POLITECNICO
MILANO 1863

Integration Test Plan Document

Software Engineering 2: “myTaxiService”

Fioratto Raffaele, Longoni Nicolò
Politecnico di Milano

A.Y. 2015/2016

Version 1.0

January 21, 2016

Contents

1 Introduction

1.1 Revision History

Version	**Date**	**Author(s)**	**Summary**
0.1	01/08/2016	Fioratto Raffaele	Document Creation
0.2	01/09/2016	Fioratto Raffaele	Added Introduction
0.2.5	01/11/2016	Fioratto Raffaele	Added Strategy (to be completed)
0.3	01/15/2016	Fioratto Raffaele	Completed Section 4
0.3.5	01/17/2016	Fioratto Raffaele	Test Plan Skeleton
0.4	01/18/2016	Longoni Nicolò	Completed section 1
0.5	01/20/2016	Longoni Nicolò	Completed section 2
0.6	01/20/2016	Longoni Nicolò	Completed section 3
0.7	01/20/2016	Fioratto Raffaele	Program Stubs Added
0.8	01/20/2016	Fioratto Raffaele	Completed section 5
0.9	01/21/2016	Fioratto Raffaele and Longoni Nicolò	Various corrections
1.0	01/21/2016	Fioratto Raffaele and Longoni Nicolò	Final version to deliver

1.2 Purpose

The purpose of an Integration Test Plan (ITP) is to describe necessary tests to verify that all the components of a software are properly assembled and work together. This document explains thoroughly tests to perform, why they have been chosen, in which order they must be executed and their expected result. The Integration Test is one of the steps involved in the Verification & Validation phase during Software Design and Development process. At this level, components of a software are treated like *black-box*es (this is in contrast to unit testing in which components are treated like *white-box*es) and tests are crafted so that components are able to interact reciprocally. Each one of these tests has a precise purpose and focuses on a particular result that must be achieved in order to consider the test passed.

1.3 Scope

The aim of this project is to create a new brand system that optimizes an existing taxi service. The system will be capable of automatizing and/or simplifying certain processes during requests or reservations of taxis. It will also guarantee a fair management of taxi queues. New passengers can sign up for services, inserting some basic information in order to use features as soon as possible. A passenger can request a taxi using web service or mobile application after registration. The system will be able to localize precisely his/her position, determining taxis that are available near him/her. The system will select a taxi and then it will forward the request to one of its drivers. Upon confirmation, the system will notify the customer about the successful completion of the operation and the ETA of the taxi. A passenger can also reserve a taxi by specifying time and date, but he/she has to do it with at least two hours in advance. Cancellation is also permitted. The system will actually process the request ten minutes before the time specified during the reservation in the same way as described previously.

1.4 Definitions and Abbreviations

- ITP: **I**ntegration **T**est **P**lan
- DD: **D**esign **D**ocument
- RASD: **R**equirement **A**nalysis and **S**pecification **D**ocument
- IDE: **I**ntegrated **D**evelopment **E**nvironment
- JEE: **J**ava **E**nterprise **E**dition
- EJB: **E**nterprise **J**ava **B**ean
- Sys: System
- Admin: Administrator

1.5 Reference Documents

- myTaxiService Project Specification Document
- myTaxiService RASD
- myTaxiService DD
- Arquillian Documentation[1]
- JUnit Documentation[2]
- Mockito Documentation[3]

[1] https://docs.jboss.org/author/display/ARQ/Reference+Guide
[2] http://junit.org/javadoc/latest/
[3] http://mockito.github.io/mockito/docs/current/org/mockito/Mockito.html

2 Integration Strategy

This section describes decisions and criteria selected for our ITP regarding the "myTaxiService" application. These ideas are fundamentals since they will be used extensively throughout the ITP document and they play a central role in its construction.

2.1 Entry Criteria

Before actually performing an Integration Testing there are some prerequisites we think that should be met.
Some items depend on a selected strategy that is used for the design of tests and they will be described in detail later on.
The list is composed as follows:

- Definition of dependencies between components.
- Proper internal method unit tests of each component and also support methods (white-box testing).
- Implementation of stubs for internal and external components that have not been implemented yet.
- Definition of input and expected output for each test.
- Definition of I/O functions, i.e. external interface methods, for each component (already described in the DD).

2.2 Elements to be Integrated

The following image illustrates all the components defined in the DD that are going to be tested for integration.

