Integration Test Plan Document (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

1. Introduction

1.0. Table of contents

1.1. 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

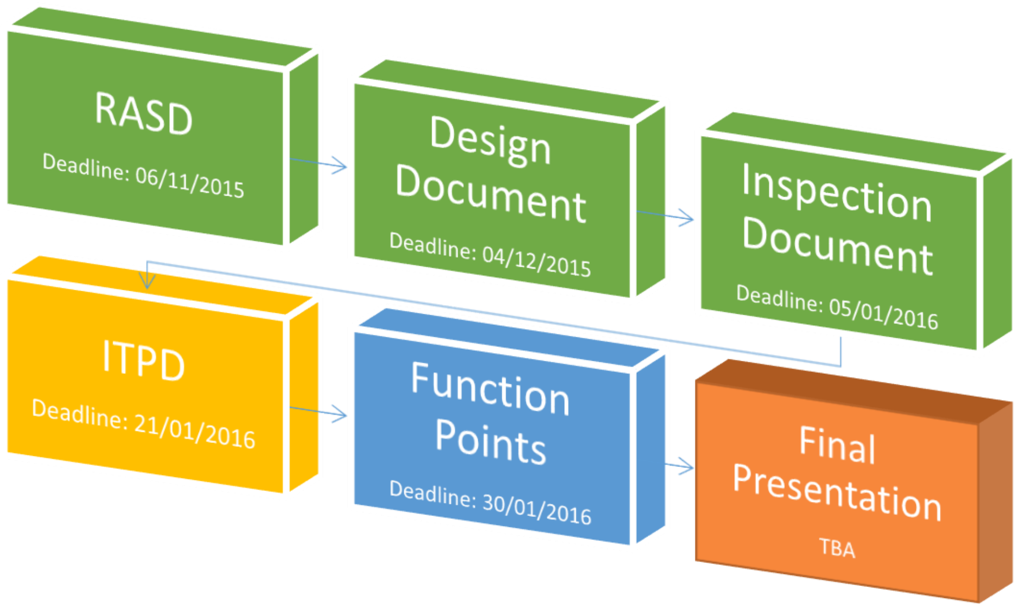
1.2. Purpose and Scope

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*).

1.3. 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*
- **GUI**: *Graphical User Interface*
- **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.
- **Regression Testing**: is a type of software testing that seeks to uncover new software bugs, or regressions, in existing functional and non-functional areas of a system after updates.

- **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.

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.*

1.4. 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/)*

2. Integration Strategy

2.1. 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.
- All the **required tools** are available and work without problems.

2.2. 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 **five clusters** (*partially overlapping*) of *myTaxiService* components for the different testing phases:

I. **Client-Client Manager interaction cluster**: contains the *user interfaces* of *Customer, Taxi Driver and SysAdmin* with the related stubs on bottom.
II. **Client-System interaction cluster**: contains *Customer Manager, Taxi Driver Manger* and the related stubs and drivers.
III. **System-External component interaction cluster:** contains *System Manager, Google Maps* and *Payment Services*.
IV. **Internal Server interaction cluster**: contains drivers of *Queue Manager, Reservation Manager, Maps Manager* and the *System Manager.*
V. **System-Data cluster**: contains *System Manager* and *Test Data*.

2.3. 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).

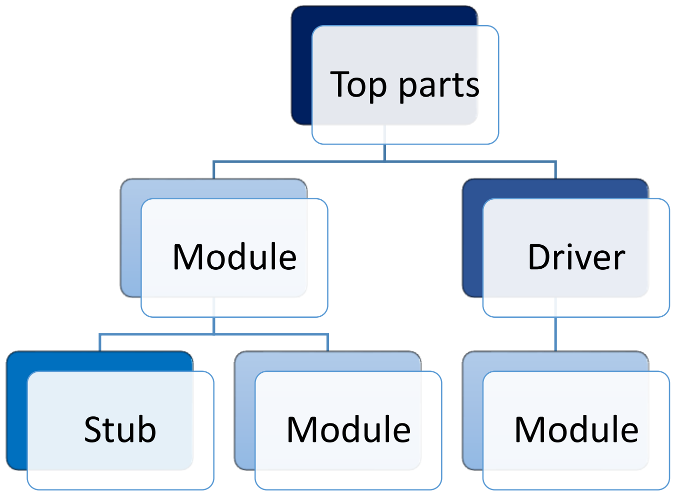
Someone may object that the creation of drivers and stubs is time-consuming and heavy from a developing point of view. Anyway, several of these instruments may be reused with only few fixes. Therefore, we will grant a **drivers/stubs recycling policy**, in order to avoid loss of time and other resources.

