LTAT.05.006: Software Testing

Lecture 07: State-Transition Testing & Exploratory Testing

Dietmar Pfahl
email: dietmar.pfahl@ut.ee

Spring 2019
Lectures (J. Liivi 2-111)

- Lecture 1 (14.02) – Introduction to Software Testing
- Lecture 2 (21.02) – Basic Black-Box Testing Techniques
- Lecture 3 (28.02) – BBT advanced: Combinatorial Testing
- Lecture 4 (07.03) – Basic White-Box Testing Techniques
- Lecture 5 (14.03) – Test Lifecycle, Test Tools, Test Automation
- Lecture 7 (28.03) – BBT advanced: State-Transition Testing & Exploratory Testing
- Lecture 8 (04.04) – BBT advanced: Security, Usability and A/B Testing
- Lecture 9 (11.04) – WBT advanced: Data-Flow Testing / Mutation Testing
- Lecture 11 (25.04) – Defect Estimation / Test Documentation, Organisation and Process Improvement (Test Maturity Model)
- 02.05 - no lecture
- Lecture 12 (09.05) – Industry Guest Lecture (to be announced)
- Lecture 13 (16.05) – Exam Preparation
Structure of Lecture 7

- Lab 7
- State-Transition Testing
- Exploratory Testing
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

• A Domain Specific Language (DSL) that helps non-programmers express requirements (features) in a structured manner
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"
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 '

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 Fri

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

<see next slide>
Cucumber – Gherkin Example

- Install Cucumber
- Create empty project
- Verify Cucumber installation by typing in terminal:

```bash
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/hellocucumber/is_it_friday_yet.feature

The Given-When-Then lines will be executed by Cucumber
Cucumber – Gherkin Example

Cucumber executes the feature file after typing:

```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 – Gherkin Example

You can implement missing steps with the snippets below:

```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 suggests test code snippets.
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();
    }
}

src/test/java/hellocucumber/Stepdefs.java
Create test code file 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.*;

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

Update test code file and execute test in Cucumber typing:

```
mvn test
```

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

Update test code file 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 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";
    }
}

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 \"([^\"]*)\"$")
    public void i_should_be_told(String expectedAnswer) {
        assertEquals(expectedAnswer, actualAnswer);
    }
}
```

Change null to "Nope"

String = "Nope"
Cucumber – Gherkin Example

Update code, make test pass, execute in Cucumber:

typing:

mvn test

SUCCESS!
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.
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.
C

TESTS

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).
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>
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
   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>&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>

SUCCESS!

All tests of the three scenarios have passed.

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

In Lab 7 you will use Gherkin with Behave and Python instead.
Lab 7 – Web-App Testing in the CI/CD Pipeline

Lab 7 (week 31: Apr 02 & 03) – Web-App Testing in the CI/CD Pipeline (10 points)

Lab 7 Instructions & Tools

Submission Deadlines:
- Tuesday Labs: Monday, 08 Apr, 23:59
- Wednesday Labs: Tuesday, 09 Apr, 23:59

- Penalties apply for late delivery: 50% penalty, if submitted up to 24 hours late; 100 penalty, if submitted more than 24 hours late
Lab 7 – Web-App Testing in the CI/CD Pipeline

Lab 7 (week 31: Apr 02 & 03) – Web-App Testing in the CI/CD Pipeline (10 points)

Lab 7 Instructions & Tools

Feature: Login form

Scenario: Access the login form with valid credentials

Given an anonymous user
When user submits a valid login page
Then user is redirected to / page # Home page
@given('an anonymous user')
def step_impl(context):
    # Creates user with given credentials and saves it to database.
    u = UserFactory(username='foo', email='foo@example.com')
    u.set_password('bar')
    u.save()

@when('user submits a valid login page')
def step_impl(context):
    # Retrieve browser instance to locate elements
    br = context.browser
    br.get(context.base_url + '/login/')

    # locate elements by name, id, tag, css class, etc
    br.find_element_by_name('username').send_keys('foo')
    br.find_element_by_name('password').send_keys('bar')
    br.find_element_by_name('submit').click()

@then('user is redirected to {url} page')
def step_impl(context, url):
    br = context.browser

    # Python's built-in asserting function
    assert url in br.current_url, 'User is redirected to a wrong url'
Lab 7: CI/CD with Bitbucket Pipelines

- Bitbucket Pipelines
  - integrated CI/CD service, built into Bitbucket.
- It allows you to automatically build, test and even deploy your code, based on a configuration file in your repository.