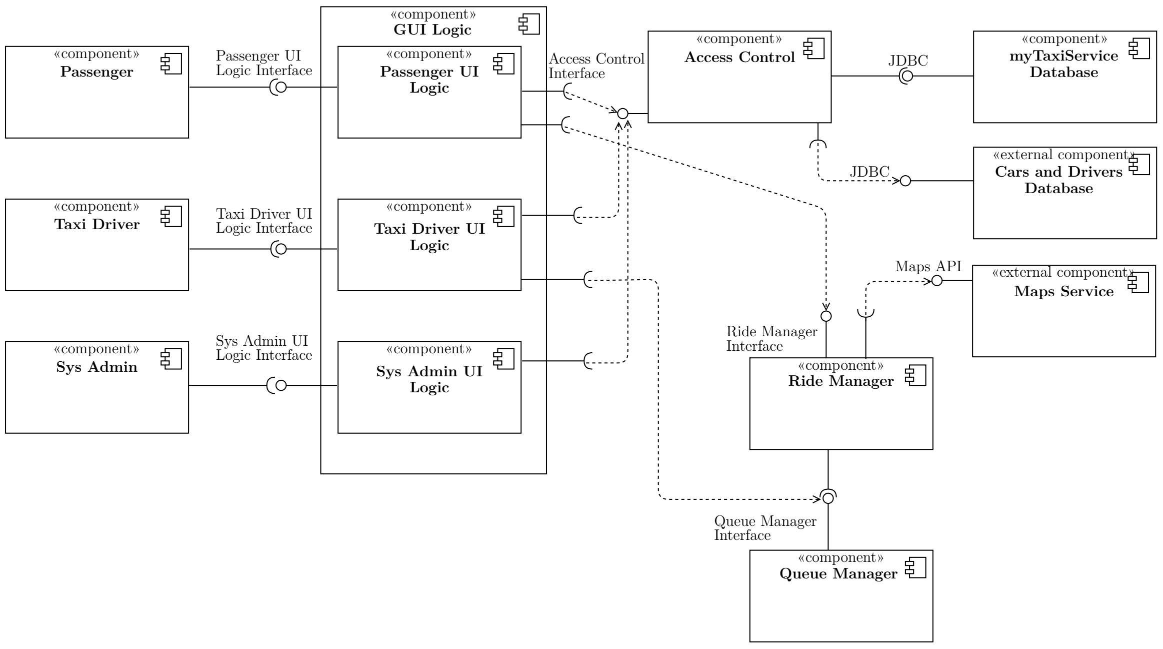

Figure 1: Component View

You can see from this picture that there are dependencies between components, i.e. which component uses methods from another external interface. This is a key concept that will guide the entire design and implementation of our ITP and it is described in later sections.

2.3 Integration Testing Strategy

The strategy chosen that will drive definition and design of Integration Tests is the top-down approach.
With this method, a hierarchy is defined such as the components with no dependences are tested first, whereas the ones that are expected to interact the most are the last to be tested. The rest of components falls in between of these two extreme categories and they are tested properly as soon as all modules that they depend on are carefully examined and tested.
If other modules have not developed yet, or to break possible dependency cycles, stubs are defined and created accordingly to implement the same interfaces by the classes that will be used in production and return canned values upon method invocations. Using this principle, we can fake the behavior of a component and focusing only on actual module to be tested, ensuring the correct interaction with its dependent components. Other benefits from using the top-down approach are:

- Advantageous if major flaws occur toward the top of the program.
- Once external interface methods are defined, representation of test cases is easier.

- Early skeletal Program allows demonstrations and boosts morale.

Of course there are also possible disadvantages in using such approach. Typically they are:

- Stub modules must be produced
- Stub modules are often more complicated than they first appear to be.
- Before external interface methods are defined, representation of test cases in stubs can be difficult.
- Test conditions may be impossible, or very difficult, to create.
- Observation of test output is more difficult.
- Allows one to think that design and testing can be overlapped.
- Induces one to defer completion of the testing of certain modules.

2.4 Sequence of Component/Function Integration

With the chosen strategy, in this section we define the actual sequence of components, based on their dependencies.
That implies the order of integration tests that should be followed and also it will guide implementation of the actual testing plan.

2.4.1 Software Integration Sequence

Since our system is not so complex we can consider a single subsystem comprising all the components so we focus only on designing a sequence for integrating modules.

