BDD Primer and Workshop

THREE DAY COURSE

Course Agenda – *Day 1*

Course Overview

BDD Basics and Principles

From TDD to BDD

Seeing it in Action

Domains and the Role of Business

Wrap-up

Course Agenda – *Day 2*

Review

BDD Tooling

Working with Gherkin Syntax

Compound Steps

Resolving Duplicates

Parameters and Templates

Course Agenda – *Day 3*

Outlines, Lists, and DataTables

Before / After Stories

Impact Mapping

Applying BDD on a Project

Final Review

Wrap-up, Q&A

Course Introductions

Name?

Role?

Background?

Primary Objective in Taking Course?

Recent Book or Movie?

Schedule and Expectations

Daily schedule

- Time frame?
- Breaks?
- Lunch?

Expectations of one another (cell phone, e-mail, mutual respect, etc.).

Any other questions before we get started?

BDD Primer & Workshop – Day 1

GOAL: ESTABLISH A BASELINE OF KNOWLEDGE SURROUNDING BEHAVIOR-DRIVEN DEVELOPMENT

Course Agenda

Course Overview

BDD Basics and Principles

From TDD to BDD

Seeing it in Action

Domains and the Role of Business

Wrap-up

BDD Origins

Test Driven Development (TDD)

- *Techniques*
- *Principles*

Domain Driven Design (DDD)

- *Concepts*

Object Oriented Analysis & Design (OOAD)

- *Concepts*

Natural Language Constructs

Domain Specific Language (DSL)

Shared Process

Shared Tools

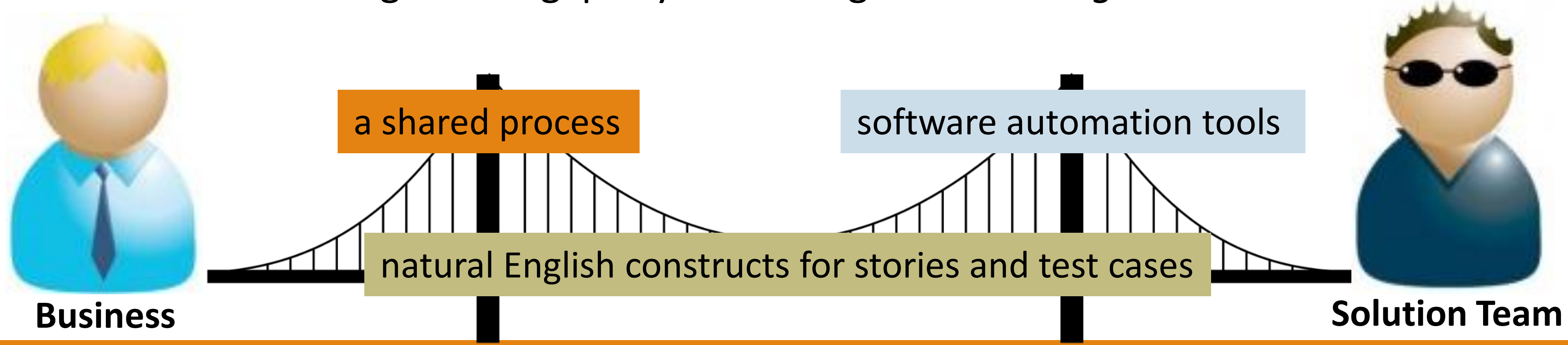
Automation

The Digital Disconnect

Software development projects experience common challenges:

- The business struggles to completely define desired outcomes
- The development team has a flawed understanding of what needs to be built
- The business is unaware of the technical challenges incurred by the requirements they identify
- The translation of business needs into technical software loses much in translation and ultimately leads to gaps in expectation and delivery

BDD strives to bridge these gaps by facilitating a *natural alignment*…


a shared process
software automation tools
natural English constructs for stories and test cases
Business
Solution Team

What’s that spell?

B - *Behavior*, the desired *behavior* of the software to be developed.

D – *Driven*, the project is oriented around and aligned with the desired behavior

D – *Development*, software is mapped directly to the declaration of behavior, supported by automated tooling.

In short, BDD is an approach for building a <u>shared understanding</u> regarding what software should be built by working through specific examples.

How’s it work?

BDD is pretty simple:

- **describe** what you want the system to do by talking through example behavior
- work from the “outside-in” to **implement** those behaviors using the examples to validate what you are creating

Bring in different perspectives for validation:

- The customer (or Product Owner) describes what they want.
- Team members ask questions to flesh out “just enough” in order to implement the behavior.
- Business analysts and quality assurance specialists can also help to identify and elaborate specific details of various scenarios.

Principles

Business Traceability – All development work can be traced back directly to business objectives.

Needs Focused – Software development meets user needs and ultimately supports current and future business objectives.

Efficient Prioritization – Business-critical features are delivered first.

Shared Understanding – All parties (business and technical) have a shared understanding of the project and thorough visibility into the project’s progression.

Imagine if you could choose…

A logical, thorough, content-heavy training course on a new topic

- Detailed definition of terms
- Well-structured thought process
- Extensive historical context
- Comprehensive philosophy regarding the motivations and drivers for this new subject
- Itemized implementation steps
- Best practices and recommendations

A pragmatic, example-rich course that aimed to SHOW you as much as TEACH you the new topic

- Establish a baseline of terminology, principles, and process
- Highlight a few benefits and motivations behind the topic's core philosophy
- Practical, real-world examples
- Collaborative exercises
- More practice, less theory

Which type of course would you prefer to participate in?

You can **always** <u>learn</u> by *doing*. You can **never** <u>do</u> by *learning*.

Simple Example

Context: Your team is working on the quote functionality within the rating engine for an insurance business.

Task: Implement a risk premium factor to quotes based upon a prospect's credit score.

Business Rules:

- Quote premium for sub-prime (less than 580 credit score) is 25%
- Quote premium for prime (more than 580 credit score) is 0%
- Quote premium for super-prime (more than 720 credit score) is -10%

Can you use this information to deliver a working feature?

How BDD Helps

How do we address the ambiguity left by the initial scenario setup?
Through concrete, behavior-driven examples.

Given a credit score of 570, when a quote is run, then the premium is 25%

Given a credit score of 580, when a quote is run, then the premium is 0%

Given a credit score of 719, when a quote is run, then the premium is 0%

Given a credit score of 720, when a quote is run, then the premium is -10%

If you take these examples, convert them into tests, and use those tests to drive (and scope) your development activity, then you've doing BDD!

Lost in Translation

Most any project (and software development in particular) has the potential to fall prey to miscommunication due to ambiguous terms.

- **Example 1** – Writing software for a bank and being asked to implement basic account management functionality. *Great! Umm… did you want 'user accounts' or 'credit/debit accounts' to be managed?*
- **Example 2** – Delivering a *policy validation* function for an insurance business. *Terrific! So…you were referring to 'security policies,' right?*

There is an inherent "cost of translation" (also known as "the feedback delay") which can range from weeks (using Agile) to months/years (using Waterfall) as we deliver working software and discover the extent of our miscommunication.

BDD aims to close the size of the feedback loop through **ubiquitous language** (a concept borrowed from domain-driven design).

Conventional BDD Format

Given-When-Then

“Given” – describes the initial context for the example

“When” – describes the triggering event performed by an actor

“Then” – describes the expected outcome (which can be validated)

This aims to mirror the natural way people describe expected behavior. When a stakeholder is asked to provide examples of what they expect for a particular capability, the naturally say something like: *“when I do X, then Y should happen.”*

BDD Story Format

CLASSIC AGILE STORY (USER FOCUS)	BDD STORY (VALUE FOCUS)
As a [type of user]	In order to [achieve some value]
I want [some goal]	As a [type of user]
So that [some reason]	I want [some goal]

BDD shifts the emphasis away from performing *functions* required by users and toward achieving *value* required by the business.

Complete Story – 1/2

Story: *New cars receive the same coverage as old cars*

In order to easily manage my policy

As a member

I want to extend the same coverage to a new car when I add it to my policy

Example: *Coverage limits should be extended to a new car*

Given a member has an existing policy

And the policy has at least one covered car

And the same coverage limits can be extended to the newly added car

When she adds a car to the policy

Then the same coverage limits should be extended to the new car

Complete Story – 2/2

Example: *Policy add-ons should be extended to the new car*

Given a member has an existing policy

And the policy has at least one covered car

And the coverage includes at least one add-on

When she adds a car to the policy

Then the add-ons should be extended to the new car

And the quote should be updated to reflect the cost of the add-ons

Ubiquitous Language

The story identifies a stakeholder, business outcome, and value.

Each example defines a precondition, trigger, and expected outcome.

Some keywords are present (Given-When-Then).

Some ambiguous, domain-specific terms exist as well. These would ultimately need to be identified, defined, and agreed-upon in collaboration between development team and business.

- Member
- Coverage
- Policy
- Quote
- Add-on
- Etc…

BDD Structure

Title: *The story should have a clear, explicit title.*

Narrative: *A short, introductory section that specifies three things…*

- who (which business or project role) is the driver or primary stakeholder of the story (the actor who derives business benefit from the story)
- what effect the stakeholder wants the story to have
- what business value the stakeholder will derive from this effect

Acceptance criteria or scenarios: *a description of each specific case of the narrative. Such a scenario has the following structure…*

- specify the initial condition that is assumed to be true at the beginning of the scenario. This may consist of a single clause, or several
- state which event trigger initiates the scenario
- state the expected outcome, in one or more clauses

Group Exercise – BDD Basics

1. Organize into teams of 2-3.
2. Define the <u>scope</u> of your project (choose a specific system, process, or feature-set you are responsible for developing).
3. Craft <u>two user stories</u> which could realistically fit within that scope.
   - **In order to**… (WHY)
   - **As a**… (WHO)
   - **I want to**… (WHAT)
4. Select <u>one story</u> and create <u>one behavior-driven scenario</u>:
   - **Given**… (describe the initial condition)
   - **When**…(triggering event performed by actor)
   - **Then**…(expected outcome)

Course Agenda

Course Overview

BDD Basics and Principles

From TDD to BDD

Seeing it in Action

Domains and the Role of Business

Wrap-up

Two Main Parts

The BDD approach can largely be divided into two main parts.

First, the practice of using examples written in *ubiquitous language* to illustrate behaviors (how users will interact with the product).

Second, the practice of using those examples as the basis of *automated tests*.

This approach ensures alignment with business needs and user expectations, with a direct traceability to system behavior as defined by the business throughout the project lifetime.

TDD Recap

Typically broken up into five different stages:

1. First the developer writes some tests.
2. Next the developer runs those tests and they fail (by design), because none of those features are actually implemented yet.
3. Next the developer actually implements those tests in code, but *just enough* to pass the tests, thus validating the tests.
4. Now the developer fleshes the implementation out to be complete, using the tests to ensure that the code remains on track.
5. Finally the developer refactors their code, adds comments, cleans it up, and is able to do so with confidence that the tests provide an objective regression meter to identify if anything breaks in the process.

From TDD to BDD

BDD borrows much of its initial philosophy from TDD and utilizes a test-centric approach as an essential thread throughout the methodology. However, it moves beyond TDD to rethink the testing approach.

Classic TDD

For each unit of software…
- define a test set for the unit first
- then implement the unit
- finally verify that the implementation of the unit makes the tests succeed

Testing in BDD

Behavior-driven development specifies that tests of any unit of software should be specified in terms of the ***desired behavior of the unit***.

What is that “desired behavior” in question? It’s the business requirements which spell out a valuable business outcome.

Within BDD practice, this is referred to as BDD being an "outside-in" activity.

Re-framing our Tests

BDD aims to rethink how we define our unit tests and acceptance tests.

Unit Tests

- BDD suggests that unit test names be whole sentences starting with a conditional verb ("should" in English, for example) and should be written in order of business value.

