Integration Test Plan Document

Alessandro Comodi, Davide Conficconi, Stefano Longari

January 21, 2016

Contents

Chapter 1

Introduction

1.1 Revision History

This section is about all the revisions that has been done to this document.

Version 1.0	First release of the document (21/01/2016)

1.2 Scope & Purpose

The Integration Test Plan Document (ITPD) is intended as a set of directives and guidelines on how to successfully test the interactions among the components described in the Design Document (DD). It includes the strategies and the various steps to be adopted and followed by the testers and developers.

1.3 Definitions & Abbreviations

Each integration test case reported in this document has a specific identification string that will be explained in the lines below.

The first letter stands for the test group to which the integration test belongs:

- G_ : General, intended as the test among the various subsystems.
- C_ : Communication, intended as the group of tests belonging to the Communication subsystem.
- A_ : Application, intended as the group of tests belonging to the Application subsystem.
- D_ : Data, intended as the group of tests belonging to the Data subsystem.

The letter "I" followed by a number is the integration test belonging to a certain group.

Lastly there is the letter "T" followed by a number that indicates the number of the test belonging to a specific integration that has to be tested.

An example of case test identification string is: G_I2T1. It means that the test is the first one of the second integration on the General system.

1.4 Reference Documents

The reference documents that have been used in order to redact the ITPD are:

- RASD v1.0 (Requirements Analysis and Specifications Document);
- DD v1.0 (Design Document);
- Software Engineering 2 Project, AA 2015/2016 Assignments 4 – Test Plan;
- Documentation regarding Mockito, on the relative website: http://site.mockito.org/
- Documentation regarding Arquillian, on the relative website: http://arquillian.org/

Chapter 2

Integration Strategy

2.1 Entry criteria

The Integration Test is an important step for the verification and validation of the software developement, and, in order to complete it successfully there are several criterias that have to be met before the Integration Test starts.

- All the Unit Tests of all the methods and functions of the entire software have been accurately done. All the bugs at the Unit Test stage must have been fixed and archived.
- All the modules necessary to the Integration Test are availble and ready to be run.
- The software is code-complete and fulfills all the requirements that have been specified in the RASD and DD.
- All the documentation about the project and the whole software is available to let the testers intervene and catch all the possible bugs that may rise, judging the correct behaviour of the various components among each

other.

- All the test cases are complete and well documented.
- All the testing software needed to proceed with this step is correctly installed in the system and works properly.

2.2 Elements to be integrated

The system can be divided in 3 big subsystems: a communication subsystem, an application subsystem and a data subsystem. The communication one is composed by the WebClient along with the WebServer component, the Mobile Client and the Connection Handler. On the other hand the QueueManager,the RideManager and the Application Controller compose the application subsystem, while the DBMS and the DataAccessManager are part of the data subsystem. All the subsystems must be integrated with the SecurityManager component.

2.3 Integration testing strategy

For the integration testing strategy we chose something in between a bottom-up approach and a functional groupings one. This has more than one reason as in the Top down approach we would have had the advantage of being able to output part of the project before finishing the testing of everything, something like a beta of our application that wouldn't have had all the functionalities. This however was not our objective, as given that the overall application is not too complex, the time spent for testing is not enough to justify a beta. Therefore the bottom-up/functional groupings approach, that gives us other advantages: first and most important is the parallel testing of the lower modules, that can lower the overall testing time. Also, the functional groupings strategy (with this

we mean that we will divide the project in functional subsystems and then test them bottom-up) will allow us to test deeply the modules alone before putting them together, assuring us the correct behaviour of each one, thing that would be harder to do in a top down/non functional approach.

2.4 Sequence of component/function integration

This section provides the order we intend to integrate each component and subsystem to compose the final system in accordance with the previous strategy defined.

2.4.1 Software integration sequence

According to the strategy the integration test plan starts with three groups of components, each one representing a set of functionalities, that could be integrated parallely.