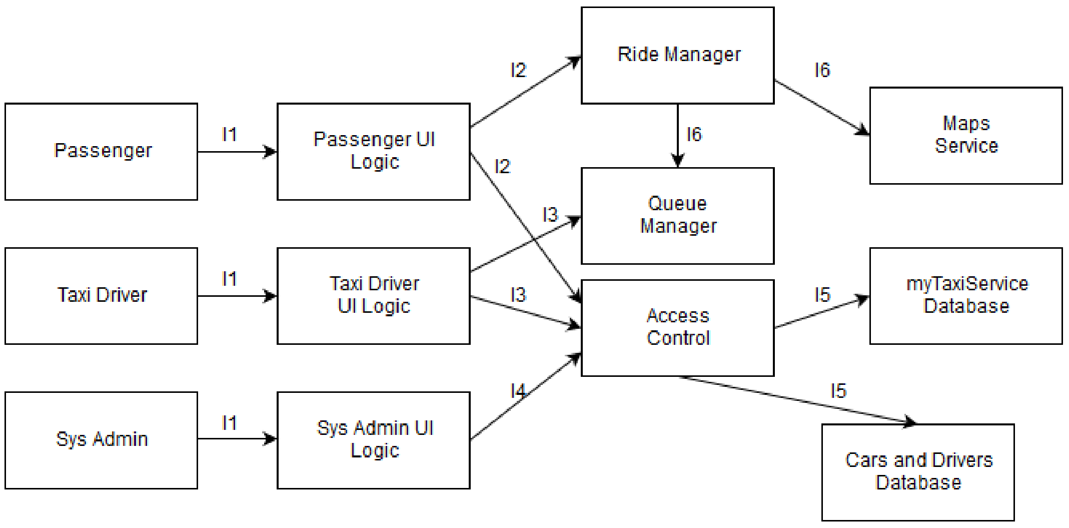

Figure 2: Integration Test Order

Starting from GUI parts because they don't interface themselves with other components. The criteria we used to choose test order is the number of components that use an interface: a part used by a certain amount of components is tested before than another that is used by less elements.
It is important to notice that as soon as a component is implemented, it will be tested for integration regardless of the implementation state of the components that it depends on, if necessary proper stubs will be created.

3 Individual Steps and Test Description

3.1 Integration test case I1

Test Case Identifier	I1T1
Test Item(s)	Passenger→Passenger UI Logic
Purpose	To control if all commands coming from a Passenger are correctly processed and all inputs send a correct request to the logic part corresponding to the one designed for
Input specification	Sequence of actions performed manually to trigger all features of Passenger's GUI Application
Output specification	Functionalities coming from the UI component consistent with requests made by passengers or an error messange in case of error
Environmental needs	Stub of Passenger UI Logic created

Test Case Identifier	I1T2
Test Item(s)	Taxi Driver→Taxi Driver UI Logic
Purpose	To control if all commands coming from a Taxi Driver are correctly processed and all inputs send a correct request to the logic part corresponding to the one designed for
Input specification	Sequence of actions performed manually to trigger all features of Taxi Driver's GUI Application
Output specification	Functionalities coming from the UI component consistent with requests made by taxi drivers or an error messange in case of error
Environmental needs	Stub of Taxi Driver UI Logic created

Test Case Identifier	I1T3
Test Item(s)	Sys Admin→Sys Admin UI Logic
Purpose	To test if all commands coming from a Sys Admin are correctly processed and all inputs send a correct request to the logic part corresponding to the one designed for
Input specification	Sequence of actions performed manually to trigger all features of Sys Admin's GUI Application
Output specification	Functionalities coming from the UI component consistent with requests made by administrator or an error messange in case of error
Environmental needs	Stub of Sys Admin UI Logic created

3.2 Integration test case I2

Test Case Identifier	I2T1
Test Item(s)	Passenger UI Logic→Access Control
Purpose	To test if all elaborated inputs coming from the GUI part are correct managed, sent in the right order and all responses coming from Access Control component are consistent with their requests
Input specification	Requests coming from GUI methods
Output specification	Information requested to Access Manager or a series of operations to do in case of error
Environmental needs	I1T1 passed, Stub of Access Control created

Test Case Identifier	I2T2
Test Item(s)	Passenger UI Logic→Ride Manager
Purpose	To test if methods of this component make a correct request of ride modifications and the answers coming from Ride Manager are in the right order and consistent
Input specification	All requests elaborated by the Passenger UI Logic part that imply a modification of rides
Output specification	A positive validation of modifications happend in Ride Manager to requests coming from Passenger UI Logic or a negative one if an error occurs
Environmental needs	I1T1 passed, Stub of Ride Manager created

3.3 Integration test case I3

Test Case Identifier	I3T1
Test Item(s)	Taxi Driver UI Logic→Access Control
Purpose	To test if all elaborated inputs coming from the GUI part are correct managed, sent in the right order and all responses coming from Access Control component are consistent with their requests
Input specification	Requests coming from GUI methods
Output specification	Information requested to Access Manager or a series of operations to do in case of error
Environmental needs	I1T2 passed, Stub of Access Control created

Test Case Identifier	I3T2
Test Item(s)	Taxi Driver UI Logic→Queue Manager
Purpose	To test if Taxi Driver UI Logic inputs that modify queues, i.e. elimination from a queue, insertion in another one and so on, use methods correctly written and answers arriving from Queue Manager are consistent with requests
Input specification	All requests elaborated by the Taxi Driver UI Logic part that imply a modification of queues
Output specification	A positive response about changings happened in Queue Manager or the launch of an error recovery procedure
Environmental needs	I1T2 passed, Stub of Queue Manager created

3.4 Integration test case I4

Test Case Identifier	I4T1
Test Item(s)	Sys Admin UI Logic→Access Control
Purpose	To test if all elaborated inputs coming from the GUI part are correct managed, sent in the right order and all responses coming from Access Control component are consistent with their requests
Input specification	Requests coming from GUI methods
Output specification	Information requested to Access Manager or a series of operations to do in case of error
Environmental needs	I1T3 passed, Stub of Access Control created

3.5 Integration test case I5

Test Case Identifier	I5T1
Test Item(s)	Access Control→myTaxiService Database
Purpose	To test if Access Control component makes correct requests that are able to modify users' profile and data obtained from myTaxiService Database are consistent
Input specification	Request coming from various users, for example registration or login
Output specification	A modification in myTaxiService Database, a set of data returned to Access Control or the activation of a procedure after an error
Environmental needs	I2T1, I3T1 and I4T1 passed, Stub of myTaxiService Database created

Test Case Identifier	I5T2
Test Item(s)	Access Control→Cars and Drivers Database
Purpose	To test if Access Control component is able to verify correspondence between data in myTaxiService Database and Cars and Drivers Database
Input specification	Requests about cab and driver profiles
Output specification	A validation about matchings or a signal of inconsistency of data in these two component. In case of error, a specific procedure that can manage it
Environmental needs	I2T1, I3T1 and I4T1 passed, Stub of Cars and Drivers Database created

3.6 Integration test case I6

Test Case Identifier	I6T1
Test Item(s)	Ride Manager→Queue Manager
Purpose	To test if Ride Manager methods are able to ask correctly to the Queue Manager a creation, modification or removal of a taxi driver from a queue
Input specification	Ride modifications that imply a changing of taxi driver status in a queue
Output specification	A positive acknoledgment about correct modifications occurred in Queue Manager after they have been requested by Ride Manager or an error message
Environmental needs	I2T2 passed, Stub of Queue Manager created

Test Case Identifier	I6T2
Test Item(s)	Ride Manager→Maps Service
Purpose	To test if Ride Manager can send a correct request to Map Service asking positions of rides and answers are consistent
Input specification	Requests coming from Ride Manager about departures and arrivals of taxi rides
Output specification	Map positions requested by Ride Manager or an error massage
Environmental needs	I2T2 passed, Stub of Maps Service created

4 Tools and Test Equipment Required

This section describes tools and techniques required in order to do all tests described in the previous sections. Moreover, all tests require additional testing equipment, thus these sections list them and motivate why they have been done.
Here there are the tools needed in order to actually perform the integration testing.

4.1 Tools

4.1.1 Arquillian

Arquillian is an integration testing tool that facilitates test writing which targets a JEE server. It basically instantiates a EJB container and injects necessary beans in order to perform a requested test, moreover, it has the ability to deploy tests directly to a JEE server. Since the back-end of our application is implemented using JEE, this tool is extremely useful and moreover, all tests can be run directly from IDE, so this enables all the benefits of testing from an IDE simplifying also debugging.

4.1.2 JUnit

The majority of tests described here can be automatized using JUnit (this is true also for unit testing), so this is another important tool that can dramatically speed up the overall testing process.
Like Arquillian, JUnit is perfectly integrated in a Java IDE and furthermore the former tool is actually designed to be tightly coupled with the latter. Eventually people who are in charge of testing have a very comfortable and powerful environment to work with, and they can only concentrate on testing instead of being worried about something else.

