Lecture 07: Exploratory Testing & Behavior Testing

Spring 2022
Lectures

- Lecture 1 (10.02) – Introduction to Software Testing
- Lecture 2 (17.02) – Basic Black-Box Testing Techniques: Boundary Value Analysis & Equivalence Class Partitioning
- Lecture 3 (03.03) – BBT advanced: Combinatorial Testing
- Lecture 4 (10.03) – Basic White-Box Testing Techniques: Control-Flow Coverage
- Lecture 5 (17.03) – BBT adv.: State-Transition, Metamorphic, Random Testing
- Lecture 6 (24.03) – Test Levels, Test Tools, Test Automation
- Lecture 7 (31.03) – BBT adv.: Exploratory Testing, Behaviour Testing
- Lecture 9 (14.04) – Security Testing of Mobile Applications
- Lecture 10 (21.04) – WBT adv.: Data-Flow Testing / Mutation Testing
- Lecture 12 (05.05) – Defect Estimation / Test Documentation, Organisation and Process Improvement (Test Maturity Model)
- Lecture 13 (12.05) – Exam Preparation
- Lecture 14 (19.05) – Advanced Topics (optional)
Structure of Lecture 7

• Exploratory Testing
• Behavior Testing

Note: There is no homework assignment next week related to this lecture (=> no labs) … but there is a quiz
Exploratory Testing

- = Error guessing, happy testing, ...
  - Not the same as ‘random testing’
  - Also called: **context-driven testing** (context => usage of system)
- Always worth doing (on top of “regular” testing)
  - Can trigger failures that systematic techniques (“checking”) miss
- Consider
  - ”What is the craziest thing we can do?”
  - Intuition / Experience / Brainstorming
  - Past failures (and faults) / Lists in literature
Exploratory Testing

• Inventors:
  • Cem Kaner, James Bach (1990s)

• Definition:
  • “Exploratory testing is simultaneous learning, test design, and test execution.”

• Elements / Variants
  • Charter: Defines mission (and sometimes tactics to use)
    • Example: “Check UI against Windows interface standards”
  • Session-based test management:
    • Defects + Notes + Interviews of the testers
Exploratory Testing - 5 Steps

One type of exploratory testing – also called session-based test management (SBTM Cycle) – has the following 5 stages:

STEP 1: Create a Bug Taxonomy (classification)

• Categorize common types of faults found in the past projects
  • E.g., security, scalability, performance, network, interoperability, …
• Analyze the root causes of the problems or faults
• Find the risks and develop ideas to test the application
Exploratory Testing - 5 Steps

STEP 2: Develop Test Charter

• Test Charter should suggest
  • what to test
  • how it can be tested
  • what needs to be looked at

• Test ideas are the starting point of exploration testing

• Test charter helps determine how the end user could use the system
Exploratory Testing - 5 Steps

STEP 3: Set a Time Box

• A single tester or a pair of testers is working not more than 90 minutes

• There should not be any interruption in those 90 minutes sessions

• Time box can be extended or reduced by 45 minutes

• This session encourages testers to react on the response from the system and prepare for the correct outcome
Exploratory Testing - 5 Steps

STEP 4: Review Results

- Evaluation of the defects
- Learning from the testing
- Analysis of coverage (functions, code)

STEP 5: Have a Debriefing

- Compilation of the output results
- Compare the results with the charter
- Check whether any additional testing is needed
Exploratory Testing – Tips

During exploratory testing, do the following:

• **Mission** of testing should be very **clear**

• **Keep notes** on what needs to be tested, why it needs to be tested and the assessment of the product quality

• **Tracking** of questions and issues raised during exploratory testing

• Better to **pair up** the testers for effective testing

• The **more we test**, the more likely we execute the right test cases for the required scenarios
Exploratory Testing – Tips

It is very important to document and monitor the following:

- **Test Coverage** - Whether we have taken notes on the coverage of test cases and improve the quality of the software
- **Risks** - Which risks needs to be covered, which are important?
- **Test Execution Log** - Recordings of the test execution
- **Issues / Queries** - Take notes on the question and issues on the system
ET Tools

http://www.softwaretestinghelp.com/tools/top-17-exploratory-testing-tools/
Example: Test Gap Analysis

Teamscale video:

https://www.youtube.com/watch?v=NwE01_84uAo
Exploratory Testing – Teamscale

Test Gap Analysis with Teamscale (by CQSE, Germany)

