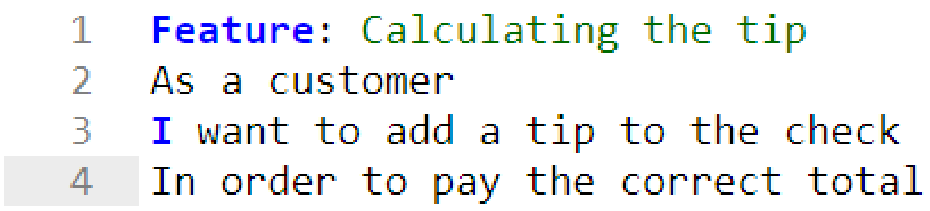
Composed of 2 Main Parts

- Feature
  - Something the user will enjoy

- Scenario
  - Objective that details out the feature (how you know your done)

Cucumber – Gherkin Feature Synta

- Features are:
  - A narrative
  - It is NOT executable
  - Describes BUSINESS VALUE
  - Something the user will enjoy when we are done developing



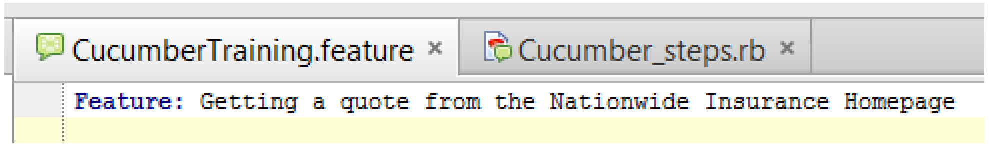
```
Feature: Calculating the tip
As a customer
I want to add a tip to the check
In order to pay the correct total
```

- Feature: <Title>
  - As a <type of user/role>
  - I want <perform some action>
  - In order to <achieve a goal – related to business value>

- This should always answer WHY. If it doesn't, it's probably not needed.
  - Best WHY's related to increasing revenue, protecting revenue, and managing costs

.feature files

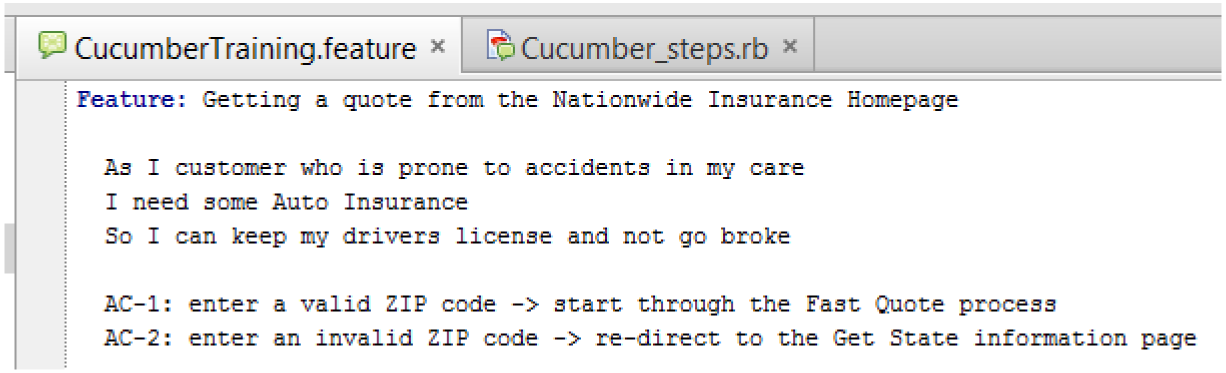
- All Gherkin Scripts go into .feature files
- Feature files must start with the keyword “Feature:”
- You should have one feature file for each
- User Story the team creates



```
Feature: Getting a quote from the Nationwide Insurance Homepage
```

- In the space between the Feature line and your 1st scenario is a comment section. You can write anything

- A good spot for (these are not required, just recommended):



```
Feature: Getting a quote from the Nationwide Insurance Homepage

  As I customer who is prone to accidents in my care
  I need some Auto Insurance
  So I can keep my drivers license and not go broke

  AC-1: enter a valid ZIP code -> start through the Fast Quote process
  AC-2: enter an invalid ZIP code -> re-direct to the Get State information page
```

- User Story
- Acceptance Criteria
- Testing design notes
- Risk analysis
- Development design notes
- Any concerns team members may have about a specific area of code

Cucumber Feature→Scenarios

Format of:

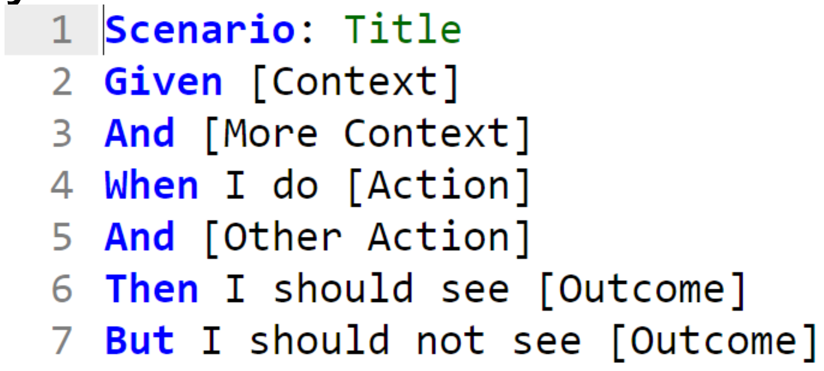
Scenario – Title

Given – Pre-requisites

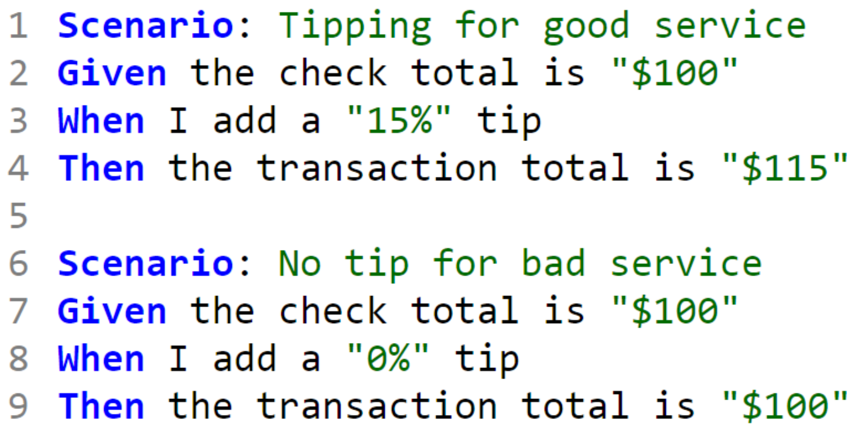
When – Action Step

Then – Results

Scenario's are your detailed objectives – how you will know you've delivered the feature



```
Scenario: Title
Given [Context]
And [More Context]
When I do [Action]
And [Other Action]
Then I should see [Outcome]
But I should not see [Outcome]
```



```
Scenario: Tipping for good service
Given the check total is "$100"
When I add a "15%" tip
Then the transaction total is "$115"

Scenario: No tip for bad service
Given the check total is "$100"
When I add a "0%" tip
Then the transaction total is "$100"
```

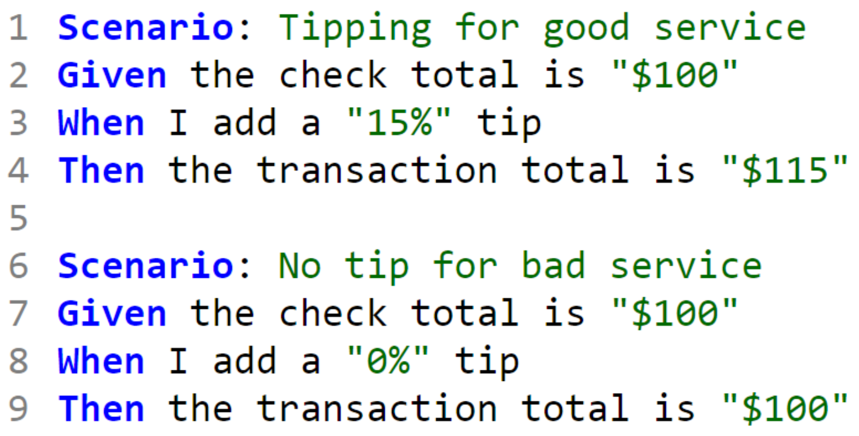
Scenarios

- Scenarios are the individual test cases to be used to prove the Acceptance Criteria have been met.