Acceptance Tests

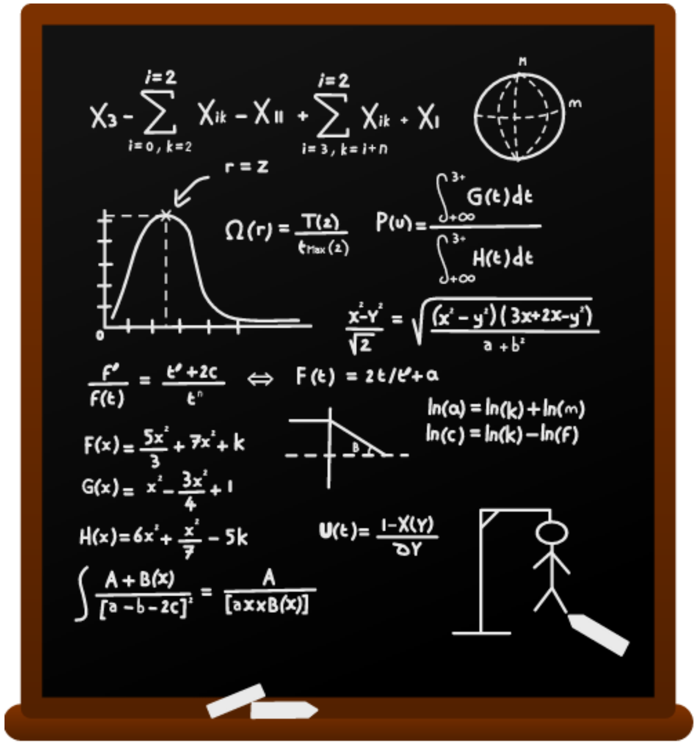
- Conventionally, BDD practitioners write acceptance tests using the standard agile framework of a User story (but with a twist): “In order to [benefit] As a [role] I want [feature]".
- *Acceptance criteria* is most often written in terms of scenarios and implemented as classes: Given [initial context], when [event occurs], then [ensure some outcomes].

The ability to read tests like a sentence is an important cognitive shift which enables you to *naturally* write better and more comprehensive tests.

Show me!

Instructor Demo #1 – *Flipper Demo*

Instructor Demo #2 – *Breaking the Demo*



Reflecting on TDD / BDD

What aspects of test-driven appeal to you?

Where do you see potential challenges or pitfalls?

Any personal experiences with a test-centric approach?

Course Agenda

Course Overview

BDD Basics and Principles

From TDD to BDD

Seeing it in Action

Domains and the Role of Business

Wrap-up

JBehave

A Java-based BDD framework developed by Dan North.

There are two varieties:

- **Core** – Includes all the essential JBehave libraries, command-line runner, and hooks for various tools (Eclipse, Maven, Ant, etc.)
- **Web** – An extension of core, providing support for web-related access or functionality. It includes a web runner and also a selenium integration module for driving verification of web application behavior

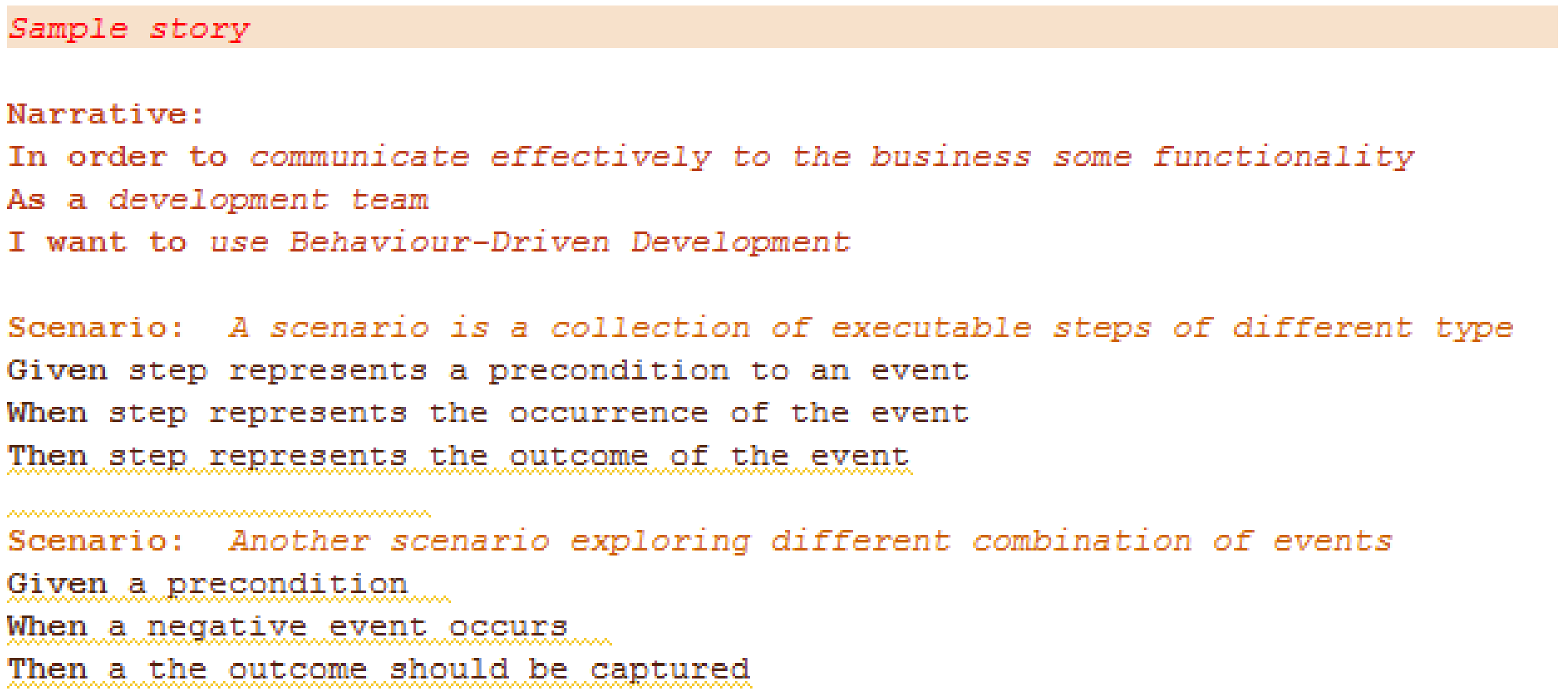
Unit Structure

Story Class – describe the expected behavior using natural language

Object Class – describe an object which implements the desired behavior

Unit Test – bind the story elements to the object elements to facilitate traceability, scoping, and automated testing

Story Structure



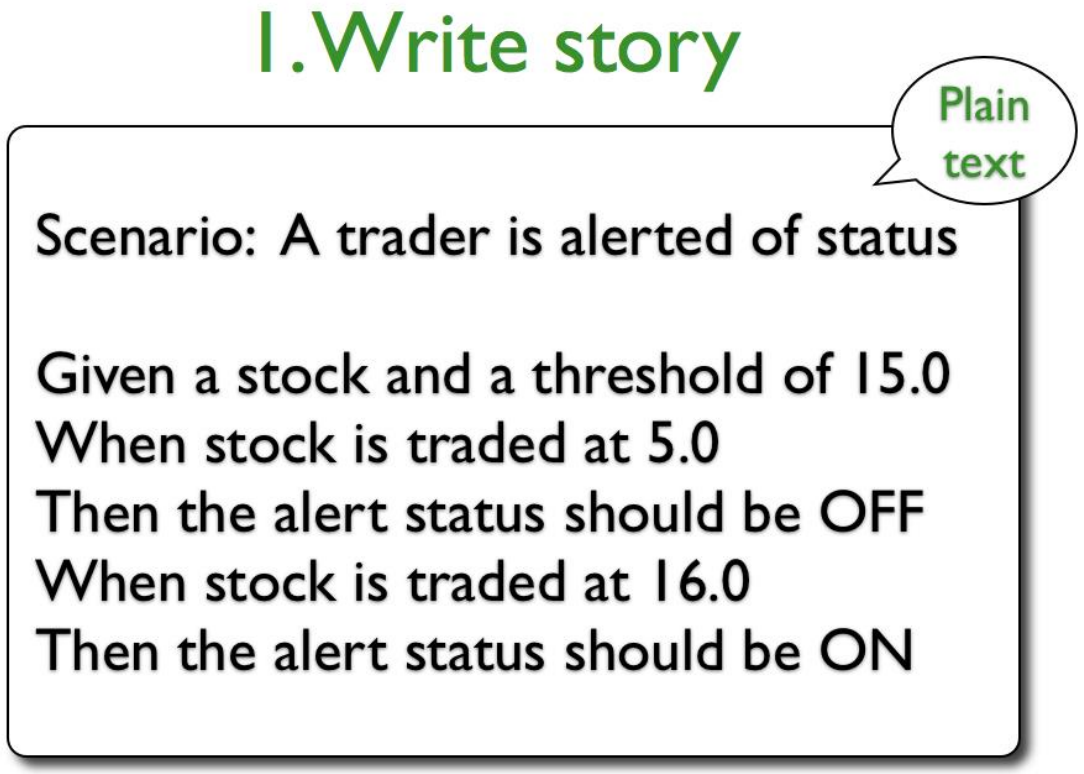
```
Sample story

Narrative:
In order to communicate effectively to the business some functionality
As a development team
I want to use Behaviour-Driven Development

Scenario:  A scenario is a collection of executable steps of different type
Given step represents a precondition to an event
When step represents the occurrence of the event
Then step represents the outcome of the event

Scenario:  Another scenario exploring different combination of events
Given a precondition
When a negative event occurs
Then a the outcome should be captured
```

Group Exercise – Refactoring

1. Get back into your teams.
2. Refactor your previous BDD scenario (if needed) now that you have a stronger understanding of the BDD structure.
3. Take your other story and elaborate two BDD scenarios (refactor the story if needed).