- https://www.youtube.com/watch?v=NwE01_84uAo

Combines static with dynamic analysis:

- Static analysis: code change in methods (Code Churn Tree Map)
- Dynamic analysis: executed methods per test (Execution Tree Map)

Can be used in automated testing and manual (exploratory) testing
Churn Tree Map 1

SUT = pinta – a paint tool

Small box = a method (size = loc)

Larger rectangles = packages or sub-systems

Gray = unchanged since baseline (e.g., last release; but date can be freely chosen)

Yellow = changed since baseline

Red = added since baseline

Pinta.Docking/DockLibrary/DockGroup.cs
Method name: GetDockTarget
Test state: Was not changed since the baseline
Churn Tree Map 2
Execution Tree Map

Since the application has not yet been executed, all boxes are gray.

Green color indicates that a method was executed.
Test 1: Gaussian Blur

Paint “z” and then select “Gaussian Blur”.

We made changes to the corresponding method and want to test it.
Test 1: Gaussian Blur

Seems to have worked well → Test passed
Test 1: Churn Tree Map

Not much change in Churn Tree Map

Changed method:
Gaussian Blur
Test 1: Execution Tree Map

Much change in Execution Tree Map

Executed method: Gaussian Blur
Test Gap Analysis

Only two of the changed/added methods have been executed

Changed and executed method: Gaussian Blur
Test 2: Radial Blur

Method:
Radial Blur Effect
Test 2: Radial Blur

After pressing “OK” we notice that nothing happens

→ Failure
Test 2: Radial Blur

Method is now green ➔ was executed
➔ but test failed
➔ must be corrected
Test 2: Radial Blur

After the code has been fixed, method is yellow again (untested)
Test 2: Radial Blur

After pressing “OK” we see that the effect is working
⇒ Test passed
Test 2: Radial Blur

Method green again:
- method executed
- this time: no failure
Exploratory Testing

**PRO**

- Useful when requirement documents are not available or only partially available
- Involves Investigation process which helps find more bugs than normal testing
- Helps to expand the imagination of testers by executing more and more test cases which finally improves productivity as well
- This testing drills down into details of the application and covers (potentially) all the requirements
- This testing covers all the types of testing and it covers various scenarios and cases
- Encourages creativity and intuition
- Generation of new ideas during test execution

**CON**

- Purely depends on the tester skills
- Limited by domain knowledge of the tester
- Not suitable for Long execution time (e.g., scientific programs)
Structure of Lecture 7

• Exploratory Testing
• Behavior Testing
Recap: Test Driven Development (TDD)

1. Write a test
2. See it fail
3. Make it run
4. Make it right (refactor)
TDD and BDD

- Developer TDD => Unit Tests

Acceptance TDD => Acceptance Tests
also called:
Behavior-driven testing (BDD)
or: Behavior Testing
Behavior Testing

Typical Process:

- Business analyst writes a user story
- (Acceptance) tester writes scenarios based on user story
- Business team reviews scenarios
- Test engineer writes the step definitions for the scenario steps
- QA team writes test scripts (to automate the scenarios)
- The test scripts are run, issues analysed and bugs fixed
- The test scripts are run as regression tests
- End user accepts the software if tests pass (acceptance criteria met)

Scenario definition language: Gherkin (DSL)
Scripting language: Java (programming language)
Test automation framework: Cucumber, JBehave
Behavior Testing – Features/Scenarios

Feature = Describes ”what” the software shall do (not: ”how”)

User Story:
As a project manager, I want to know what day is today, so I don’t make planning mistakes

Example/Scenario:
   Given today is Monday
   When I ask whether it is Friday yet
   Then I should get the answer ”nope”
Gherkin

• Gherkin is a Domain Specific Language (DSL) helping non-programmers express requirements (features) in a structured manner.

• Gherkin code is interpreted line by line.
Gherkin – Example

**Feature:** Is it Friday yet?
PMs want to know whether it's Friday

**Scenario:** Monday isn't Friday
  
  *Given* today is Monday
  *When* I ask whether it's Friday yet
  *Then* I should be told "Nope"

is_it_friday_yet.feature
Gherkin – Example

**Feature:** Is it Friday yet?  
PMs want to know whether

**Scenario:** Monday isn't Friday  
*Given* today is Monday  
*When* I ask whether it's Friday  
*Then* I should be told "Nope"

The first line of this file starts with the keyword **Feature** followed by a name. Features will be saved in *.feature* files in Cucumber. It’s a good idea to use a name similar to the file name.

