My Taxi Service:

TEST PLAN

Julián David Gallego García

Politecnico di Milano january 21, 2016


POLITECNICO
MILANO

CONTENTS

1. Introduction

1.1. Purpose and scope

The goal of this project is to optimize the taxi service in a large city through a web application and a mobile app. These applications allow the passengers to request a taxi in an easy way and the taxi drivers to have a fair management of taxi queues. This document has the goal to present the tests to be applied to the system to review its performance to guaranty that it is responsive and reliable, in general and each subsystem, ensuring that all its subsystems interacts properly, and will be focused on the specifications and tools needed to implement these test.

This document is the Integration Test Plan Document for the system My Taxi Service project. This document is intended for the development team, highlighting the tests needed to be done, the sequence to do them and in witch tools.

1.2. List of Definitions and Abbreviations

1.2.1. Definitions

User: a person who requests a service from the system. It can be a visitor or a passenger.

Visitor: a person who is not registered in the application.

Passenger: a person who is registered in the application.

Taxi driver: a taxi driver who access the application with a specific ID.

Request: the request of a taxi in a certain area and position in the city made by a user.

Stubs: are considered as the dummy modules that always simulate the low level modules.

Drivers: are considered as the form of dummy modules, that is handled in bottom up integration testing.

1.2.2. Acronyms and abbreviations

RASD: Requirements analysis and specification document.

DD: Design Document.

DBMS: Database Management System

JEE: Java Enterprise Edition

OS: Operative System

UML: Unified Modeling Language

RMI: Remote Method Invocation

EIS: Enterprise Information systems.

1.3. List of Reference Documents

RASD: Julian Gallego, Requirements analysis and specification document for My taxi service.

DD: Julian Gallego, Design document for My taxi service.

Specification document: Software Engineering 2 Project, Integration Test Plan document.

2. Integration Strategy

2.1. Overview

In this chapter the integration strategy will be described. In the section 2.2 the prerequisites for the tests will be presented. The section 2.3 is dedicated to the elements to be integrated in the system to execute some tests. Finally, in the sections 2.4 and 2.5 the strategy used to test the integrations will be discussed, paying attention to the order.

2.2. Entry Criteria

These are the criteria that must be respected for any component before the integration testing phase may begin:

The component or subcomponent to be tested must be complete and functional.

The component satisfies the requirements and the assumptions specified in the RASD.

The component satisfies the architecture and the design specified in the DD.

Code Inspection is complete and revised.

Test environment, test cases and test data are complete and ready to use.

Stubs/drivers/oracles needed already developed.

2.3. Elements to be integrated

These components refer to the ones specified in the Component View in chapter 2.3 of the DD. From the deployment view of the system, its have been selected witch components have to be integrated and the order of integration, according to the strategy adopted (thread). Its have been identified the main subsystems to be integrated:

-The **Logic** component, which includes the modules that take care of the major functionalities of the application.

-The **Status/Information Manager** subcomponent, which includes the software module needed to manage all the information of the users.

-The **User interface** subcomponent, witch includes the software module needed to manage all the interface that can see and interact the user.

-The **EIS or DBMS Manager**, which is the software module that manages the communication between the DBMS and the application.