THE TRADER EXAMPLE

The Trader Example – 1/5

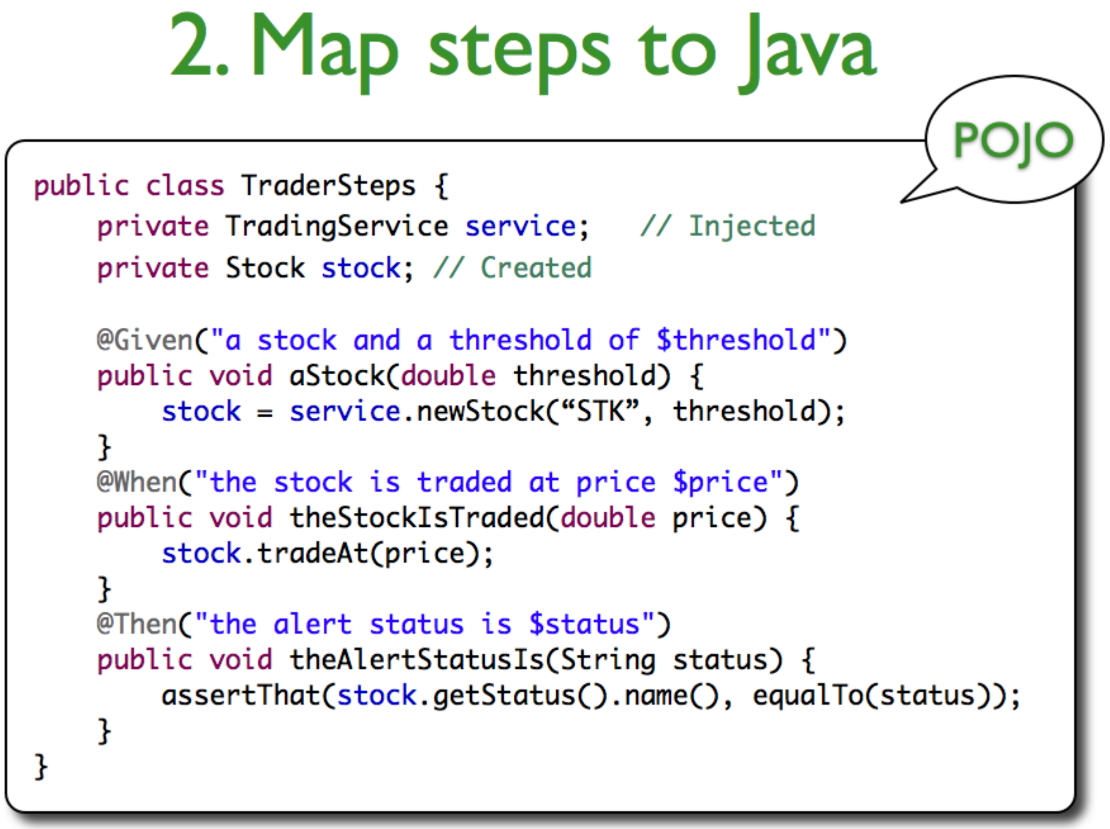


1. Write story

Scenario: A trader is alerted of status

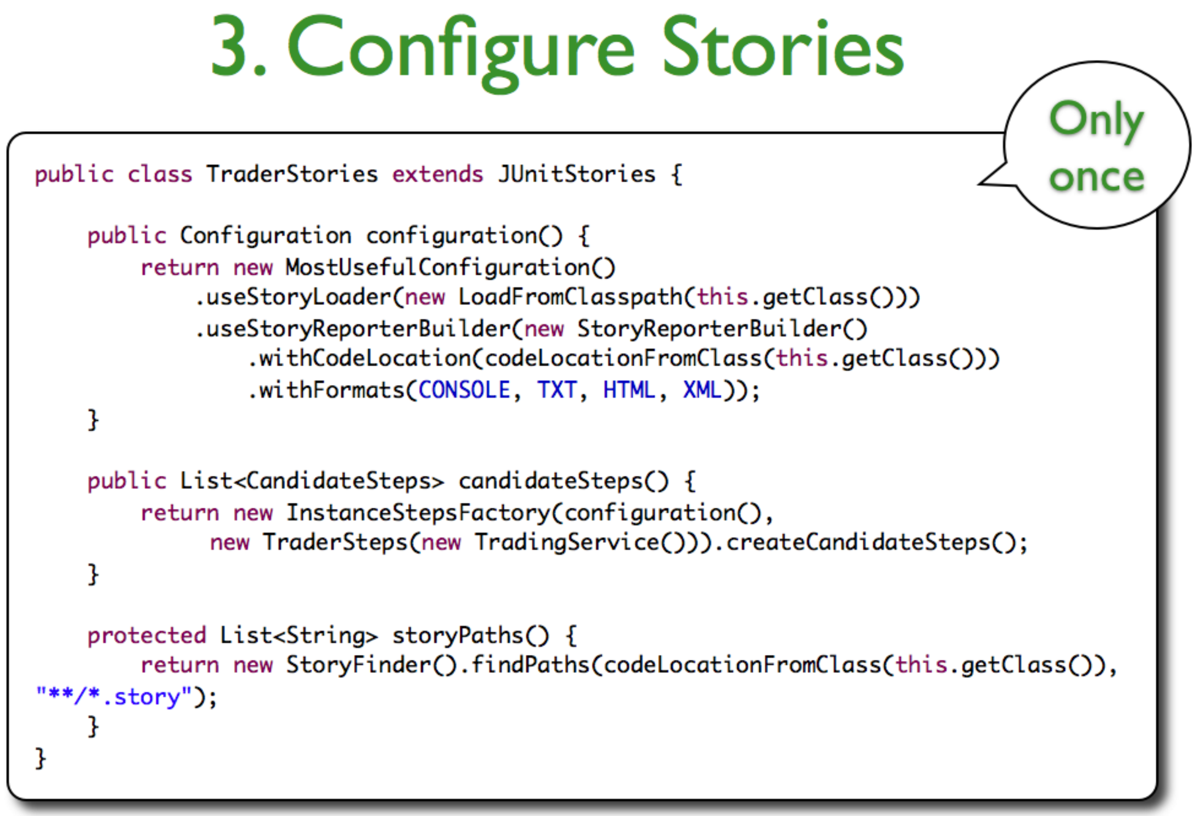
Given a stock and a threshold of 15.0
When stock is traded at 5.0
Then the alert status should be OFF
When stock is traded at 16.0
Then the alert status should be ON

The Trader Example – 2/5


2. Map steps to Java

```
public class TraderSteps {
    private TradingService service;   // Injected
    private Stock stock; // Created

    @Given("a stock and a threshold of $threshold")
    public void aStock(double threshold) {
        stock = service.newStock("STK", threshold);
    }
    @When("the stock is traded at price $price")
    public void theStockIsTraded(double price) {
        stock.tradeAt(price);
    }
    @Then("the alert status is $status")
    public void theAlertStatusIs(String status) {
        assertThat(stock.getStatus().name(), equalTo(status));
    }
}
```

The Trader Example – 3/5



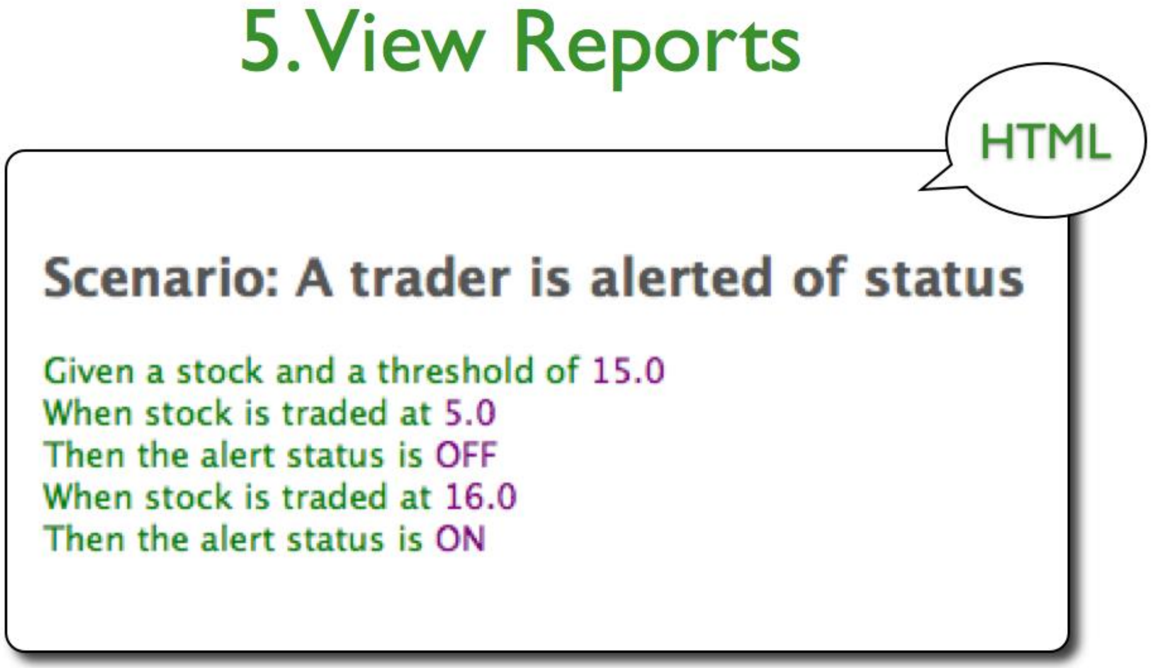
3. Configure Stories

```
public class TraderStories extends JUnitStories {

    public Configuration configuration() {
        return new MostUsefulConfiguration()
            .useStoryLoader(new LoadFromClasspath(this.getClass()))
            .useStoryReporterBuilder(new StoryReporterBuilder()
                .withCodeLocation(codeLocationFromClass(this.getClass()))
                .withFormats(CONSOLE, TXT, HTML, XML));
    }

    public List<CandidateSteps> candidateSteps() {
        return new InstanceStepsFactory(configuration(),
             new TraderSteps(new TradingService())).createCandidateSteps();
    }

    protected List<String> storyPaths() {
        return new StoryFinder().findPaths(codeLocationFromClass(this.getClass()),
"**/*.story");
    }
}
```

The Trader Example – 4/5

4. Run Stories
With any of
<APACHE ANT>
eclipse
THE ECLIPSE PROJECT
JUnit
IntelliJIDEA
maven

The Trader Example – 5/5



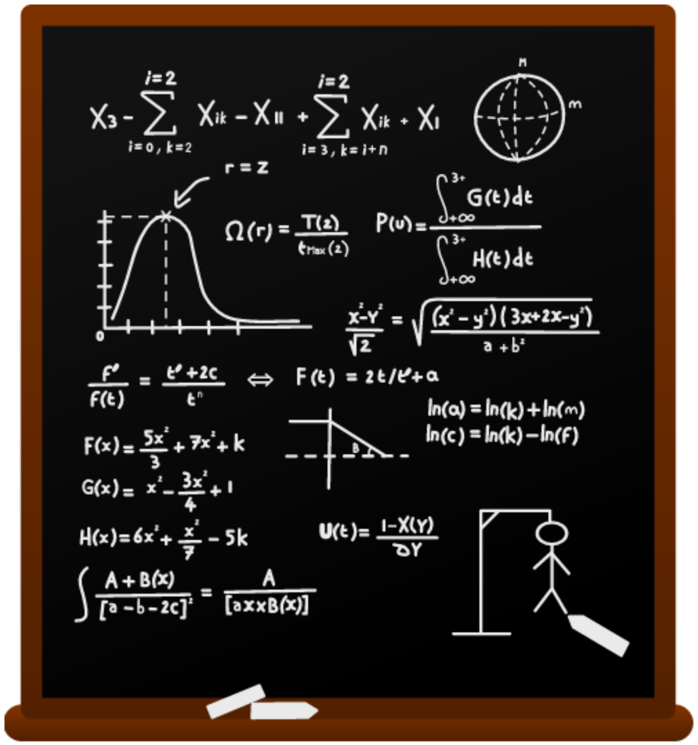
Scenario: A trader is alerted of status

Given a stock and a threshold of 15.0
When stock is traded at 5.0
Then the alert status is OFF
When stock is traded at 16.0
Then the alert status is ON

Demo

Let’s look at the Trader code from JBehave…

***NOTE**: Pay special attention to handling of variables, tables, etc.*



Group Exercise – Software Binding

1. Get back into your teams.
2. Take one BDD scenario and use pseudo code or annotated comments to create a wireframe of what the corresponding object class and testing class would look like for this scenario.
   - Identify a method for each element of the story (GIVEN-WHEN-THEN) that would need to exist within the testing class.
   - For each identified method in the testing class, briefly describe how you would perform that step.
   - For each THEN clause, make sure you have a testable outcome to measure against.
   - Define the basic structure of the object class (fields and methods).
   - *Examples follow…*

Group Exercise Examples

Take one BDD scenario and use pseudo code or annotated comments to create a wireframe of the corresponding object class and testing class.

- Identify a method for each element of the story (GIVEN-WHEN-THEN) that would need to exist within the testing class.

```
@Given("a customer exists")
customerExistsFunction()
```

- For each identified method in the testing class, briefly describe how you would perform that step.

```
//Here we will create a Customer object
```

- For each THEN clause, make sure you have a testable outcome to measure against.

```
@Then("the account balance will be 100")
checkAccountBalance() {
    AssertEquals( 100, account.getBalance() );
}
```

- Define the basic structure of the object class (fields and methods).

```
Customer { name, account balance, account number }
Bank { routing number, account number, customer list}
```

Course Agenda

Course Overview

BDD Basics and Principles

From TDD to BDD

Seeing it in Action

Domains and the Role of Business

Wrap-up

Outside-in Development

Focus – Satisfy the needs of stakeholders.

Premise – In order to create successful software, you must have a clear understanding of the goals and motivations of your stakeholders.

Aim – Produce software that is highly consumable, meeting or even exceeding the needs of your stakeholders.

