Politecnico di Milano

Department of Electronics, Information and Bioengineering

Master Degree course in Computer Science Engineering

Integration Test Plan Document (ITPD)

myTaxiService

Instructor: Prof. Elisabetta Di Nitto

Authors:	*Code:*
Luca Luciano Costanzo	789038
Simone Disabato	852863

Document version 1.01, released on January 21, 2016

Contents

Chapter 1

Introduction

In this chapter the purpose and the scope of the document will be presented in the section 1.1. Then, other useful information are made available, for instance the list of definitions and abbreviations and the reference documents. Finally, in the section 1.4 an overall description of the document structure will be presented.

1.1 Purpose and Scope

The purposes of this document are principally two. The first one is to describe the sequence of myTaxiService component's implementation. The sequence of the implementation of the components described in the Design Document is important for the testing phase.
The second main objective of this document is to define the testing phase of the components' interactions: an accurate description will be provided with the required new components and the tests' definitions, each in a dedicate section.
Finally, in order to accomplish the tests some tools are obviously needed. In this document we will pay particular attention to which tool and how they will be used.

Up to now, we have described the purposes of the document. The global scope of the project follows, as it has been explained in the previous documents, to

make the document easy to understand.
Users, once registered, are able to ask for an immediate ride or to book one of them.

The system provides the user with a complete map of the city and its suburbs within the taxi service is available. The current position of the user is obtained by localization services of the user's smartphone if it's possible, otherwise the user notifies his position directly on the map with a marker or by a searching box. The destination is also chosen either graphically or by a research. The user can view the suggested path and then he must confirm the request.

When a user asks for a ride, the system checks the availability of a taxi driver near the current position, by splitting the city in several areas and using a FIFO (First In First Out) policy to manage the assignment of the ride's driver. The selected driver can accept or decline the ride. In the former case the system informs the user about waiting time, estimated travelling time, prices and cab car-code.

The system gives also the possibility to book a ride with at least two hours in advance. As the user does when he asks for a ride, he selects the desired starting venue and the destination. Afterwards, the system gives a calendar where the customer can choose the date (at most 30 days in advance) and the starting hour. Ten minutes before the meeting time the system starts all the operations described before in order to assign a taxi-driver.

A reservation from the app or the website can be undone until the system confirmation of the availability of a taxi, while a booking can be cancelled at most fifteen minutes before the meeting hour.

After those deadlines the ride is considered bought by the customers and an eventual absence on the established venue forbids other possibilities to book or to take a ride.

1.2 List of Definitions and Abbreviations

Up to now, no definitions or acronyms or abbreviation have been used in the document. Hence, this section is empty.

1.3 List of Reference Documents

The reference documents are now listed. Note that, all the documents related on the *myTaxyService* project are written by the same authors of this one, whereas the other documents have a reference of their author when this information is available.

- Software Engineering 2 Project, AA 2015-2016 Assignments 4 - Test plan (available on beep platform only for registered students of Politecnico of Milan);
- The Requirements Analysis and Specification Document (RASD) for *myTaxiService* - v1.2, released on 6th November 2015;
- The Design Document (DD) for *myTaxiService* - v1.0, released on 4th December 2015;
- Mockito reference document, available at
  http://site.mockito.org/mockito/docs/current/org/mockito/Mockito.html or
  http://mockito.github.io/mockito/docs/current/org/mockito/Mockito.html
- Arquillian reference document, available at
  https://docs.jboss.org/author/display/ARQ/Reference+Guide?_sscc=t

1.4 Overall Description

In the chapter 2 the preconditions, the required new components, the temporal development/testing order and the testing strategies are analysed in details.
In the chapter 3 each test will be analysed with a brief description and the expected results.
In the chapter 4 a presentation of the used tools we will be provided with the

reasons for their use. Finally, in the chapter 5 we will discuss about the special test data or program stubs required for each integration step.

Chapter 2

Integration Strategy

In this chapter the integration strategy will be described. In the section 2.1 the prerequisites for the tests will be presented. The section 2.2 is dedicated to the required components to be integrated in the system to execute some tests. Finally in the sections 2.3 and 2.4 the strategy used to test the integrations will be discussed, paying attention to the order.

2.1 Entry Criteria

The criteria which are required to perform each test are easy. The involved components (and eventually needed program stubs) must be developed and fully tested before executing the integration test.
In addition, for some test, further preconditions may be required. These prerequisites will not be described here, but in the chapter 3, dedicated to test's description.

