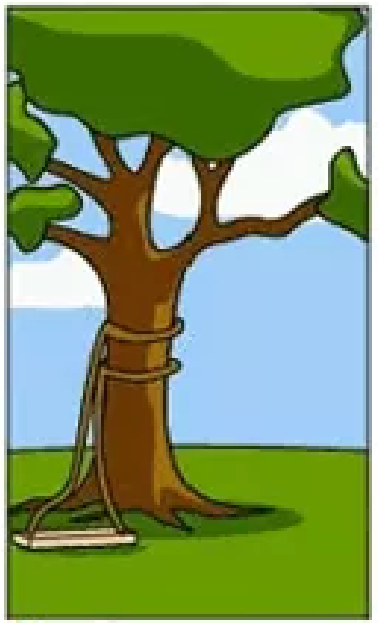
BDD & Cucumber

Behavior Driven Development

How the customer explained it

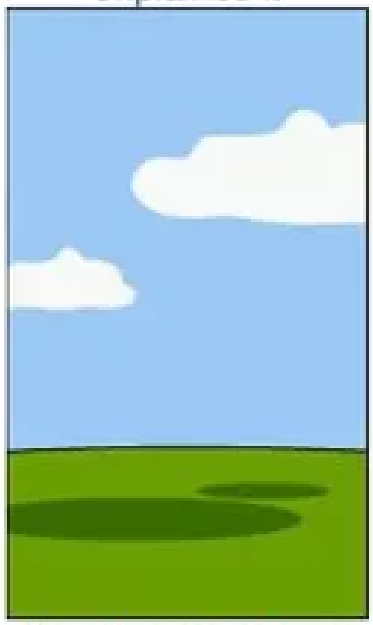
How the project leader understood it

How the engineer designed it



How the programmer wrote it

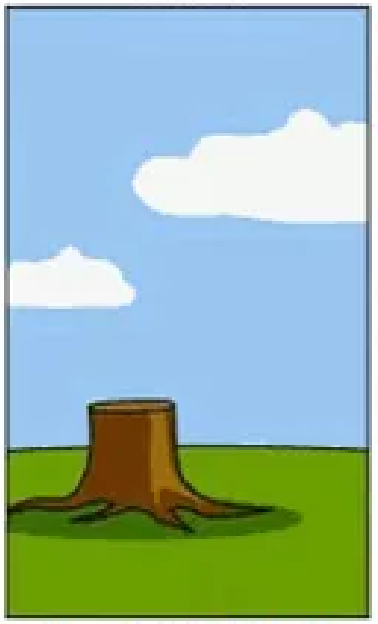
How the sales executive described it



How the project was documented

What operations installed

How the customer was billed



How the helpdesk supported it

What the customer really needed

Behavior Driven Development

What if we could write software in a way that would let us discover and focus our efforts on what really matters?

BDD Aspects

BDD helps teams:

Focus their efforts on identifying, understanding, and building valuable features that matter to businesses

Makes sure that these features are well designed and well implemented.

BDD Practitioners

BDD practitioners:

Use conversations around concrete examples of system behavior to help understand how features will provide value to the business

BDD Collaboration

BDD tools can help:

Turn these requirements into automated tests

Guide the developer verify the feature

Document what the application does

What BDD is Not.

BDD is not:

A software development methodology.

A replacement for:

- Scrum
- XP
- Kanban
- RUP

BDD Does Enhance Process

BDD enhances your existing process by:

Incorporating, building on, and enhancing ideas from many of these methodologies

BDD by Excruciating Means

1. Business Analyst has requirements
2. Requirements set on word processor
3. Developer translates into code and unit tests
4. QA then creates tests that verify the requirements
5. Code is retranslated into documentation

BDD By Easiness

1. Business Analyst, QA, Developer share a list of requirements, and work on features.
2. Developer uses BDD tool (JBehave, RSpec, Cucumber) into automated tests that verify that a feature is complete
3. QA uses the completed test for further testing
4. The BDD Tests are low-level documentation
5. The BDD Tests are also a checklist of features