The first group of components that represents the Communication subsystem includes the WebClient and the Web Server, the MobileClient component and the CommunicationHandler component. The second group represents the Data layer of the system and is the join of the DataAccessManager component and the DBMS, the Data subsystem. Lastly we spot that the RideManager, the QueueManager and the ApplicationController can be seen as a third group of component's functionalities, the Application subsystem.

A view of the subsystem described above can be seen in Figure 2.1.

2.4.2 Subsystem integration sequence

The groups of component described before can be seen as a subsystem. Therefore the subsystem integration sequence begins with an integration of the Com-

munication susbsystem and the Security Manager. After this kind of merge we integrate the previous subsystem with the subsystem that concerns the Data part of the application and the Application susbsystem. In this way we obtain a system that is completely integrated tested.

The view on the integration steps can be seen in both Figures 2.1 and 2.2. The lables on the arcs indicate the integrations to be made.

Figure 2.1: Detailed view of the various Subsystems described in section 2.4.1

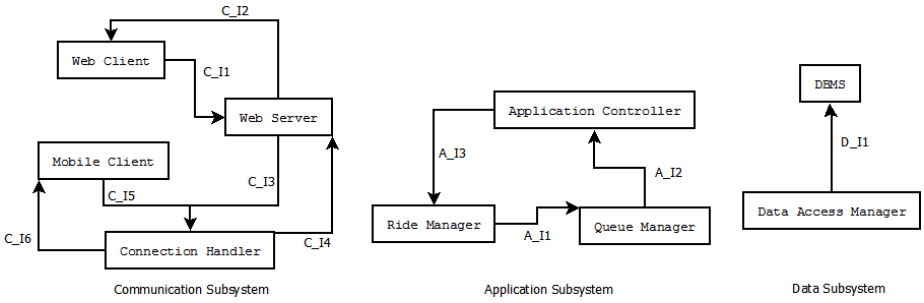
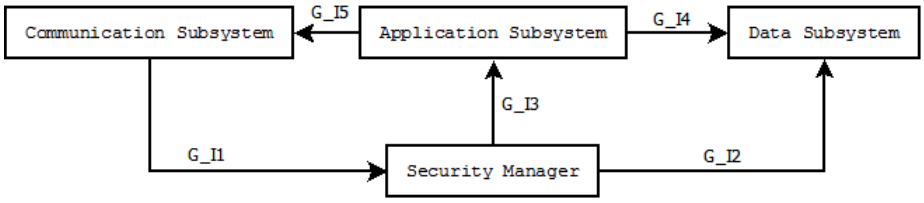


Figure 2.2: Subsystems view



Chapter 3

Individual Steps & Test Description

In this chapter will be analyzed each individual step of the integration test plan.

3.1 Communication subsystem tests

This section deals with the integration tests concerning the communication subsystem.

3.1.1 Web Client → Web Server

Test case identifier	C_I1T1
Components involved	Web Client → Web Server
Input specifications	Create typical requests by a Web Browser
Purposes	Test the correct generation of an answer to a request by the Web Client
Output specifications	The Web Server interprets in the right way a request by the Web Client and is ready to send the right answer
Environment needed	The Web Client and Web Server components are correctly implemented, and one ready to send a message and the other one to interpret in the right way

3.1.2 Web Server → Web Client

Test case identifier	C_I2T1
Components involved	Web Server → Web Client
Input specifications	Create typical answers of the Web Server
Purposes	Test the correct decodification of a the Web Server's answer
Output specifications	A representation of the Web Server message
Environment needed	The implementation of the Web Server that generates an answer and the Web Client that translates the message

3.1.3 Web Server → Connection Handler

Test case identifier	C_I3T1
Components involved	Web Server → Connection Handler
Input specifications	Typical messages forwarded by the Web Server
Purposes	A correct interpretation of the messages by the Connection Handler
Output specifications	The right method called by the Connection Handler
Environment needed	The implementation of both components that guarantees to the Web Server a right forward of a message and the right decodification by the Connction Handler