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:

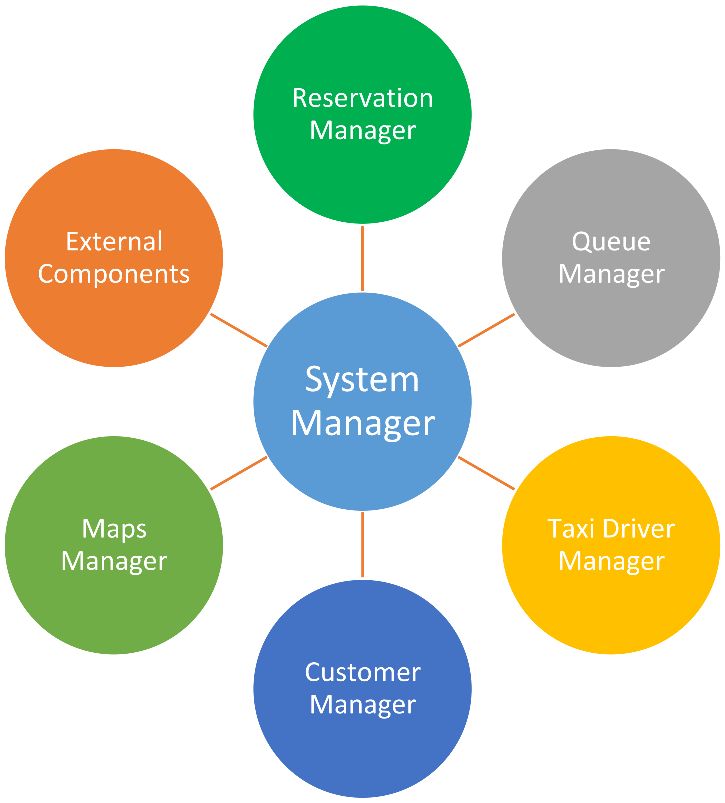
Sandwich Integration Strategy



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)*.

Note that integration testing is usually very hard to perform in **big enterprise projects**, such *myTaxiService*. As in our case, there are more alternatives for the plan. **Classes coupling** and **functional dependencies** can often cause problems in the integration phase. Anyway, we adopted **different ad-hoc strategies** in order to obtain better results.

In addition, we strongly suggest to perform periodically **regression tests** in the future evolutions of the software. See next paragraphs for detailed information about our integration test plan.

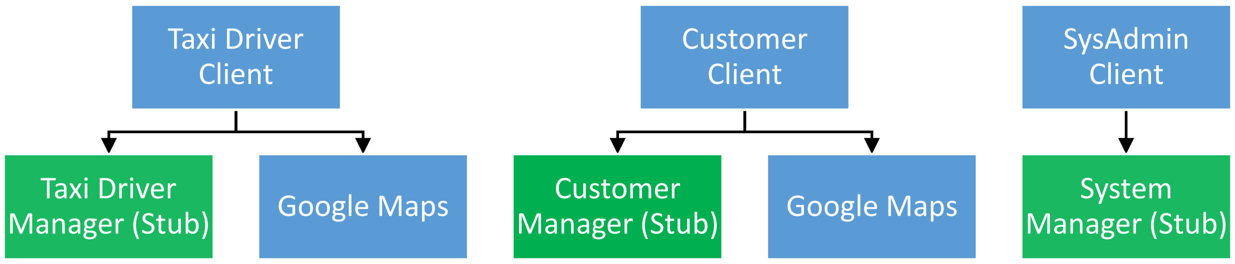
2.4. Sequence of Component/Function Integration

Here you can see how we will integrate different component with respect to the clusters.

2.4.1. Software Integration Sequence

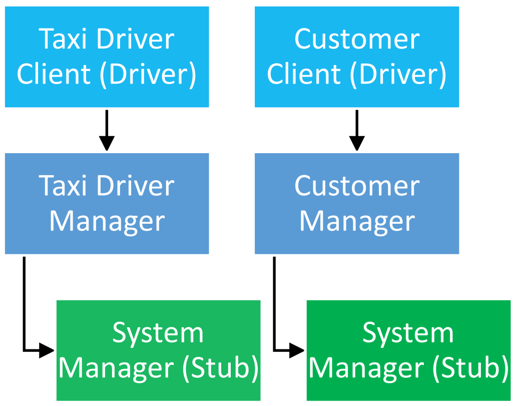
➢ **Integration tests of Client-Client Manager interaction cluster (Top Down)**

ID	Integration Test
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/Thread)**

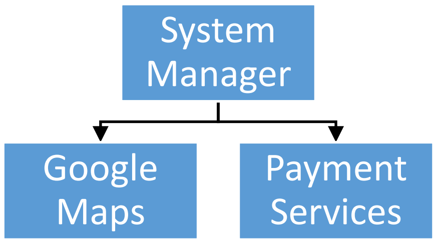
ID	Integration Test
I4	Customer Client (Driver) -> Customer Manager
I5	Client Manager -> System Manager (Stub)
I6	Taxi Driver Client (Driver) -> Taxi Driver Manager
I7	Taxi Driver Manager -> System Manager (Stub)



- **Integration tests of System-External components cluster (Top Down)**

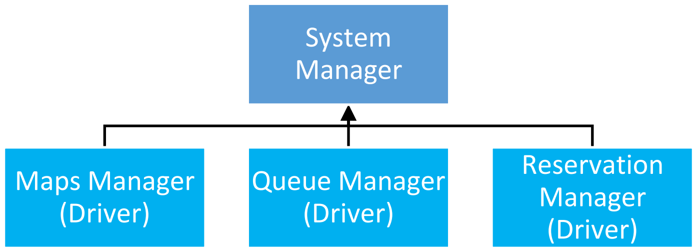
ID	Integration Test
I8	System Manager -> Google Maps
I9	System Manager -> Payment Services (Stub)*

***** 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.



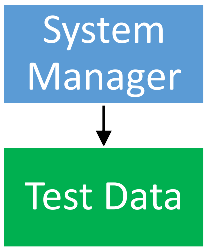
- **Integration tests of Internal Server interaction cluster (Bottom Up)**

ID	Integration Test
I10	Reservation Manager (Driver) -> System Manager
I11	Queue Manager (Driver) -> System Manager
I12	Maps Manager (Driver) -> System Manager



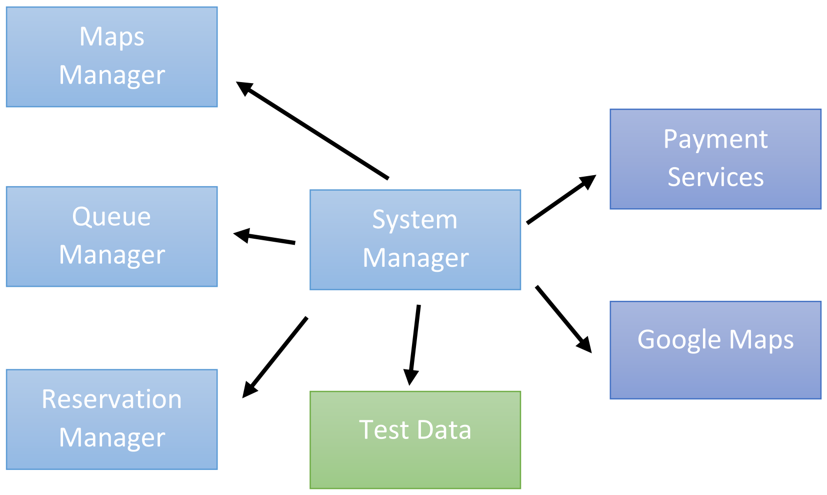
- **Integration tests of System-Data cluster (Top Down)**