<u>Four categories of stakeholder for outside-in projects</u>

- **Principals** – people with the money
- **End users** – people who interact with your product
- **Partners** – people who make your product work in real life (ops and SI teams)
- **Insiders** – people within your company who impact how your team develops

Importance of Goal Alignment

If you break it down, **software development is merely a way of supporting human behaviors** by automating them, replacing them, or creating them so a human doesn't have to perform that activity. Moreover, **every change in behavior should have a tangible impact by helping the organization to achieve one or more of its goals** (e.g.

- increasing revenue
- reducing waste
- improving quality
- reducing time-to-market
- etc.

Whatever the desired business outcome looks like should be the driving force behind your project. Every story and scenario should tie back to one or more business goals.

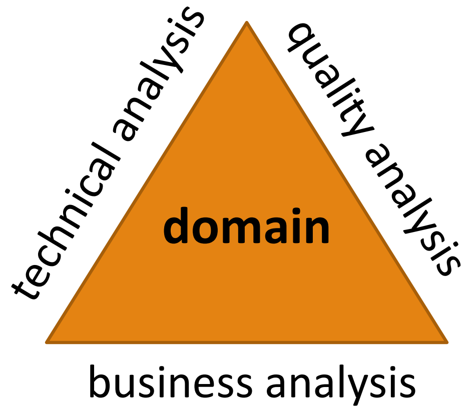
This is precisely why BDD has us flip the script on classic agile story structure and lead with the WHY. It keeps the focus on the goal!

The Three Amigos

One of the biggest (and most common mistakes) is to assume that writing out the scenarios is the most import part of the process and that this responsibility can or should be given to a single person.

BDD is intended to be a **highly collaborative process**. Talking in examples requires multiple inputs and varied perspectives in order to be efficient.

In order to effectively strike the right balance of breadth and depth, business and technical elements, you need three fundamental perspectives.

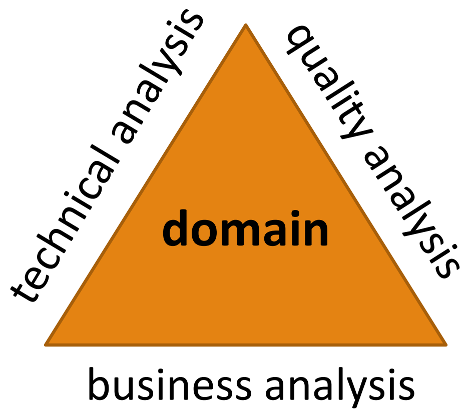


Share your experiences

Do you currently have a “three amigos” step in your process?

How have you seen the interplay of technical, business, and quality perspectives on a project?

What happens when one of these is non-existent or perhaps excessively dominant?



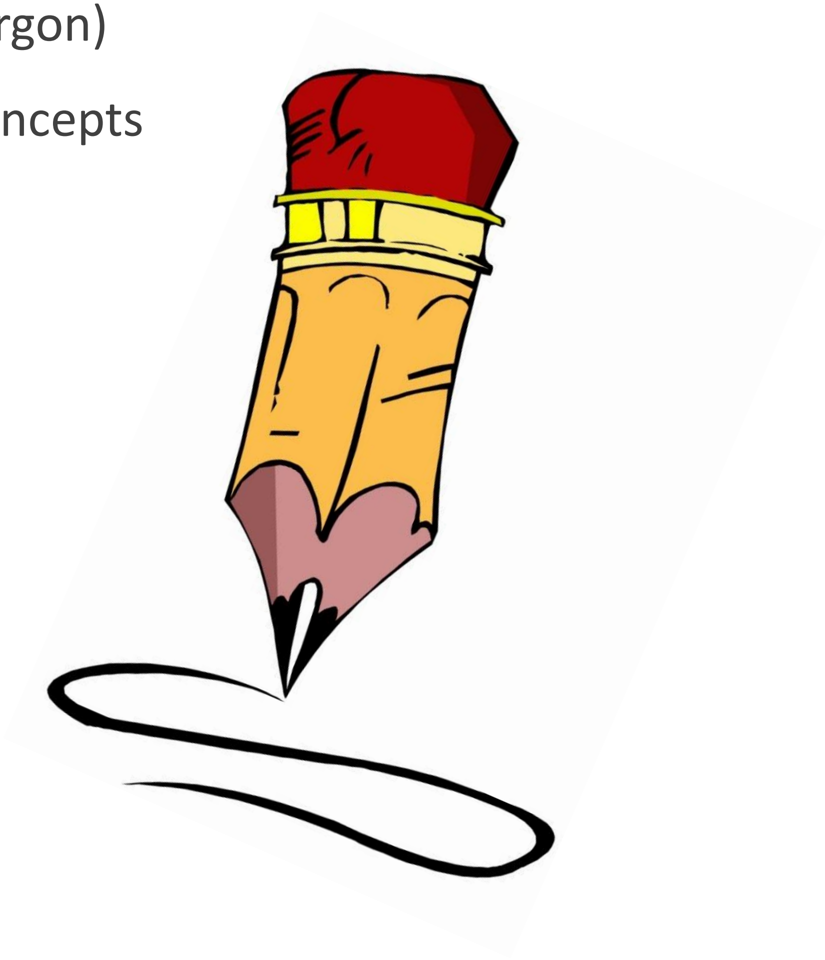
Effective story/scenario writing

Employ natural language (not technical jargon)

Incorporate domain-specific terms and concepts

Clearly convey the business goals

Customer involvement



Getting the right level of abstraction

Typically, a developer’s description is going to be more fine-grained than the customer’s description.

- When I fill my first name into the field “first name”
- When I fill my last name into the field “last name”
- When I fill….

In contrast, from a business standpoint, these elements might be more abstracted described in the following way.

- When I fill out the form…

<u>Rule of thumb:</u> *all of the scenarios belonging to a certain feature or user story should be described at the same abstraction level.*

The importance of shared capital

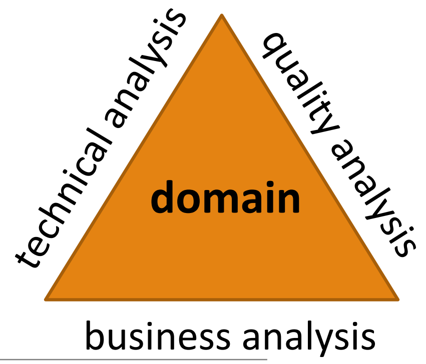
Why does testing so often get cut from the budget?

The answer may be blindingly simple: tests are often not seen as *shared capital*; they are not thought of as having value to the project, because "they're just for the engineers", or similarly, providing value to a particular department or one team.

How do we get buy-in and change the perception of tests to become valuable? By embracing not only test-driven development (TDD), but behavior-driven development (BDD).

BDD changes the conversation, because we aren't creating a fancy way to validate results. Instead, we are driving consensus **by sharing expected behaviors** <u>across all members of the team</u>.

Communication



Ultimately, successful Behavior Driven Development comes down to communication and collaboration.

- A technical person walks through an item’s functionality with the business person and analyzes the system through its internals (how the objects fit together internally).
- In parallel, the business person analyzes the system from the outside in terms of its business value.
- When possible, a third perspective is introduced in terms of quality and objective measures.

In this way, Behavior-Driven Development is a tool to aid collaboration and **start a conversation** within the team. It’s also a way to clarify the scope of a desired feature and get better estimates.

- Developers will have technical considerations (“What exactly do you mean when you say, ‘day’?”)
- Business people will have their own considerations (“Please don’t use the term ‘overdue’ here, that means something different”).

Course Agenda

Course Overview

BDD Basics and Principles

From TDD to BDD

Seeing it in Action

Domains and the Role of Business

Wrap-up

Wrap-up and Q&A

What aspects of BDD are still a bit fuzzy for you at this point?

Preview of next two days…

- BDD Tooling
- Gherkin Syntax
- Compound Steps
- Resolving Duplicates
- Parameters and Templates
- Outlines, Lists, and DataTables
- Before / After Stories
- Impact Mapping
- Applying BDD on a Project
- Final Review, Wrap-up, Q&A

BDD Primer & Workshop – Day 2

GOAL: GAIN HANDS-ON EXPERIENCE WITH BDD SOFTWARE TOOLS

Course Agenda

Day 1 Review

BDD Tooling

Working with Gherkin Syntax

Compound Steps

Resolving Duplicates

Parameters and Templates

BDD in a Nutshell

BDD is pretty simple:

- **describe** what you want the system to do by talking through example behavior
- work from the “outside-in” to **implement** those behaviors using the examples to validate what you are creating

Bring in different perspectives for validation:

- The customer (or Product Owner) describes what they want.
- Team members ask questions to flesh out “just enough” in order to implement the behavior.
- Business analysts and quality assurance specialists can also help to identify and elaborate specific details of various scenarios.

Two main parts

The BDD approach can largely be divided into two main parts.

First, the practice of <u>using examples</u> written in *ubiquitous language* to illustrate behaviors (how users will interact with the product).

Second, the practice of <u>using those examples</u> as the basis of *automated tests*.

This approach ensures alignment with business needs and user expectations, with a direct traceability to system behavior <u>as defined by the business</u> throughout the project lifetime.

BDD Story / Scenario Elements

Story

“In order to” – lead with the business value

“As a” – identify the role

“I want to” – describe the functionality

BDD Scenario

“Given” – describes the initial context for the example

“When” – describes the triggering event performed by an actor

“Then” – describes the expected outcome (which can be validated)

Example – 1 / 2

Story: Account Holder withdraws cash

In order to easily access my money

As an Account Holder

I want to withdraw cash from an ATM

Scenario 1: *Account has sufficient funds*

Given the account balance is \$100

And the card is valid

And the machine contains enough cash

When the Account Holder requests \$20

Then the ATM should dispense \$20

And the account balance should be \$ 80

And the card should be returned

Example – 2 / 2

Scenario 2: *Account has insufficient funds*

Given the account balance is \$10

And the card is valid

And the machine contains enough cash

When the Account Holder requests \$20

Then the ATM should not dispense cash

And the account balance should be \$10

And the card should be returned

Scenario 3: *Card has been disabled*

Given the card is disabled

When the Account Holder requests \$20

Then the ATM should retain the card

And the ATM should say the card has been retained

Group Exercise – BDD Reflections

1. Get back into your teams
2. Discuss BDD and its key elements, core disciplines, and techniques.
   - What one or two aspects of BDD do you believe are essential to get right?
   - What are some potential pitfalls that teams must pay close attention to when utilizing BDD?
   - What do you consider to be the greatest value BDD delivers for an organization?
3. Prepare a brief 2-3 minute presentation to share your group's findings with the class.

