TU/e

Project FINGERPAINT

UTP-1.0

Unit Test Plan

Authors:
Tessa Belder (0739377)
Lasse Blaauwbroek (0749928)
Thom Castermans (0739808)
Roel van Happen (0751614)
Benjamin van der Hoeven (0758975)
Femke Jansen (0741948)
Hugo Snel (0657700)

Junior Management:
Simon Burg
Areti Paziourou
Luc de Smet

Senior Management:
Mark van den Brand, MF 7.096
Lou Somers, MF 7.145

Technical Advisor:
Ion Barosan, MF 7.082

Customer:
Patrick Anderson, GEM-Z 4.137

Eindhoven - June 23, 2013

Abstract

This document is the Unit Test Plan (UTP) of Group Fingerpaint. This project is part of the Software Engineering Project (2IP35) and is one of the assignments at Eindhoven University of Technology. The document complies with the UTP from the Software Engineering Standard, as set by the European Space Agency [1]. The UTP provides the main guidance for the Unit Tests (UT) during the Detailed Design (DD) phase for the FINGERPAINT application. It describes the environment needed to perform the UT. When this environment is set up, all test cases must be executed according to their corresponding test procedures. After a test has been performed a report will be written.

Contents

Document Status Sheet

Document Status Overview

General

Document title:	Unit Test Plan
Identification:	UTP-1.0
Author:	Roel van Happen, Benjamin van der Hoeven, Femke Jansen, Hugo Snel
Document status:	Internally approved

Document History

Version	*Date*	*Author*	*Reason of change*
0.0	19-Jun-2013	Roel van Happen, Benjamin van der Hoeven, Femke Jansen, Hugo Snel	Initial version.

Document Change Records Since Previous Issue

General

Date:	19-Jun-2013
Document title:	Unit Test Plan
Identification:	UTP-1.0

Changes

Page	*Paragraph*	*Reason to change*
-	-	Initial version.

Chapter 1

Introduction

This chapter lists general information about this document.

1.1 Purpose

This document describes the plan for testing the developed software units against the detailed design, defined in the DDD [2]. The unit tests make sure that FINGERPAINT complies with the design in the Detailed Design (DD) phase of FINGERPAINT as described in the ESA software engineering standard [1].

1.2 Overview

Chapter 2 gives an overview of all items that will be tested, and the general criteria for the UT. Chapter 3 lists all the Unit Tests, chapter 4 specifies the test procedures and chapter 5 reports the test results.

1.3 List of definitions and abbreviations

1.3.1 Definitions

Ant Tool to build Java applications.

API A specification of how some software components should interact with each other.

Eclipse A universal toolset for software development.

GWT A Java software development toolkit by Google for building and optimizing browser-based applications.

HTML A language for creating web pages and other information that can be displayed in a web browser.

Javadoc A documentation generator for generating API documentation in HTML format from Java source code.

JUnit A unit testing framework for the Java programming language.

Local storage The storage from a web browser where data can be stored persistently.

Selenium Tool to automate browser input.

1.3.2 Abbreviations

2IP35	The Software Engineering Project
API	Application Programming Interface
DD	Detailed Design
DDD	Detailed Design Document
GUI	Graphical User Interface
GWT	Google Web Toolkit
HTML	HyperText Markup Language
JDK	Java Development Kit
SEP	Software Engineering Project
SVVP	Software Validation and Verification Plan
URD	User Requirements Document
UT	Unit Test(s)
UTP	Unit Test Plan

1.4 List of references

[1] ESA, *ESA Software Engineering Standards.* ESA, March 1995.

[2] Group Fingerpaint, "Detailed design document," *SEP*, 2013.

[3] Group Fingerpaint, "Software validation and verification plan," *SEP*, 2013.

[4] Group Fingerpaint, "User requirements document," *SEP*, 2013.

Chapter 2

Test plan

2.1 Test items

The software to be tested is the FINGERPAINT application. As this application is created using the Google Web Toolkit, the unit tests are run with GWT JUnit tests. The source code for the FINGERPAINT application is written in Java, but compiled to JavaScript, which means the tests should be run twice. Firstly, the Java bytecode is tested, using the GWT JUnit tests in development mode. Secondly, the tests are run as compiled JavaScript. The same GWT JUnit test can be run both in development mode and in production mode, albeit not at the same time. Therefore, tests need only be written once.

2.2 Features to be tested

The FINGERPAINT application should meet the design as described in the DDD [2]. Each component should adhere to the interfaces given in the DDD [2].

2.3 Test deliverables

Before the testing can commence, the following items must be completed:

- SVVP [3].
- DDD [2].
- UTP (this document, excluding chapter 5).
- UT input data.

After completing the testing the following items must be completed:

- UT report (chapter 5 of this document).
- UT output data.
- Problem reports (if applicable).

2.4 Testing tasks

Before any testing in the UT phase can take place, the following tasks need to be completed:

- Designing the unit tests.
- Tracing all test cases to components.
- Covering all components mentioned in the DDD [2] by test cases.
- Creating the UT input data.
- Ensuring that all environmental needs for the UT have been satisfied.

When these tasks have been completed, a UT can be performed according to the procedures described in chapter 4.

2.5 Environmental needs

To be able to perform the UT, the following resources are needed:

- One or more web browsers supported by the FINGERPAINT application (see the URD [4] for a list).
- A client device with Ant, JUnit, a JDK and GWT installed.

See also the constraints described in the DDD [2].

2.6 Test case pass/fail criteria

An overall UT pass can only be achieved when all tests described in chapter 3 have been performed and passed.

Chapter 3

Test case specifications

The specifications for the test cases can be found in the source code repository, more specifically in the `fingerpaint/test/src/nl/tue/fingerpaint/*` folder, where * represents the structure of the section names below. In the section below, a full list of test cases can be found. For an explanation of a test case – i.e. what it does and what it checks – refer to the Javadoc for that test in either the source code or the DDD [2].

3.1 List of test cases

The following tests are included.

3.1.1 Client

gui

UT1 NumberSpinnerTest.java

model

UT2 ApplicationStateTest.java

UT3 DrawingToolTest.java

UT4 RectangleGeometryTest.java

storage

UT5 FingerpaintJsonizerTest.java

UT6 FingerpaintZipperTest.java

UT7 ResultStorageTest.java

3.1.2 Shared

UT8 ServerDataResultTest.java

model

UT9 MixingProtocolTest.java

UT10 RectangleMixingStepTest.java

simulator

UT11 SimulationResultTest.java

UT12 SimulationTest.java

utils

UT13 ColourTest.java

3.1.3 Server

simulator

UT14 NativeCommunicatorTest.java

3.2 Suites

The suites allow for multiple test cases to be run at once. Every suite described below runs all the tests listed in their respective sections.

3.2.1 ClientUnit.java

- ApplicationStateTest
- DrawingToolTest
- FingerpaintJsonizerTest
- FingerpaintZipperTest
- NumberSpinnerTest
- RectangleGeometryTest
- ResultStorageTest

3.2.2 SharedUnit.java

- ColourTest
- MixingProtocolTest
- RectangleMixingStepTest
- ServerDataResultTest
- SimulationResultTest
- SimulationTest

3.2.3 ServerUnit.java

- NativeCommunicatorTest

Chapter 4

Test procedures

This chapter describes the procedures that have to be followed when writing and executing unit tests. Some general procedures that are applicable to server and client side testing are discussed in section 4.1. The specific procedures that have to be adhered for client side testing can be found in section 4.2, and the procedures that have to be adhered for client side testing can be found in section 4.3.
A procedure on how to execute these tests is given in section 4.4.

4.1 General procedures

For both client and server side functionality, the author of the code of a certain component should also write test cases accordingly for this component. It should never be the case that code is committed before it has been tested. Moreover, whenever existing functionality is re-factored, the person who changes the code must make sure that all the test cases still pass after the re-factoring.
As a rule of thumb, each class from the `src` folder should have its own test class in the `test` folder. The name of the test class should clearly indicate the test class under consideration and should be suffixed with `Test`. The test classes should only test functionality of the class under consideration. However, it might be necessary to include functionality of other classes (such as instantiating other classes) when testing a particular class. If a new testcase has been written, it should be added to one of the testsuites present in the code, in order to enable automatic testing from the command line.

4.2 Client side testing

On the client side, functionality can be tested with either GWT JUnit tests or with Selenium tests. The GWT JUnit tests can be used whenever a specific component has to be tested without the use of a graphical user interface. For example, tests regarding saving to and loading from the local storage can be tested internally: this can be tested by calling the appropriate store and retrieve functionality from the storage-related classes. Testing via the graphical user interface can be achieved through Selenium tests. These tests are written in Java and can be executed on several browsers of choice. After running the tests, the results for the selected browsers can be compared automatically, using a screen-shot comparator. All the GWT JUnit tests can be executed from two test suites, which can be found at

`ClientSuite.java` and `SharedSuite.java`. In a similar fashion, all Selenium tests can be executed via `SeleniumSuite.java`.
As mentioned before, some special settings are required to run the unit tests. This also applies to the test suites, as these contain all the individual unit tests regarding specific functionality. All the unit tests in the test suites will only pass, when they are executed in manual mode.

4.3 Server side testing

Server side testing can be done through JUnit tests, in a similar way as the GWT JUnit tests in section 4.2. Again, a test suite is available to execute all the server tests at once: `ServerSuite.java`.

4.4 Executing the tests

To execute the tests, it is assumed the environment of the system is setup correctly as described in ATP appendix A. Assuming the application is present in <APP-ROOT>, navigate the command-line to <APP-ROOT>/FINGERPAINT. To run the tests, one can now execute the following command:

```
ant test
```

This command compiles the application. After it is finished, a URL is presented on the command-line. Navigate the browser you would like to use to run the tests to this URL. The GWTTestCases wil now be run in this browser. Selenium tests and standard Junin tests will also be automatically run. After all tests are finished, a test report will be printed on the command line.

Chapter 5

Test reports

This chapter shows the test reports after executing the unit tests.

5.1 client

5.1.1 gui

NumberSpinnerTest

UT1	*Date: 19-june-2013*
testDefaultAboveMax	pass
testDefaultBelowMin	pass
testInitialisation	pass
testListener	pass
testMinBiggerMax	pass
testNotRound	pass
testOverMax	pass
testRound	pass
testUnderMin	pass

5.1.2 model

ApplicationStateTest

UT2	*Date: 19-june-2013*
testConstructor	pass
testDistribution	pass
testNrSteps	pass
testProtocol	pass
testSegregation	pass
testSetGeometry	pass
testSetMixer	pass
testStepsize	pass

DrawingToolTest

UT3	*Date: 19-june-2013*
testGetTool	pass
testRadius	pass

RectangleGeometryTest

UT4	*Date: 19-june-2013*
testInitCanvas	pass
testIsInsideBottomWall	pass
testIsInsideDrawingArea	pass
testIsInsideTopWall	pass
testIsOutsideDrawingArea	pass
testMixingSteps	pass

5.1.3 storage

FingerpaintJsoniserTest

UT5	*Date: 19-june-2013*
testJsonizeHashMap	pass
testJsonizeIntArray	pass
testJsonizeProtocol	pass
testJsonizeResultStorage	pass

FingerpaintZipperTest

UT6	*Date: 19-june-2013*
testZip	pass

ResultStorageTest

UT7	*Date: 19-june-2013*
testGetDistribution	pass
testGetGeometry	pass
testGetMixer	pass
testGetNrSteps	pass
testGetProtocol	pass
testGetSegregation	pass

5.2 shared

ServerDataResultTest

UT8	*Date: 19-june-2013*
testConstructor	pass
testSetters	pass

5.2.1 model

MixingProtocolTest

UT9	*Date: 19-june-2013*
testAddStep	pass
testAddStepException	pass
testConstructor	pass
testGetStepExecution	pass
testMoveStepBack	pass
testMoveStepException1	pass
testMoveStepException2	pass
testMoveStepForward	pass
testRemoveStep	pass
testRemoveStepException	pass
testToString	pass

RectangleMixingStepTest

UT10	*Date: 19-june-2013*
testGetStepSize	pass
testRounding	pass
testSetStepSize	pass
testToString	pass

5.2.2 simulator

SimulationResultTest

UT11	*Date: 19-june-2013*
testConstructor	pass
testGetSegregation	pass
testGetVectors	pass

SimulationTest

UT12	*Date: 19-june-2013*
testConstructor	pass
testGetConcentrationVector	pass
testGetMixer	pass
testGetProtocol	pass
testGetRuns	pass
testIntermediateVectors	pass

5.2.3 utils

ColourTest

UT13	*Date: 19-june-2013*
testConstructor	pass
testEquals	pass
testHex	pass
testPad	pass
testSetters	pass

5.3 server

5.3.1 simulator

NativeCommunicatorTest

UT14	*Date: 19-june-2013*
testSegregation	pass