ID	Integration Test
I13	System Manager -> Test Data

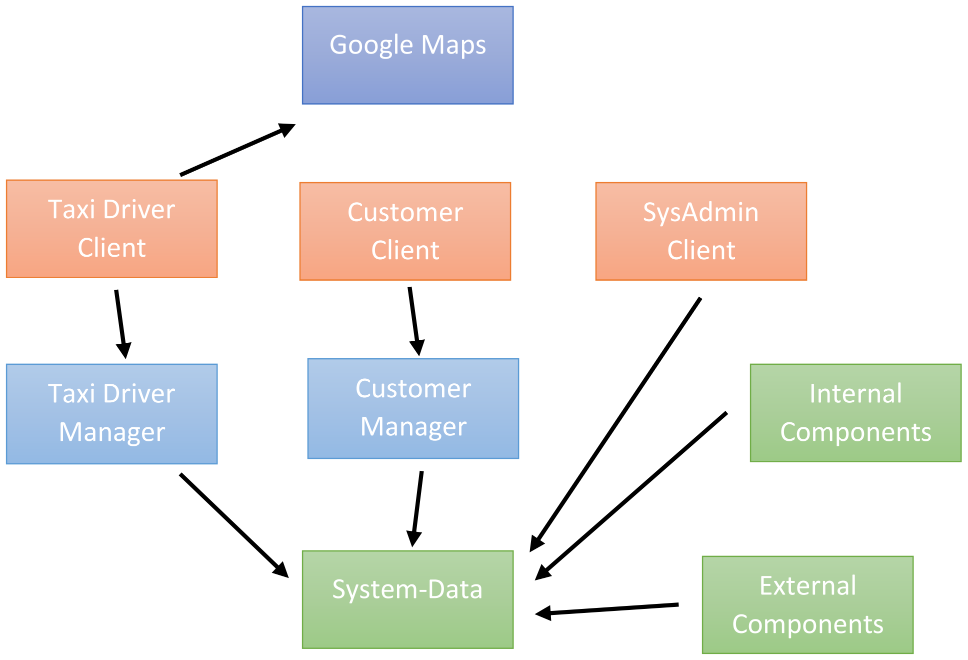


2.4.2. Subsystem Integration Sequence

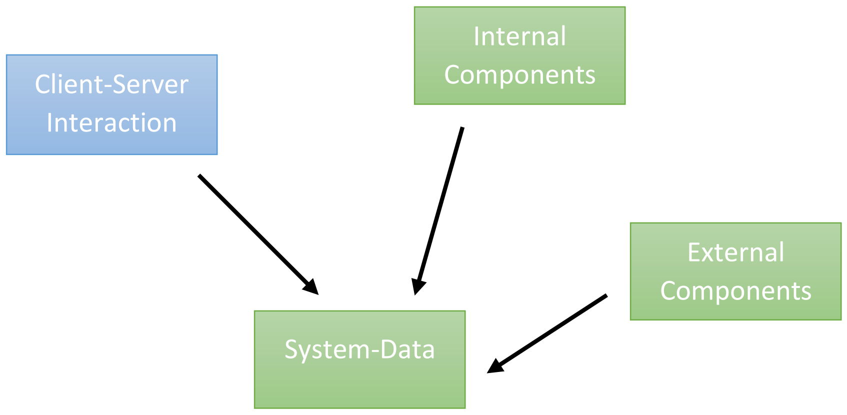
ID	Integration Test
I14	System-Data -> Internal Server, External Components



ID	Integration Test
I15	Client-Client Manager -> Client-System



ID	Integration Test
I16	Whole mTS



3. Individual Steps and Text Description

3.1. Software Integration

Test Case Identifier	I1T1
Test Item(s)	Customer Client -> Client Manager (Stub), Google Maps
Input Specification	Specific inputs are given to the Customer Client that will require the client application to interface with mTS Server.
Output Specification	Customer Client is able to properly handle user inputs, to send them to the Client Manager stub and show simulated responses generated by the stub.
Environmental Needs	Client Manager stub

Test Case Identifier	I1T2
Test Item(s)	Customer Client -> Google Maps
Input Specification	Maps/GPS position is requested to the Customer Client.
Output Specification	Customer Client is able to require and show information from Google Maps external component.
Environmental Needs	N/A

Test Case Identifier	I2T1
Test Item(s)	Taxi Driver Client -> Taxi Driver Manager (Stub)
Input Specification	Specific inputs are given to the Taxi Driver Client that will require the taxi driver application to interface with mTS Server.
Output Specification	Customer Client is able to properly handle user inputs, to send them to the Taxi Driver Manager stub and show simulated responses generated by the stub.
Environmental Needs	Taxi Driver Manager stub