Course Agenda

Day 1 Review

BDD Tooling

Working with Gherkin Syntax

Compound Steps

Resolving Duplicates

Parameters and Templates

BDD Frameworks

There are a lot of them out there. They all have the same basic qualities:

- Declare the Given/When/Then structure
- Automate the execution of tests
- Provide a reporting mechanism for viewing results

We'll do a quick compare/contrast of three frameworks:

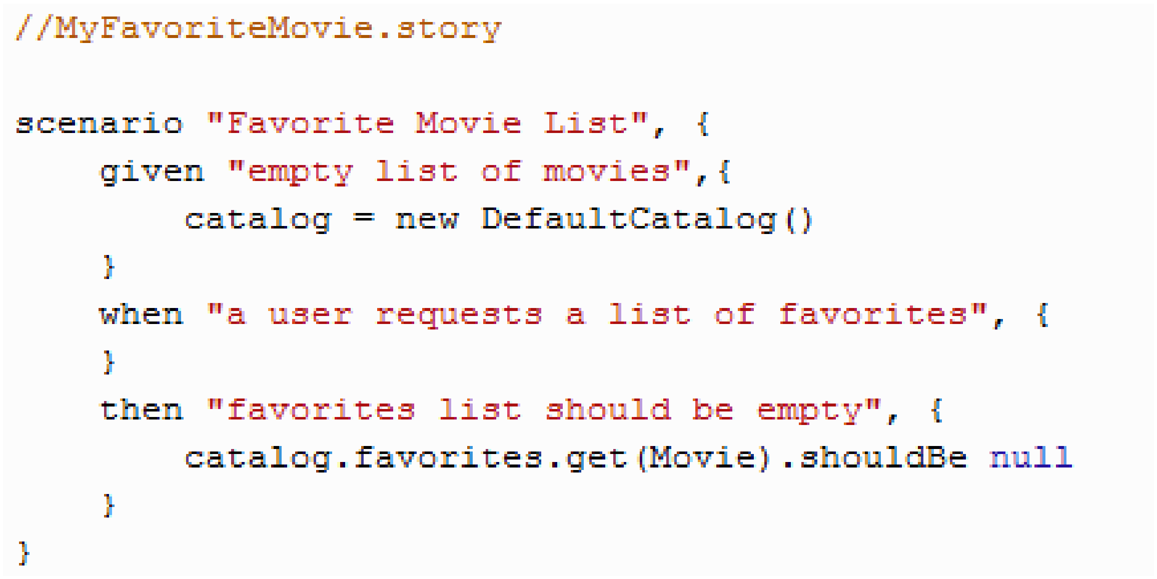
- Easyb
- JBehave
- Cucumber

Easyb

Language: Groovy

Execution Options: Maven Plugin

Binding style: Story files and test code woven together in same file



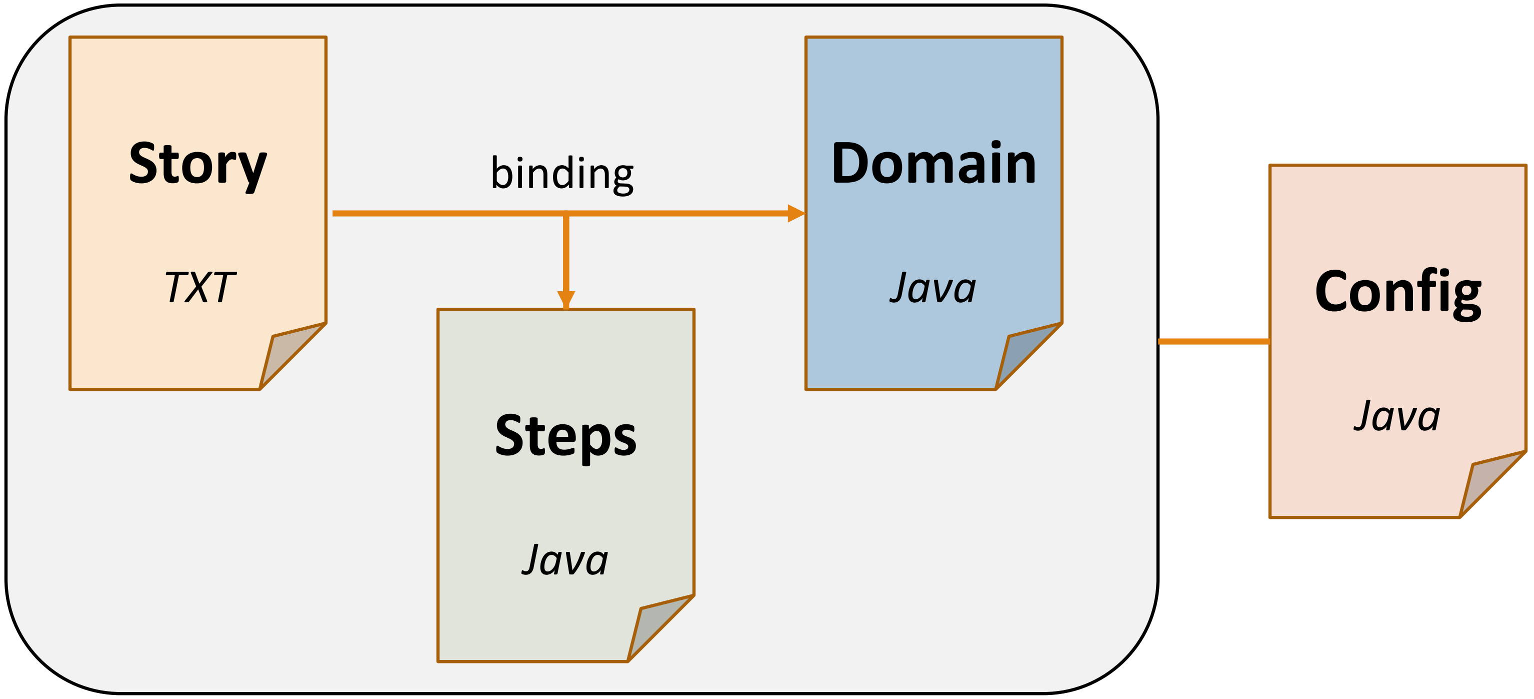
```
//MyFavoriteMovie.story

scenario "Favorite Movie List", {
    given "empty list of movies",{
        catalog = new DefaultCatalog()
    }
    when "a user requests a list of favorites", {
    }
    then "favorites list should be empty", {
        catalog.favorites.get(Movie).shouldBe null
    }
}
```

Jbehave & Cucumber

Language: Java

Execution Options: JUnit, Ant tasks, Maven goals, Eclipse, IntelliJ

Binding style: Story files, Steps classes, Test config, and Java code



JBehave vs Cucumber

They are actually quite similar, it is really a matter of style and preference.

- JBehave supports @Alias annotations whereas Cucumber leans more heavily upon RegEx expressions.
- JBehave is pure Java whereas Cucumber is originally Ruby-based and comes with extensive support for integrating with other languages.
- JBehave’s implementation style feels a bit more like a traditional Java.
- Cucumber’s Maven-style download/configuration is a bit smoother and lighter weight, fitting with its overall lightweight footprint. Jbehave is a bit more of a large toolbox of capabilities and lends towards a more traditional “download the toolkit” type of approach.

Other Tools

JUnit

Se

<APACHE ANT>

Jenkins

maven

BDD With / Without Tools

Behavior Driven Development is just a technique that can be used without any tools.

- You can write tests in BDD style - e.g. start test methods with “should”.
- Map methods to individual story/feature steps.
- Use comments to describe the Given/When/Then annotations.

You can choose to use tools, but make sure you don’t lose sight of the core BDD philosophy:

- Natural Language Constructs
- Domain Specific Language (DSL)
- Shared Process
- Value-driven, business-centric approach

Lab 0 – Lab Setup

1. Open Command Prompt (**`cmd`**)

   `java -version`

   Confirm Java version is 1.7 or higher

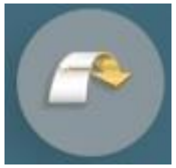
   `mvn -version`

   Confirm Maven version is 3.2 or higher

2. Configure maven proxy settings

   Copy `Lab0/settings.xml` to `${user.home}/.m2/`**`settings`**`.xml`

3. Open Eclipse

4. Click the Workbench icon 

5. File > New > Maven Project

6. Click "Next" x2

7. Group Id: Tester

8. Artifact Id: Tester

9. Click "Finish"

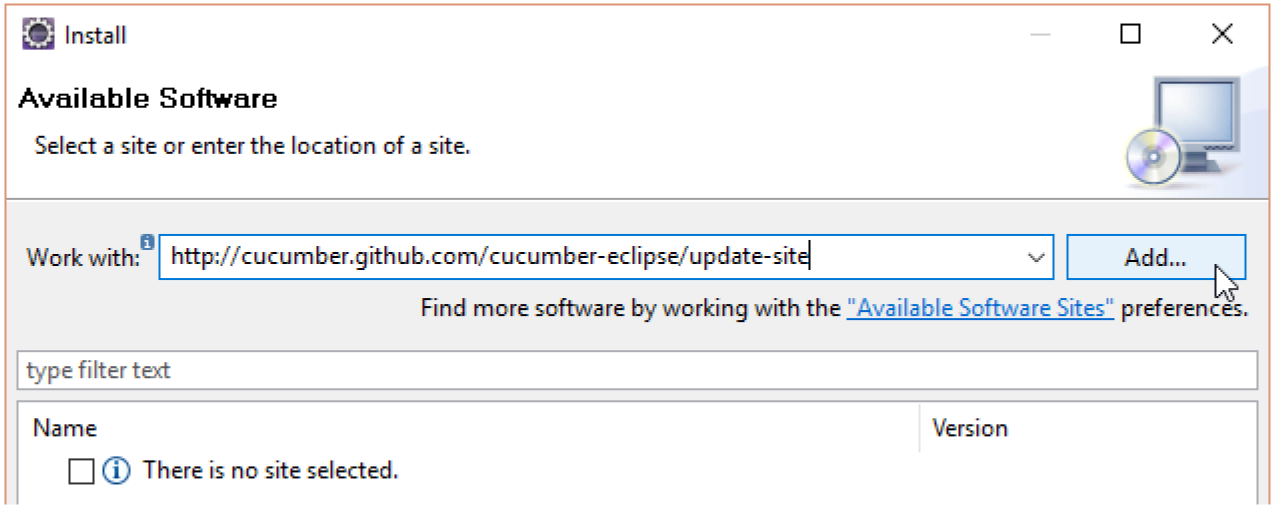
10. Right-click project

11. Run-As > Maven Install

12. Confirm "Build Success"

Lab 0b – Lab Setup Continued

Help > Install New Software


Install

Available Software

Select a site or enter the location of a site.

Work with: http://cucumber.github.com/cucumber-eclipse/update-site

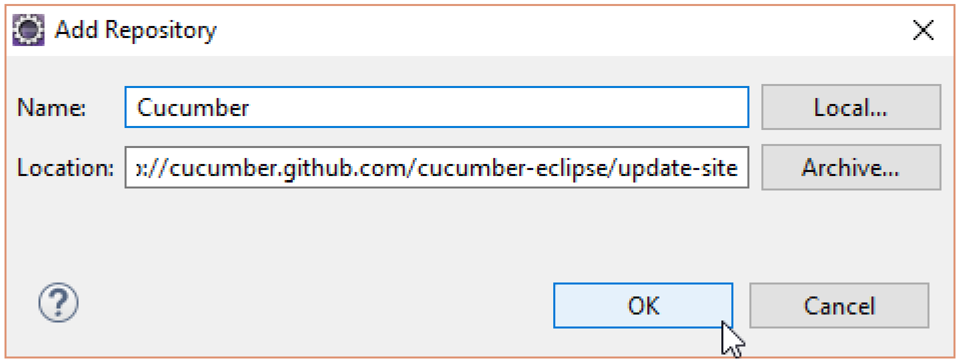
Find more software by working with the "Available Software Sites" preferences.

Name	Version
There is no site selected.	

Click the “Add” button

[Continued….]

Lab 0b – Lab Setup Continued



Name: Cucumber

Location: ɔ://cucumber.github.com/cucumber-eclipse/update-site

Check the "Cucumber Eclipse Plugin" box

“Next” x2

"I accept..."

“Finish”

May take a while. If you receive a security warning, click “OK”

Restart? Yes.

Course Agenda

BDD Review

BDD Tooling

Working with Gherkin Syntax

Compound Steps

Resolving Duplicates

Parameters and Templates

Gherkin Syntax – Given

Given

- Describes the initial condition(s) in place prior to the event occurring
- Within the testing context, the given statement(s) put the system into a known state to support the test
- If this were a use case, givens would be pre-conditions
- Avoid talking about workflow or user actions

<u>In order to</u> drive increased customer interaction

<u>As a</u> customer

<u>I want to</u> share company content through social media

What would be a couple of examples of valid Given statements?

Gherkin Syntax – When

When

- Describes the triggering event
- Within the testing context, the when statement(s) interact with the established pre-conditions from the Given statements
- May behave as a state transition
- Is performed by an actor

In order to drive increased customer interaction

As a customer

I want to share company content through social media

What would be a couple of examples of valid ***When*** statements?