To set up Pipelines you need to create and configure the file `bitbucket-pipelines.yml` in the root directory of your repository.
The `bitbucket-pipelines.yml` file holds all the build configurations for your repository. YAML is a file format that is easy to read, but writing it requires care. Indenting must use spaces, as tab characters are not allowed.

There is a lot you can configure in the `bitbucket-pipelines.yml` file, but at its most basic the required keywords are:

- **pipelines**: contains all your pipeline definitions.
- **default**: contains the steps that run on every push.
- **step**: each step starts a new Docker container with a clone of your repository, then runs the contents of your **script** section.
- **script**: a list of commands that are executed in sequence.

**Bitbucket Pipelines Documentation:**

Black-Box Testing Techniques

- Equivalence class partitioning (ECP)
- Boundary value analysis (BVA)
- Cause-effect graphing
- Combinatorial testing
- State transition testing (State-based testing)
- Exploratory testing
- Usability testing
- A/B testing (UX)
Structure of Lecture 7

• Lab 7
• State-Transition Testing
• Exploratory Testing
State-Transition Testing - Example

Use Case Diagram

- Check Account
- Withdraw Money
- ...

ATM
State-Transition Testing - Example

Use Case Description: Check Account

Role: Customer
Goal: Customer wants to check the amount of money in his/her accounts

Scenario (actions):
1. ATM asks for customer card
2. Customer enters card
3. ATM asks for PIN code
4. Customer enters PIN code
5. …
State-Transition Diagram

State-Transition Graph

Card inserted
Ask for PIN

Invalid PIN
Beep

Valid PIN
Ask amount

Now create a set of test cases that trigger each state-transition at least once
<table>
<thead>
<tr>
<th>Input (Event)</th>
<th>State</th>
<th>Wait for Card (S1)</th>
<th>Wait for PIN (S2)</th>
<th>Next (S3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Card inserted</td>
<td>Ask for PIN -&gt; S2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Invalid PIN</td>
<td>-</td>
<td>Beep -&gt; S2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Valid PIN</td>
<td>-</td>
<td>Ask amount -&gt; S3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Cancel</td>
<td>-</td>
<td>Return card -&gt; S1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
**State Table**

4 Test Cases:
- S1 -> 'Card inserted' / 'Ask for PIN' -> S2
- S2 -> 'Invalid PIN' / 'Beep' -> S2
- S2 -> 'Valid PIN' / 'Ask amount' -> S3
- S2 -> 'Cancel' / 'Return card' -> S1

<table>
<thead>
<tr>
<th>Input (Event)</th>
<th>State</th>
<th>Wait for Card (S1)</th>
<th>Wait for PIN (S2)</th>
<th>Next (S3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Card inserted</td>
<td>Ask for PIN -&gt; S2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Invalid PIN</td>
<td>-</td>
<td>Beep -&gt; S2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Valid PIN</td>
<td>-</td>
<td>Ask amount -&gt; S3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Cancel</td>
<td>-</td>
<td>Return card -&gt; S1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Test scenario (=sequence of test cases):
- S1 -> 'Card inserted' / 'Ask for PIN' -> S2 -> 'Cancel' / 'Return card' -> S1
State-Transition Testing: Example 2

Extract of a Specification Doc:

Parameters

- PORT_A: calling phone
- PORT_B: called phone

PORT_A identifies the connection from which a call is to be set up. The actual state of the call setup is globally available. Depending on this a new state arises after the evaluation of the transferred event. The delivered state is DISCONNECTED, if the call setup was terminated, it is DIALING, if the call setup is in progress but not completed yet. It is CONNECTED, if the call setup was successfully completed. In this case PORT_B delivers the connection of the selected subscriber, otherwise the data content of PORT_B is undefined. A call setup requires the sequence UNHOOK (DIGIT_N)* and the digit sequence must represent a valid number. HANG UP always leads to the complete termination of the call. If TIMEOUT occurs, HANG UP brings the software back into the initial state (DISCONNECTED)
State-Transition Testing: Example 2

State Chart
State-Transition Testing: Example 2

The minimal test strategy is to cover each state at least once.

