Politecnico di Milano
Software engineering 2

myTaxiService

Integration Test Plan Document (ITPD)

Version 1.0
Release date 21st january 2016

790021 Antenucci Sebastiano
852461 Buonagurio Ilaria

Prof.ssa Di Nitto Elisabetta

Table of contents

1 Introduction

1.1 Revision history

At the actual state of things, version 1.0 released the 21st of January of 2016, there were no changes to the Integration Test Plan Document.
This is the first release of this document.

1.2 Purpose and scope

1.2.1 Purpose

The purpose of this document is to describe the plans for the creation of the Integration Test of the system. In particular it focuses on the Integration Test of the components described in the Design Document.
The Integration Test Plan Document is addressed to all the people who contribute to the Integration Test.

1.2.2 Scope

Foremost the aim of the project is to optimize the taxi service of a large city and in particular it is to simplify passengers' access to the service and to guarantee a fair management of the queues by creating a new software called myTaxiService.
Users can request a reservation through a web or mobile application and they are notified by the system with the code of the incoming taxi and the waiting time.
There are a two kind of reservation: the reservation, made at least two hours in advance of the meeting time by specifying the departure and arrival of the ride; or the "quick" one, by specifying only the origin of the ride. In the second case is supposed that the taxi is requested immediately.
In support to taxi drivers there is a mobile application that allows them to inform the system about their availability and they can also confirm or dismiss a certain call.
The system also simplifies queues management and taxi distribution on the territory, in order to make the

service more efficient and to serve a larger number of passengers.
Since the system must be reliable, available, maintainable, secure and portable it is important to specify an integration test plan for the system before actually write the code of the system.

1.3 Definitions and abbreviations

In this section we provide an useful glossary in order to better understand this documents. In particular the definitions are the ones directly linked to the project, as we assume that the readers of this document are able programmers.

1.3.1 Definitions

- Quick call: the users can only specify the origin of the ride and is supposed that the taxi is request immediately

- Reservation: the users can do this reservation at least two hours in advance of the meeting time by specifying the departure and arrival of the ride

- Taxi driver: a person whose job is to drive a taxi. It works for the government of the city

- Drivers' reserved mobile application: an application that is given to taxi drivers from the taxi company. It allows the driver to assert their disponibility and can answer to a call: in a positive way, by accepting the request, or in a negative way by dismissing it.

- Taxi identification code: each taxi has an unique identification code written on each backdoor that allows the user to easily find their own taxi.

- Reservation number: unique id that represent the user reservation.

- Driver's calls list: it contains all the drivers call up to a month before the date of the visualization. It shows the reservation number, the date, the departure location and time and the arrival location and time of the call.

- User's rides history: it lists all the user's reservations and quick call, ordered by date. It shows the reservation number, the taxi identification code, the price, the date, the departure location and time and the arrival location and time of the call.

- Taxi driver's next call: it is the call that the driver has accepted but has not accomplished. There can only be on next call at the time, so that a driver can not get more than one call at the time.

- User's last unfulfilled request: it is the last request that the user has made that has not been fulfilled. In other words it is a quick call or reservation in which the taxi driver has not picked up the user yet.

- Notifications: pop up messages shown to the user or to the driver in order to give them information about rides.

1.3.2 Abbreviations

RASD : Requirements Analysis and Specifications Document

DD : Design Document

ITPD: Integration Test Plan Document

1.4 Reference documents

- Specification Document: Assignments 1 and 2 (RASD and DD).pdf and Assignment 4 (integration test plan).pdf.

- The integration test plan example provided

- RASD - S. Antenucci, I.Buonagurio

- DD - S. Antenucci, I.Buonagurio
- Slides from Software Engineering 2 course of Politecnico di Milano, academic year 2015/2016.
- http://arquillian.org/guides/
- http://site.mockito.org/mockito/docs/
- http://junit.org/cookbook.html

All documents can be found at:

https://github.com/sebastianoantenucci/AntenucciBuonagurio

https://beep.metid.polimi.it/web/3343933/

2 Integration strategy

2.1 Entry criteria

All the public classes must be tested in JUnit before the Code Inspection. In particular the lines of code must be covered at least at the 90% in order to discover major issues in the code, facilitate changes and guiding the design.
However unit testing is not part of this document and will be performed independently, in order to ensure that dependencies work independently from each other.
Moreover, in order to simplify the testing, the JavaDoc of all the classes, methods, interfaces and files, must be up to date and complete.