Gherkin Syntax – Then

Then

- Describes the business outcome
- Must be objective and measureable (true / false, on / off, equal / unequal)
- Verify *Given + When* is or is not in the output
- Value should be observable

In order to drive increased customer interaction

As a customer

I want to share company content through social media

What would be a couple of examples of valid ***Then*** statements?

Lab 1 – Basic BDD Application

Locate *Lab 1 - Basic BDD Application.pdf* and follow the instructions.

Be sure and ask for help if you get stuck.

Prepare to share your results with the class.

Course Agenda

BDD Review

BDD Tooling

Working with Gherkin Syntax

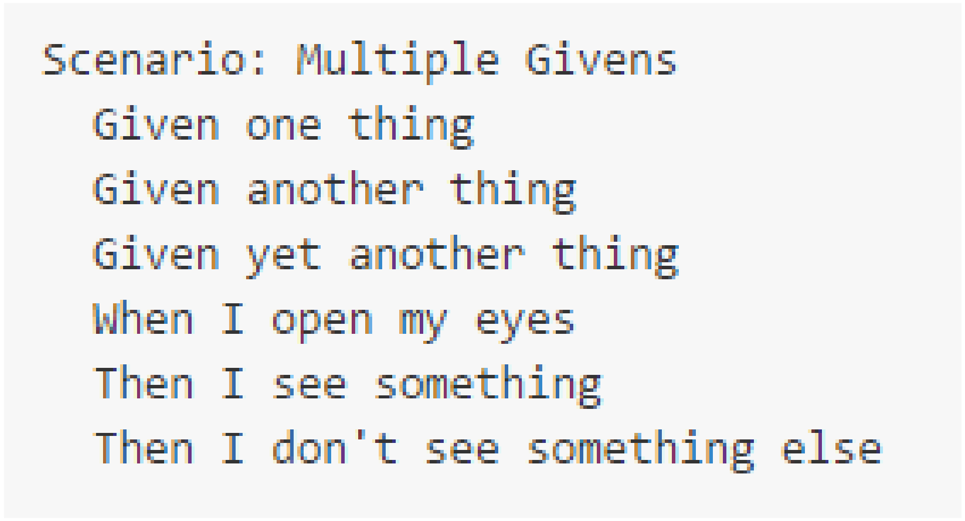
Compound Steps

Resolving Duplicates

Parameters and Templates

Compound Steps

If you have several givens, whens or thens you can write:



```
Scenario: Multiple Givens
  Given one thing
  Given another thing
  Given yet another thing
  When I open my eyes
  Then I see something
  Then I don't see something else
```

Lab 2a –BDD Syntax

Locate *Lab 2 – Working with BDD Syntax.pdf* and follow the instructions.

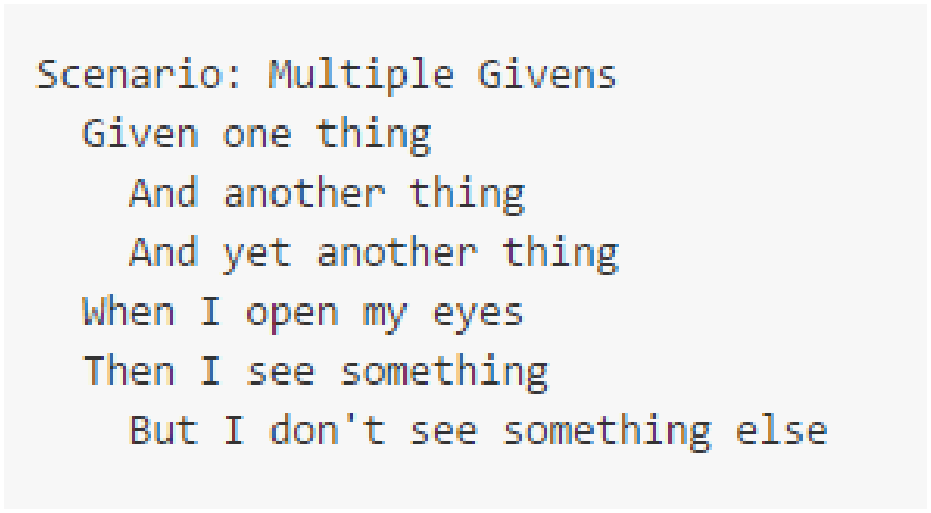
NOTE: Only complete section A of the lab for right now

Be sure and ask for help if you get stuck.

Prepare to share your results with the class.

Compound Steps (continued)

If you don’t like having repeated Given/When/Then statements, you can make it read more fluently by writing with “And”/”But”:



```
Scenario: Multiple Givens
  Given one thing
    And another thing
    And yet another thing
  When I open my eyes
  Then I see something
    But I don't see something else
```

Note: Cucumber treats “And” and “But” identically. They have no different in behavior, merely readability.

Lab 2b – BDD Syntax

Locate *Lab 2 – Working with BDD Syntax.pdf* and follow the instructions.

NOTE: Complete section b of the lab for now

Be sure and ask for help if you get stuck.

Prepare to share your results with the class.

Course Agenda

BDD Review

BDD Tooling

Working with Gherkin Syntax

Compound Steps

Resolving Duplicates

Parameters and Templates

Repeat Repeat Repeat

There's a lot of duplication in our content right now.

- Story / Feature class
  - Duplicate Given statements
  - Nearly identical Then statements
- Steps class
  - Repetitive methods

There are several tools in our tool belt for refactoring our code to deal with the repetition. We'll look at them each individually.

JBehave Alias

With an @Alias annotation, multiple Given/When/Then statements can be mapped to a single method call.

```
@Given(“a user has an account")
@Alias("a customer with an existing account")
public void hasAccount() {
     account = new GenericAccount();
}
```

Cucumber RegEx

Cucumber does not have an alias construct, preferring RegEx expressions instead:

```
@Given("a .* account")
public void hasAccount() {
     account = new GenericAccount();
}
```

Course Agenda

BDD Review

BDD Tooling

Working with Gherkin Syntax

Compound Steps

Resolving Duplicates

Parameters and Templates

RegEx Wildcards

.*	matches anything (or nothing), literally “any character (except a newline) 0 or more times”
.+	matches at least one of anything
[0-9]* or d*	matches a series of digits (or nothing)
[0-9]+ or d+	matches one or more digits
"[^"]*"	matches something (or nothing) in double quotes
an?	matches a or an (the question mark makes the n optional)

Capturing Part of the Input

Parenthesis allow you to capture part of the input for later use (such as storing in a parameter).

```
@Then("(\\d+) movies? should have been found" )
public void verifyMoviesFound(int quantity) {
     //method definition
}
```

Lab 3 – Refactoring

Locate *Lab 3 – Refactoring Scenarios.pdf* and follow the instructions.

Be sure and ask for help if you get stuck.

Prepare to share your results with the class.

RegEx String Anchors

^	anchors the expression to the front of the string
$	anchors the expression to the end of the string

The regular expression:

```
I'm logged in
```

matches:

```
I'm logged in
I'm logged in as an admin
```

to avoid ambiguity:

```
^I'm logged in$
```

RegEx String Wildcards

.+	matches any string of characters
(.+)	*matches any string of characters and saves the match for later use as a non-string type*
‘(.+)’	*matches string of characters and saves for later use as a string type*

```
@Given(".+book with the title '(.+)', authored by '(.+)'")
public void addBook(final String title, final String author) {
     Book book = new Book(title, author);
     library.addBook(book);
}
```

Lab 4 – RegEx Cleanup

Refactor `MovieSearchSteps.java` using RegEx substitution so that you have one single Given statement mapped to a single Java method.

Remember:

.+	*matches any string of characters*
(.+)	*matches any string of characters and saves the match for later use as a non-string type*
'(.+)'	*matches string of characters and saves for later use as a string type*

Non Capture Groups

Scenario Steps:

```
When I’m logged in as an ‘Admin’

When I log in as a ‘Manager’
```

Annotation Matching (invalid):

```
@When(“^(I'm logged|I log) in as an? ‘(.*)’$")]
public void LogInAs( String role )

{ // log in as the given role }
```

Annotation Matching (valid):

```
@When(“^(?:I'm logged|I log) in as an? ‘(.*)’$")]
public void LogInAs( String role )

{ // log in as the given role }
```

Day 2 Retrospective

Good and Better If…

Good – *What worked well that you would like to see us continue with tomorrow?*

Better If – *What could have been better if we had done it differently?*

BDD Primer & Workshop – Day 3

GOAL: ENHANCE KNOWLEDGE OF BDD TOOLING AND EXPLORE BUSINESS AND PROJECT IMPACTS

Day 3 – Agenda

Review of Day 2

Scenario Outlines, Lists, and DataTables

Before / After Stories

Other Features

Impact Mapping

Steps to Incorporate BDD into a Project

Final Review, Wrap-up, Q&A

Review of Day 2

- Core BDD concepts/elements
- BDD Story vs Classic Agile Story
- Qualities of a good/bad *Given* clause
- Qualities of a good/bad *When* clause
- Qualities of a good/bad *Then* clause
- Name the four types of files used by JBehave / Cucumber
- Using regular expressions to define annotations
  - Wildcards (string, digits, other)
  - Optional characters
  - String anchors

Day 3 – Agenda

Review of Day 2

Scenario Outlines, Lists, and DataTables

Before / After Stories

Other Features

Impact Mapping

Steps to Incorporate BDD into a Project

Final Review, Wrap-up, Q&A

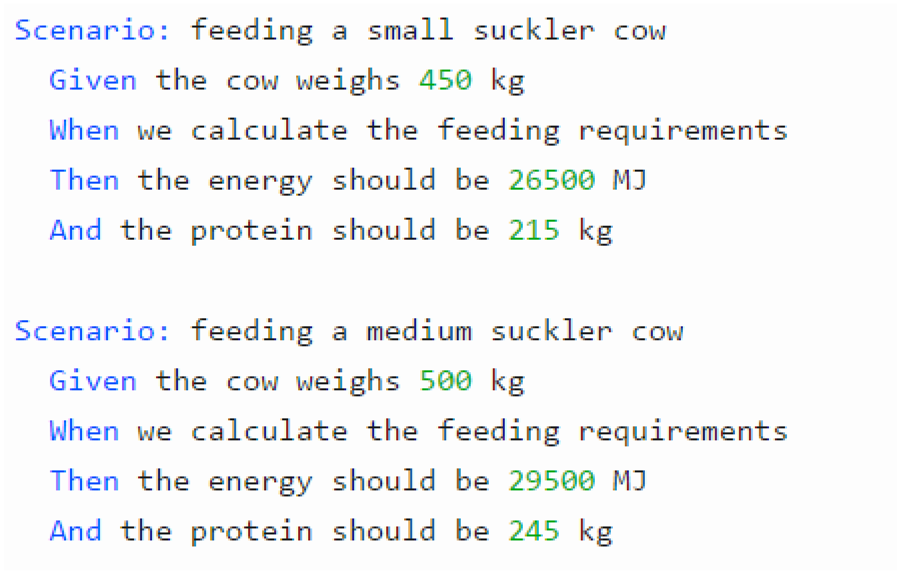
Given…several things

When defining features/stories, its common to have a stack of data you want to use in setting up pre-conditions.