4.1.3 Mockito

Mockito is a tool that simplifies the creation of mock classes during tests. These are useful when you want to test lines of code already written, but they depend on some other code that is missing because it hasn't been implemented yet. This tool is more useful during unit testing rather than in integration one, but you can reuse mocks created in unit testing phase as stubs for this phase.

4.1.4 Manual Testing

Some components of the system are difficult to test using an automated approach, so there are a few cases in which manual testing comes into play.
One of the possible examples is GUI testing: it is very complex to test automatically and indeed it is easier to do it using direct interaction between humans who can provide additional feedbacks and possible improvements of the interface in order to enhance the overall end-user experience.

4.2 Testing Equipment

4.2.1 Testing Environment

During the Integration Test phase, actual testing will be performed in a specific environment called Testing Environment that is different from Production one. A reason for doing this thing is to have easier and controlled facilities in order to neglect some possible issues that can arise during initial stages of Integration Testing.
Of course we should also consider testing in the final environment but this will be done later in more advanced phases, so overall we have a more step-by-step approach. Another benefit coming from this choice is, in general, a speed increase of the overall process.
Conversely there are of course possible drawbacks with this method: one of the most important to consider is less power of Testing Environment in respect to the Production one, this might not highlight any issues that eventually will be unnoticed during this phase. These can be hopefully discovered during final testing or in worse cases, after the actual final deployment of the software ready to be distributed.

4.2.2 Back-end

For the back-end, some PCs will be set up with the software and hardware already described in the RASD, furthermore IDEs will be also installed in order to enable testing on these machines. The structure of the overall system will be simple in accordance with what is described in previous section, but it will replicate at least the n-tier structure described in the DD to have a system that closely resembles the production one.
The software will be up-to-date in order to have the most recent and stable versions for each of them.

4.2.3 Front-end

Since the front-end of the application is designed to be showed from mobile phones, it is necessary they accomplish tasks regarding testing Passenger, Taxi Driver and Sys Admin components that interact with the GUI Logic as well.
We think that at least three different models for each Mobile OS should be taken into account: in the RASD we defined the minimum version for each Operating System to run the mobile application, thus we should check the actual compatibility of the app with those versions.
Moreover we expect that the application should run flawlessly also with the most up-to-date versions for both Android and iOS, so other smart-phone models that executes these Operating System versions should be used to test the application.
At last, even if it is not mandatory, we should check the correct application behavior with version that fall in between this two categories, since a good percentage of users doesn't usually execute on their phone the last available version of a specific Operating System.

5 Program Stubs and Test Data Required

5.1 Program Stubs

The following table summarizes stubs used throughout the ITP.

Stub	**Test(s) Involved**	**Description**
Passenger UI Logic	I1T1	Upon methods calling, it returns sample views and instructions to perform requested operations. It also provides fake notifications to Passenger that are supposed to be generated by back-end components.
Taxi Driver UI Logic	I1T2	It shows sample views and notifications to Taxi Driver.
Sys Admin UI Logic	I1T3	It shows sample views and messages to System Administrator.
Access Control	I2T1, I3T1, I4T1	It returns sample responses upon sign in, sign up of both Passenger and Taxi Driver, also login for System Administrator.
Ride Manager	I2T2	It returns sample notifications to the front-end about new rides freshly assigned or correctly reserved. Notifications for exceptional conditions (such as multiple declines of a same request) are also provided.
Queue Manager	I3T2, I6T1	It provides fake free taxis to be assigned to an incoming ride and also sample notifications of requests supposed to be declined by a Taxi Driver.
myTaxiService Database	I5T1	It returns sample data supposed to be stored in the database, for instance Passengers already registered and Taxi Drivers.
Cars and Drivers Database	I5T2	It returns sample data supposed to be stored in the database, for instance valid Taxi Drivers licenses.
Maps Service	I6T2	It returns fake Passenger and Taxi Driver locations and sample ETAs as well.

5.2 Test Data Required

In order to test correctly some features it is mandatory to have data that should be managed by the system during Integration Testing.
Data must be already present in our internal database, i.e. some passengers and drivers

already registered and a system administrator as well in order to test login features and access right management.

Appendices

A Tools

- Document written in LaTeX
- Basic MiKTeX 2.9.5721 64-bit – `http://miktex.org/`
- Texmaker 4.5 – `http://www.xm1math.net/texmaker/index.html`
- draw.io – `https://www.draw.io/`

B Hours of Work

- Raffaele Fioratto: 21 hours
- Nicolò Longoni: 19 hours