A better strategy is to cover each transition at least once, which leads, e.g., to the following test scenarios …
State-Transition Testing: Example 2

State Chart

DISCONNECTED, unhook -> DIALING, hang up -> DISCONNECTED
State-Transition Testing: Example 2

State Chart

- DISCONNECTED, unhook ->
- DIALING, timeout ->
- TIMEOUT OCCURRED, hang up ->
- DISCONNECTED
State-Transition Testing: Example 2

State Chart

- DISCONNECTED, unhook -> DIALING, Digit 0..9 ->
- DIALING, Digit 0..9 -> DIALING, dialed number valid -> CONNECTED, hang up -> DISCONNECTED
State-Transition Testing: Example 2

State Chart

- DISCONNECTED, unhook -> DIALING, Digit 0..9 -> DIALING, Digit 0..9 -> DIALING, dialed number invalid -> INVALID NUMBER, timeout -> TIMEOUT OCCURRED, hang up -> DISCONNECTED
State-Transition Testing: Example 2

The minimal test strategy is to cover each state at least once.

A better strategy is to cover each transition at least once, which leads, e.g., to the following test scenarios (or, short: tests):

- DISCONNECTED, unhook -> DIALING, hang up -> DISCONNECTED
- DISCONNECTED, unhook -> DIALING, timeout -> TIMEOUT OCCURRED, hang up -> DISCONNECTED
- DISCONNECTED, unhook -> DIALING, Digit 0..9 -> DIALING, Digit 0..9 -> DIALING, dialed number valid -> CONNECTED, hang up -> DISCONNECTED
- DISCONNECTED, unhook -> DIALING, Digit 0..9 -> DIALING, Digit 0..9 -> DIALING, dialed number invalid -> INVALID NUMBER, timeout -> TIMEOUT OCCURRED, hang up -> DISCONNECTED

… and so on ...
State-Transition Testing: Example 2

The minimal test strategy is to cover each state at least once.

A better strategy is to cover each transition at least once, which leads, e.g., to the following tests:

- DISCONNECTED, unhook -> DIALING, hang up -> DISCONNECTED
- DISCONNECTED, unhook -> DIALING, timeout -> TIMEOUT OCCURRED, hang up -> DISCONNECTED
- DISCONNECTED, unhook -> DIALING, Digit 0..9 -> DIALING, Digit 0..9 -> DIALING, dialed number valid -> CONNECTED, hang up -> DISCONNECTED
- DISCONNECTED, unhook -> DIALING, Digit 0..9 -> DIALING, Digit 0..9 -> DIALING, dialed number invalid -> INVALID NUMBER, timeout -> TIMEOUT OCCURRED, hang up -> DISCONNECTED

Furthermore, it is useful to test all events, if transitions can be initiated by more than one event. The result is a strength-hierarchy of test techniques:

all states ≤ all transitions ≤ all events

Important: Do not forget to test the failure treatment!
## State-Transition Testing: Example 2

<table>
<thead>
<tr>
<th>Event</th>
<th>DISCONNECTED</th>
<th>DIALING</th>
<th>CONNECTED</th>
<th>INVALID NUMBER</th>
<th>TIMEOUT OCCURRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>unhook</td>
<td>DIALING</td>
<td>FAILURE</td>
<td>FAILURE</td>
<td>FAILURE</td>
<td>FAILURE</td>
</tr>
<tr>
<td>hang up</td>
<td>FAILURE</td>
<td>DISCONNECTED</td>
<td>DISCONNECTED</td>
<td>DISCONNECTED</td>
<td>DISCONNECTED</td>
</tr>
<tr>
<td>digit_0</td>
<td>DISCONNECTED</td>
<td>DIALING</td>
<td>CONNECTED</td>
<td>INVALID NUMBER</td>
<td>TIMEOUT OCCURRED</td>
</tr>
<tr>
<td>digit_9</td>
<td>DISCONNECTED</td>
<td>DIALING</td>
<td>CONNECTED</td>
<td>INVALID NUMBER</td>
<td>TIMEOUT OCCURRED</td>
</tr>
<tr>
<td>timeout</td>
<td>FAILURE</td>
<td>TIMEOUT OCCURRED</td>
<td>FAILURE</td>
<td>TIMEOUT OCCURRED</td>
<td>TIMEOUT OCCURRED</td>
</tr>
<tr>
<td>dialed number valid</td>
<td>FAILURE</td>
<td>CONNECTED</td>
<td>FAILURE</td>
<td>FAILURE</td>
<td>FAILURE</td>
</tr>
<tr>
<td>dialed number invalid</td>
<td>FAILURE</td>
<td>INVALID NUMBER</td>
<td>FAILURE</td>
<td>FAILURE</td>
<td>FAILURE</td>
</tr>
</tbody>
</table>
State-Transition Testing: Example 2

State Chart
with FAILURE state
State-Transition Testing
(state-transition coverage)

vs.