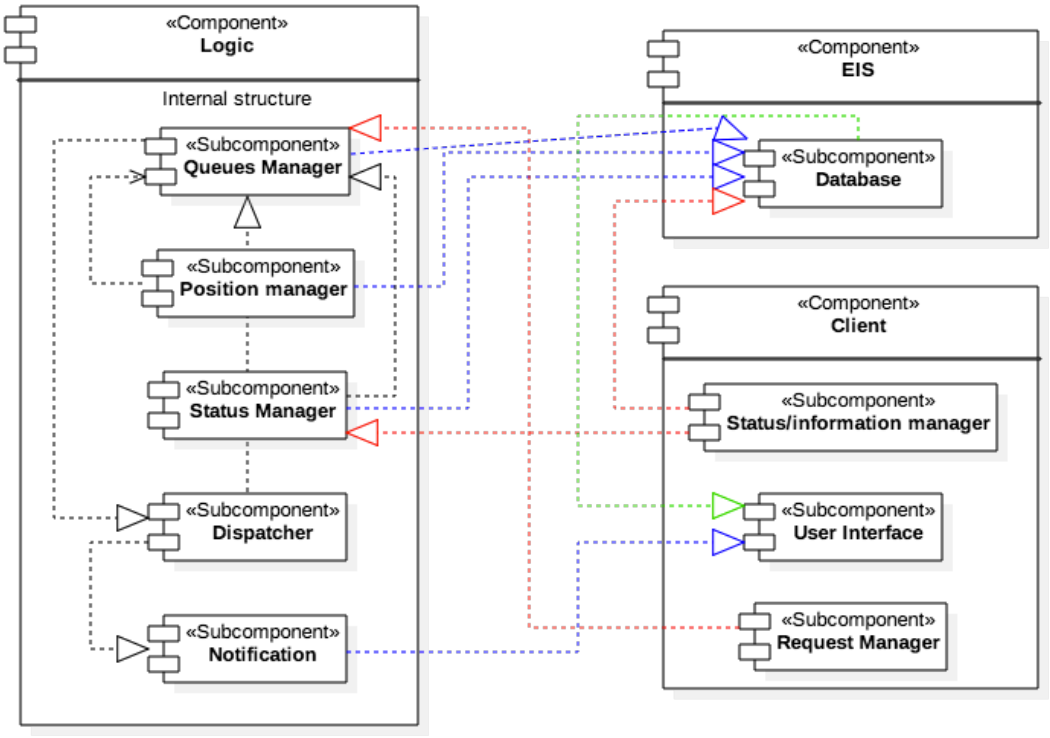

Figure 1. Deployment view

2.4. Integration Testing Strategy

It will use a thread. It is chosen such approach because the type and the complexity of the application that it is been designed, the main advantage of the Bottom-Up approach is that bugs are more easily found. It's chosen thread over top-down because it is easier to test the the system and its components in such away due to the modular approach developed for the application but because we use several portion of every module to accomplish an action we can't use bottom up, that is why its followed a thread. Besides, the development of many stubs (required in top-down analysis) will require a lot of time and a higher complexity building drivers for the bottom-up and top down analysis, than in this way.

2.5. Sequence of /Function Integration

To apply the strategy of integration first will be integrated by functionality (subsystem), data storage management, Logics, Visualization and data management, then when all the subcomponents of each subsystem are integrated, the subsystems will be integrated to have at the end all the system, integrated.

2.5.1. Software Integration Sequence

Integration Tests of the Data Storage Management subsystem:

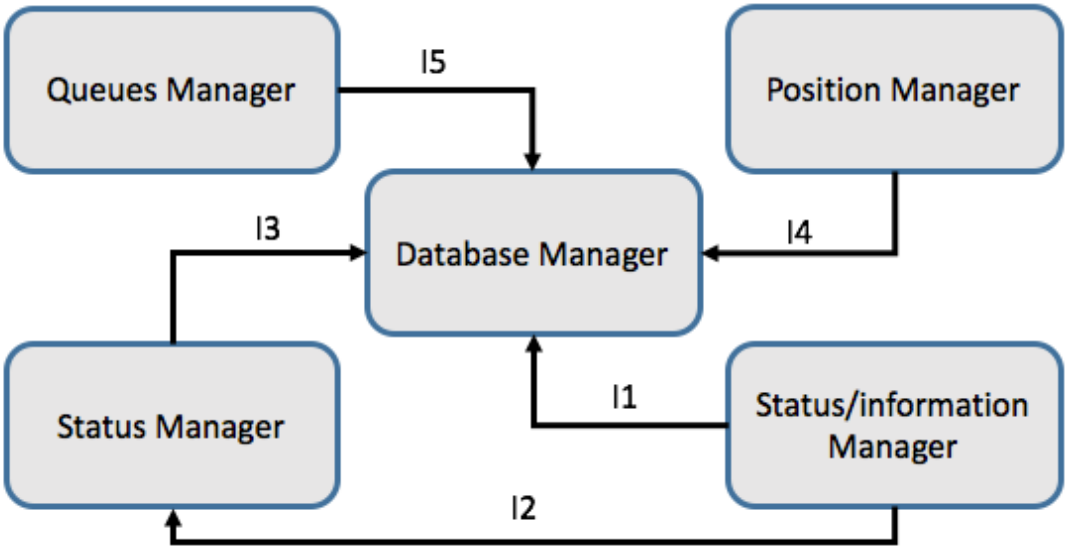

Figure 2. Components of the Data storage management subsystem.