2.2 Elements to be Integrated

First of all, no further components need to be integrated to perform integration tests.
Then, to perform the tests described in the following pages, all the components of the system need to be implemented. Furthermore, it is not required that all

the components must be implemented before executing the first integration test: in the subsection 2.4.1 the order of implementation is well-presented focusing both on the subsystems and on the reasons for the required order.

2.3 Integration Testing Strategy

We have decided to test our system's interactions with a bottom-up approach. In the section 2.4 it is possible to see the components' order of implementation defined by us with specific criteria. As soon as two components are available, the tests associated to their interactions have to be applied. Before starting the development of the following part of the system, all the tests should be successful.
Obviously, to perform some tests, we need to use a *non-available* component (it has not been implemented yet): to simulate this components we will use some particular program stubs, if any, and they will be described in detail in the chapter 5.

2.4 Sequence of Component/Function Integration

In this section we point out the features related to the order of the components' development. Which is our order of implementation? Why do we need to develop the component *A* before *B*? These two questions are fully discussed from now.

2.4.1 Software Integration Sequence

The "subsystems" of *myTaxiService* are two, both important.
The *Client and User Handler* has the specific role to bind the users' requests to the corresponding services on the system by enqueueing the request on the dispatcher. Then it is able to show the results or to perform the following interactions of the involved service.
To handle these operations it has three components, with the names *Authentica-*

tion, *ServiceShow* and *ClientInterface*[1]. In the Figure 2.1 the order of implementation is shown[2].

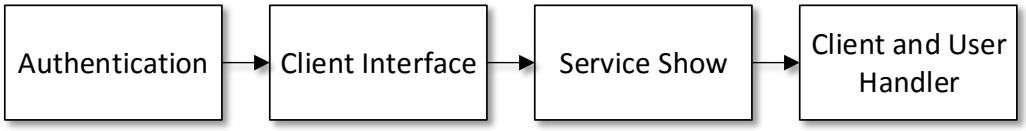

Figure 2.1: Integration Sequence of Client and User Handler

The *Ride Allocator* handles everything that is related to a ride, so the taxy drivers' queues, their assignments and the identification of their positions (by GPS coordinates or an address).
A more precise description is available in the Design Document, as usual, whereas here we are interested only in the implementation order shown in Figure 2.2

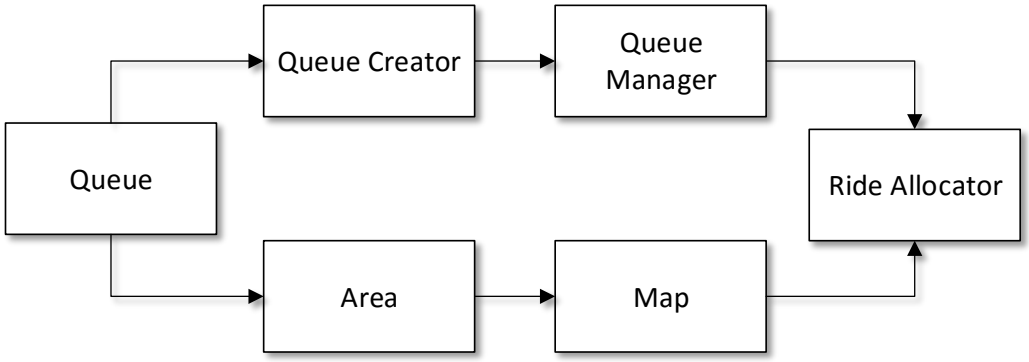

Figure 2.2: Integration Sequence of Ride Allocator

The interactions and the integration inside these two subsystems are not tested in this project phase, but when the single components are implemented. In fact, each component is tested as soon as its first implementation has been

[1]To have a brief description of each of them read the Design Document.

[2]According to the Design Document the reader may expect to see all the interfaces of the Client and User Handler. To simplify the representation all the interfaces and the classes have been grouped and inserted into the related component.

completed and until all the tests are successful the development can not be considered done.
This kind of tests are defined in a dedicated document, called *Unit Test Plan Document*, not available now for *myTaxiService*.

2.4.2 Subsystem Integration Sequence

In this section we will present the global interactions between the *myTaxiService*'s core subsystems and we will define all the integration tests and their order.
In the Figure 2.3 we can see the components' implementation order, as we suppose it is better: first of all the DBMS, then, in this order, all related to data checking or handling, to the users, to the security, to the rides and, finally, the dispatcher (the most important component in our system).