BDD

1. Shared goals
2. Avoiding YAGNI*
3. Understand Stakeholders Goals
4. Manage Uncertainty
5. Reduce Risk

* You Ain't Gonna Need It

BDD History

Invented by Dan North

Done as an extension of TDD

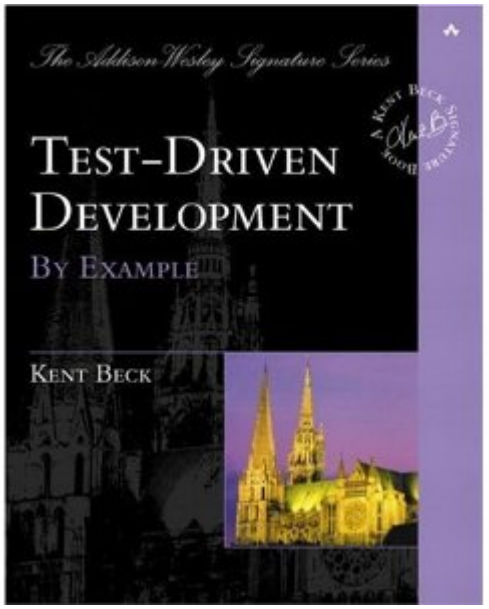
Developed circa 2000

TDD History

Started as a migration from Smalltalk

Kent Beck

Test First



TDD Kent Beck Recipe

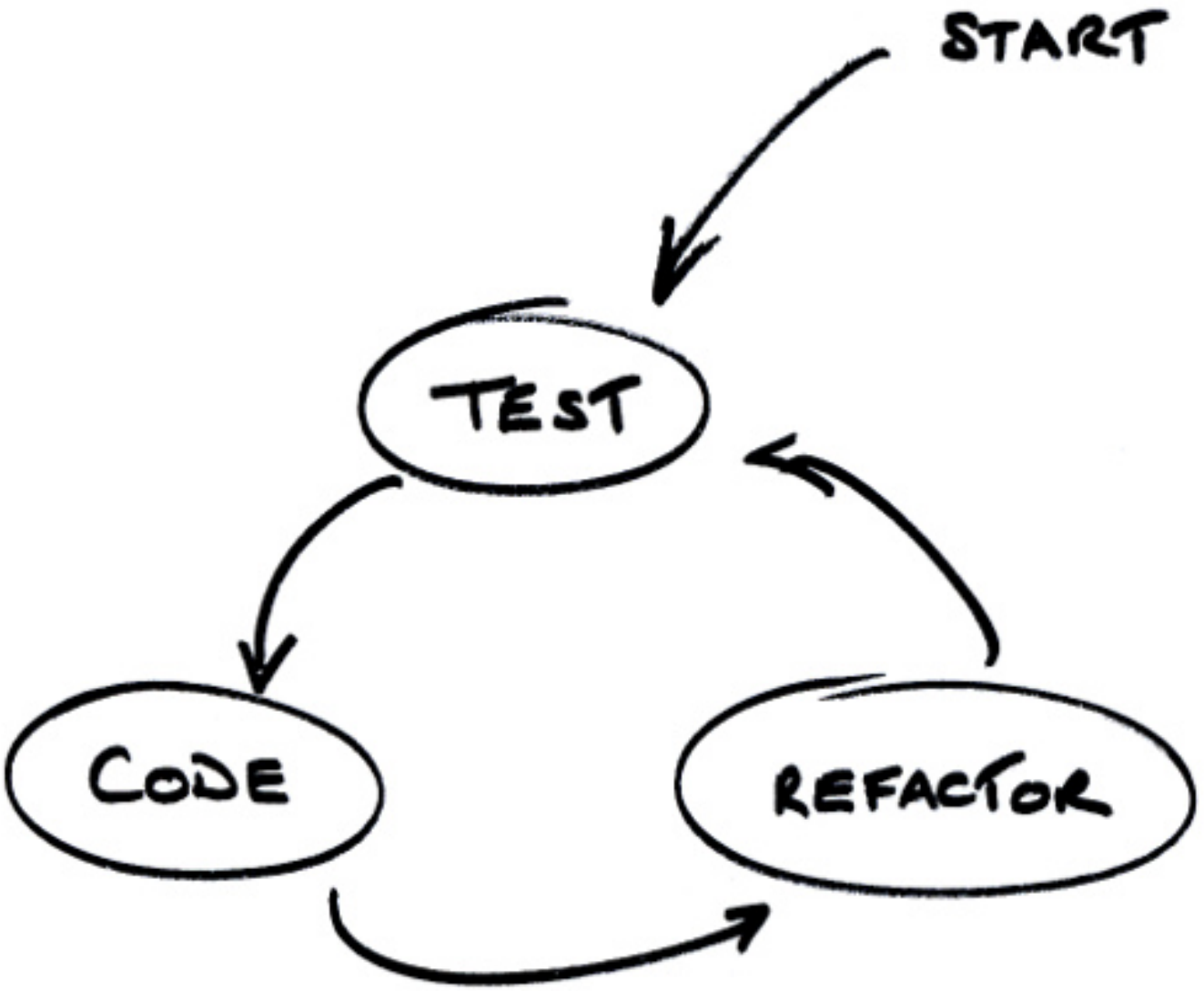
1. Quickly add a test.
2. Run all tests and see the new one fail.
3. Make a little change.
4. Run all tests and see them all succeed.
5. Refactor to remove duplication.

Kent Beck – Test Driven Development By Example 2003

TDD Ford Recipe

1. Write a failing test.
2. Write code to make it pass.
3. Repeat steps 1 and 2.
4. Along the way, refactor aggressively.
5. When you can't think of any more tests, you must be done.

Neal Ford – Evolutionary Architecture and Emergent Design 2009


START
TEST
CODE
REFACTOR

TDD vs. BDD

Rather than make the API the goal
...you make the business the goal

TDD is *often* coupled with BDD

Back to BDD

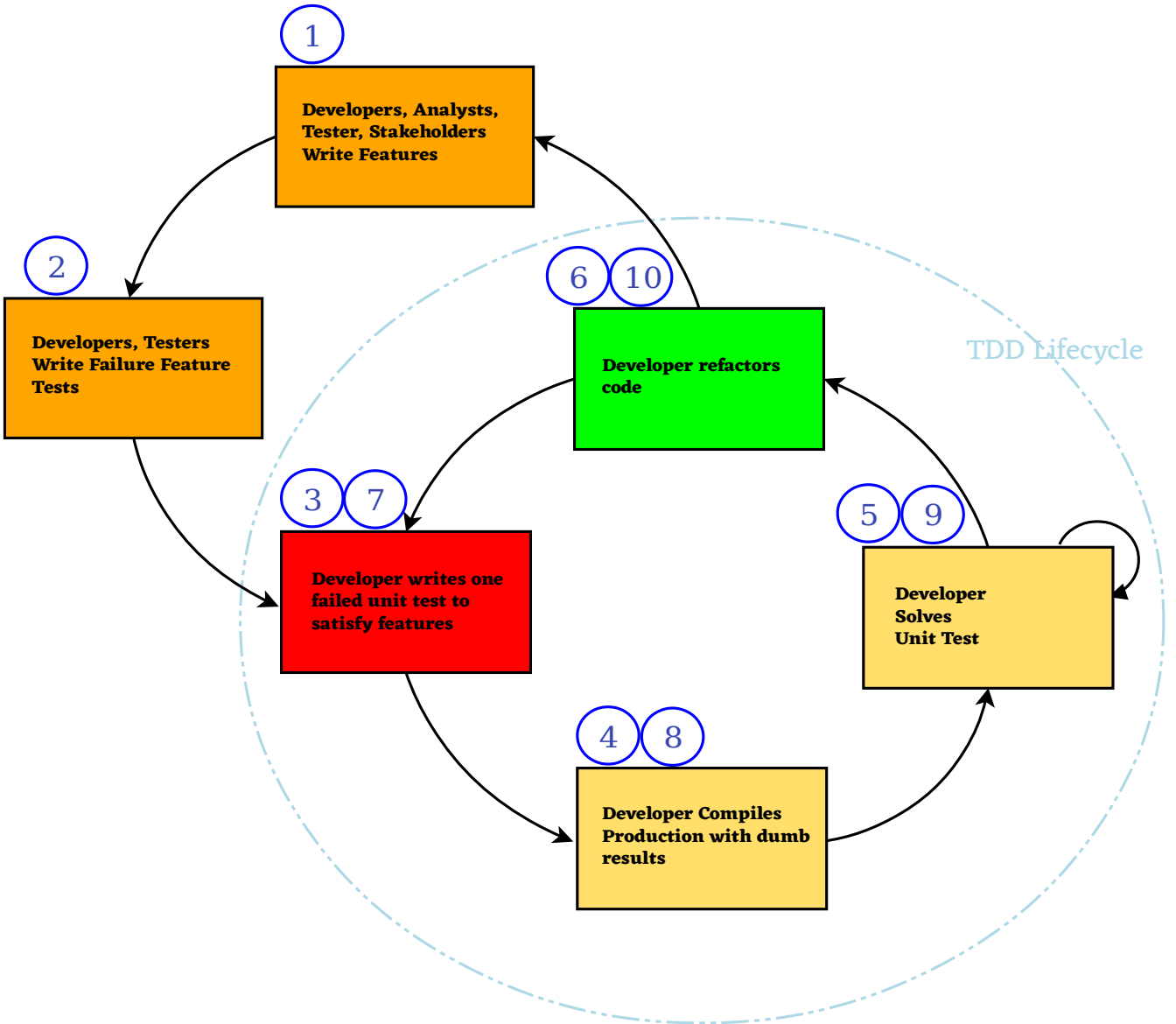
Dan North North observed that a few simple practices...