- **Scenario outline** – *template-driven parameter substitution*
- **Data lists** – *a one-dimensional list / enumeration*
- **Data table** – *a multi-dimensional map of name/value pairs*

Scenario Outline – 1/2

You may find you have several scenarios which have a repetitive structure, but differ in terms of their specific values.



```
Scenario: feeding a small suckler cow
  Given the cow weighs 450 kg
  When we calculate the feeding requirements
  Then the energy should be 26500 MJ
  And the protein should be 215 kg

Scenario: feeding a medium suckler cow
  Given the cow weighs 500 kg
  When we calculate the feeding requirements
  Then the energy should be 29500 MJ
  And the protein should be 245 kg
```

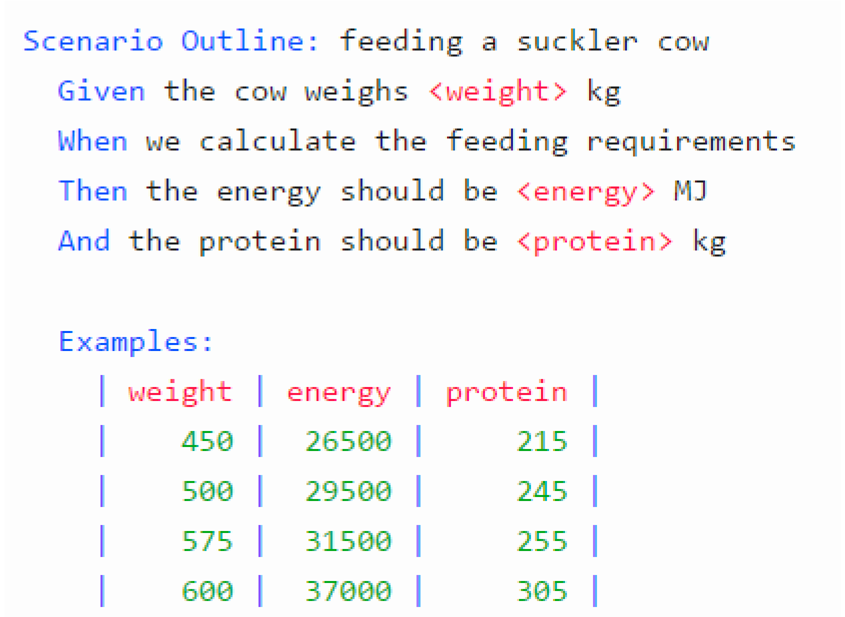
This is best solved by creating a template in the form of a *Scenario Outline*…

Scenario Outline – 2/2

Variables in the outline are indicated using angle brackets <VARIABLE>.

The first row of the table must be a header row.

Each column header must match to a variable in the outline.

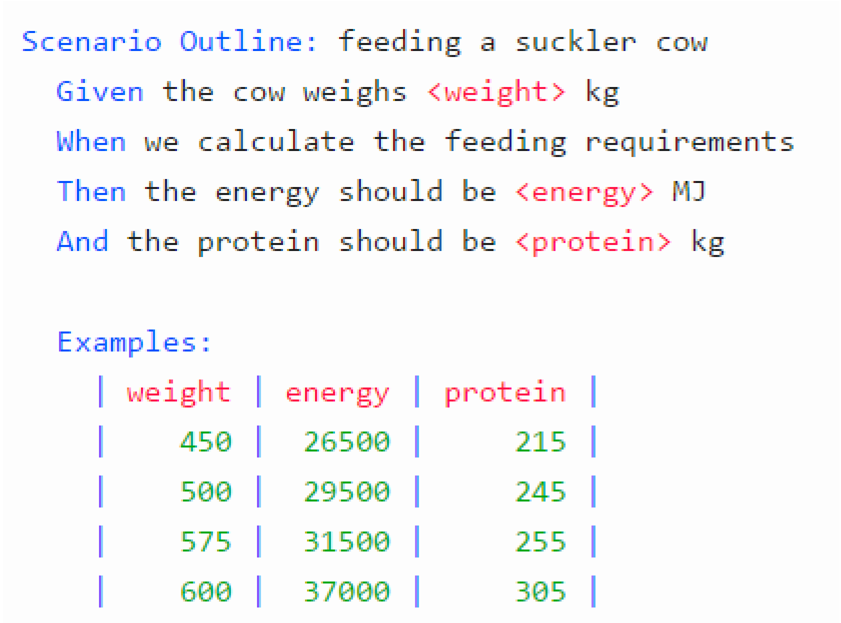


```
Scenario Outline: feeding a suckler cow
  Given the cow weighs <weight> kg
  When we calculate the feeding requirements
  Then the energy should be <energy> MJ
  And the protein should be <protein> kg

  Examples:
    | weight | energy | protein |
    |    450 |  26500 |     215 |
    |    500 |  29500 |     245 |
    |    575 |  31500 |     255 |
    |    600 |  37000 |     305 |
```

IMPORTANT: *Your step definitions will never have to match a placeholder. They will need to match the values that will replace the placeholder.*

Lab 5 – Scenario Outline

1. Open `search_movie.feature` file
2. Refactor the *"Search movies by title"* scenario to use a Scenario Outline structure
3. Save and test it.



```
Scenario Outline: feeding a suckler cow
  Given the cow weighs <weight> kg
  When we calculate the feeding requirements
  Then the energy should be <energy> MJ
  And the protein should be <protein> kg

  Examples:
    | weight | energy | protein |
    |    450 |  26500 |     215 |
    |    500 |  29500 |     245 |
    |    575 |  31500 |     255 |
    |    600 |  37000 |     305 |
```

Data Lists

Data lists allow you to pass a simple list of Strings into the Steps class:

```
Given the following pets: cat, dog, lizard, monkey
```

In the Steps class, the values pass in as a List of type String:

```
@Given("the following animals: (.*)")
public void the_animals(List<String> animals) {}
```

Data Table

Data Tables look similar to a Scenario Outline, but are really designed to pass a set of data in as a direct part of a step.

```
Given the following users exist:

| name   | email              | twitter         |
| Aslak  | aslak@cucumber.io  | @aslak_hellesoy |
| Julien | julien@cucumber.io | @jbpros         |
| Matt   | matt@cucumber.io   | @mattwynne      |
```

Define a custom object type to represent the columns of the table.

```
@Given("a user list exists")

public void userList(List<User> userList) {
       this.userList = userList;

}
```

Lab 6 – Catalog Status Project

1. Import the Story/Feature file "Catalog Status" from the *Lab6* folder
2. Define two scenarios for this story
   - Add a couple movies and validate the size of the catalog has increased by 2 from what it was previously
   - Add no movies and validate the size of the catalog is zero
3. Define a `CatalogStatusSteps.java` class
   - Use the `MovieSearchStatus.java` class as a guide
   - Create a `Catalog` and `List<Movie>` instances as well
   - Define only the annotated methods needed for this story
4. Save and Test!

BONUS: Find an opportunity to fold in a Scenario Outline, Data List, or Data Table into this lab.

Day 3 – Agenda

Review of Day 2

Scenario Outlines, Lists, and DataTables

Before / After Stories

Other Features

Impact Mapping

Steps to Incorporate BDD into a Project

Final Review, Wrap-up, Q&A

Background

Occasionally you'll find yourself repeating the same Given steps in all of the scenarios in a feature file. Since it is repeated in every scenario it is an indication that those steps are not essential to describe the scenarios, they are incidental details.

You can literally move such Given steps to the background by grouping them under a Background section before the first scenario:

```
Background:
  Given something present in multiple scenarios
  And something else which appears frequently
```

Lab 7 – Background

Refactor the Search Movie feature to utilize a Background definition within the Feature class.

```
Feature: feature title
narrative story description (in order to / as a /
I want)


Background:

      //Place common Given statements here


/* Scenario Definitions Next */
```

Day 3 – Agenda

Review of Day 2

Scenario Outlines, Lists, and DataTables

Before / After Stories

Other Features

Impact Mapping

Steps to Incorporate BDD into a Project

Final Review, Wrap-up, Q&A

Other Features

Calling step definitions from within a step

Flexible reporting capabilities

Selenium integration

Jenkins integration

Hooks for before, after, around, etc.

Tags for running a subset of available stories

Day 3 – Agenda

Review of Day 2

Scenario Outlines, Lists, and DataTables

Before / After Stories

Other Features

Impact Mapping

Steps to Incorporate BDD into a Project

Final Review, Wrap-up, Q&A

Goal Alignment

BDD places an emphasis upon business goals. **Every new or changed behavior should have a tangible impact upon the business**. A clearly identifiable value should be the goal of each story/scenario.

Whatever the desired business outcome looks like should be the driving force behind your project. Every story and scenario should tie back to one or more business goals.

Goal alignment is achieved in several ways:

- Story structure leads with the WHY
- Each scenario should represent a verifiable business outcome
- Feature files include the story and comments to reinforce the value
- On the front-end, brain-storming work should be goal-centric (see *impact mapping* technique, discussed next)

Good Goals vs Bad Goals

A good business goal is specific, measurable, achievable, realistic and time-bound (SMART).

Bad example: We are going to generate more revenue by working faster

Good example: We are planning to achieve a 15% increase in revenue by reducing our average time-to-market for new capabilities from 3 months to 6 weeks

Having a SMART goal provides <u>focus</u>, <u>measurable impact</u>, and a <u>direct linkage</u> back to what drives the business.

How do we achieve our goals?

Once you have a defined goal, you’re ready to plan how you will go about achieving that target state.

Classic answer? *“We can achieve our goals by delivering new features on our applications.”* What could possibly go wrong? As it turns out…a lot.

As Dan North states: *“Delivering features for the sake of delivering features is a common problem in software development, and is the reason behind many of the most infamous project failures. Instead of trying to deliver as many features as possible, you need to identify how the features will support the user’s behaviours or the business.”*

Since the hundreds of potential features are likely to require different levels of time, energy, and skill, it is important to systematically navigate through you growing backlog. An increasingly popular strategy for this is known as **Impact Mapping**.

Impact mapping

Impact Mapping is a technique that helps you to outline all of the alternative ways to reach your decided goal and works by analyzing users' behavior. Since software cannot achieve business goals independently; the only way software can get us closer to the business goal is by supporting particular human behaviors.

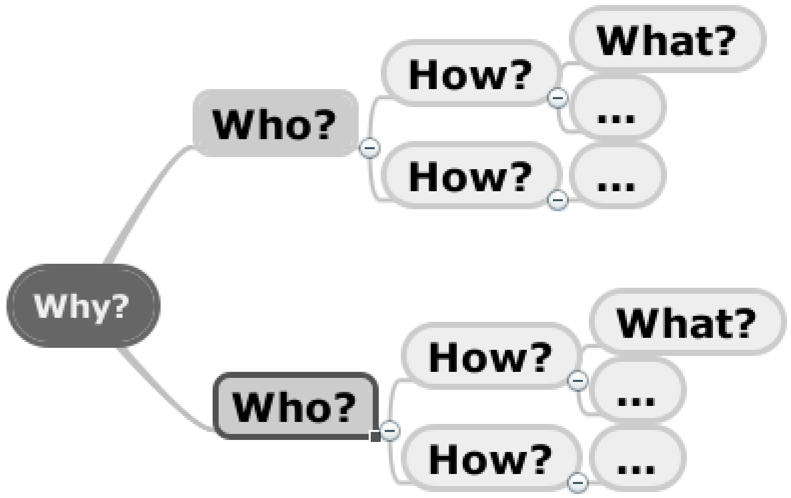
For example, an online shop doesn't bring you money on its own - people bring you money by purchasing products from it. Similarly, better system logging doesn't improve performance of your website on its own - technical specialists using it do. Looking at software through a prism of those who use it is a very valuable way to plan and direct the project delivery.