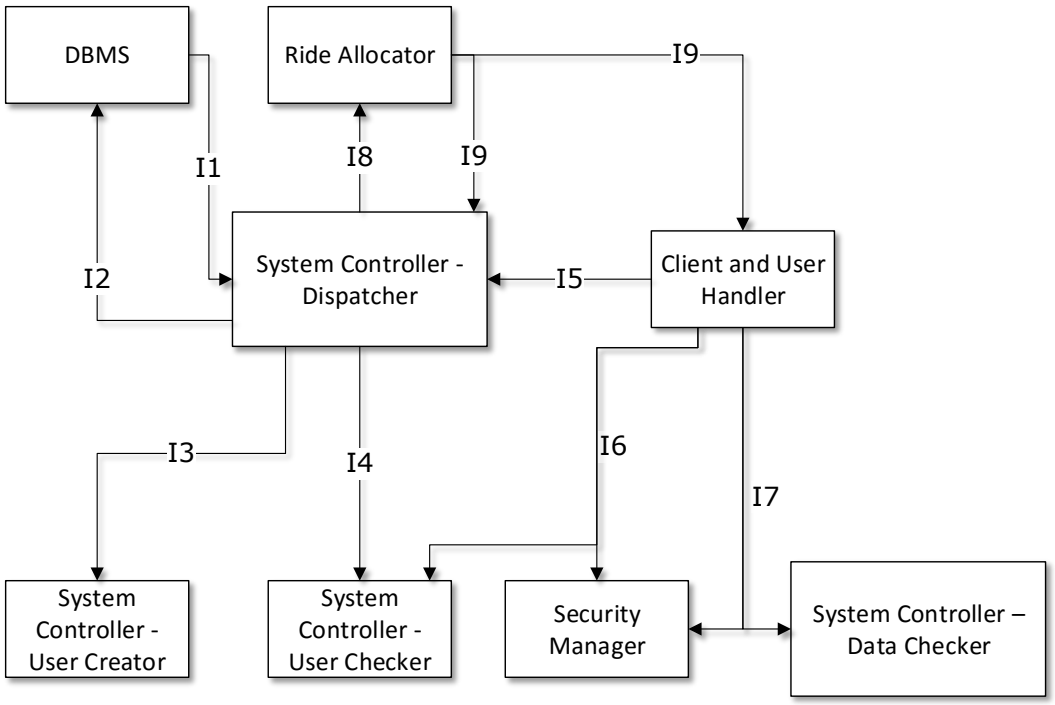

Figure 2.3: Integration Test Order

Integration tests.

ID	Integration test
I1	DBMS → System Controller - Dispatcher
I2	System Controller - Dispatcher → DBMS
I3	System Controller - Dispatcher → User Creator
I4	System Controller - Dispatcher → User Checker
I5	Client and User Handler → System Controller - Dispatcher
I6	Client and User Handler → User Checker , Security Manager
I7	Client and User Handler → System Controller - Data Checker , Security Manager
I8	System Controller - Dispatcher → Ride Allocator
I9	Ride Allocator → System Controller - Dispatcher , Client and User Handler

Chapter 3

Individual Steps and Test Description

In this chapter we are going to fully describe each test, pointing out the pre- and the post-conditions, the purposes and the environment needed.

3.1 DBMS → System Controller - Dispatcher

Test Case Identifier	I1
Components Involved	DBMS → System Controller - Dispatcher
Input Specifications	Create all possible requests by the DBMS.
Purposes	The purposes of this test are: • monitors the proper queuing of requests. • simulates a request generation.
Output Specifications	Check that the request are put into the dispatcher's queue only when they are generated.
Environment Needed	System Dispatcher Stub. DataBaseStub, to simulate the interactions between the DBMS and the database.

3.2 System Controller - Dispatcher → DBMS

Test Case Identifier	I2
Components Involved	System Controller - Dispatcher → DBMS
Input Specifications	We should generate typical data, but in this case it is sufficient to generate desired data for each DBMS interface method[1]
Purposes	The purposes of this test are: • find a data. • saves data into the database. • add/remove data from the database.
Output Specifications	We check the correctness of the searched/modified/added/removed data
Environment Needed	System Dispatcher Stub. DataBaseStub, to simulate the interactions between the DBMS and the database.

3.3 System Controller - Dispatcher → User Creator

Test Case Identifier	I3
Components Involved	System Controller - Dispatcher → User Creator
Input Specifications	None.
Purposes	We want to test the creation of each kind of user.
Output Specifications	Checks if a user is correctly created.
Environment Needed	System Dispatcher Stub. Note that the User Creator returns the created user and the dispatcher has the role to save it into the database.