The second line is a brief description of the feature. Cucumber does not execute this line, it’s just documentation.

The fourth line, **Scenario**: Sunday is not Friday is a scenario, which is a concrete example illustrating how the software should behave.

The last three lines starting with **Given**, **When** and **Then** are the steps of our scenario. This is what Cucumber will execute.
Gherkin – Example

Feature: Is it Friday yet?
PMs want to know when it's Fri

Scenario: Monday isn't Friday

Given today is Monday

When I ask whether it's Frid

Then I should be told "Nope"

The purpose of Given steps is to put the system in a known state before the user (or external system) starts interacting with the system (in the When steps). Avoid talking about user interaction in givens. If you have worked with use cases, givens are your preconditions.

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

The purpose of Then steps is to observe outcomes. The observations should be related to the business value/benefit in your feature description. Thus, it should be related to something visible from the outside (behavior).
Gherkin – Template

**Feature:** Some terse yet descriptive text of what is desired

In order to realize a named business value
As an explicit system actor
I want to gain some beneficial outcome which furthers the goal

**Scenario:** Some determinable business situation

Given some precondition

And some other precondition
When some action by the actor
And some other action
And yet another action
Then some testable outcome is achieved
And something else we can check happens too

**Scenario:** A different situation

...
Gherkin – Template

**Feature:** Some terse yet descriptive text of what is desired
In order to realize a named business value
As an explicit system actor
I want to gain some beneficial outcome

**Scenario:** Some determinable business situation
Given some precondition
And some other precondition
When some action by the actor
And some other action
And yet another action
Then some testable outcome is achieved
And something else we can check

**Scenario:** A different situation ...

Gherkin is a line-oriented language that uses indentation to define structure.
Line endings terminate statements (called steps).
Finally, most lines in Gherkin start with a special keyword:
Gherkin – Another Example using ‘And’

**Feature:** Serve coffee

In order to earn money
Customers should be able to buy coffee at all times

**Scenario:** Buy last coffee

*Given* there are 1 coffees left in the machine
*And* I have deposited 1 dollar
*When* I press the coffee button
*Then* I should be served a coffee
Gherkin – Yet Another Example

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."
Gherkin – Yet Another Example

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."

Feature files can contain several scenarios.

Note: Scenarios that are to be used across features should be collected in a common.feature file.
Gherkin – Yet Another Example

Scenario: Eat 5 out of 12
  Given there are 12 cucumbers
  When I eat 5 cucumbers
  Then I should have 7 cucumbers

Scenario: Eat 5 out of 20
  Given there are 20 cucumbers
  When I eat 5 cucumbers
  Then I should have 15 cucumbers

Adding scenarios with just different values is tiresome.
Gherkin – Yet Another Example

Scenario Outline: Eating

Given there are <start> cucumbers
When I eat <eat> cucumbers
Then I should have <left> cucumbers

Examples:

<table>
<thead>
<tr>
<th>start</th>
<th>eat</th>
<th>left</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>15</td>
</tr>
</tbody>
</table>

Solution: Use Scenario Outlines instead!
Cucumber – Gherkin Tutorial

A 10 min tutorial that explains how to:

• Install Cucumber
• Write your first Scenario using the Gherkin syntax
• Write your first step definition in Java
• Run Cucumber
• Learn the basic workflow of Behaviour-Driven Development (BDD)

Can be found here:

https://docs.cucumber.io/guides/10-minute-tutorial/
Cucumber – Gherkin Example

- Install Cucumber
- Create empty project "hellocucumber"
- Verify Cucumber installation by typing in terminal:
  
  mvn test
Cucumber – Gherkin Example

- Install Cucumber
- Create empty project
- Verify Cucumber installation in terminal:
  ```
  mvn test
  ```

Cucumber’s output is telling us that it didn’t find anything to run.
Cucumber – Gherkin Example

Create a .feature file and write a scenario:

Feature: Is it Friday yet?
    Everybody wants to know when it's Friday

Scenario: Sunday isn't Friday
    Given today is Sunday
    When I ask whether it's Friday yet
    Then I should be told "Nope"

File path:
src/test/resources/hello cucumber/is_it_friday_yet.feature
Cucumber – Gherkin Example

Cucumber executes the feature file after typing:

```sh
mvn test
```

<Cucumber output on next slide>
Cucumber – Gherkin Example

Cucumber executes the feature file after typing:
```
mvn test
```
Cucumber detects the scenario file but notices that neither the scenario nor the Given-When-Then steps have been implemented as tests.