ID	Integration test
I5	Queues Manager→Database Manager
I3	Status Manager→Database Manager
I4	Position Manager→Database Manager
I1	Status/ information manager→Database Manager

I2	Status/ information manager→Status Manager→Database Manager

Integration Tests of the Logic subsystem:

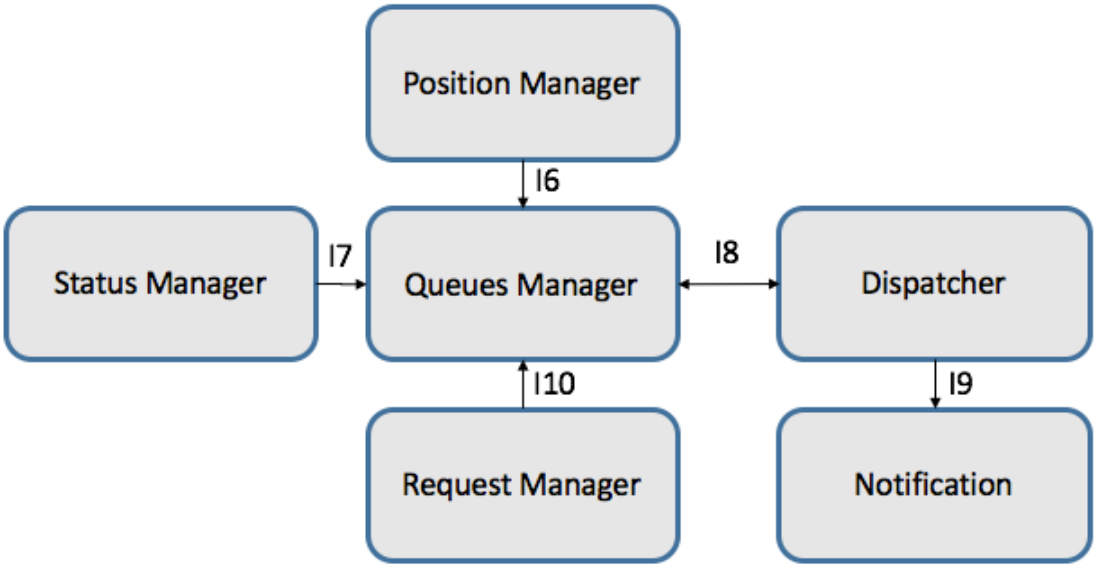

Figure 3. Components of the Logic subsystem.

ID	Integration test
I6	Position Manager → Queues Manager
I7	Status Manager→ Queues Manager
I8	Dispatcher→ Queues Manager
I9	Queues Manager→Dispatcher→Notification
I10	Request Manager→ Queues Manager

Integration Tests of the visualization subsystem:

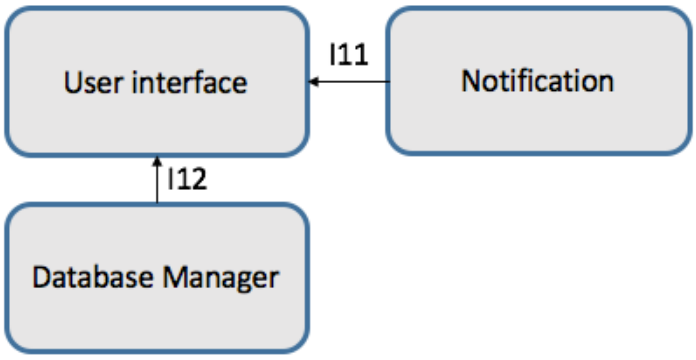

Figure 4. Components of the visualization subsystem.

ID	Integration test
I11	Database Manager → User interface

I12	Notification→ User interface

Integration Tests of the Information Management subsystem:

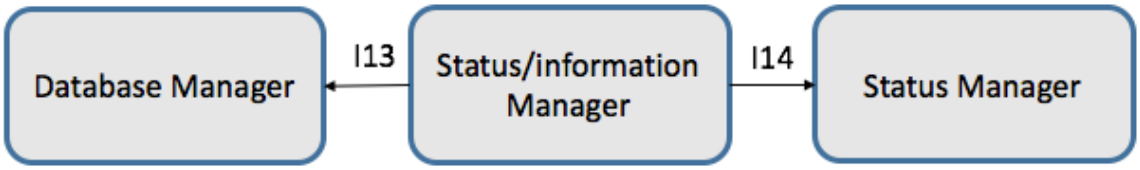

Figure 5. Components of the Information Management subsystem

ID	Integration test
I13	Status/information→Manager Database Manager
I14	Status Manager→ Status/information Manager