[1] see the section 2.6 of the Design Document.

3.4 System Controller - Dispatcher → User Checker

Test Case Identifier	I4
Components Involved	System Controller - Dispatcher → User Checker
Input Specifications	We want to create each kind of user.
Purposes	We want to test the correct identification of the users. in the following way. First we pass a correct type of user, then a wrong type and the component should detect both cases.
Output Specifications	Check if correct result is given.
Environment Needed	System Dispatcher Stub. User Creator to create the users.

3.5 Client and User Handler → System Controller - Dispatcher

Test Case Identifier	I5
Components Involved	Client and User Handler → System Controller - Dispatcher
Input Specifications	We want to simulate all kinds of user's request.
Purposes	We test the way each request is enqueued in the dispatcher, thus the method called and the parameters specification. Then we test the following interactions for each request.
Output Specifications	Check if a request is correctly enqueued and handled in the right way.
Environment Needed	System Dispatcher Stub. Note that the exceptions or the alternative execution flows (for instance when a wrong data is inserted by the user and the system has to notify it) cannot be tested because the stub gives fixed and positive answers without performing any actions.

3.6 Client and User Handler → User Checker, Security Manager

Test Case Identifier	I6
Components Involved	Client and User Handler → User Checker , Security Manager
Input Specifications	We simulate different user's requests.
Purposes	The purpose of this group of tests is to check the correct authentication procedure, thus if for each request generated by the user, the Client and User Handler tries to identify it. The tests are several: we start from login (both user and driver) to check the shown homepage/services; then we check various request to verify if an additional check is performed before the interaction with the dispatcher.
Output Specifications	Check if the correct methods into the User Checker are called and check the results.
Environment Needed	System Dispatcher Stub to simulate the requests' queuing. User stub to simulate the interactions with the external world (both the stubs have fixed behaviour for each actions, but they are useful to be identified.).

3.7 Client and User Handler → System Controller - Data Checker , Security Manager

Test Case Identifier	I7
Components Involved	Client and User Handler → System Controller - Data Checker , Security Manager
Input Specifications	Possible inputs by users, both valid and invalid.
Purposes	The purposes are: • check the invocation of data checking procedures on each kind of input. • check the request of encryption of a password only. • check the secure connection request during a login.
Output Specifications	Checks the identification of data errors (invalid emails, names, surnames or the presence of SQL injection). Check if the encryption is called after a password insertion. Check the invocation of secure connections.
Environment Needed	User stub to simulate the interactions with the external world.

3.8 System Controller - Dispatcher → Ride Allocator

Test Case Identifier	I8
Components Involved	System Controller - Dispatcher → Ride Allocator
Input Specifications	We create a ride request into the System Dispatcher.
Purposes	We want to test if the correct methods inside the Ride Allocator are called.
Output Specifications	Checks the correct methods invocation inside the Ride Allocator, ignoring the effects of the calls.
Environment Needed	Here, the System Dispatcher has been implemented.

3.9 Ride Allocator → System Controller - Dispatcher, Client and User Handler

Test Case Identifier	I9
Components Involved	Ride Allocator → System Controller - Dispatcher , Client and User Handler
Input Specifications	We want to simulate the rides' booking.
Purposes	We want to simulate all the ride's booking procedure, both for a future ride and for a zerotime ride. In the latter case we test all the assignment procedure.
Output Specifications	Check if the correct driver is called, check the correct methods invocations and check the system answers in each phase.
Environment Needed	User and Driver stubs to simulate the requests.

3.10 System functionalities test.

Finally, in the chapter 2 we do not focus on the services handling, because the System Dispatcher is the last developed component. In each test where this component is not available, it has been simulated by an appropriate stub defined in the chapter 5.
Now, we want to check the correctness of each service. These tests are not described here because they are related only to one component functionalities, even if it interacts with some other components. Furthermore, these tests are described in the Unit Test Document, not available for *myTaxyService*.
To have an idea, with the stubs described in the chapter 5 each service will be tested independently, knowing that each other involved component works correctly[2].

[2]This is the reason why these tests are unit ones and not integration ones.