Cucumber example:
```
Feature: Is it Friday yet?
 Everybody wants to know when it's Friday

Scenario: Sunday isn't Friday
 Given today is Sunday
 When I ask whether it's Friday yet
 Then I should be told "Nope"

1 Scenarios (1 undefined)
3 Steps (3 undefined)
```
You can implement missing steps with the snippets below:
Cucumber – Gherkin Example

Cucumber executes the feature file after typing `mvn test`.

Cucumber suggests test code snippets.

```java
@Given("^today is Sunday$")
public void today_is_Sunday() {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^I ask whether it's Friday yet$")
public void i_ask_whether_it_s_Friday_yet() {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^I should be told \"([^\"]*)\"$")
public void i_should_be_told(String arg1) {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```
Cucumber –

Create test code file and execute test in Cucumber typing:

mvn test

```
import cucumber.api.java.en.Given;
import cucumber.api.java.en.When;
import cucumber.api.java.en.Then;
import static org.junit.Assert.*;

public class Stepdefs {

    @Given("^today is Sunday$")
    public void today_is_Sunday() {
        // Write code here turning the phrase above into action
        throw new PendingException();
    }

    @When("^I ask whether it's Friday yet$")
    public void i_ask_whether_it_s_Friday_yet() {
        // Write code here turning the phrase above into action
        throw new PendingException();
    }

    @Then("^I should be told \"([^\"]*)\"$")
    public void i_should_be_told(String expectedAnswer) {
        // Write code here turning the phrase above into action
        throw new PendingException();
    }
}
```

Accepts any string

```
src/test/java/hellocucumber/Stepdefs.java
```
Cucumber – Gherkin Example

Create test code file and execute test in Cucumber typing:

```
import cucumber.api.java.en.Given;
import cucumber.api.java.en.When;
import cucumber.api.java.en.Then;
import static org.junit.Assert.*;

public class Stepdefs {
    @Given("^today is Sunday$")
    public void today_is_Sunday() {
        // Write code here turning the phrase above into action
        throw new PendingException();
    }

    @When("^I ask whether it's Friday yet$")
    public void i_ask_whether_it_s_Friday_yet() {
        // Write code here turning the phrase above into action
        throw new PendingException();
    }

    @Then("^I should be told "(\[^\"]\)*)"$")
    public void i_should_be_told(String expectedAnswer) {
        // Write code here turning the phrase above into action
        throw new PendingException();
    }
}
```

cucumber.java.listeners.RSpecTestListener

Import the Gherkin steps in a .java file:

```
import org.junit.runner.RunWith;
import cucumber.api.junit.Cucumber;
@RunWith(Cucumber.class)
public class RunCucumberTest {
    @CucumberOptions(features="src/test/java/hellocucumber/feature")
    public static void main(String[] args) {
    }
}
```

Cucumber detects the scenarios (in .feature file) and the Given-When-Then steps (in .java file).
Cucumber – Gherkin Example

Create test code file and execute test in Cucumber typing:

```java
mvn test
```

```java
import cucumber.api.java.en.Given;
import cucumber.api.java.en.When;
import cucumber.api.java.en.Then;
import static org.junit.Assert.*;

public class Stepdefs {
    @Given("^today is Sunday$")
    public void today_is_Sunday() {
        // Write code here turning the phrase above into action
        throw new PendingException();
    }

    @When("^I ask whether it's Friday yet$")
    public void i_ask_whether_it_s_Friday_yet() {
        // Write code here turning the phrase above into action
        throw new PendingException();
    }

    @Then("^I should be told " \$(\[^\"]*\) \$")
    public void i_should_be_told(String expectedAnswer) {
        // Write code here turning the phrase above into action
        throw new PendingException();
    }
}
```

Since an exception is thrown in the first step (Given), the next steps are skipped.

Tests

Running hellocucumber.RunCucumberTest
Feature: Is it Friday yet?
    Everybody wants to know when it's Friday

Scenario: Sunday isn't Friday # hellocucumber/is_it_friday_yet.feature:4
    Given today is Sunday # Stepdefs.today_is_Sunday()
        cucumber.api.PendingException: TODO: implement me
            at hellocucumber.Stepdefs.today_is_Sunday(Stepdefs.java:12)
            at *.today is Sunday(hellocucumber/is_it_friday_yet.feature:5)

    When I ask whether it's Friday yet # Stepdefs.i_ask_whether_it_s_Friday_yet()
    Then I should be told "Nope" # Stepdefs.i_should_be_told(String)

1 Scenarios (1 pending)
3 Steps (2 skipped, 1 pending)
0m0.188s
Cucumber – Gherkin Example

Update test code file and execute test in Cucumber typing:

mvn test

