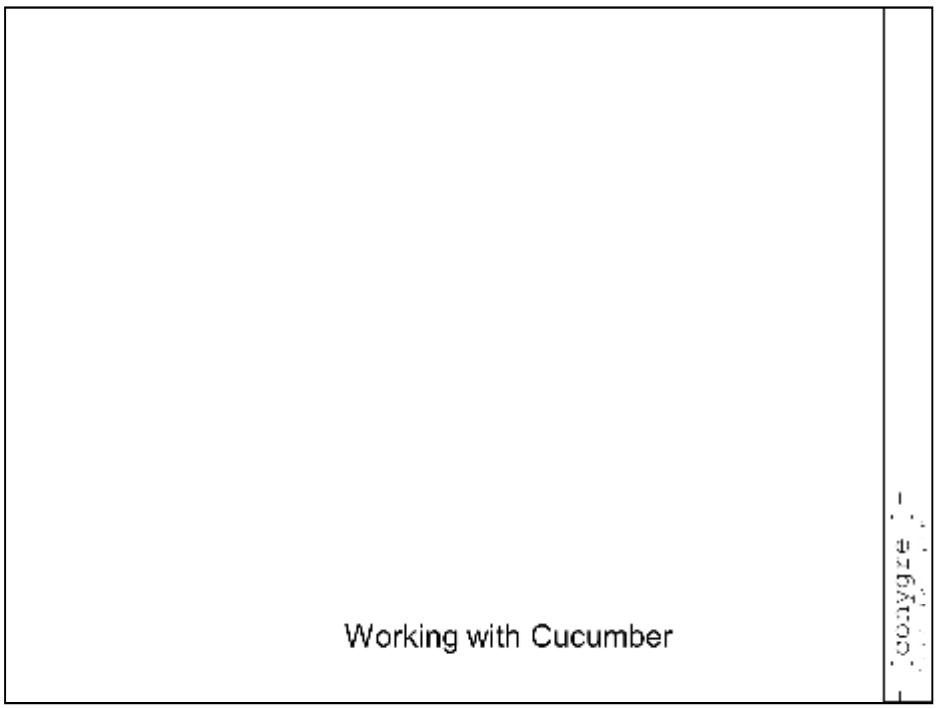


Objectives

- What is Behaviour Driven Development (BDD)
- The Benefits of BDD
- The Relationship between BDD and Other Agile Practices
- What is Cucumber?
- Defining Features and Scenarios
- Linking Features and Scenarios to Tests

What is BDD?

- Behavior-driven development (BDD) is a software development methodology in which an application is specified and designed by describing how its behavior should appear to an outside observer.
  - *techtarget.com*

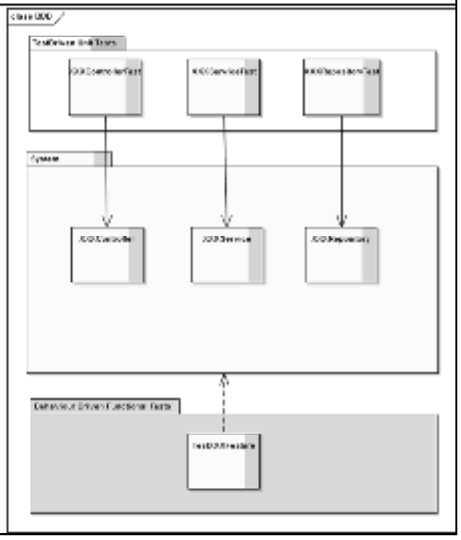
Benefits of BDD

- Automated functional testing
- Completeness of functional testing
- Application only does what is required
- Behavioural requirements are captured in a format understood by technologists and business people alike
- Test cases are defined up front and the requirements are also the test cases

-[conygre]-

Relationship with Other Practices

- BDD tests are written first, then the implementation of the system can then be built out using Test Driven Development



BDD and User Stories

- User Stories, which are a common way of capturing requirements in Agile projects, have acceptance criteria specifying the specific testable requirements for a feature
- The acceptance criteria can be captured as the behaviours required when applying BDD
  - But how? This is where testing tools like Cucumber come in

-[conygre]-

Introducing Cucumber

- Cucumber is a tool that facilitates the creation and running of BDD tests
- Cucumber supports many different languages and platforms including
  - Java
  - C#
  - C++
  - Scala

-[conygre]-

Cucumber Feature Files

- The key artefact in Cucumber is the **Feature File**
- The Feature file specifies the required behaviour as a series of **scenarios** which provide ***examples*** of how the system must work
  - This results in *example based specifications*