"Should" can help developers write more meaningful tests

Focus on what the class should do.

Focus on the behavior of the applications

BDD By Diagram



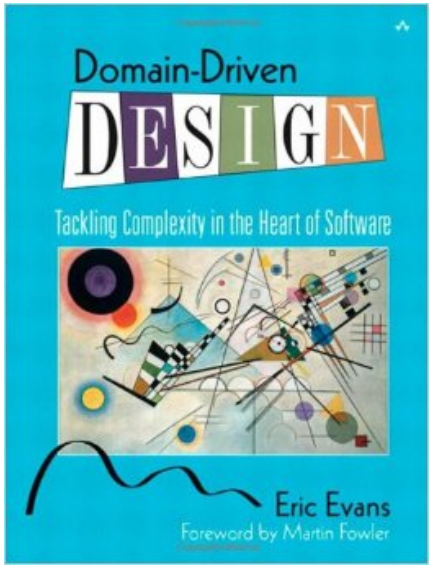
TDD: Fast Example

BDD, DDD, & Gherkin

DDD & Gherkin

Eric Evans

- Domain Driven Design
- Developed a business language....
- Something that business analysts can use unambiguously
- The language is called *Gherkin*



Gherkin by Simple Example

Given a insurance policy

When a customer renews the policy

Then the policy must refer to record of payment

And the policy must update the expiration to one year from the current date

Gherkin with Scenarios!

Scenario: Wilson posts to his own blog
  Given I am logged in as Wilson
  When I try to post to "Expensive Therapy"
  Then I should see "Your article was published."

Scenario: Wilson fails to post to somebody else's blog
  Given I am logged in as Wilson
  When I try to post to "Greg's anti-tax rants"
  Then I should see "Hey! That's not your blog!"

Scenario: Greg posts to a client's blog
  Given I am logged in as Greg
  When I try to post to "Expensive Therapy"
  Then I should see "Your article was published."

http://docs.behat.org/en/v2.5/guides/1.gherkin.html

Gherkin Scenarios

Feature: Feedback when entering invalid credit card details

In user testing we've seen a lot of people who made mistakes entering their credit card. We need to be as helpful as possible here to avoid losing users at this crucial stage of the transaction.

Scenario: Credit card number too short

Given I have chosen some items to buy

And I am about to enter my credit card details

When I enter a card number that's only 15 digits long

And all the other details are correct

And I submit the form

Then the form should be redisplayed

And I should see a message advising me of the correct number of digits

Scenario: Expiry date must not be in the past

Given I have chosen some items to buy

And I am about to enter my credit card details

When I enter a card expiry date that's in the past

And all the other details are correct

And I submit the form

Then the form should be redisplayed

And I should see a message telling me the expiry date must be wrong

Lab: Pair off Scenarios

Now that you have some idea on some scenarios, pair off and *determine a problem you need to solve* in your domain, and create some scenarios.

Gherkin with Backgrounds

Feature: Feedback when entering invalid credit card details

In user testing we've seen a lot of people who made mistakes entering their credit card. We need to be as helpful as possible here to avoid losing users at this crucial stage of the transaction.

Background:

Given I have chosen some items to buy

And I am about to enter my credit card details

Scenario: Credit card number too short

When I enter a card number that's only 15 digits long

And all the other details are correct

And I submit the form

Then the form should be redisplayed

And I should see a message advising me of the correct number of digits

Scenario: Expiry date must not be in the past

When I enter a card expiry date that's in the past

And all the other details are correct

And I submit the form

Then the form should be redisplayed

And I should see a message telling me the expiry date must be wrong

Lab: Finding a Background

Given your scenarios, find commonality, and extract or refactor a `background`

Gherkin Complete List of Keywords

```
Feature
Background
Scenario
Given
When
Then
And
But
*
Scenario Outline
Examples
```

BDD by any other name

Acceptance-Test-Driven Development

Acceptance Test-Driven Planning

Specification by Example

BDD Benefits

Shared ownership of a project

Shared appreciation of progress

Features

Establishing Features

Tangible

Deliverable Piece of Functionality

Encompasses business goals

Start an ongoing conversation

Must address the business goals

Must be conversed with stakeholders

Describing Features Candidate 1

1. What outcomes are you trying to achieve?
2. Who needs it?
3. What must you do to help achieve this outcome?

Describing Features Candidate 1

Feature: In order to achieve this goal, as a stakeholder, we need the this feature

Describing Features Candidate 2

1. Who will benefit from this feature?
2. What does the feature do?
3. What business value will the stakeholder get out of this feature?

Describing Features Candidate 2

Feature: As a stakeholder, I want this feature, so I can achieve this goal.

Example Features

Feature: Automatic Discount for long term customers

Our goal is to provide an automatic discount on insurance after a customer has been with us for any number of consecutive years

BDD Evolutions

It is not important to have everything known

Change is certain to happen

Early feedback from the users and
stakeholders needed to
ensure that they’re on track

**Never wait until the end of the project to ask
stakeholders if it meets specification**

Scenarios

BDD Scenarios

After establishing features....

Establish scenarios of what is meant to be delivered.

Users define a set of concrete examples that show outcomes

Each scenario must make sense and be able to be executed independently of any other scenario.

BDD Scenarios

In other words....

Money deposited in one scenario should not be in the next....

BDD Scenarios Begets Scenarios

Scenarios can create other Scenarios

Be flexible with your Scenarios

Some Scenarios may not need to exist anymore

- ...because things cannot be automated
- ...because things are just too hard
- ...because the payoff isn't that great

Translate the story to scenarios

Feature: Automatic Discount for Customer after three consecutive years

Our goal is to provide an automatic discount on insurance after a customer has been with us for three consecutive years. This process needs to happen for web client and for internal systems.

Scenario:

Given a customer

And a new payment receipt ...

....But WAIT why do they receive a payment receipt!?!?!?!?

BDD allows you to put information into words.

When you physically write or type down what you require it...

...avoids invoking needless resources on a mistake!

Translate Story to a Set of Tests

Feature: Automatic Discount for Customer after three consecutive years
Our goal is to provide an automatic discount on insurance after a customer has been with us for three consecutive years. This process needs to happen for web client and for internal systems.