Impact Mapping Elements

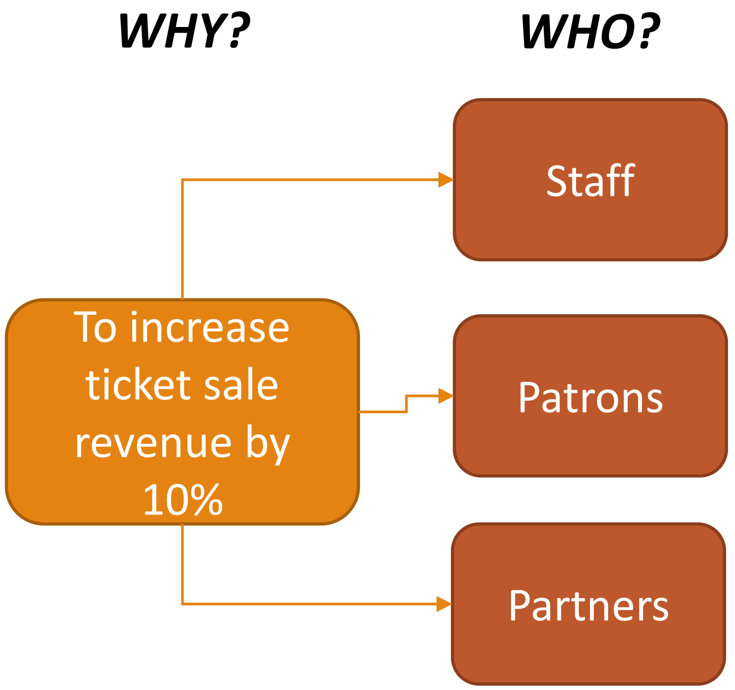
Impact Mapping is a simple technique built on top of conventional mind mapping, with a bit of twist.

An impact map includes four distinct levels of detail:

- Business goal [WHY]
- One or more actors [WHO]
- One or more impacts [HOW]
- Ways to support/prevent these impacts [WHAT]



Impact Example – 1/2

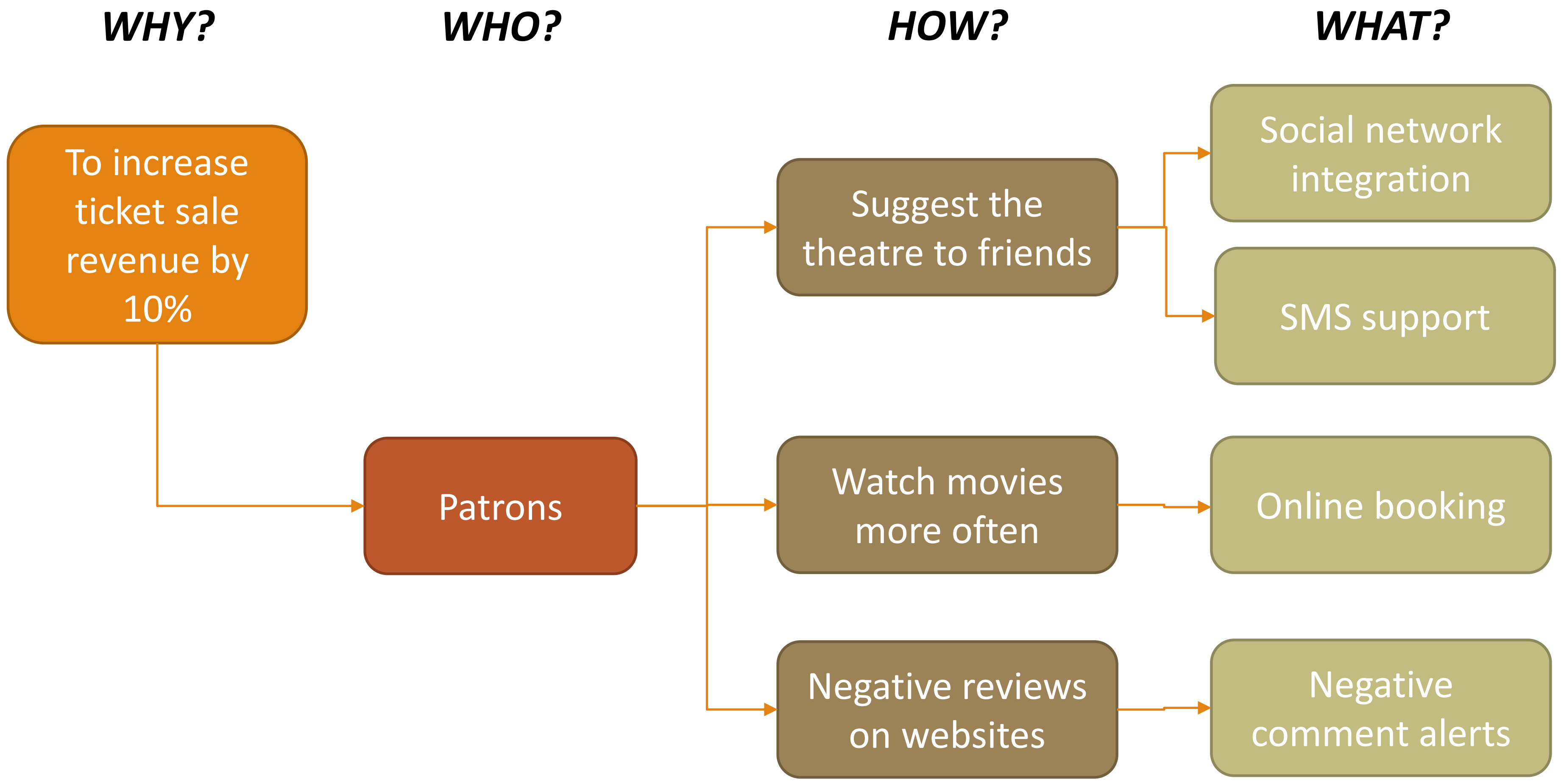


HOW?

How can this actor *<u>positively</u>* ***impact*** *my goal?*

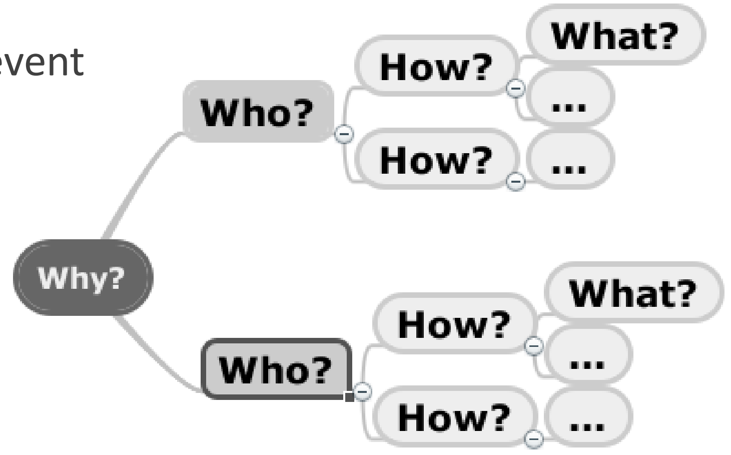
How can this actor *<u>negatively</u>* ***impact*** *my goal?*

Impact Example – 2/2


WHY?
WHO?
HOW?
WHAT?
To increase ticket sale revenue by 10%
Patrons
Suggest the theatre to friends
Watch movies more often
Negative reviews on websites
Social network integration
SMS support
Online booking
Negative comment alerts

Class Exercise – Goal Focus

1. As a class, let's utilize the Impact Mapping technique.
2. What project scope would you like to use?
3. Define a SMART goal within the context of your project scope.
   Specific, Measureable, Achievable, Realistic, Time-bound
4. Build out an Impact Map
   - Select at least two actors
   - Identify 1-3 impacts for each actor
   - Capture at least one way to support / prevent the impact (these ultimately map to epics/stories of functionality)



Food for thought – *How does this change the way you think about identifying stories and scenarios?*

Connecting the Dots

OK, so we've got this Why/Who/How/What Impact Mapping framework. What do we do with it though?

Stories are defined to support the impacts we have identifed for the project. They represent threads of activity in scenario form for effecting a change in the current environment.

Some people choose to strictly pull all the elements of an impact map into each story (i.e. the WHY/WHO/HOW/WHAT) while others use it as a framework to guide the high-level conversation and then create stories at the lowest level. Both approaches are valid.

Day 3 – Agenda

Review of Day 2

Scenario Outlines, Lists, and DataTables

Before / After Stories

Other Features

Impact Mapping

Steps to Incorporate BDD into a Project

Final Review, Wrap-up, Q&A

Ways to include BDD

Incorporate elements or include it all!

- BDD as a way to drive a shared understanding
- BDD with fully automated tooling
- With / without the goal-focus
- With / without impact analysis

Remember, it's a spectrum!

How do you anticipate the adoption of BDD looking for your team?

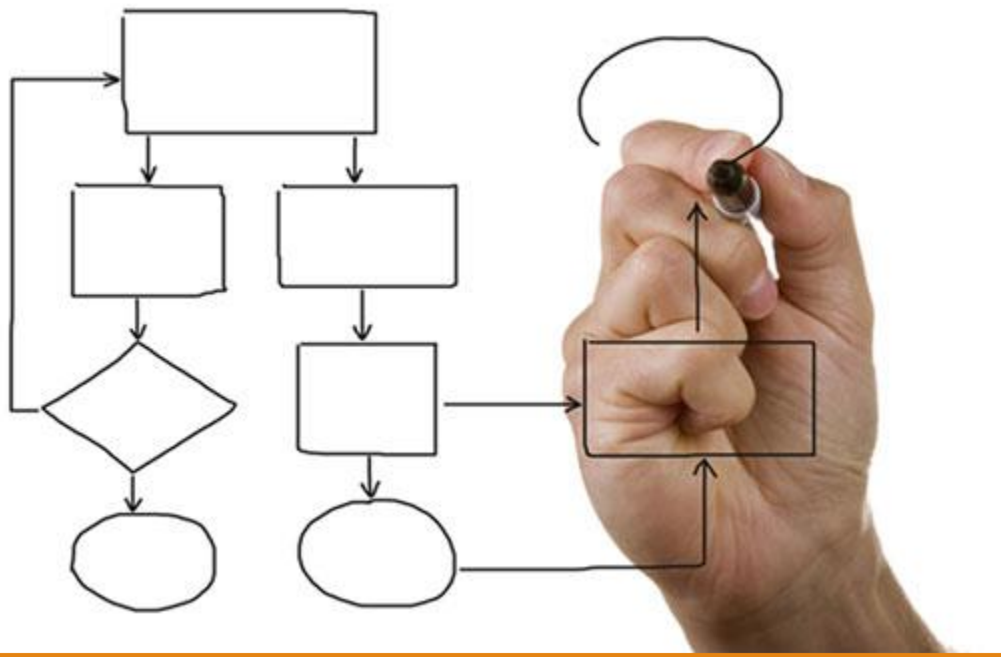
Where in your process does BDD make sense?

Early – *impact mapping and goal-focus*

Middle – *shared understanding through scenario-based examples*

Late – *test-centric alignment between business and technical roles*

Hybrid – *mixture of the above*



Day 3 – Agenda

Review of Day 2

Scenario Outlines, Lists, and DataTables

Before / After Stories

Other Features

Impact Mapping

Steps to Incorporate BDD into a Project

Final Review, Wrap-up, Q&A

Complete BDD Story Structure

Title: *The story should have a clear, explicit title.*

Narrative: *A short, introductory section that specifies three things…*

- who (which business or project role) is the driver or primary stakeholder of the story (the actor who derives business benefit from the story)
- what effect the stakeholder wants the story to have
- what business value the stakeholder will derive from this effect

Acceptance criteria or scenarios: *a description of each specific case of the narrative. Such a scenario has the following structure…*

- specify the initial condition that is assumed to be true at the beginning of the scenario. This may consist of a single clause, or several
- state which event trigger initiates the scenario
- state the expected outcome, in one or more clauses

OPTIONAL (if time allows)

Final Project

1. Organize into teams of 2-3 (make sure you have at least one technical person per group).
2. Define the scope of your project (choose a simple domain to work within such as an “online directory” or “food ordering kiosk”).
3. Craft two user stories in the context of that scope.
   - **In order to…As a…I want to…**
4. Select one story and create one behavior-driven scenario:
   - **Given…When…Then…**
5. Write the necessary code to minimally demonstrate this BDD scenario
   - Start with a blank project
   - Write the files from scratch (feel free to inspect the other project or even copy files over as a starting point when necessary)

Q&A

What was your favorite nugget?

What concept or feature of BDD did you enjoy most?

If you could only go and apply one BDD technique on an up-coming project, which technique would it be, and why?

THANK YOU!