2.2 Elements to be integrated

As stated in the Design Document the subsystems that need to be considered in the integration test are:

- Client tier: contains web application and mobile application for user and taxiDriver;
- Web tier: contains the web pages servlet which will be used from the web application; it has the task to handle client request to the server and to generate response;
- Business tier: will be used either for the app and web application; this layer manage the user session, keeping him logged in into the server while using the application;
- EIS tier: represent the database part (DBMS) which contains the user and taxi driver data, it contains also all the information that allow the system to manage the queue, the taxi request and distribution on the territory.

In particular, first of all we need to test the integration of each subsystem (as the testing of different components in the same subsystem) and then the relation between subsystems.

2.3 Integration testing strategy

We are dealing with integration testing, so we need to test the interfaces and modules interaction. We can use black-box testing since the specifications will be usually smaller that the code and they will help identifying missing functionalities in the system. Moreover, black-box testing is suitable for our integration test because the models used in specification and design have a structure and we can devise test cases to check the actual behavior specified by the model.
We have excluded a-priori the Big Bang integration test due to its copious issues.
We chose a structural orientated testing, even if it requires more planning.
As stated in point 2.1, we assume to already have the unity testing, and so we could select between bottom-up or sandwich strategies.
The elected testing approach is the bottom-up one, since the sandwich one is flexible and adaptable, but more complex to plan rather than integrating subsystems, as requested in the bottom-up approach.

2.4 Sequence of component / function integration

We decided to test the components and the subsystems from the less independent one to the most independent one. By doing so we can provide the testing of the needed parts to the needing one because its test has already been done. Moreover with this approach we can delete the dependence problems and avoid the creation of useless stubs.

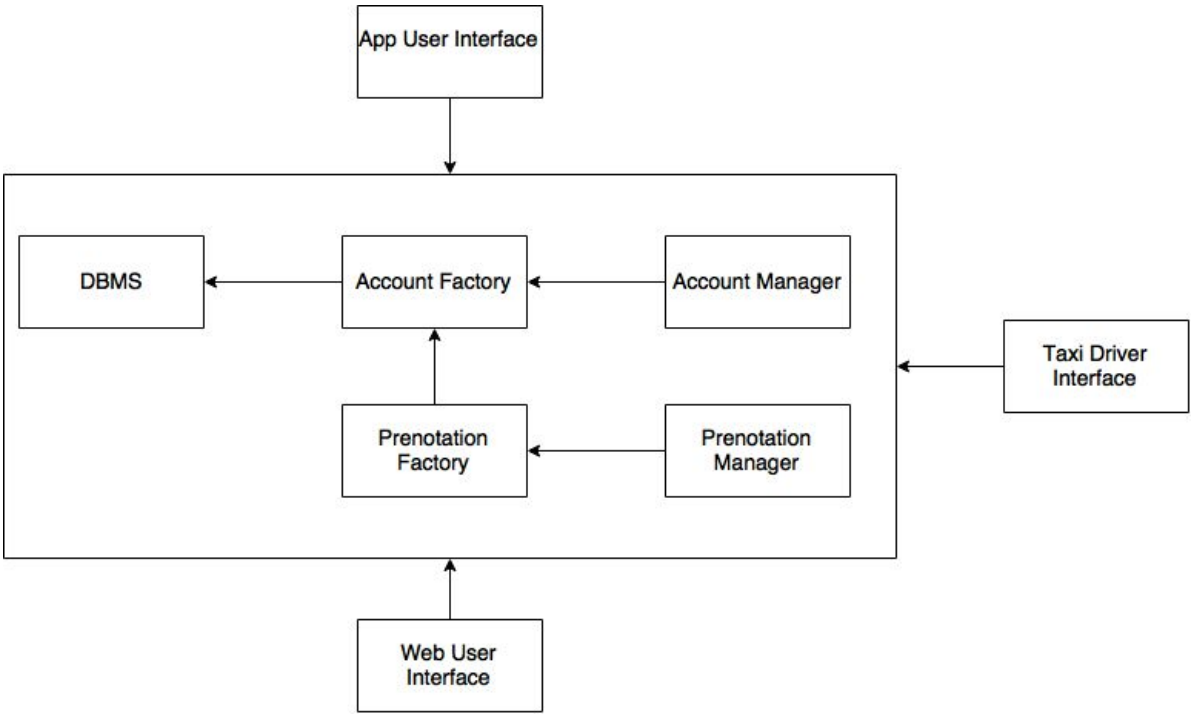
2.4.1 Software integration sequence

Here is presented the integration sequence we chose for the integration of the components that compose each subsystem.