White-Box Testing
(control-flow-coverage)
Stack Example

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void push(int x) {
        if (size >= values.length) {
            resize(); }
        values[size++] = x;
    }

    int pop() {
        if (size > 0) {
            return values[size--]; }
        throw new EmptyStackException();
    }

    private void resize() {
        int[] tmp = new int[values.length * 2];
        for (int i = 0; i < values.length; i++) {
            tmp[i] = values[i];
        }
        values = tmp;
    }
}
```

push(elem1)
pop() -> elem1
pop() -> exception

push(elem1)
...
push(elem4) -> resize()
pop() -> elem4
Stack Example

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void push(int x) {
        if (size >= values.length) {
            resize();
        }
        values[size++] = x;
    }

    int pop() {
        if (size > 0) {
            return values[size--];
        }
        throw new EmptyStackException();
    }

    private void resize() {
        int[] tmp = new int[values.length * 2];
        for (int i = 0; i < values.length; i++) {
            tmp[i] = values[i];
        }
        values = tmp;
    }
}
```

100% branch coverage with:

Stack stack0 = new Stack();  
try {
    stack0.pop();
} catch (EmptyStackException e) {
}

Stack stack0 = new Stack();  
int int0 = -55;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();
```
Stack Example

100% branch coverage with:

Stack stack0 = new Stack();
try {
    stack0.pop();
} catch (EmptyStackException e) {
}

Stack stack0 = new Stack();
int int0 = -55;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();
Stack Example

100% branch coverage with:

```java
Stack stack0 = new Stack();
try {
    stack0.pop();
} catch (EmptyStackException e) {
}
```

```java
Stack stack0 = new Stack();
t1
try {
    stack0.pop();
    stack0.pop();
} catch (EmptyStackException e) {
}
```

```java
Stack stack0 = new Stack();
t2
int int0 = -55;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();
```
Stack Example

```java
Stack stack0 = new Stack();
try {
    stack0.pop();
} catch (EmptyStackException e) {
}
Stack stack0 = new Stack();
t(int0) = -55;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();
```

100% branch coverage with:

```
Stack stack0 = new Stack();  // T1
try {
    stack0.pop();
} catch (EmptyStackException e) {
}
```

```
Stack stack0 = new Stack();  // T2
int int0 = -55;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();
```
Stack Example

```
Stack stack0 = new Stack();
try {
    stack0.pop();
} catch (EmptyStackException e) {
}
```

```
Stack stack0 = new Stack();
int int0 = -55;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();
```

100% branch coverage with:

- **T1**: 
  ```java
  Stack stack0 = new Stack();
  try {
      stack0.pop();
  } catch (EmptyStackException e) {
  }
  ```

- **T2**: 
  ```java
  Stack stack0 = new Stack();
  int int0 = -55;
  stack0.push(int0);
  stack0.push(int0);
  stack0.push(int0);
  stack0.push(int0);
  stack0.pop();
  ```
Stack Example

100% branch coverage with:

Stack stack0 = new Stack();
try {
    stack0.pop();
}
catch(EmptyStackException e) {
}

Stack stack0 = new Stack();
int int0 = -55;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();
Stack Example

100% branch coverage with:

Stack stack0 = new Stack();
try {
    stack0.pop();
} catch (EmptyStackException e) {
}

Stack stack0 = new Stack();
int int0 = -55;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();
Stack Example

100% branch coverage with:

Stack stack0 = new Stack();
try {
    stack0.pop();
} catch(EmptyStackException e) {
}

Stack stack0 = new Stack();
text = -55;
stack0.push(text);
stack0.push(text);
stack0.push(text);
stack0.push(text);
stack0.pop();

T1

Stack stack0 = new Stack();
int int(0) = -55;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();

T2
Stack Example