Worked Example – Time To Text

- Consider the example of a system that provides a text value for a given time of day
  - 12pm returns the text "midday"
  - 12am returns the text "midnight"
- This can be captured in the form of a feature file

Feature File

- Below is a Feature file for our system

Feature: SpeakingClock
In order to get the time as text
I want to be able to pass in a Datetime
So that I can get a suitable String back

Scenario: ConvertMidnightToText
Given the time is 0 hours and 0 minutes
When I request the time
Then the result should be midnight

Scenario: ConvertMiddayToText
Given the time is 12 hours and 0 minutes
When I request the time
Then the result should be midday

SpeakingClock.feature

Feature File Explained

- The top level Feature describes the overall functionality to be implemented

```
Feature: SpeakingClock
In order to get the time as text
I want to be able to pass in a Datetime
So that I can get a suitable String back
```

- This is not read by a machine, but will appear in your automated test results

The Scenarios

- Each Scenario follows a standard convention

Keyword	Purpose	
Given	Arrange the test	Arrange
And	Additional aspects for arranging the test (if required)	Arrange
When	Specify the action that is to be done that requires testing	Act
Then	Assert what you expect the result to be	Assert

- For those of you familiar with unit testing, you might recognise the sections as equivalents of
  - ***Arrange, Act, Assert***

Scenario Example

- Below is an example scenario for out text to time engine

Scenario: **ConvertMidnightToText**
Given the time is 0 hours and 0 minutes
When I request the time
Then the result should be midnight

Creating Feature Files

- Scenarios can be created by business people and technical people
- Each feature will have multiple scenarios that capture examples of how the system must work
- Those involved in the process must agree on a common terminology for the domain in which they are working
  - The agreed terms are then used consistently in the scenario text (important for automation)

– [conygre] –

What about the Code?

- In order for the feature file to run as an automated test against the system, a code artefact needs to be created
- These code artefacts are called **Step Flow Definitions**
- These can be written in one of the supported languages

-[conygre]-

Step Flow Definition

- Below is the step flow definition for our example written in C#

```
public class SpeakingClockSteps  {
   [Given(@"the time is (.*) hours and (.*) minutes")]
   public void GivenTheTimeIsHoursAndMinutes(int hours, int minutes)  {
            // set up a clock
   }
   [When(@"I request the time")]
   public void WhenIRequestTheTime() {
         //request the time from the clock
   }
   [Then(@"the result should be (.*)")]
   public void ThenTheResultShouldBeTimeAsText(string expectedTimeAsText) {
         // assert what you expect to happen
  }
}
```

Steps Explained

- Each step is processed by Given / When / Then attributes
- Each attribute contains a regular expression with a pattern matching the text in the scenario
- Parameter values in the text are placed in parentheses which are then automatically available as method parameters

```
[Given(@"the time is (.*) hours and (.*) minutes")]
public void GivenTheTimeIsHoursAndMinutes(int hours, int minutes)
```

– [conygre] –

Within the Methods

1. Arrange the system so it is ready for testing

```
[Given(@"the time is (.*) hours and (.*) minutes")]
pub ic void GivenTheTimeIsHoursAndMinutes(int hours, int minutes) {
    dateTime = new DateTime(2016,10,27,hours,minutes,0);
    speakingClock = new SpeakingClock();
}
```

2. Exercise the system to see what happens

```
[When(@"I request the time")]
public void WhenIRequestTheTime() {
  actualTimeAsText = speakingClock.GetTimeAsText(dateTime);
}
```

3. Verify the result

```
[Then(@"the result should be (.*)")]
public void ThenTheResultShouldBe(string expectedTimeAsText) {
    Assert.AreEqual(expectedTimeAsText, actualTimeAsText);
}
```

– [conygre] –

Finally - What was Tested

- Below is the simple class we were testing

```
public class SpeakingClock : ISpeakingClock
{
    public string GetTimeAsText(DateTime dateTime  {
        if (dateTime.Hour == 0 && dateTime.Minute == 0)
            return "midnight";
        else if (dateTime.Hour == 12 && dateTime.Minute == 0)
            return "midday";
        return null;
    }
  }
}
```

Setting up Cucumber in C#

- In Visual Studio you can install the SpecFlow Cucumber tools
- In the .NET project then add the following references
  - TechTalk.SpecFlow
  - TechTalk.SpecFlowRunner
- You will then have wizards to help create the Feature files
  - The StepFlow class stubs can be generated automatically from the feature file