2.5.2. Subsystem Integration Sequence

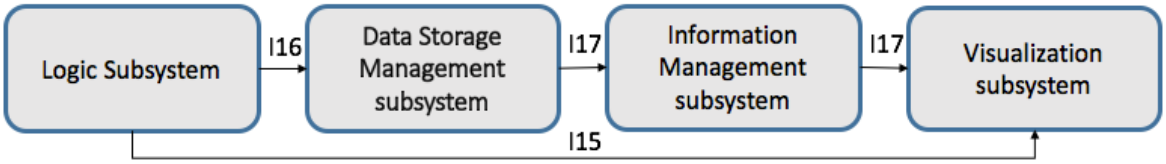

Figure 6. Subsystem Integration Sequence

ID	Integration test
I15	Logic Subsystem→ Visualization Subsystem
I16	Logic Subsystem→Data Storage Management Subsystem
I17	Data Storage Management Subsystem→Information Management subsystem→Visualization Subsystem

3. Individual Steps and Test Description

3.1. Integration test cases

3.1.1. Integration test case I1

Test Items: Status/ information manager→Database Manager
Input Specification: update data related to information about a user
Output Specification: Check that the data were correctly update on the database on the correct fields.
Environmental Needs: Create a driver that change information, in the cellphone app and web page, about a user and send it.

3.1.2. Integration test case I2

Test Items: Status/ information manager→ Status Manager
Input Specification: change the status of the driver.
Output Specification: Check that the data correctly trigger the status manager routine.
Environmental Needs: Create a driver that change the status of the driver, in the cellphone app and web page and send it.

3.1.3. Integration test case I3

Test Items: Status Manager→Database Manager
Input Specification: update data related to a state of a driver
Output Specification: Check that the data were correctly update on the database on the correct fields.
Environmental Needs: I1 and I2 success and create a Dispatcher driver to create a change on the taxi driver when he accepts a ride.

3.1.4. Integration test case I4

Test Items: Position Manager→Database Manager
Input Specification: update data related to the position and zone of a driver
Output Specification: Check that the data were correctly update on the database on the correct fields.
Environmental Needs: Create a driver that send from a phone different positions of the GPS.

3.1.5. Integration test case I5

Test Items: Queues Manager→Database Manager
Input Specification: update data related to the position and zone of a driver
Output Specification: Check that the data were correctly update on the database on the correct fields.
Environmental Needs: Success I1, I2, I3, I4 and Dispatcher driver to create a change from a ride taken.

3.1.6. Integration test case I6

Test Items: Position Manager → Queues Manager
Input Specification: update data related to the position and zone of a driver
Output Specification: Check that the data were correctly update on the queues manager and made the changes on the zone and queues if were necessary.
Environmental Needs: Create a driver that send from a phone different positions of the GPS.

3.1.7. Integration test case I7

Test Items: Status Manager→ Queues Manager
Input Specification: update data related to a state of a driver

Output Specification: Check that the data were correctly update on the queues manager and made the changes on the queue to erase the driver or add it to one.
Environmental Needs: Create a driver that send from a phone different positions of the GPS to add a driver to a queue and also a position manager to do it, create a Dispatcher driver to create a change on the taxi driver when he accepts a ride and Create a driver that change the status of the driver, in the cellphone app and web page and send it.

3.1.8. Integration test case I8

Test Items: Dispatcher→ Queues Manager
Input Specification: update data related to the queue and driver information
Output Specification: Check that the data were correctly update on the queues manager and made the changes on the queue to erase the driver that take the ride, and remove the user from waiting.
Environmental Needs: create a driver to accept a ride from a taxi driver.

3.1.9. Integration test case I9

Test Items: Queues Manager→Dispatcher→Notification
Input Specification: send an offer of a ride to a driver.
Output Specification: Check that when reject the queues manager send an other request to other driver if necessary, the notification to the driver change the info displayed on his screen.
Environmental Needs: Create a passenger offer for a ride.

3.1.10. Integration test case I10

Test Items: Request Manager→ Queues Manager
Input Specification: update data related to a state of a driver when he changes it.
Output Specification: Check that the data were correctly update on the queues manager and made the changes on the queue to erase the driver or add it to one.
Environmental Needs: Create a driver that send from a phone different positions of the GPS to add a driver to a queue.

3.1.11. Integration test case I11

Test Items: Database Manager → User interface
Input Specification: send data from the database to the interface.
Output Specification: Check that the data send to the interface is the same and it's seen in the correct places.
Environmental Needs: Non.

3.1.12. Integration test case I12

Test Items: Notification→ User interface
Input Specification: send data when the driver accepts the ride.

Output Specification: Show on the users the waiting time and update the time and the position to the driver and the passenger for the remaining time.
Environmental Needs: Create a driver to simulate the accept of a ride from a taxi driver.

3.1.13. Integration test case I13

Test Items: Status/ information manager→Database Manager
Input Specification: update data related to information about a user
Output Specification: Check that the data were correctly update on the database on the correct fields.
Environmental Needs: Create a driver that change information, in the cellphone app and web page, about a user and send it.

3.1.14. Integration test case I14

Test Items: Status Manager→ Status/information Manager
Input Specification: Send a change of status of the driver
Output Specification: Check that trigger an alert on the Status/information manager from the change.
Environmental Needs: create a Dispatcher driver to create a change on the taxi driver when he accepts a ride.

3.1.15. Integration test case I15

Test Items: Logic Subsystem→ Visualization Subsystem
Input Specification: from a request of a ride send the proposal and response
Output Specification: Check that the driver and the passenger changes on the interface, correspond the data given.
Environmental Needs: create a Driver to make a request of a taxi, a Dispatcher driver to create the changes when he accepts a ride, I6, I7, I8, I9, I10, I11, I12 success.

3.1.16. Integration test case I16

Test Items: Logic Subsystem→Data Storage Management Subsystem
Input Specification: from a request of a ride send the proposal and response
Output Specification: Check that were made the changes on the database and at the end recorded the corresponded data.
Environmental Needs: I2, I3,I4,I5,I6, I7, I8, I9, I10 success.

3.1.17. Integration test case I17

Test Items: Data Storage Management Subsystem→Information Management subsystem→Visualization Subsystem
Input Specification: changes of a data on from the information manager into the database
Output Specification: Check that in the interface the data are changed correctly.
Environmental Needs: I1, I2, I3,I4,I5, I11, I12, I13, I14 success.

3.2. Test Procedures

3.2.1. Integration test procedure TP1

Purpose: This procedure verifies the data recorded:
-Save the data of the Queue after make changes
-Save the data of the current queue of the taxi drivers and position.
-change the data from the database when an update occurs.
-change the information about status and personal information of the users.

Procedure steps: I1, I2, I3, I4, I5

3.2.2. Integration test procedure TP2

Purpose: This procedure verifies the internal logics:
-Manage the Queues.
-Send offers of rides.
-Receive and manage request of rides.
-Manage zones and positions of drivers.
-Manage the status of the drivers.

Procedure steps: I6, I7, I8, I9, I10

3.2.3. Integration test procedure TP3

Purpose: This procedure verifies the visualization:
-Can access and show data from the database property
-Can show the data of the remaining time and distance of a waiting ride
-Can display correct information on the web-and cellphone interface
Procedure steps: I11, 12.

3.2.4. Integration test procedure TP4

Purpose: This procedure verifies the information manage:
-Save the data of the taxi drivers and position on the database.
-Change/record the data on the database when an update occurs in the personal information.
-Change triggers when a change of status occurs.

Procedure steps: I13, I14, I15.

4. Tools and Test Equipment Required

. Glassfish Server open source edition 4.1
. Oracle database 12c
. Java EE.7
. NetBeans IDE 8.1 or Eclipse 4.5.1
. Any OS which supports JDK 7 and JRE 7
. Android simulator /IOS Simulator of different versions
. Mockito tool, to design and to implement the program stubs.
. Arquillian to perform the integration tests.
. For the test it will be used for simulate a user using the service on the web application, a computer and a mac, with the System Windows 7 or greater, Lion or greater or GNU/Linux Ubuntu and Mac OS X 10.8 Mountain. The System must be able to run a web browser to run the application on the web as Firefox, Microsoft Edge, Google Chrome, Opera or Safari. The computer must have a processor core i3 and 1 GB of ram.
. Latest version of most used browser (Google Chrome, Mozilla, Firefox, Safari) for the testing of the web application

5. Program Stubs and Test Data Required

Since we have used a bottom-up approach, we need several drivers for testing
our components and some simulation of the interaction of the user with the system. In particular, it is needed:

Database with some data.
Dispatcher Driver
Position Manager Driver
A simulation of a request of a driver
A simulation of an offer a ride.
A simulation of a request of a driver.
A simulation of a registration.
A simulation of a profile management.