Scenario:

Given a customer
And their history of policies
When a quote is generated for a customer
And that customer has been with us for 3 years
Then the quote should take 10% off our normal rate auto

Scenario:

Given a customer
And their history of policies
When a quote is generated for a customer
And that customer has been with us for 5 years
Then the quote should take 15% off our normal rate auto

Setting up a Cucumber Project

Creating a Maven Project

Ensure that Maven is installed

```
> mvn -v

Apache Maven 3.3.9
(bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-
10T09:41:47-07:00)

Maven home: /usr/lib/mvn/apache-maven-3.3.9

Java version: 1.8.0_65, vendor: Oracle Corporation

Java home: /usr/lib/jvm/jdk1.8.0_65/jre

Default locale: en_US, platform encoding: UTF-8
```

Create a project

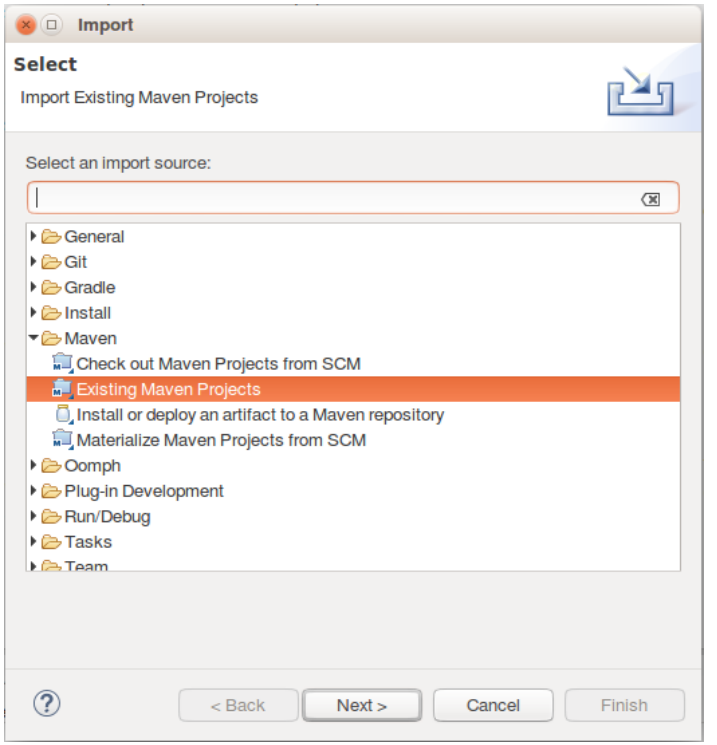
1. `mvn archetype:generate`
2. Wait for list to complete
3. Hit `Enter` to select the default
4. Select the latest archetype `(1.1)`
5. Enter `groupID (com.xyzcorp)`
6. Enter `artifactID (cucumber-jvm)`
7. Enter version of `1.0-SNAPSHOT`
8. Confirm
9. `cd cucumber-jvm`

Setting up Eclipse

Check Eclipse Maven Plugin Install

To check if Maven is installed

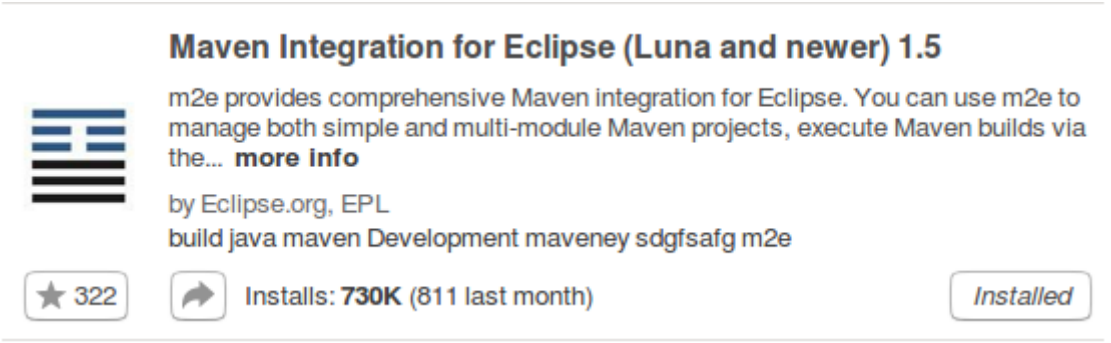
- *File > Import*
- Locate Maven Section
- If there, it is installed.
- If not we will need to install, see the next slide.



Install Maven Plugin if Necessary

In Eclipse:

- *Help > Eclipse Marketplace*
- In Search, type: *Maven*
- Click *Next*
- Select the Plugin if not selected
- Accept License Agreement
- Restart if requested



Install Eclipse Cucumber Plugin

Help > Install New Software

Enter `http://cucumber.github.com/cucumber-eclipse/update-site`

Select the Cucumber Eclipse Plugin

Select *Next, Next*

Accept License Agreement

Finish

Restart Eclipse

Importing Project Into Eclipse

File > Import

Select *Maven > Existing Maven Projects*

Click *Browse...*

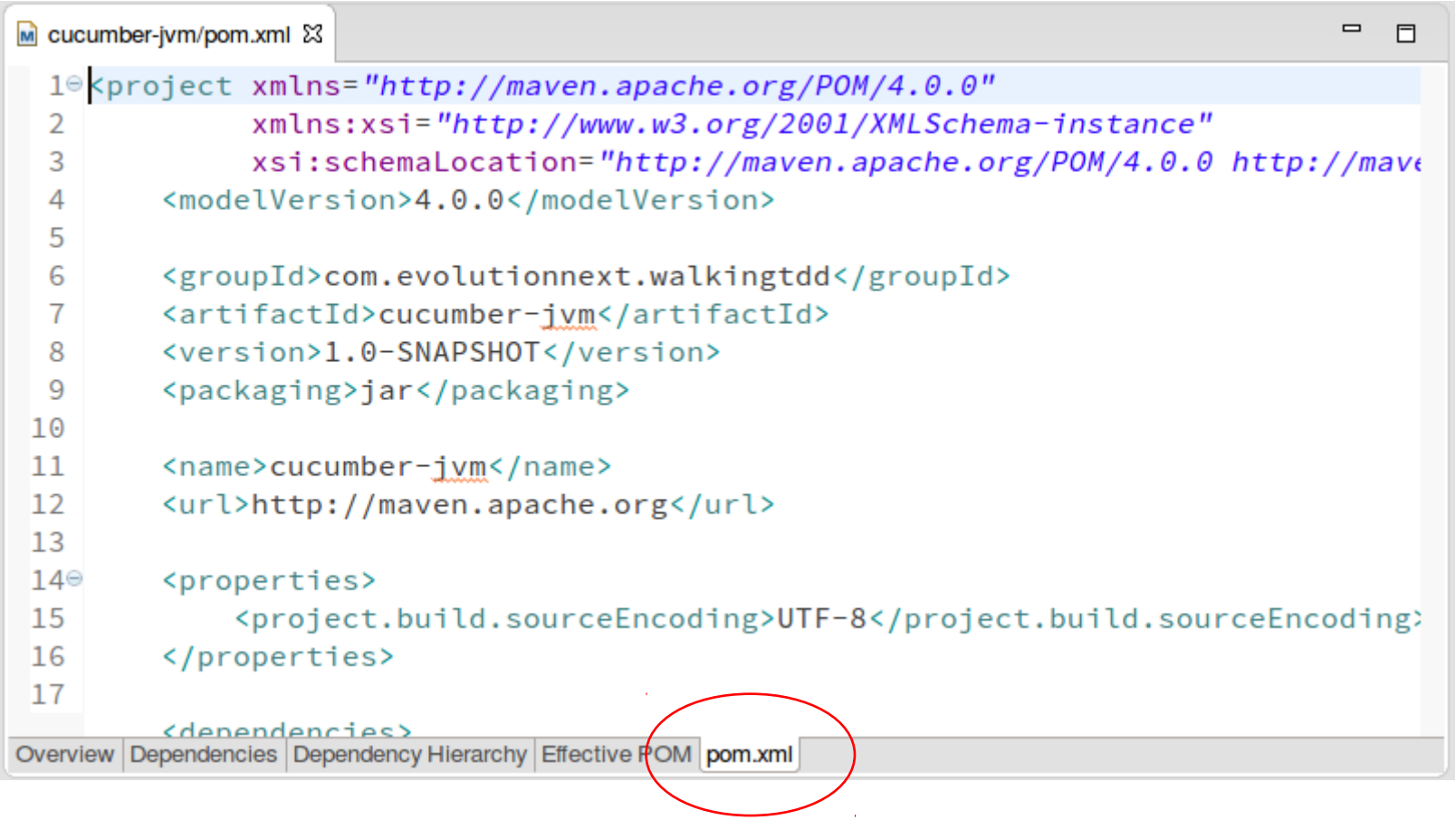
Locate `cucumber-jvm` directory

Click *Ok*

Ensure that the `pom.xml` file is found

Click *Finish*

Open up the pom.xml editor



Upgrade JUnit

The standard junit is old, upgrade to the latest

Find the latest at `search.maven.org`

Add it as a dependency to the `pom.xml`

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

Setting up Cucumber Java

Locate the latest `cucumber-java` from `search.maven.org`

Add it as a dependency to the `pom.xml`

```
<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>1.2.4</version>
    <scope>test</scope>
</dependency>
```

Setting up Cucumber Java 8

Locate the latest `cucumber-java8` from `search.maven.org`

Add it as a dependency to the `pom.xml`

```
<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-java8</artifactId>
    <version>1.2.4</version>
    <scope>test</scope>
</dependency>
```

Setting up Cucumber JUnit

Locate the latest `cucumber-junit` from `search.maven.org`

Add it as a dependency to the `pom.xml`

```
<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>1.2.4</version>
    <scope>test</scope>
</dependency>
```

Setting up Picocontainer

Locate the latest `cucumber-picocontainer` from `search.maven.org`

Add it as a dependency to the `pom.xml`

```
<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-picocontainer</artifactId>
    <version>1.2.4</version>
    <scope>test</scope>
</dependency>
```

Setting up AssertJ-Core

Locate the latest `assertj-core` from `search.maven.org`

Add it as a dependency to the `pom.xml`

```
<dependency>
    <groupId>org.assertj</groupId>
    <artifactId>assertj-core</artifactId>
    <version>3.4.1</version>
    <scope>test</scope>
</dependency>
```

Update Project to use Java 8

Locate the latest compiler plugin `maven-compiler-plugin` from `search.maven.org`

Add it as a **plugin** to the `pom.xml`

```
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-compiler-plugin</artifactId>
   <version>3.5.1</version>
   <configuration>
       <source>1.8</source>
       <target>1.8</target>
   </configuration>
</plugin>
```

Refresh Maven Settings

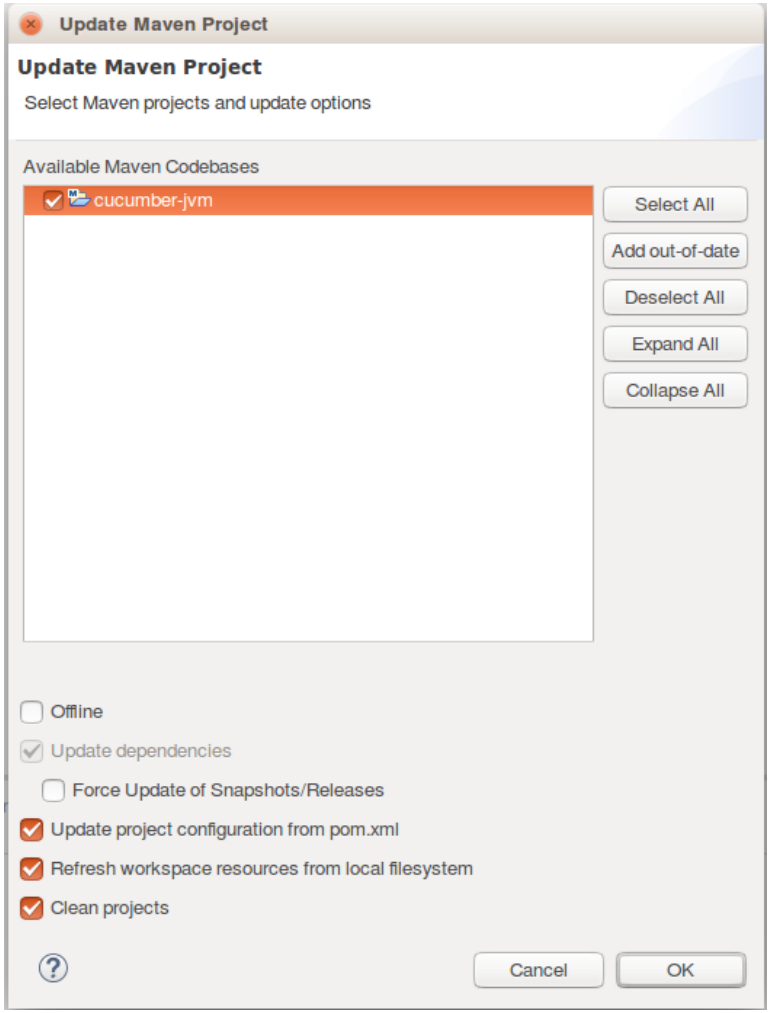
When all has been added and configured either perform:

ALT-<F5>

or

Right Click on Project

Maven > Update Project



Creating a Feature File

About `.feature` files

Located in the `src/test/resources`

Typically, and recommended they be in the same package as the test and production code.

Files end in `.feature`

Lab: Setting up a Resources Folder

- Right click on the `src/test` ***folder*** in the Package Explorer.
- Select *New > Folder*
- Enter the folder name: `resources`
- Right click on *resources* and select *Build Path*
- Select *'Use as source folder'*

Lab: Create a `.feature` file

Create package `com.xyzcorp` in `src/test/resources`

Create a file `health_quote.feature` inside of `src/test/resources`

Delete all the helper material

Our first feature

Lab: Create the first feature

Create a feature in `health_quote.feature`:

```
Feature: As a stake holder, we should analyze our accepted
risk given each plausible scenario to better serve the
public and to maximize all possible sources of revenue.
```

Our first scenario

Lab: Create a Scenario

Create a feature in `health_quote.feature`:

```
Feature: As a stake holder, we should analyze our accepted
risk given each plausible scenario to better serve the
public and to maximize all possible sources of revenue.

Scenario: A person, under 30, in excellent health, should be
provided a quote with a 1 risk.
```

Our first step

Lab: Create a Scenario

Create a scenario in `health_quote.feature`:

```
Feature: As a stake holder, we should analyze our accepted
risk given each plausible scenario to better serve the
public and to maximize all possible sources of revenue.

Scenario: A person, under 30, in excellent health, should be
provided a quote with a 1 risk.
    Given a prospect
    And a birth date of 1993-02-01
    And a current date of 2016-05-01
    And the health risk list is empty
    When all the factors are taken into consideration
    Then the quote risk should be 1
```

Can't we use the system clock?

We want consistency

Each Non-UI BDD test, and every TDD Test must also be isolated.

Non-UI BDD Tests and TDD Tests are meant to be fast.

Lab: Create a 2nd Scenario

Create a scenario in `health_quote.feature`:

```
Scenario: A person, under 30 with high blood pressure,
should be provided a quote with a 3 risk.
    Given a prospect
    And a birth date of 1993-02-01
    And a current date of 2016-05-01
    And the health risk includes High-Blood-Pressure
    When all the factors are taken into consideration
    Then the quote risk should be 3
```

Lab: Create a 3rd Scenario

Create a scenario in `health_quote.feature`:

```
Scenario: A person, under 30 with high blood pressure, and
an active smoker be provided provided a quote with a 5 risk.
    Given a prospect
    And a birth date of 1993-02-01
    And a current date of 2016-05-01
    And the health risk includes High-Blood-Pressure
    And is an active smoker
    When all the factors are taken into consideration
    Then the quote risk should be 3
```

Lists

- Lists can be create in Gherkin
- Either as a comma delimited list in a statement
  - or as a list with a bar
- The Lists in Gherkin can be translated into a `java.util.List` or `java.util.Set` or `cucumber.api.DataTable`

Lab: Create a 4th Scenario

Create a scenario in `health_quote.feature`:

```
Scenario: A person, over 30 and under or equal to 45, with high blood pressure, and high blood sugar should be provided a quote with a 5 risk.
    Given a prospect
    And a birth date of 1993-02-01
    And a current date of 2016-05-01
    And the health risk includes High-Blood-Pressure, High-Blood-Sugar
    When all the factors are taken into consideration
    Then the quote risk should be 5
```

Lab: Create a 5th Scenario

Create a scenario in `health_quote.feature`:

```
Scenario: A person, over 30 and under or equal to 45, with
high blood pressure, and high blood sugar, and moderate
cholesterol they should be provided a quote with a 5 risk.
    Given a prospect
    And a birth date of 1993-02-01
    And a current date of 2016-05-01
    And the health risk includes
      | High-Blood-Pressure  |
      | High-Blood-Sugar     |
      | Moderate-Cholesterol |
    When all the factors are taken into consideration
    Then the quote risk should be 5
```

Scenario Outline

- Perfect when you have several scenarios that follows the exact same number of steps, except with vastly different input and output
- Makes use of placeholders within the scenario outline as brackets
- Place holders are were you want the values to be inputted.

Scenario Outline

- Requires an `examples` section.
- One Feature can have multiple Scenario Outlines
- One Scenario Outline can have multiple examples for categorization
- Scenario Outlines are just compiled down to multiple Scenarios
- Try to maintain only *key examples* to avoid example bloat

Lab: Create a 6th Scenario

Create a scenario in `health_quote.feature`:

```
Scenario Outline: A person with a certain table of elements, should be
  provided with the following table of risks
    Given a prospect
    And a birth date of <birth-date>
    And a current date of <current-date>
    And the health risk includes <health-risks>
    When all the factors are taken into consideration
    Then the quote risk should be <quote-risk>

    Examples:
      | birth-date | current-date | health-risks         | quote-risk |
      | 2000-01-30 | 2016-07-14   |                      | 0          |
      | 1978-01-30 | 2016-07-14   | high-blood-pressure | 1          |
```

Lab: Add Labels and More Examples

Add more examples to the same scenario outline in `health_quote.feature`:

```
Scenario Outline: A person with a certain table of elements, should be
  provided with the following table of risks
    Given a prospect
    And a birth date of <birth-date>
    And a current date of <current-date>
    And the health risk includes <health-risks>
    When all the factors are taken into consideration
    Then the quote risk should be <quote-risk>

    Examples: Low Risk
      | birth-date | current-date | health-risks        | quote-risk |
      | 2000-01-30 | 2016-07-14   |                     | 0          |
      | 1978-01-30 | 2016-07-14   | high-blood-pressure | 1          |

    Examples: Moderate Risk
      | birth-date | current-date | health-risks                         | quote-risk |
      | 2000-01-30 | 2016-07-14   | Asthma, high-blood-sugar             | 2          |
      | 1978-01-30 | 2016-07-14   | high-blood-pressure, high-blood-sugar | 4          |
```

Background

Allow you to specify a set of steps that are common to every scenario

Instead of having to repeat the steps over and over.

If you need to change the elements you only need to change them to one place.

When run in the report they show that they were integral to that steps of that scenario

Must appear before any `scenario` or `scenario outline` elements

Background Caveats

Don’t use Background to set up complicated state

Keep it short and sweet

Remember to try to tell a story

If the background is more than 4 lines think about either another feature, or crafting higher level steps

If the scenarios don’t compliment the background move the scenarios to their own feature.