- You could have one, you could have many for each Acceptance Criteria.
- There are no limits on the number of Scenarios in a Feature file
  - Use Risk to determine depth of script coverage
  - If the number of Scenarios in a Feature file is becoming large, think about separating them into multiple Feature files

- Length of each Scenario does not matter
  - Key: one test per Scenario

- Syntax:
- Scenario: <scenario title>
- <Gherkin Script>



```
Scenario: Tipping for good service
Given the check total is "$100"
When I add a "15%" tip
Then the transaction total is "$115"

Scenario: No tip for bad service
Given the check total is "$100"
When I add a "0%" tip
Then the transaction total is "$100"
```

What are Gherkin Scripts?

Gherkin Scripts: connects the human concept of **cause and effect** to the software concept of **input/process/output**.

> **Given – indicates something that we accept to be true in a scenario**
> **When – indicates the event in a scenario**
> **Then – indicates an expected outcome**

Can also use “And” and “But” steps to help make a Given, When, or Then more descriptive

- In the “language of the domain” – scripts are written using the language, terms and definitions of the domain.

- It is “stakeholder readable” – if a script is written for business acceptance it should be written so the business users can understand it and follow what is happening. If a script is written for an Integration Test of a Web Service it should be written so the developers and testers of the Web Service can understand it and follow what is happening.

Given

The purpose of Given steps is to put the system in a known state before the user (or external system) starts interacting with the system (in the When steps)

Indicates something that we accept to be true in a scenario

Sometimes know as preconditions

GREAT place to set-up test data

Hint: no functionality should be changing in the Given step. So…you should rarely have an error in the Given steps.

> Examples:

```
Given the Online Baking User logs into Online Banking
And navigates to the Account History page
And selects a Premier Checking account

 Given I am entitled to Initiate Payment And Payee Maintenance
   And I have 1 Bill Pay account
   And I have accounts entitled to Bill Pay
   And I have a funding account
```

When

The purpose of When steps is to describe the key action the user performs

It is what the business is requesting
It is what the developers are developing
It is what the testers are testing

The MONEY step: it is what everyone is being paid
to build & test

> **Examples:**
> When the user Logs in
> When I view the Pay Bills tab
> When I select a Bill Pay account

Then

- The purpose of Then steps is to observe outcomes
    - The expected results of the When statement

- The observations should be related to the business value/benefit in your feature description

- The observations should also be on some kind of *output* – that is something that comes *out* of the system (report, user interface, message) and not something that is deeply buried inside it (that has no business value)

    > Examples:

    Then I will see a highlighted Pay Bills tab

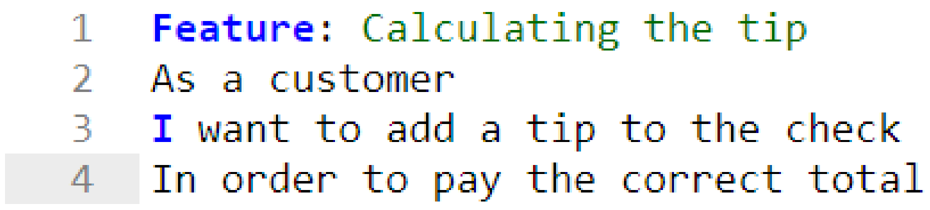
    And I will see a header that reads "Enter Payment Criteria“

    And I will see a table containing payment method icons to the left of each transaction

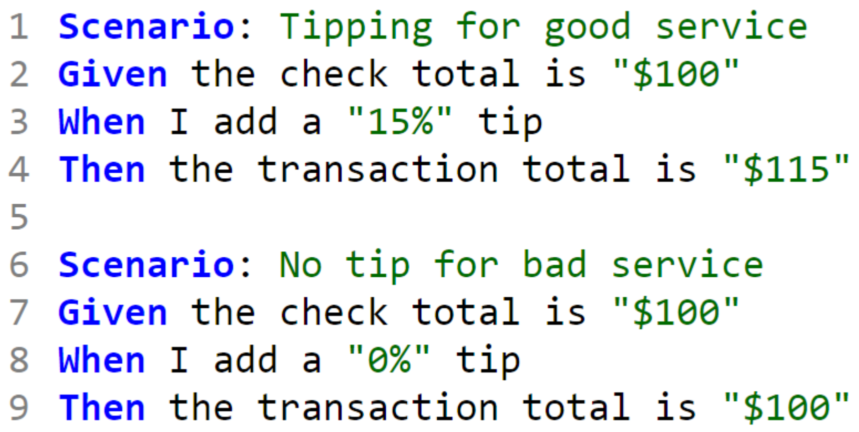
    Then the too many characters error message appears

A word about Features and Scenari

- Feature
  - Something the user will enjoy
- Scenario
  - Objective that details out the feature (how you know your done)



```
Feature: Calculating the tip
As a customer
I want to add a tip to the check
In order to pay the correct total
```



```
Scenario: Tipping for good service
Given the check total is "$100"
When I add a "15%" tip
Then the transaction total is "$115"

Scenario: No tip for bad service
Given the check total is "$100"
When I add a "0%" tip
Then the transaction total is "$100"
```

- This is no EASY answer as to how high level a feature should be
  - Single benefit
  - Single activity
  - Single Story

Work breakdown is the HARDEST part

Leverage Tables for Data

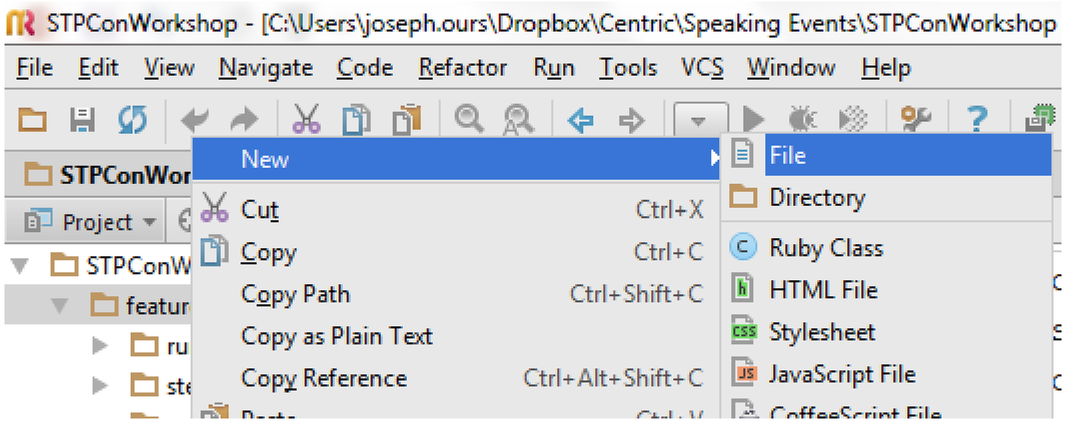
Scenario outlines allow us to express examples through a template with placeholders. These use Cucumber keywords `Scenario Outline`, `Examples`, and `<>`.

```
Scenario Outline: eating
Given there are <start> cucumbers
When I eat <eat> cucumbers
Then I should have <left> cucumbers
Examples:
| start        | eat          | left         |
| 12           | 5            | 7            |
| 20           | 7            | 13           |
```

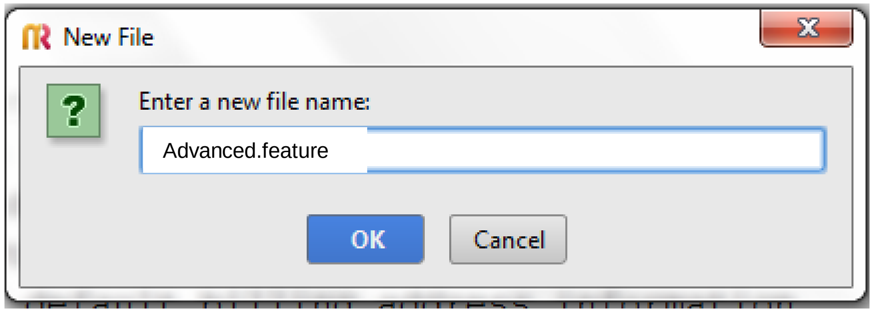
This will change your step definition to allow for the passing of values. Let's try one out

Create a New Feature File

1. Select the Feature Folder
2. Right Click
3. Select New
4. Select File


STPConWorkshop - [C:\Users\joseph.ours\Dropbox\Centric\Speaking Events\STPConWorkshop
File Edit View Navigate Code Refactor Run Tools VCS Window Help
STPConWor
Project
STPConW
featur
ru
ste
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy as Plain Text
Copy Reference Ctrl+Alt+Shift+C
File
Directory
Ruby Class
HTML File
Stylesheet
JavaScript File
CoffeeScript File

1. Enter FileName – like advanced.feature


New File
Enter a new file name:
Advanced.feature
OK
Cancel