```
import cucumber.api.java.en.Given;
import cucumber.api.java.en.When;
import cucumber.api.java.en.Then;
import static org.junit.Assert.*;

class IsItFriday {
    static String isItFriday(String today) {
        return null;
    }
}

public class Stepdefs {
    private String today;
    private String actualAnswer;

    @Given("^today is Sunday$")
    public void today_is_Sunday() {
        this.today = "Sunday";
    }

    @When("^I ask whether it's Friday yet$")
    public void i_ask_whether_it_s_Friday_yet() {
        this.actualAnswer = IsItFriday.isItFriday(today);
    }

    @Then("^I should be told \"([^\"]*)\"$")
    public void i_should_be_told(String expectedAnswer) {
        assertEquals(expectedAnswer, actualAnswer);
    }
}
```

Full implementation of Stepdefs.java file.
Cucumber – Gherkin Example

Update test code file and execute test in Cucumber typing:

mvn test

Full implementation of Stepdefs.java file.
Cucumber – Gherkin Example

Update code file and execute test in Cucumber typing:

```
mvn
```

```java
import cucumber.api.java.en.Given;
import cucumber.api.java.en.When;
import cucumber.api.java.en.Then;
import static org.junit.Assert.*;

class IsItFriday {
    static String isItFriday(String today) {
        return null;
    }
}

public class Stepdefs {
    private String today;
    private String actualAnswer;

    @Given("^today is Sunday$")
    public void today_is_Sunday() {
        today = "Sunday";
    }

    @When("^I ask whether it's Friday yet$")
    public void i_ask_whether_it_s_Friday_yet() {
        actualAnswer = IsItFriday.isItFriday(today);
    }

    @Then("^I should be told "Nope"$")
    public void i_should_be_told(String expectedAnswer) {
        assertEquals(expectedAnswer, actualAnswer);
    }
}
```

Full implementation of Stepdefs.java file.

That’s progress! The first two steps are passing, but the last one is failing.
**Cucumber – Gherkin Example**

Update test code file to make test pass and execute test in Cucumber typing:

```
mvn test
```

```java
import cucumber.api.java.en.Given;
import cucumber.api.java.en.When;
import cucumber.api.java.en.Then;
import static org.junit.Assert.*;

class IsItFriday {
    static String isItFriday(String today) {
        return "Nope";
    }
}

class Stepdefs {
    private String today;
    private String actualAnswer;

    @Given("^today is Sunday$")
    public void today_is_Sunday() {
        this.today = "Sunday";
    }

    @When("^I ask whether it's Friday yet$")
    public void i_ask_whether_it_s_Friday_yet() {
        this.actualAnswer = IsItFriday.isItFriday(today);
    }

    @Then("^I should be told "([\^\"]*)\"$")
    public void i_should_be_told(String expectedAnswer) {
        assertEquals(expectedAnswer, actualAnswer);
    }
}
```

*Change null to “Nope”*
Cucumber – Gherkin Example

Update test code to make test pass and execute test in Cucumber typing:

```
mvn test
```

SUCCESS!
Cucumber – Gherkin Example

The next thing to test for would be that we also get the correct result when it is Friday.
Feature: Is it Friday yet?
   Everybody wants to know when it's Friday

Scenario: Sunday isn't Friday
   Given today is Sunday
   When I ask whether it's Friday yet
   Then I should be told "Nope"

Scenario: Friday is Friday
   Given today is Friday
   When I ask whether it's Friday yet
   Then I should be told "TGIF"

Add new scenario in feature file.
Cucumber – Gherkin Example

Feature: Is it Friday yet?
   Everybody wants to know when it's Friday

Scenario: Sunday isn't Friday
   Given today is Sunday
   When I ask whether it's Friday yet
   Then I should be told "Nope"

Scenario: Friday is Friday
   Given today is Friday
   When I ask whether it's Friday yet
   Then I should be told "TGIF"

Add corresponding Step definition in .java file.