Don’t put other mechanisms into the background like starting databases, for that there are hooks

Refactoring Definition

After testing is structured and in place...

After the tests pass, both unit and features...

Refactoring gives you opportunity to refine and clean up code.

Refactor to Background

Don’t start with background, refactor to it

If you find repeated steps in your scenarios then it is a good time to refactor a `background`.

Notice the common steps and create a `background` from those.

Lab: Create a background

In `health_quote.feature`, refactor and extract the **`Given a prospect step`** as a background.

```
Feature: As a stake holder, we should analyze our
accepted risk given each plausible scenario to
better serve the public and to maximize all
possible sources of revenue.

  Background:
    Given a prospect
```

Comments

Comments start with a #

Has to be the first and only line in the comment

No comments after a Gherkin statement is allowed.

```
#First Scenario simple case
Scenario: A person, under 30, in excellent health,
should be provided a quote with a 1 risk.
```

Lab: Come together, right now

Group together as a mix of stakeholders, business analyst, testers, developers, all the better.

Create your own Gherkins

Be sure to word your Features, Scenarios, and your steps.

Use examples relevant to your own domain

Creating a Test Runner

Test Runners

Cucumber integrates with JUnit and TestNG

We can run our scenarios using the same tooling as our unit tests

Test Runners go inside of `src/test/java`

Lab: Create a TestRunner

Create a test runner in `CukesRunnerTest.java`:

```
package com.xyzcorp;

import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(plugin = {"pretty",
        "html:target/site/cucumber-html-report",
        "progress:target/site/cucumber-progress-report"},
        features = "src/test/resources")
public class CukesRunnerTest {}
```

Step Definitions/Glue Code

Step Definitions/Glue Code

Step Definition Code houses the code, that is engineered between Testers, and Developers

At this time we will need to define the expected API that will be called

This can change – Prognostications are useless. Be able to accept change. Unit tests will likely reveal hidden truths about the code.

Automated Step Creation

Eclipse currently has no support for automated step creation.

IntelliJ IDEA offers automated step creation merely by typing ALT+ENTER on each step.

If you favor Eclipse try some tools like **Tidy Gherkin** on the Chrome Browser.

Lab: Create Step Definitions

- Install Tidy Gherkin on Google Chrome
- Copy the contents of your feature into Tidy Gherkin
- Highlight, Java Steps and view your new step definitions

Lab: Create a step definition file

Create a class called `HealthQuoteSpecification` (or whatever you would like) inside of `src/test/java`

Copy the contents inside of the `StepDefinitions` class in Tidy Gherkin into the `HealthQuoteSpecification` class

What is `PendingException`

`PendingException` tells the testing framework that your Specifications are still a work in progress

Intro Regular Expressions

Capture Groups

When you surround with parenthesis it becomes part of a capture group

```
@Given("I have deposited \\$(100) in my Account")
public void iHaveDeposited$100InMyAccount(int amount) {
  // TODO: code goes here
}
```

Alternation

Regular Expression that express different options separated by a pipe character

```
@Given("I have deposited \\$(100|250) in my Account")
public void iHaveDeposited$InMyAccount(int amount) {
  // TODO: code goes here
}
```

Dot

The dot

- *metacharacter*, meaning it has magical powers in a regular expression.
- A dot means match any single character

```
@Given("I have deposited \\$(...) in my Account")
public void iHaveDeposited$InMyAccount(int amount) {
  // TODO: code goes here
}
```

Star

A repetition modifier takes a character and tells us how many times it will appear

0 or more

Mental trick, a * is round, and shaped like a zero.

Greedy

"Given I have deposited $1 and a cucumber in my Account", will capture "1 and a cucumber"

```
@Given("I have deposited \\$(.*) in my Account")
public void iHaveDeposited$InMyAccount(int amount) {
  // TODO: code goes here
}
```

Character Classes

Character classes allow you to tell the regular expression engine to match one of a range of characters.

You just place all of the characters you would accept inside square brackets

```
@Given("I have deposited \\$([01234567890]*) in my Account")
public void iHaveDeposited$InMyAccount(int amount) {
  // TODO: code goes here
}
```

```
@Given("I have deposited \\$([0-9]*) in my Account")
public void iHaveDeposited$InMyAccount(int amount) {
  // TODO: code goes here
}
```

Shorthand Character Classes

For items like `[0-9]` there are some short hand that you can use.

`[0-9]`	`\\d`
`[A-Za-z0-9_]`	`\\w`
`[\t\r\n]`	`\\s`
Word Boundary	`\\b`

```
@Given("I have deposited \\$(\\d*) in my Account")
public void iHaveDeposited$InMyAccount(int amount) {
  // TODO: code goes here
}
```

Capitalizing each one will do the reverse.
e.g. `\\D` is the reverse of `\\d`

The Plus Modifier

Plus represents one or more

Mind trick + looks as if it has a cross of 1s therefore starts with 1

```
@Given("I have deposited \\$(\\d+) in my Account")
public void iHaveDeposited$InMyAccount(int amount) {
  // TODO: code goes here
}
```

Carrots before the Cash

The Carat ^ starts the line and will not match anything on the inside of the line

The $ will match the end of the line and will ensure that the entire line is matched

^$ will match an empty line, not that that is useful for cucumber

Mind Trick: "You must grow carrots before you can sell them"

Multi-item Captures

You can match two or more groups whenever you would need more than 1 of any item.

```
@Given("I have deposited \\$(\\d+) in my (\\w+) Account")
public void iHaveDeposited$InMyAccount(int amount, String accountType) {
  // TODO: code goes here
}
```

Flexible Words

Flexible words

If you want to match either a plural or a single word you can use a ? character

Character ? actually means the character preceding is optional.

```
@Given("I have (\\d+) cucumbers? in my basket")
public void iHaveCucumbersInMyBasket(int number) {
  // TODO: code goes here
}
```

Non-capturing groups

Great if you want to use alternation, but don’t necessary want to capture the items

Uses the operator `?:`

That also means since it is not captured that it will not be processed by the method.

```
@When("I (?:visit|go to) the homepage")
public void iVisitTheHomepage() {
  // TODO: code goes here
}
```

DataTables 

DataTables

DataTables give you a way to extend a Gherkin step to more items

Make use of the pipe to construct tables

Cucumber will remove the pipes and whitespace and plug the data into the step method

DataTables are accepted as a parameter in the step method.

DataTables are immutable

DataTable Diff

DataTables can be diffed to compare one table vs. another.

To do so invoke the diff method of the DataTable object.

Lab: Retrofit the Steps with Regex

Given your knowledge of regular expressions retrofit your steps for it to work correctly with your steps

Be sure to include parameters for to match the steps

For Dates, use `java.util.Date` for now.