When we planned the integration sequence we used the logic of integrating first the components with less dependencies and last the one with the most number of dependencies for each subsystem. By doing so, we managed to easily integrate the components because we respected the dependencies between components.

- EIS tier: we don't need to test it, we assume that it works correctly and we only test the interaction with other components
- Business tier: we tested in order: account factory, account manager, prenotation factory, prenotation manager.
- Web tier: we tested how it responds to the interaction with the client and with other subsystems and in particular we tested in the order: web user interface, app user interface, taxi driver user interface.
- Client tier: we tested the application on each kind of device. This test is done manually and is the last one to be performed.

ID	***Component***	***Dependence***
C1	Database	N/A
C2	Account manager	C3
C3	Account factory	C1
C4	Prenotation factory	C1
C5	Prenotation manager	C5
C6	Web user interface	C2, C3, C4, C5
C7	App user interface	C2, C3, C4, C5
C8	Taxi driver user interface	C5


App User Interface
DBMS
Account Factory
Account Manager
Taxi Driver Interface
Prenotation Factory
Prenotation Manager
Web User Interface

2.4.2 Subsystem integration sequence

Here is presented the integration sequence we chose for the integration of the subsystems.
When we planned the integration sequence we used the logic of integrating first the subsystem with less dependencies and last the one with the most number of dependencies. By doing so, we managed to easily integrate the subsystems because we respected the dependencies between subsystems.
First of all we decided to integrate the EIS tier, that represents the DBMS, since it does not depend from any other subsystem.
Then, we decided to integrate the business tier, that represents the applicative logic, since it only depends from the EIS tire.
After that we decided to integrate the web tier, since the client tier depends from it and so has to be integrated as the last one.

ID	*Subsystem*	*Dependence*
S1	EIS tier	N/A
S2	Business tier	S1
S3	Web tier	S1, S2
S4	Client tier	S1, S2, S3

3 Individual steps and test description

In this section we will deal with the description of each test case. The test cases are presented in 2.4.1 and 2.4.2 and in this section are treated more in detail.

Components

Test Case Identifier	TC1
Test Item(s)	Account manager → Database
Input Specification	Managing an account
Output Specification	Check if the tuple related to the account managed is correctly changed
Environmental Needs	TC2 passed

Test Case Identifier	TC2
Test Item(s)	Account factory → Database
Input Specification	Creation of an account
Output Specification	Check if the tuple related to the account is correctly created
Environmental Needs	DBMS correctly working

Test Case Identifier	TC3
Test Item(s)	Prenotation factory → Database
Input Specification	Creation of a prenotation
Output Specification	Check if the tuple related to the prenotation is correctly created
Environmental Needs	DBMS correctly working

Test Case Identifier	TC4
Test Item(s)	Prenotation manager → Database
Input Specification	Managing a prenotation
Output Specification	Check if the tuple related to the prenotation managed has changed
Environmental Needs	TC3 passed

Test Case Identifier	TC5
Test Item(s)	Web user interface → Account factory
Input Specification	Creation of an account
Output Specification	Check if the account is correctly created
Environmental Needs	TC2 passed

Test Case Identifier	TC6
Test Item(s)	App user interface → Account factory
Input Specification	Creation of an account
Output Specification	Check if the account is correctly created
Environmental Needs	TC2 passed

Test Case Identifier	TC7
Test Item(s)	Web user interface → Account manager
Input Specification	Manage an account data
Output Specification	Check if the account is correctly modified
Environmental Needs	TC1, TC5 passed

Test Case Identifier	TC8
Test Item(s)	App user interface → Account manager
Input Specification	Manage an account data
Output Specification	Check if the account is correctly modified
Environmental Needs	TC1, TC6 passed

Test Case Identifier	TC9
Test Item(s)	Web user interface → Prenotation factory
Input Specification	Creation of a quick call
Output Specification	Check if the quick call is correctly created
Environmental Needs	TC3 passed

Test Case Identifier	TC10
Test Item(s)	App user interface → Prenotation factory
Input Specification	Creation of a quick call
Output Specification	Check if the quick call is correctly created
Environmental Needs	TC3 passed

Test Case Identifier	TC11
Test Item(s)	Web user interface → Prenotation factory
Input Specification	Creation of a reservation
Output Specification	Check if the reservation is correctly created
Environmental Needs	TC3 passed

Test Case Identifier	TC12
Test Item(s)	App user interface → Prenotation factory
Input Specification	Creation of a reservation
Output Specification	Check if the reservation is correctly created
Environmental Needs	TC3 passed

Test Case Identifier	TC13
Test Item(s)	Prenotation manager → Prenotation factory
Input Specification	Managing the newly created ride
Output Specification	Check if the correct methods are called in prenotation manager
Environmental Needs	TC3, TC4, TC9, TC10, TC11, TC12 passed

Test Case Identifier	TC14
Test Item(s)	Web user interface → Prenotation manager
Input Specification	Managing a ride and notification
Output Specification	Check if the ride is correctly managed and the notification created and delivered
Environmental Needs	TC3, TC4, TC9, TC10, TC11, TC12, TC13 passed

Test Case Identifier	TC15
Test Item(s)	App user interface → Prenotation manager
Input Specification	Managing a ride and notification
Output Specification	Check if the ride is correctly managed and the notification created and delivered
Environmental Needs	TC3, TC4, TC9, TC10, TC11, TC12, TC13 passed

Test Case Identifier	TC16
Test Item(s)	Taxi driver user interface → Prenotation manager
Input Specification	Managing a ride and notification
Output Specification	Check if the ride is correctly managed (accepted or discarded) and the notification created and delivered
Environmental Needs	TC3, TC4, TC9, TC10, TC11, TC12, TC13 passed

Subsystems

Test Case Identifier	TC17
Test Item(s)	Business tier → EIS tier
Input Specification	Interaction of the system with the database
Output Specification	Check if the EIS tier is correctly invoked
Environmental Needs	N/A

Test Case Identifier	TC18
Test Item(s)	Web tier → EIS tier
Input Specification	Interaction of the web application with the database
Output Specification	Check if the EIS tier is correctly invoked
Environmental Needs	TC17, TC20 passed

Test Case Identifier	TC19
Test Item(s)	Client tier → EIS tier
Input Specification	Interaction of the client application with the database
Output Specification	Check if the EIS tier is correctly invoked
Environmental Needs	TC17, TC18, TC20, TC22 passed

Test Case Identifier	TC20
Test Item(s)	Web tier → Business tier
Input Specification	Interaction between the web application and the system
Output Specification	Check if the systems responds correctly
Environmental Needs	N/A

Test Case Identifier	TC21
Test Item(s)	Client tier → Business tier
Input Specification	Interaction between the client application and the system
Output Specification	Check if the systems responds correctly
Environmental Needs	TC22, TC20 passed

Test Case Identifier	TC22
Test Item(s)	Client tier → Web tier
Input Specification	Interaction between the client application and the web application
Output Specification	Check if the web application responds correctly
Environmental Needs	N/A

4 Tools and test equipment required

In order to perform the test presented in section 2 and 3, we will use:

- http://arquillian.org
  Arquillan is an integration test framework that can be used to perform testing inside a remote or embedded container, or deploy an archive to a container so the test can interact as a remote client.
  We will use it for the integration test of subsystem, together with JUnit and Mockito.

- http://junit.org/ unit test
  JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.
  We will use it for the unit test of components and integration test of subsystems,togheter with Mockito and Arquillan.

- http://site.mockito.org
  Mockito allows you to create and configure mock objects. Using Mockito simplifies the development of tests for classes with external dependencies significantly. It is an open source testing framework for Java released under the MIT License that allows the creation of mock objects, stubs and drivers in automated unit tests.
  We will use it for the unit test of components and the integration test of subsystem, together with JUnit and Arquillan.

- Manual test: some test can not be performed by automated tool. For example the testing of the client application or driver application on devices. For this kind of test we will rely on manual test.

5 Program stubs and test data required

According to the integration test strategy and design proposed in this document, we will not deal with Big Bang testing, so we will test the system when it is not completed.
In order to correctly execute the integration test, we will need some stubs in order to simulate project parts that are not already built.
In particular, to better manage the testing, we can assume to have a "light" system: few users that perform few operations. This means to also have a smaller DBMS, still structured as the final one though.
For the testing of the Business tier and Web tier without the Client tier we can use a stub. Another stub will be used in order to test the Business tier when the Web tier is not completely developed.