Setting up Cucumber in Java

- Both IntelliJ and Eclipse have plugins and extensions for Cucumber
- Projects built with Maven can automatically run your cucumber tests

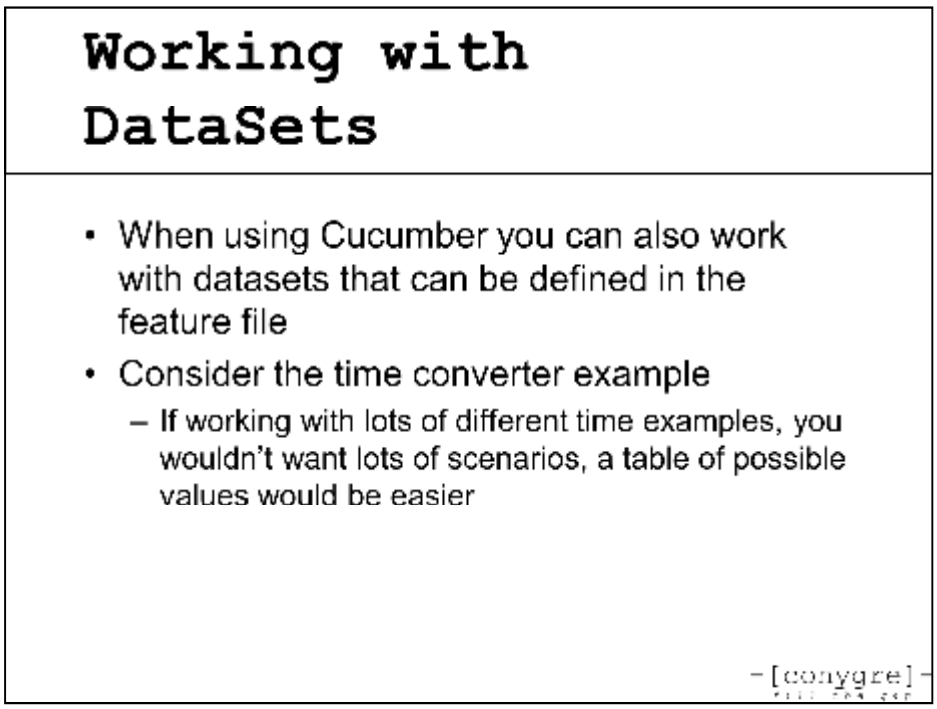
```
<dependency>
  <groupId>info.cukes</groupId>
  <artifactId>cucumber-java</artifactId>
  <version>${cucumber.version}</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>info.cukes</groupId>
  <artifactId>cucumber-junit</artifactId>
  <version>${cucumber.version}</version>
  <scope>test</scope>
</dependency>
```

-[conygre]-

Java Step Flow Definitions

- Java step flows are almost identical to the C# ones except annotations are used instead of attributes

```
@When("^the customer searches for items published between (\\d+) and (\\d+)$")
public void setSearchParameters(@Format("yyyy") final Date from, @Format("yyyy") final Date to) {
  // act
}
```



DataSet Example C#

- Below is a scenario written with a set of data

```
Scenario: ConvertMultipleTimesToText
Given I have been provided this set of times and expected results

  | Hours | Minutes | ExpectedText |
  | 0     | 0       | midnight     |
  | 12    | 0       | midday       |

When I request all the times as text
Then all the text should match
```

- Create a C# class to match the table

```
public class TimeTable  {
   public int Hours { get; set; }
   public int Minutes { get; set; }
   public string ExpectedText  { get; set; }
}
```

Processing the Set

- To process the set, the method with the Given attribute is different

```
private List<TimeTable> timeTables;
[Given(@"I have been provided this set of times and expected results")]
public void GivenIHaveBeenProvidedThisSetOfTimesAndExpectedResults(Table tableOfTimes)
{
  timeTables = tableOfTimes.CreateSet<TimeTable>().ToList();
}
```

- SpecFlow provides a **Table** class that has methods to convert your table into a regular List<YourType> so long as the properties of the type match the table columns

DataSet Example Java

- Java is even easier than C#, just create a class matching your table with getters/setters
- Then have the @Given method to take in a parameter of a List<YourType>

```
@Given("^the system is initialized with the following data$")
public void initializeTheSystem(final List<TimeTable> timeTables) {
  // arrange
}
```

-[conygre]-

Summary

- What is Behaviour Driven Development (BDD)
- The Benefits of BDD
- The Relationship between BDD and Other Agile Practices
- What is Cucumber?
- Defining Features and Scenarios
- Linking Features and Scenarios to Tests