Creating an API

Now that the glue code has been set and steps made....

Testers and Developers consider a standard API to use

Lab: Design an API

Developers and Testers should start developing an API based on the methods and the order that they are called.

Determine the best regular expression to match each step.

Use parameters that best match the regular expression provided.

Transformers

Work on captured arguments

Transforms a given parameter into something more meaningful.

Lab: Create a Transformer

Create a Transformer in `src/test/java` called `Java8LocalDateConverter.java` with the following content.

```
package com.xyzcorp;
import cucumber.api.Transformer;
import java.time.LocalDate;

public class Java8LocalDateConverter extends Transformer<LocalDate> {
    @Override
    public LocalDate transform(String value) {
        return LocalDate.parse(value);
    }
}
```

Lab: Convert the arguments

Convert the arguments in `HealthQuoteSpecification` from `java.util.Date` to `java.time.LocalDate`

Note: Feature Subfolders

Features can be organized by category, domain, or however you would like

It is best to also refactor into those Subfolders.

Lab: Test Driven Development

Create Unit Tests with Test Driven Development that drive to the specification.

This is done by developers only

UI Testing

Selenium

www.seleniumhq.org

Automated Browser navigation and test suite

Suite runs on a browser

Use Selenium WebDriver for running UI Web testing

Use Selenium IDE to create some raw scripts to run web automated testing

Word of Warning on UI BDD

Selenium Tests/UI Tests are extremely brittle

Overtime they are harder to maintain

The cost *may* outweigh the benefits

Consider validating against Web Service instead

Effective Selenium

Work on getting `id` and `name` elements in the tags for anchoring

Browsers are different, test different ones.

Lab: Bring in Selenium

In your pom.xml file bring in the last selenium library and a connection for Firefox.

```
<dependency>
   <groupId>org.seleniumhq.selenium</groupId>
   <artifactId>selenium-firefox-driver</artifactId>
   <version>2.53.0</version>
 </dependency>
```

Lab: Install Firefox & Selenium IDE

If you do not have it, please install the Firefox browser.

Install the Selenium IDE Plugin from `https://addons.mozilla.org/en-US/firefox/addon/selenium-ide/`

Lab: Create a UI Feature

Inside of `src/test/resources` create a file inside of the `com.xyzcorp` packages named `quote_ui.feature` with the following:

```
Feature: When we go to the main page of our website we should get a quote
  when providing our zip code.

  Scenario: Adding our zip code correctly on Firefox
    Given we are connecting with Firefox
    And we are going to http://www.libertymutual.com
    When we provide an zip code of 68520
    And click on the "Get a quote" button
    And wait for no more than 10 seconds
    Then we should be given a window that will ask us for our address
    And it should have the city filled in as Lincoln
    And it should have the state filled in as NE
```

Lab: Create the Steps

Given current knowledge create the step code for the steps created in a new Specification titled:

Hooks

Hooks are supported methods that run before and after each scenario

Analogous to `setUp` and `tearDown` of xUnit

Tagged Hooks

Accept a tag expression which allows hooks to be used only for certain situations

```
@Before(value=”nightly”)
```

Ordered Hooks

Specifies the order in which they should run.

Cucumber runs Ordered tags from low to high for `@Before`

e.g. a Hook with an order of `10` will run before `20`

Cucumber run Ordered tags from night to low `@After`

A Hook with an order of `20` will run before `10`

Lab: Set up Selenium Hooks

Given current knowledge create the step code for the steps created in a new Specification titled `QuoteUISpecification`:

```
public class QuoteUISpecification {

    private WebDriver driver;

    @Before
    public void startFirefoxDriver() {
        driver = new FirefoxDriver();
    }


    @After
    public void shutdownDriver() {
        driver.quit();
    }
}
```

Lab: First Selenium Steps

Add the following steps to `QuoteUISpecification`:

```
@Given("^we are connecting with Firefox$")
public void connectingWithFirefox() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
}

@And("^we are going to (.*)$")
public void gotoWebsite(String website) throws Throwable {
    driver.get(website);
}
```

Lab: Next Selenium Steps

Add the following steps to `QuoteUISpecification`:

```
@When("^we provide an zip code of (.*)$")
public void weProvideAZipCodeOf(String zipCode) throws Throwable {
    driver.findElement(By.id("zipcode")).clear();
    driver.findElement(By.id("zipcode")).sendKeys(zipCode);
}

@And("^click on the \"Get a quote\" button$")
public void clickOnTheButton() throws Throwable {
   driver.findElement(By.xpath("(//button[@type='submit'])[2]")).click();
   driver.findElement
     (By.id("ESalesCommon_view_widget_layout_pane_SliderSet_0")).click();
}
```

Lab: More Selenium Steps

Add the following steps to `QuoteUISpecification`:

```
@And("^wait for no more than (\\d+) seconds$")
public void waitForNoMoreThanSeconds(int seconds) throws Throwable {
    Thread.sleep(seconds * 1000);
}

@And("^it should have the city filled in as (.*)$")
public void verifyCity(String city) throws Throwable {
    assertThat(driver.findElement
        (By.id(
            "FormElement_customer_city_nav-your_info_YI01"))
                .getAttribute("value")).isEqualTo(city);
}
```

Lab: Last Selenium Steps

Add the following steps to `QuoteUISpecification`:

```
@And("^it should have the state filled in as (.*)$")
public void itShouldHaveTheStateFilledInAsNE(String state) throws Throwable {
    assertThat(driver.findElement(
            By.id("FormElement_customer_state_nav-your_info_YI01"))
            .getAttribute("value")).isEqualTo(state);
}

@Then("^we should be given a window that will ask us for our address$")
public void weShouldBeGivenAWindowThatWillAskUsForOurAddress()
        throws Throwable {
    assertThat(driver.findElement(
      By.cssSelector(
       "#FormElement_customer_streetAddress_nav-your_info_YI01")))
        .isNotNull();
}
```

Lab: Run the Selenium Test

Last Items

Doc Strings

Doc Strings allow a larger piece of text than could fit in a single line

This can be applied to any step.

Perfect for XML or JSON loads cautiously

Don’t get too crazy.

Tags

- Tags mark important non important marks to qualify a test
- Tags are done with a `@` character followed by a name of the tag
- Tags can be used to filter certain features or scenarios
- Some popular ones
  `@ui @web-service @unit @nightly @slow @fast`
- You can also tags Scenario Outline elements and the individual elements under them

Reasons for Tagging

Documentation - Label them with an ID from any agile tool

Filtering - Select which tests to run

Hooks - Run a block of code whenever scenario with particular tag is about to start

Fin