3.1.4 Connection Handler → Web Server

Test case identifier	C_I4T1
Components involved	Connection Handler → Web Server
Input specifications	Typical messages forwarded by the Connection Handler
Purposes	Check the correct decodification of a message by the Web Server
Output specifications	The correct representation of the messages by the Web Server
Environment needed	Implementation of the Connection Handler and of the Web Server that ensure a communication from the former component to the latter one

3.1.5 Mobile Client → Connection Handler

Test case identifier	C_I5T1
Components involved	Mobile Client → Connection Handler
Input specifications	Typical messages sent from a Mobile Client
Purposes	A correct interpretation of the messages by the Connection Handler
Output specifications	A correct interpretation of the messages by the Connection Handler
Environment needed	Each component implemented at least for the generation, for the Mobile client, and the transaltion, for the Connection Handler, of the messages

3.1.6 Connection Handler → Mobile Client

Test case identifier	C_I6T1
Components involved	Connection Handler → Mobile Client
Input specifications	Typical messages forwarded by the Connection Handler
Purposes	The right translation of the message by the Mobile Client
Output specifications	The correct representation and visualization by the component
Environment needed	The implementation of the Connection Handler and the Mobile Client that allows the communication

3.2 Application subsystem tests

This section deals with the integration tests concerning the application subsystem.

3.2.1 Ride Manager → Queue Manager

Test case identifier	A_I1T1
Components involved	Ride Manager → Queue Manager
Input specifications	New ride request
Purposes	Check if the ride manager asks and receives correctly to the Queue Manager a taxi driver to be allocated to the ride
Output specifications	The right taxi driver assigned
Environment needed	In order to connect the Ride Manager with the Queue Manager a driver that simulates the Application Controller is needed. The Ride Manager and Queue Manager must be correctly implemented

3.2.2 Queue Manager → Application Controller

Test case identifier	A_I2T1
Components involved	Queue Manager → Application Controller
Input specifications	Record to be updated
Purposes	Check if the update requests coming from the Queue Manager (updating the queue, setting the driver allocations, ect.) are correctly implemented by the Application Controller
Output specifications	Updated record
Environment needed	Queue Manager and Application Controller are correctly implemented

3.2.3 Application Controller → Ride Manager

Test case identifier	A_I3T1
Components involved	Application Controller → Ride Manager
Input specifications	Ride request to be passed to the Ride Manager
Purposes	Check if the Ride Manager, given the request of a new ride from the Application Controller invokes the right methods in order to create a new ride
Output specifications	The new created ride
Environment needed	Tests I1 and I2 has to be successfully completed. Ride Manager and Application Controller correctly implemented

3.3 Data subsystem tests

This section deals with the integration tests concerning the data subsystem.

3.3.1 Data Access Manager → DBMS

Test case identifier	D_I1T1
Components involved	Data Access Manager → DBMS
Input specifications	General Query for DBMS
Purposes	Check if Data Access Manager is able to communicate with and query the DBMS
Output specifications	Correct generation of the answer by the DBMS
Environment needed	None

3.4 General system tests

This section deals with the integration tests concerning the whole three subsystem among each other as described in Figure 2.2.

3.4.1 Communication Subsystem → Security Manager

Test case identifier	G_I1T1
Components involved	Communication Subsystem → Security Manager
Input specifications	Request from a righteous client to do an action
Purposes	Check if the Security Manager is able to forward actions from clients that have the rights to do them
Output specifications	Security Manager calls the right methods necessary to forward the action
Environment needed	Communication Subsystem and Security Manager Tests succeeded

3.4.2 Communication Subsystem → Security Manager

Test case identifier	G_I1T2
Components involved	Communication Subsystem → Security Manager
Input specifications	Request from a non-righteous client to do an action
Purposes	Check if the Security Manager can block actions from clients that don't have the rights to do them
Output specifications	Security Manager calls the right methods necessary to block the action and alert the client
Environment needed	Communication Subsystem and Security Manager Tests succeeded