```java
@Given("^today is Friday$")
public void today_is_Friday() {
    this.today = "Friday";
}
```
Of course, the new scenario fails.
We haven’t yet fully implemented the logic!
Cucumber – Gherkin Example

```java
static String isItFriday(String today) {
    if (today.equals("Friday")) {
        return "TGIF";
    }
    return "Nope";
}
```

Change code in helper class to reflect the logic.

```java
static String isItFriday(String today) {
    if (today.equals("Friday")) {
        return "TGIF";
    }
    return "Nope";
}
```

… then run Cucumber again.
Running helloCucumber.RunCucumberTest
Feature: Is it Friday yet?
   Everybody wants to know when it's Friday

Scenario: Friday is Friday
   Given today is Friday
   When I ask whether it's Friday yet
   Then I should be told "TGIF"

Scenario: Sunday isn't Friday
   Given today is Sunday
   When I ask whether it's Friday yet
   Then I should be told "Nope"

2 scenarios (2 passed)
6 steps (6 passed)
0m0.255s

SUCCESS!
Cucumber – Gherkin Example

Now let’s make our tests more flexible using ‘Scenario Outline’ und ‘Example’ (Data-driven Testing).
Cucumber – Gherkin Example

Feature: Is it Friday yet?
   Everybody wants to know when it's Friday

   Scenario Outline: Today is or is not Friday
   Given today is "<day>"
   When I ask whether it's Friday yet
   Then I should be told "<answer>"

Examples:
<table>
<thead>
<tr>
<th>day</th>
<th>answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friday</td>
<td>TGIF</td>
</tr>
<tr>
<td>Sunday</td>
<td>Nope</td>
</tr>
<tr>
<td>anything else!</td>
<td>Nope</td>
</tr>
</tbody>
</table>

File: src/test/resources/hellocucumber/is_it_friday_yet.feature
Cucumber – Gherkin Example

package hellocucumber;
import cucumber.api.java.en.Given;
import cucumber.api.java.en.When;
import cucumber.api.java.en.Then;
import static org.junit.Assert.*;

class IsItFriday {
    static String isItFriday(String today) {
        if (today.equals("Friday")) {
            return "TGIF";
        }
        return "Nope";
    }
}

public class Stepdefs {
    private String today;
    private String actualAnswer;

    @Given("^today is \("\^[\"]\)*\"\$"")
    public void today_is(String today) {
        this.today = today;
    }

    @When("^I ask whether it's Friday yet$"")
    public void i_ask_whether_it_s_Friday_yet() {
        this.actualAnswer = IsItFriday.isItFriday(today);
    }

    @Then("^I should be told \("\^[\"]\)*\"\$"")
    public void i_should_be_told(String expectedAnswer) {
        assertEquals(expectedAnswer, actualAnswer);
    }
}

Merge two steps into one.
Cucumber – Gherkin Example

---

TESTS
---

Running hellocucumber.RunCucumberTest

Feature: Is it Friday yet?
   Everybody wants to know when it's Friday

Scenario Outline: Today is or is not Friday # hellocucumber/is_it_friday_yet.feature:4
   Given today is <day> # hellocucumber/is_it_friday_yet.feature:5
   When I ask whether it's Friday yet # hellocucumber/is_it_friday_yet.feature:6
   Then I should be told <answer> # hellocucumber/is_it_friday_yet.feature:7

Examples:

<table>
<thead>
<tr>
<th>day</th>
<th>answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Friday&quot;</td>
<td>&quot;TGIF&quot;</td>
</tr>
<tr>
<td>&quot;Sunday&quot;</td>
<td>&quot;Nope&quot;</td>
</tr>
<tr>
<td>&quot;anything else!&quot;</td>
<td>&quot;Nope&quot;</td>
</tr>
</tbody>
</table>

3 scenarios (3 passed)
9 steps (9 passed)
0m0.255s

SUCCESS!

All tests of the three scenarios have passed.
Cucumber – Gherkin Tutorial

A 10 min tutorial that explains how to:
• Install Cucumber
• Write your first Scenario using the Gherkin syntax
• Write your first step definition in Java
• Run Cucumber
• Learn the basic workflow of Behaviour-Driven Development (BDD)

Can be found here:
https://cucumber.io/docs/guides/10-minute-tutorial/
Next Week

• Quiz 7 → Moodle

• Finish up and submit HW 6:
  – Automated Web Application Testing (Selenium)

• Lecture 8: