

**I**ntegration **T**est **P**lan **D**ocument **(ITPD)**

Computer Science and Engineering (CSE)

Software Engineering 2 Project

Year 2015/16

*Date: 21/01/2016 – Version: 1.0*



**STUDENTS:**

*Martino Andrea (****788701****)*

*Marchesani Francesco (****852444****)*

**PROFESSOR:**

*Mirandola Raffaela*

# Introduction

## Table of contents

1. Introduction 2

1.0. Table of contents 2

1.1. Revision History 3

1.2. Purpose and Scope 3

1.3. List of Definitions and Abbreviations 4

1.4. List of Reference Documents 5

2. Integration Strategy 5

2.1. Entry Criteria 5

2.2. Elements to be integrated 6

2.3. Integration Testing Strategy 6

2.4. Sequence of Component/Function Integration 7

3. Individual Steps and Text Description 7

4. Tools and Test Equipment Required 8

5. Program Stubs and Test Data Required 9

6. Appendix 10

6.1. Hours of work 10

## Revision History

We will keep the **revision history** of the **Integration Test Plan Document** (**ITPD**) in this chapter.

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Summary** |
| 1.0 | 21/01/2016 | Andrea Martino, Francesco Marchesani | Document  Creation |

## Purpose and Scope

This **Integration Test Plan Document** (**ITPD**) contains information about the **test plan**, the **integration strategy** and **other required material** of *myTaxiService*.

This **document** is coherent with *the official template* of the project on the *Beep platform* (see *Assignment 4 – integration test plan.pdf*).

As we said for the **RASD** and the **DD**, it is important to underline that some parts of this document may evolve in the future (this may occurs for several causes).

Anyway, we will try to maintain coherence as much as possible.

Here is a resume of the steps of the project, with the related deadlines (in green documents already delivered, in yellow the current document):

The main scope of this **ITPD** (*Integration Test Plan Document*) is to give an overall guidance to the **testing phase** of the **project**, which is *myTaxiDriver* (**Software Engineering 2 project** of year 2015/16 - **Politecnico di Milano**).

* We described the main **goals** and **objectives** of the project in the previous *Requirements Analysis and Specification Document*.
* We also specified the **general architecture**, with the **components** of the system in the other previous document (*Design Document*).

## List of Definitions and Abbreviations

* **RASD**: *Requirements Analysis and Specification Document*
* **DD**: *Design Document*
* **ITPD**: *Integration Test Plan Document*
* **mTS**: *myTaxiService*
* **SE**: *Software Engineering*
* **IDE**: Integrated Development Environment
* **JEE**: Java Enterprise Edition
* **Mockito**: tool for mockups creation (useful in *Unit Testing*).
* **Arquillian**: tool for integration testing.
* **ShrinkWrap**: Java API for archive manipulation. It powers the *arquillian* deployment mechanism.
* **NetBeans**: open source IDE.
* **Unit Testing (UT)**: a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.
* **Integration Testing (IT)**: is the phase in software testing in which individual software modules are combined and tested as a group.
* **System Testing**: is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.
* **Performance Testing**: is a testing practice performed to determine how a system performs in terms of responsiveness and stability under a particular workload.
* **Load Testing**: is the process of putting demand on a software system or computing device and measuring its response.
* **Stress Testing**: is a test that put a greater emphasis on robustness, availability, and error handling under a heavy load, rather than on what would be considered correct behavior under normal circumstances.
* **Test Plan**: is a document detailing the objectives, target market, internal beta team, and processes for a specific beta test for a software product.
* **Top-Down Strategy**: is an integration strategy that starts from the highest level of abstraction (user view in our application) and gradually adds levels of detail.
* **Bottom-Up Strategy**: is an integration strategy that starts from the lowest level of abstraction and gradually combines elements to add levels of abstraction.
* **Sandwich Strategy**: (also called *Mixed Strategy*) is an integration strategy that mixes both *Top-Down* and *Bottom-Up*.
* **Stub**: A *method stub* or simply *stub* in software development is a piece of code used to stand in for some other programming functionality of *lower* levels of abstraction.
* **Driver**: A *method driver* or simply *driver* in software development is a piece of code used to stand in for some other programming functionality of *higher* levels of abstraction. //MAYBE SOME EXAMPLES?

**Other stuff, of course!**

**Note:** *for the full Glossary may be helpful to see also the paragraph 1.5 of the RASD 2.0 and paragraph 2.3 of DD.*

## List of Reference Documents

Here is a list of the **reference documents** for the *Integration Test Plan Document* of *myTaxiService*:

* **Project Description** (from *Beep* platform)
* **RASD 2.0 [RASD Revision]** (hosted on *GitHub Repository*)
* **Design Document [DD]** (hosted on *GitHub Repository*)
* **JUnit Documentation** (*http://junit.org/javadoc/latest/)*
* **Arquillian Documentation** (*http://docs.jboss.org/arquillian/aggregate/latest/)*
* **NetBeans Documentation** (*https://netbeans.org/kb/)*
* /\* Documentation of other tools used for testing \*/

# Integration Strategy

## Entry Criteria

It is important to underline the **entry criteria** before the application of the integration testing process. This is a list of the required entry criteria, with respect to *myTaxiService* project:

* COMPLETED Functions must been have **unit tested**, otherwise there is a high probability of issues with the standalone units (without looking at their interactions, as target of the integration tests).
* *Requirements Analysis and Specification Document* (**RASD**) and *Design Document* (**DD**) must be completed.
* The code has a **proper documentation**, in order to be readable from the point of view of the testers. Otherwise, they may occur problems with the readability of some parts of the code.
* All the **required tools** are available and work without problems.
* OTHERS?

## Elements to be integrated

We want to integrate the **components** described in the *Design Document* in order to test incrementally the integration of the elements.

We identified the **clusters** of elements to be integrated in our integration strategy. The integration will be coherent with the clustering aggregation.

This are the **four** **clusters** (*partially overlapping*) of *myTaxiService* components for the different testing phases:

//DA AGGIORNARE

1. **Client-Server interaction cluster**: contains *Customer Manager, Taxi Driver Manger* and the related stubs and drivers.
2. **Client-System interaction cluster**: contains the *user interfaces* of *Customer, Taxi Driver and SysAdmin* with the related stubs on bottom.
3. **Internal Server interaction cluster**: contains *Queue Manger, Reservation Manager, Maps Manager* and a *System Manager Stub.*
4. **System interactions cluster**: is the biggest cluster to be integrated. It contains the *System Manager* within the *Data Layer* and several stubs (*Maps Manager, Reservation Manager, Queue Manager, Google Maps API* and *Payment Services*).

?

We want to remark the integration of the **two external APIs** (*Payment API* and *Google Maps API*). We will create only *little stubs* in these special cases because they are external services **easy to integrate** by default.

## Integration Testing Strategy

We decided to choose a **Sandwich** (or **Mixed**) *integration testing strategy*, so we will use both **Top-Down** and **Bottom-Up** depending on the specific case of integration.

The main advantage of this approach is that testers may reach a **high level of parallelism** in testing. This will improve **teamwork**, **work subdivision** and will reduce the **total length** of the integration testing process. In a world where *time is money*, make this phase shorter will assure benefits for sure.

With the *Sandwich approach*, we will use both **drivers** (for *bottom-up phases*) and **stubs** (for *top-down phases*) gradually, of course (see *chapter 5. Program Stubs and Test Data Required* for more information).

Someone may object that the creation of drivers and stubs is time-consuming and heavy from a developing point of view. Anyway, several of these instruments may be reused with only few fixes. Therefore, we will grant a **drivers/stubs** **recycling policy**, in order to avoid loss of time and other resources.

We can summarize with the following sentence:

*“Similar stubs/drivers in different tests, with parallelism: this is the key.”*

Let us see the general structure of **Sandwich approach**, with a graphical representation:

Sandwich Integration Strategy

Now let us focus on the *myTaxiService* specific project. As it is possible to see from the *component diagram* there is “*star structure*” for the central components.

The core of this structure is the **System Manager** component. It directly interacts with *Customer Manager, Taxi Driver Manager, Maps Manager, Reservation Manager*, *Queue Manager* and the *External Components* (*Google Maps* and *Payment Services*).

Let us see a graphical representation:

Our testing strategy considers this structure as a starting point. Then we will apply the **Sandwich approach** following the **clusters** identified in *Chapter 2.2 Elements to be integrated.* For the full *Component Diagram*, with the related components and interfaces see the *Design Document (DD)*.

## Sequence of Component/Function Integration

Here you can see how we will integrate different component. For every cluster

### Software Integration Sequence

* **Integration tests of Client-Client Manager interaction cluster (Top Down)**

|  |  |  |
| --- | --- | --- |
| ID | Integration Test | Paragraphs |
| I1 | Customer Client -> Client Manager (Stub), Google Maps |  |
| I2 | Taxi Driver Client -> Taxi Driver Manager (Stub), Google Maps |  |
| I3 | SysAdmin Client -> System Manager (Stub) |  |

* **Integration tests of Client-System interaction cluster (Sandwich)**

|  |  |  |
| --- | --- | --- |
| ID | Integration Test | Paragraphs |
| I1 | Customer Client (Driver) -> Customer Manager |  |
| I2 | Client Manager -> System (Stub) |  |
| I3 | Taxi Driver Client (Driver) -> Taxi Driver Manager |  |
| I4 | Taxi Driver Manager -> System Manager (Stub) |  |

* **Integration tests of System-External components cluster (?)**

|  |  |  |
| --- | --- | --- |
| ID | Integration Test | Paragraphs |
| I1 | System Manager -> Google Maps |  |
| I2 | System Manager -> Payment Services (Stub)\* |  |

\*As we have seen in *Design Document*, *Payment Services* is an abstract external component that handles monetary transaction. However, in real case scenarios, *Payment Services* will represent an SDK/interface provided by an external company. Usually the SDK should give the instruments to simulate transaction. If not a proper stub must be created.

* **Integration tests of Internal Server interaction cluster (Bottom Up)**

//DA CORREGGERE ORDINE COMPONENTI

|  |  |  |
| --- | --- | --- |
| ID | Integration Test | Paragraphs |
| I1 | Reservation Manager (Driver) -> System Manager |  |
| I2 | Queue Manager (Driver) -> System Manager |  |
| I3 | Maps Manager (Driver) -> System Manager |  |

* **Integration tests of System-Data cluster (Top Down)**

|  |  |  |
| --- | --- | --- |
| ID | Integration Test | Paragraphs |
| I1 | System Manager -> Test Data |  |

\* BLOCKS OF COMPONENTS INVOLVED \*

### Subsystem Integration Sequence

\* IDEM SEE ABOVE \*

# Individual Steps and Text Description

1st kind of tables

|  |  |
| --- | --- |
| **Test Case Identifier** |  |
| **Test Item(s)** |  |
| **Input Specification** |  |
| **Output Specification** |  |
| **Environmental Needs** |  |

2nd kind of tables

|  |  |
| --- | --- |
| **Test Procedure Identifier** |  |
| **Purpose** |  |
| **Procedure Steps** |  |

# Tools and Test Equipment Required

In this chapter, we will show the **tools** and the **test equipment** required for the *Integration Testing*.

Note that this section regards only the Integration Testing. In fact, we will not talk about other useful tools like **mockito**,which are used in the **Unit Testing** **phase** (before the Integration Testing, as said in *Chapter 2.1 Entry Criteria*). In addition, we will also not talk about tools related to other types of testing, such as *System Testing*, *Performance Testing*, *Load Testing*, *Stress Testing* and so on.

We used the following tools and test equipment:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Logo** | **Website** | **Function** |
| **Arquillian** |  | *http://arquillian.org/* | Integration testing framework for containers |
| **JUnit** |  | *http://junit.org/* | Framework to write repeatable tests |
| **NetBeans** |  | *https://netbeans.org/* | IDE for manual integration testing |

Now it is useful to give a **motivation** related to the use of these tools in practice:

* **Arquillian** combines a unit testing framework (*JUnit* in our case), *ShrinkWrap*, and one or more supported target containers (*Java EE containers*) to provide a simple, flexible and pluggable integration testing environment. We selected this tool because it is a helpful open source standard for integration testing of big projects.
* We know that **JUnit** should be used before Integration testing (so in *Unit Tests*). Anyway, we will also use it to do integration testing when possible. In fact, it is a versatile tool and may be helpful for testing in several cases. We will use it mainly for assertions for testing expected results.
* **NetBeans** is an open source IDE for several programming languages, as JEE. We selected this IDE as testing environment because it is optimized for big enterprise projects like *myTaxiService*. It does not require special plugins to deal with JEE. See also [*https://netbeans.org/enterprise/index.html*](https://netbeans.org/enterprise/index.html)for more details.
* We will also consider **manual testing** for some part of the code. Sometimes, in fact, it may add knowledge to other systematic ways of testing. In this case, the tester directly plays the role of the end user. We will focus only on “*smart*” *test cases*, in order to avoid waste of time. See *chapter 5* for more details.

# Program Stubs and Test Data Required

From Wikipedia:

**STUB**

*A method stub or simply stub in software development is a piece of code used to stand in for some other programming functionality. A stub may simulate the behavior of existing code (such as a procedure on a remote machine) or be a temporary substitute for yet-to-be-developed code. Stubs are therefore most useful in porting, distributed computing as well as general software development and testing.*

**TEST DATA**

*Test data is data which has been specifically identified for use in tests, typically of a computer program.*

*Some data may be used in a confirmatory way, typically to verify that a given set of input to a given function produces some expected result. Other data may be used in order to challenge the ability of the program to respond to unusual, extreme, exceptional, or unexpected input.*

*Test data may be produced in a focused or systematic way (as is typically the case in domain testing), or by using other, less-focused approaches (as is typically the case in high-volume randomized automated tests). Test data may be produced by the tester, or by a program or function that aids the tester. Test data may be recorded for re-use, or used once and then forgotten.*

# Appendix

## Hours of work

* **Andrea Martino**: Hours
* **Francesco Marchesani**: Hours