3.4.3 Security Manager → Data Subsystem

Test case identifier	G_I2T1
Components involved	Security Manager → Data Subsystem
Input specifications	Request from the Security Manager for a tuple in the DBMS
Purposes	Check if the Data Subsystem can receive Security Manager requests for data
Output specifications	Check if the DBMS chooses the right tuple
Environment needed	Data Subsystem and Security Manager Tests Succeeded

3.4.4 Security Manager → Application Subsystem

Test case identifier	G_I3T1
Components involved	Security Manager → Application Subsystem
Input specifications	Request from the Security Manager to do an action
Purposes	Check if the Application Subsystem is able to execute the right action requested by the Security Manager
Output specifications	The Application subsystem called the methods needed
Environment needed	Application Subsystem and Security Manager Tests Succeeded

3.4.5 Application Subsystem → Data Subsystem

Test case identifier	G_I4T1
Components involved	Application Subsystem → Data Subsystem
Input specifications	Request from the Application Subsystem for data
Purposes	Check if the Data Subsystem can generate the correct queries after being questioned by the Application Subsystem
Output specifications	The Data Subsystem called the right methods
Environment needed	Application Subsystem Tests and Data Subsystem Tests Succeeded

3.4.6 Application Subsystem → Communication Subsystem

Test case identifier	G_I5T1
Components involved	Application Subsystem → Communication Subsystem
Input specifications	Notification from the Application subsystem directed to a client
Purposes	Check if the Communication Subsystem receives the right informations from the Application Subsystem
Output specifications	Notification received correctly
Environment needed	Application Subsystem and Communication Subsystem tests succeeded

Chapter 4

Tools & Test Equipment Required

In this chapter there is a description of what tools are used in order to accomplish the integretion test.

4.1 Mockito

Figure 4.1: Mockito logo

We use the Mockito tool for the Interaction Testing: we want to verify and ensure that interaction between components happens and happens in the right way. If something in the implementation does not work as we expect and there is something missing with this tool we're able to create stubs and check the interaction and the integration in a faster way.

4.2 Arquillian

Figure 4.2: Arquillian logo

We use Arquillian which provides an integration testing framework for containers: it allows to execute test cases inside and against the container and to let it manage the lifecycle of the containers and it is also an useful instrument to inject the Contexts and Dependecy Injection(CDI) beans, Enterprise JavaBeans(EJB).

Chapter 5

Program Stubs & Test Data Requirement

5.1 Stubs and Drivers

As a Bottom-up strategy has been chosen there's no need for stubs of any kind, as the bottom components has already been tested and can be used without being in need of any stub.

We're going to need only one driver, for the connection between Ride and Queue manager. As a matter of fact as we're using a bottom-up approach we'll need to test the connections between those two managers before testing the Application controller, so we will need a Driver that fakes the operations done from the Application controller to the request from the Ride manager and forwards it to the Queue manager.

5.2 Special test data

Special test data are instead useful in more than one occasion:

For the Application Subsystem there's the need for a fake ride request and some Taxi Drivers to test the connection between Ride and Queue manager. As the connection between DBMS and the two managers is not yet implemented, the Drivers and Rides will have to be implemented directly in the managers as they would appear after being requested to the DBMS.

For the Data Subsystem there's the need for some fake information in the DBMS and a fake request from the Data Management System. As in the Application Subsystem the request will be implemented as it arrives from the Application controller.

For the Communication Subsystem the need is for messages that go from the server to the clients, as for example in test I6 the server needs a fake message that will be translated by the mobile client. There's no need for special data from clients to server as everything can be called directly from an actual client.

For the connection between the subsystems the necessity is for a fake request from the security manager to the Data subsystem and some data inserted in the DBMS, a fake request from the security manager to the application subsystem to do an action, another one from the application subsystem to the data one and last one from the application subsystem to the communication one to check the functionality of the latter one.

Chapter 6

Other Info

6.1 Working Hours

	Alessandro	Davide	Stefano
ITPD hours	10	10	12
Total hours	66	68	66.5

6.2 Used Tools

- Text editors;
- GitHub;
- LyX and LaTeX.