Chapter 4

Tools and Test Equipment Required

In this chapter we are going to present the tools that will be used to perform the integration test. They will be presented with their name, a link to their reference documents in the chapter 1 and a short explanation concerning the reasons why and the situations where they will be used[1].
The first one is *Mockito*, available at the website (it is a GitHub repository) `http://mockito.github.io`. This tool is especially useful to design and to implement the program stubs, described in the chapter 5, because it allows to specify predefined *answer* to each method calling with a few lines of code.

The second tool is *Arquillian*, developed to make easy the integration tests. Hence, its use will allow us to perform the integration tests.
Further information about this tool are available at its website, `http://arquillian.org`.

[1]The presented tools are all referred to a project developed in Java Enterprise Edition, whereas the Design Document is not refer to any particular development platform. This choice was made to fulfil this document showing real tools. Obviously, if the project will not be realized using Java Enterprise Edition, the presented tools cannot be used. Nevertheless, the integration tests are not related to one these tools, so they can be easily adapted to another tool.

Finally, an important test about the functionalities of *myTaxiService* is about its capacities and response times during normal or intense executions. This kind of test can be performed by using *Apache JMeter*[2] tool and with a little different client stubs w.r.t. the one defined in the chapter 5 (able to return random, but valid data instead of returning always the same ones). Despite the importance of this kind of tests, they are not argument of this document.

[2]Further information about this tool are available at `http://jmeter.apache.org`.

Chapter 5

Program Stubs and Test Data Required

This chapter is dedicated to a detailed description of each required program stub to the integration tests.

5.1 Program Stubs

The Integration Test of *myTaxyService* requires four program stubs to perform some interactions with the external environment and/or components, like the database.

5.1.1 User Simulator

The User Simulator is a stub able to act as a normal user. It offers the following methods:

- *register*, to simulate a registration with fixed data;
- *login*, to simulate a login with the same credentials defined in the registration phase;
- *profileManagement*, to simulate the management of the user's personal profile. The change is only one for each execution and it is fixed as for the

user's data.

- *zerotimeReservation*, that accepts as parameter GPS coordinates and an address. It simulate a booking of a zerotime ride. The starting position of the ride is given as in the case where the user has activated the GPS and, to be precise, that value is the one passed as parameter.
- *futureReservation*, to perform a future ride booking. For each calling of the method the departure is exactly three hours after the current time, whereas the two addresses are the same passed as parameter to the method[1].

Note that the two remaining functionalities (check the reservations and read the alerts) are not provided by this stub.

5.1.2 Driver Simulator

The Driver Simulator is a stub able to act as a taxy driver. It offers two main methods:

- *accept*, to simulate the acceptance of a System request for a ride's assignment. In each method invocation the action of accepting a ride is performed. An other version of this method, called *randomRide* can either accept or deny a ride, if someone wants to test this specific case.
- *startWaiting*, that accepts a GPS position and with this value it simulates the *Start Waiting Time* functionality.

Note that the working hours handling is not provided.

5.1.3 DataBase Stub

This stub is able to work as a real database for the other components. The methods offered by this stub allow to save and receive data as in a real interaction: in the former case the fake database perform no actions, whereas in the latter case fixed data are returned by each method. We do not point out on each

[1]Both for this method and for the previous one, the addresses are not fixed as other data. The reason for this choice is easy: we can simulate the booking with valid values, depending on the simulated city.

method definition: to have an idea see the DBMS interface into the Design Document and imagine an application of them to the rides' data.

5.1.4 Dispatcher Stub

This stub is able to act as the real System Dispatcher. The methods offered by this stub allow to enqueue the requests coming from other components and replies to the first request in the queue by simulating a fixed response for each possible method. In other words we provide the simplest answer for each method, without interacting with other components: for instance, when the Client and User Handler asks for a ride, the stubs gives a generic positive answer without really interacting with the Ride Allocator.

Chapter 6

Other Info

This chapter contains information about the used tools and the hours of work by the members of the working group.

6.1 Working hours

Date	Costanzo's hours	Disabato's hours
2016/1/9	-	1h
2016/1/10	-	1h
2016/1/13	1h	2h
2016/1/14	-	1h
2016/1/15	2h	1h
2016/1/16	1h	-
2016/1/16	3h	3h
Total ITPD	7h	9h
Global	71h	73h

6.2 Tools

For this assignment the following tools were used:

- LaTeX and TexStudio editor
- Microsoft Visio 2013

6.3 Changes (w.r.t. version 1.0)

In this release two changes have been made to correct a few errors: first of all, in the chapter 4 we have added a footnote to clarify that the project has not been thought for a particular development platform.
Second, a section has been re-written to make it more coherent w.r.t. the whole document.