Test Case Identifier	I2T2
Test Item(s)	Taxi Driver Client -> Google Maps
Input Specification	Maps/GPS position is requested to the Taxi Driver Client.
Output Specification	Customer Client is able to require and show information from Google Maps external component.
Environmental Needs	N/A

Test Case Identifier	I3T1

Test Item(s)	SysAdmin Client -> System Manager (Stub)
Input Specification	Commands are given to SysAdmin Client.
Output Specification	SysAdmin correctly sends commands to System Manager and properly handles responses.
Environmental Needs	System Manager stub

Test Case Identifier	I4T1
Test Item(s)	Customer Client (Driver) -> Customer Manager
Input Specification	Simulated user input are given from the Customer Client driver to the Customer Manager.
Output Specification	Customer Manager is able to create a proper response and to send it to the Customer Client driver.
Environmental Needs	Customer Client driver, I5 succeeded

Test Case Identifier	I5T1
Test Item(s)	Customer Manager -> System Manager (Stub)
Input Specification	Customer Manager methods that require interaction with the System Manager are called.
Output Specification	Simulated response from the stub are correctly handled by the Customer Manager.
Environmental Needs	System Manager stub

Test Case Identifier	I6T1
Test Item(s)	Taxi Driver Client (Driver) -> Taxi Driver Manager
Input Specification	Simulated user input are given from the Taxi Driver Client driver to the Taxi Driver Manager.
Output Specification	Taxi Driver Manager is able to create a proper response and to send it to the Taxi Driver Client driver.
Environmental Needs	Taxi Driver Client driver, I7 succeeded

Test Case Identifier	I7T1
Test Item(s)	Taxi Driver Manager -> System Manager (Stub)
Input Specification	Taxi Driver Manager methods that require interaction with the System Manager are called.
Output Specification	Simulated response from the stub are correctly handled by the Taxi Driver Manager.
Environmental Needs	System Manager stub

Test Case Identifier	I8T1
Test Item(s)	System Manager -> Google Maps

Input Specification	System Manager specific methods are called to request information from Google Maps component.
Output Specification	Method are successfully executed, requests are forwarded to Google Maps and System Manager properly handles responses.
Environmental Needs	I12 succeeded

Test Case Identifier	I9T1
Test Item(s)	System Manager -> Payment Services
Input Specification	Some simulated payment request are made to the System Manager.
Output Specification	System Manager properly handles payment requests, forwards them to Payment Services and gets an expected response.
Environmental Needs	Specific payment service stub, if transaction simulation is not available.

Test Case Identifier	I10T1
Test Item(s)	Reservation Manager (Driver) -> System Manager
Input Specification	Some method that will require interaction with the System Manager are called on the Reservation Manager driver.
Output Specification	The System Manager correctly forwards the requests to other drivers and gives a proper response to the initial caller.
Environmental Needs	Reservation Manager driver

Test Case Identifier	I11T1
Test Item(s)	Queue Manager (Driver) -> System Manager
Input Specification	Some method that will require interaction with the System Manager are called on the Queue Manager driver.
Output Specification	The System Manager correctly forwards the requests to other drivers and gives a proper response to the initial caller.
Environmental Needs	Queue Manager driver

Test Case Identifier	I12T1
Test Item(s)	Maps Manager (Driver) -> System Manager
Input Specification	Some method that will require interaction with the System Manager are called on the Maps Manager driver.

Output Specification	The System Manager correctly forwards the requests to other drivers and gives a proper response to the initial caller.
Environmental Needs	Maps Manager driver

Test Case Identifier	I13T1
Test Item(s)	System Manager -> Test Data
Input Specification	Method that interacts with persistent data are called on System Manager.
Output Specification	Data handling is performed without problems.
Environmental Needs	Test Data

3.2. Subsystem Integration

Test Case Identifier	I14T1
Test Item(s)	System-Data -> Internal Server
Input Specification	Inputs are given to and from the System-Data cluster to test the interaction with the others internal components.
Output Specification	Every components properly reacts to every input given.
Environmental Needs	Test Data, I10, I11, I12, I13 succeed

Test Case Identifier	I14T2
Test Item(s)	System-Data -> External Components
Input Specification	Inputs are given to and from the System-Data cluster to test the interaction with the other external components.
Output Specification	Every components properly reacts to every input given.
Environmental Needs	Test Data, I8, I9, I12 succeeded

Test Case Identifier	I15T2
Test Item(s)	Client-Client Manager -> Client-System
Input Specification	Inputs are given to Clients and Client-System to test the interactions between the two clusters.
Output Specification	Every components properly reacts to every input given.
Environmental Needs	Test Data, I1, I2, I3, I4, I5, I6, I7, I14 succeeded

Test Case Identifier	I16T1
Test Item(s)	Whole mTS
Environmental Needs	Test Data, I14, I15 succeeded

Note: There is no need to specify that the whole system must correctly work with every possible input to pass this test. We understand that this phase will not necessarily end before the release of *mTS*, given that in a complex system is practically impossible to discover every possible malfunctions during the testing phase.

3.3. Test Procedures

Test Procedure Identifier	TP1
Purpose	This test procedure verifies whether the System Manager: • can handle command-line input by SysAdmin Client • can handle GUI input by SysAdmin Client • can handle customer input transmitted via Customer Manager • can handle taxi driver input transmitted via Taxi Driver Manager • can output requested information to a customer via Customer Manager • can output requested information to a taxi driver via Taxi Driver Manager
Procedure Steps	Execute 13T1 after I13T1

Test Procedure Identifier	TP2
Purpose	This test procedure verifies whether the Customer Client software: • can handle input (mobile Application User Interface) • can handle input (web app User Interface) • can output requested information to the Customer Manager and then to the System Manager • can output information to the Client Manager and then to the System Manager
Procedure Steps	Execute I5T1 after I4T1

Test Procedure Identifier	TP3
Purpose	Purpose This test procedure verifies whether the Taxi Driver Client software:

	• can handle input (taxi driver User Interface) • can output requested information to the Taxi Driver Manager and then to the System Manager • can output information to the Taxi Driver Manager and then to the System Manager
Procedure Steps	Execute I7T1 after I6T1

4. 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	arquillian	*http://arquillian.org/*	Integration testing framework for containers
JUnit	JUnit	*http://junit.org/*	Framework to write repeatable tests
NetBeans	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* 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.

5. Program Stubs/Drivers and Test Data Required

As you can see in section 2 and section 3 we have used some stubs and drivers to test the integration between a complete module and a yet-to-be completed module.

- **Client Manager (Stub), Taxi Driver Manager (Stub)**: Those consist in components that are deployed in mTS Server and their role is to simulate respectively Customer Manager and Taxi Driver Manager component and provide a basic Client – Server interaction to decouple Client developing from Server developing. In this way it is possible to build and test functioning Clients (with, if required, the GUI) without waiting the Server part of mTS to be completed.

- **System Manager (Stub)**: System Manager dummy component. Note that it is not recommended to build a stub capable of covering every aspect of the System Manager because, as we have seen in Design Document, System Manager is a complex module that "glues" all the internal component. In fact, System Manager Stubs are built in I3 to test SysAdmin Client component, in I5 to test Customer Manager Component and I7 to test Taxi Driver Component.

- **Taxi Driver Client (Driver), Customer Client (Driver)**: Exactly the same case of Client Manager Stub and Customer Client stub, but with inverted roles. In fact, using those drivers it is possible to build and test functioning Managers without having to wait the Client part of mTS to be completed.

- **Reservation Manager (Driver), Queue Manager (Driver), Maps Manager (Driver)**: These are the most important drivers. In fact, they are use to build and test System Manager Component.

- **Test Data**: It is important to define some sample data to test every required functionality of *mTS*. For example it is required to have an address table of a portion of the city mTS service wants to cover (see Design Document for a brief explanation on how an address is considered valid or not), some fake reservation to test the correct behavior of the Reservation Manager (the ability to provide an easy interface to manage reservation and update other components of some reservations that need to be handled) and System-Data cluster (that is, the interaction between System Manager and Data layer) and so on.

6. Appendix

6.1. Hours of work

- **Andrea Martino**: ~ 20 Hours
- **Francesco Marchesani**: ~ 20 Hours