```
Stack stack0 = new Stack();
try {
    stack0.pop();
} catch (EmptyStackException e) {
}
Stack stack0 = new Stack();
type {
    int int0 = -55;
    stack0.push(int0);
    stack0.push(int0);
    stack0.push(int0);
    stack0.push(int0);
    stack0.pop();
}
```

100% branch coverage with:

T1
```
Stack stack0 = new Stack();
try {
    stack0.pop();
} catch (EmptyStackException e) {
}
```

T2
```
Stack stack0 = new Stack();
type {
    int int0 = -55;
    stack0.push(int0);
    stack0.push(int0);
    stack0.push(int0);
    stack0.push(int0);
    stack0.pop();
```
Stack Example

public class Stack {
    int[] values = new int[3];
    int size = 0;

    void push(int x) {
        if (size >= values.length) {
            resize(); } 
        values[size++] = x;
    }

    int pop() {
        if (size > 0) {
            return values[size--]; }
        throw new EmptyStackException();
    }

    private void resize() {
        int[] tmp = new int[values.length * 2];
        for (int i = 0; i < values.length; i++) {
            tmp[i] = values[i]; }
        values = tmp;
    }
}

Specification:
- When Stack has been created, it is empty
- When elements are added (push), the stack is not empty
- When Stack is full and an element added, it will be resized
- When the last element has been removed (pop) from Stack, it is empty
- When Stack is empty and an element shall be removed, an error occurs.
Note: *resize* is a private method

\[ length := length \times 2 \]
Stack stack0 = new Stack();
try {
    stack0.pop();
} catch(EmptyStackException e) {
}

Note: resize() is a private method
length := length*2
Note: resize() is a private method
length := length*2
**T3**
(to cover all state-transitions)

- **non_exist**
- **Stack()**
- **empty** (size = 0)
- **pop()**
- **error**
- **push(x)**
- **not_empty** (0 < size <= length)
- **[size = 1] pop() / size--**
- **[size > 1] pop() / size--**
- **[size = length] push(x) / resize(); size++**

Note: **resize() is a private method**
length := length*2
Merge T1 / T2 / T3 $\Rightarrow$ T1*

---

**T1**

```java
Stack stack0 = new Stack();
try {
    stack0.pop();
} catch (EmptyStackException e) {
}
```

---

**T2**

```java
Stack stack0 = new Stack();
int int0 = -55;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();
```

---

**T3**

```java
Stack stack0 = new Stack();
int int0 = -55;
stack0.push(int0);
stack0.pop();
```
Merge T1 / T2 / T3 ➞ T1*
Merge T1 / T2 / T3 ➔ T1*

NON_EXIST -> Stack() -> EMPTY -> push() ->
NOT_EMPTY -> push() -> NOT_EMPTY ->
push() -> NOT_EMPTY -> push() ->
NOT_EMPTY -> pop() -> NOT_EMPTY -> pop() ->
NOT_EMPTY -> pop() -> NOT_EMPTY -> pop() ->
EMPTY -> pop() -> ERROR

[length = 21]

Stack stack0 = new Stack();
int int0(0) = -55;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();
stack0.pop();
stack0.pop();
stack0.pop();
try {
    stack0.pop();
} catch(EmptyStackException e) {
Structure of Lecture 7

- Lab 7
- State-Transition Testing
- Exploratory Testing
Exploratory Testing

- = Error guessing (?), happy testing, ...
- Not the same as ‘random testing’
- Always worth doing (on top of “regular” testing)
- Can trigger failures that systematic techniques miss
- Consider
  - ”What is the craziest thing we can do?”
  - Intuition / Experience / Brainstorming
  - Past failures / Lists in literature
- Tools
  - http://www.softwaretestinghelp.com/tools/top-17-exploratory-testing-tools/
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
- 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 less 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 areas

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

During exploratory execution, following needs to be done:

• 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, more likely to execute right test cases for the required scenarios
Exploratory Testing

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 and which are all important ones?

• Test Execution Log - Recordings on the test execution

• Issues / Queries - Take notes on the question and issues on the system
ET Tool Selection

http://www.softwaretestinghelp.com/tools/top-17-exploratory-testing-tools/
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
Churn Tree Map 2
Execution Treemap

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”
Test 1: Gaussian Blur

Seems to have worked well
→ Test passed
Test 1: Tree Maps

No change in Churn Tree Map
Test 1: Tree Maps

Much change in Execution Tree Map
Test Gaps

Only two of the changed/added methods have been executed
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**

- Usefult 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 drill down to smallest part of application and covers 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)
Next Week

• Quiz 7 → Moodle

• Lab 7:
  – Web-Application Testing in the CI/CD Pipeline

• Lecture 8: