HINDSIGHT
JOIN THE TESTING REVOLUTION

Test Automation with Cucumber-JVM

Boston Java Meetup, 13/06/2013

Alan Parkinson

@alan_parkinson

Cucumber is a **tool** that executes **plain-text** functional descriptions as **automated tests**

- Tests written in business-facing text
  - Supports over 40 spoken languages
  - This test is called Gherkin
- Automation written in target language
  - Ruby, Java, Scala, Groovy, etc.. .net, python, PHP

Behaviour Driven Development

Software - What can go wrong?

IMPLEMENTATION DEFECT

- Program does not do what it is intended
- Fixed in development/QA

REQUIREMENTS DEFECT

- Program does what is intended
- Unstated, misunderstood or tacit requirements

A woman asks her husband, a programmer, to go shopping.

She tells him “please, go to the nearby grocery store to buy some bread. Also, if they have eggs, buy 6”

50-60%

of issues identified by testers can be chalked down as

Requirements Defects

100-200%

more expensive to fix than other defects because

the code is already written

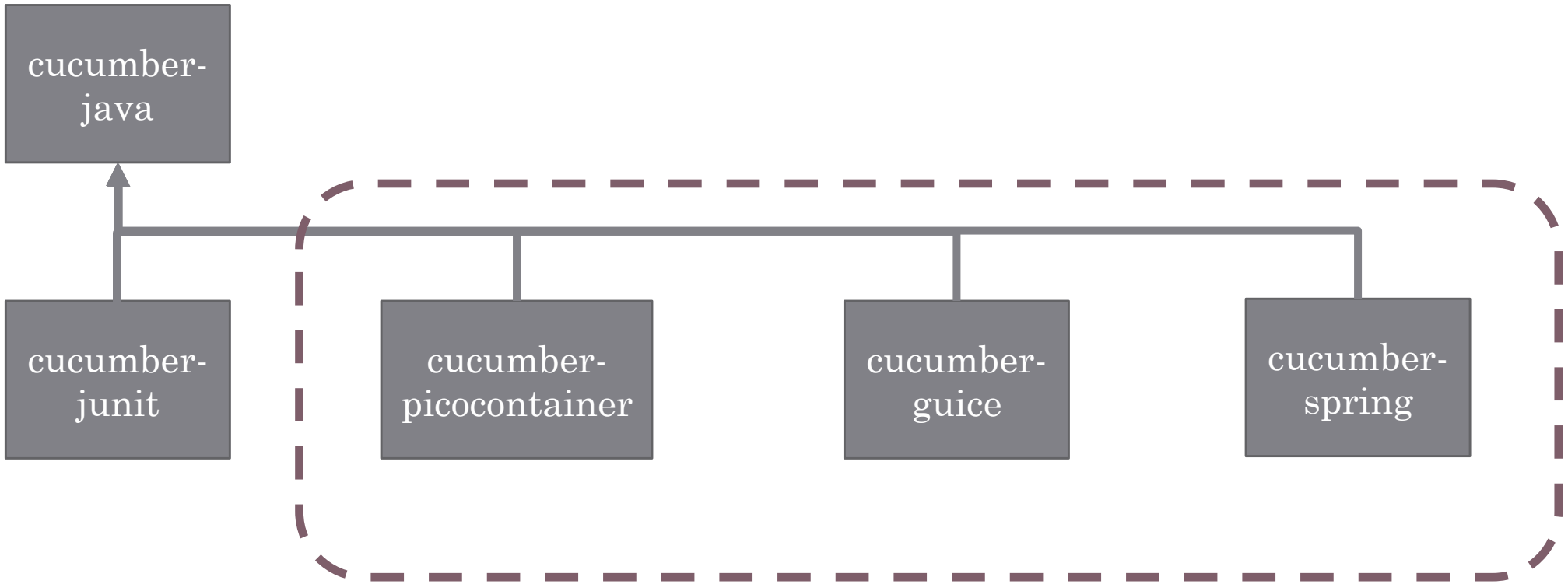
How does BDD help?

- All stakeholders involved in the discussion drives out clear and concise specifications centred around **business value**.
- Write specifications in **natural language** so everyone is on the same page
- Turn the specification into **acceptance tests** that guarantees the software does as it is intended
- Use those tests as the software evolves to guarantee that it **continues to work as intended.**
- Specifications are **documentation** – each scenario describes how the system is being used

Cucumber, The Origins

- Cucumber
  - Ruby
  - Born 2007-2008
- Cuke4Duke
  - First seen in 2008
  - JRuby
- Cucumber-JVM
  - Released April 2012
  - Java, Groovy, Clojure, Scala, Ioke, Jython, Rhino (JavaScript) and JRuby

Maven Dependencies



Maven Dependencies

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

Scenarios - Given When Then

Given an owner with a registered pet called “Sly“

When the owner registers a new pet called “Sly“

Then the user should be warned that the pet already exists

Writing Scenario's in Cucumber

- Scenarios are organized together into features
- Each feature is represented as a plain text file.
- Feature files must have the “.feature” extension
- Each feature can contain many scenarios

JUnit Runner

```
import org.junit.runner.RunWith;
import cucumber.junit.Cucumber;

@RunWith(Cucumber.class)
public class RunnerIT {
}
```

JUnit Runner 2

- Body of the class should be empty
  - No @Test annotated methods
  - No @Before or @After methods
- Arguments can be provided with an additional annotation

JUnit Runner - Arguments

```
@RunWith(Cucumber.class)
@Cucumber.Options(
  features = "src/test/resources/features“
)
public class RunnerIT {
}
```

Step Definitions

- A single Method representing a Step from a Feature file
- The Method is annotated with information about the step it represents
- Cucumber generates “Code Snippet's” for Step Definition it can’t find

Step Annotations

import cucumber.annotation.en.*;

Each Gherkin step keyword has a Annotation

```
@When("^we want to stop traffic$")
public void methodName() { ... }
```

Passing and Failing

- Each step has it own result state
  - Passed
  - Pending
  - Skipped
  - Failed
- Individual step result state combined determine the Scenarios success or failure
- Steps can be failed on exceptions or JUnit Assertion failures

Regex Lesson

Cucumber uses Regular Expressions to match steps to definitions.

```
@When("^we want to stop traffic$")
```

^ Matches the starting position

$ Matches the ending position

Capturing Arguments

```
@Given("^an owner with a pet called \"Sly\"$")
public void an_owner_with_a_pet_called() {
        …
}

@Given("^an owner with a pet called \"([^\"]*)\"$")
public void an_owner_with_a_pet_called(String name) {
        …
}
```

Capturing Arguments 2

- The pair of brackets is a capture group
- Everything captured is passed to method parameter
- Cucumber will automatically convert to the method parameter type
- Equal number of method parameters and capture groups

Type Conversion

- If the captured string is not in the correct format for type conversion an error will be thrown
- Use regular expressions to capture the correct format string for the type

Note: Escape regular expression characters with backslash

Typical Capture Groups

(.+) Good for capturing strings and dates

\"([^\"]*)\" Captures a string within Double quotes

(\\d) Capture decimal numbers only

(\\d+\\.\\d+) Capture floating point numbers

(true|false) Captures a Boolean value

Date Formats

```
@Given ("^today is (.+)$")
public void today_is(Date date) {
        ....
 }
```

Dates are recognized if it’s a format from the DateFormat class

Date Formats

SHORT is completely numeric, such as 12.13.52 or 3:30pm

MEDIUM is longer, such as Jan 12, 1952

LONG is longer, such as January 12, 1952 or 3:30:32pm

FULL is pretty completely specified, such as Tuesday, April 12, 1952 AD or 3:30:42pm PST.

Date Formats

```
import cucumber.DateFormat;


@Given ("^today is (.+)$")
public void today_is(
    @DateFormat ("yyyy-MM-dd") Date date) {
      ....
}
```

Tag use cases

- Grouping Scenarios together
  - Identify slow running test that shouldn't be run frequently
  - Identify tests that require a particular library or server

Running Scenarios with a Tag

```
@Cucumber.Options(
    tags = {"@WebDriver"}, ... )
```

Running Scenarios without a Tag

Cucumber can exclude Scenarios with a particular tag by inserting the tilde character before the tag

For the following command will run all Scenarios without the WebDriver tag

```
@Cucumber.Options(
        tags = {"~@WebDriver"}, ... )
```

Tag expressions – Logical OR

Separate a list of tags by commas for a Logical OR tag expression

```
@Cucumber.Options(
        tags = {"@WebDriver,@email"}, ... )
```

Tag expressions – Logical AND

Specifying multiple tag arguments creates a logical AND between each tag expression.

```
@Cucumber.Options(
        tags = {"@WebDriver", "@email"}, ... )
```

Tag expressions – AND OR

Run scenarios that are tagged with @WebDriver or @email but not @slow

```
@Cucumber.Options(
        tags = {"@WebDriver,@email", "~@slow"}, ... )
```

Further information

Official website: cukes.info

Github: github.com/cucumber/cucumber-jvm

Hindsight Software: hindsighttesting.com

HINDSIGHT
JOIN THE TESTING REVOLUTION

Thank you

Alan Parkinson

CEO and Co-founder, Hindsight Software Ltd

alan.parkinson@hindsightsoftware.co.uk

@alan_parkinson