Politecnico di Milano

A.A. 2015-2016

Software Engineering 2: “myTaxiService”

Integration **T**est **P**lan **D**ocument

Davide Azzalini (855185), Fabio Azzalini (855182)

1 Introduction

In this first chapter we give an introduction and a overall description of the whole document. In particular we present the purpose and the scope of the integration testing. We also provide some other useful information such as a list of the definitions, abbreviations and acronym used.

1.1 Purpose

This document represent the Integration Test Plan Document (ITPD). The individual software modules (components) are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the costumer.

Integration testing ensures that the unit-tested modules interact correctly.

This document is supposed to be written before the integration test actually happens.

1.2 Scope

The government of a large city aims at optimizing it's taxi service. At this stake myTaxiService will be created, a brand-new application available in two variations: web application and mobile application.

In particular, the attention will be focused on making the access to the service easier from the passengers and on guaranteeing a fairer distribution of the work load among all taxi drivers.

The application will allow passengers to request a ride and taxi drivers to decide weather or not to give the ride. Once a taxi driver has been allocated the passenger will be informed about the waiting time and the code of the incoming taxi.

Furthermore, the application will ensure a more efficient management of the service since each taxi is provided with a GPS tracker and the city is divided in zones, as a result a reduction of waiting times it is expected.

In addition to the standard taxi service (in which the costumer contacts the taxi driver in the exact moment he/she needs the service), will also be possible for the costumers to make reservations for future rides.

Important is also for the system to be developed in such a way that a future addition of some new feature can be made easily and without too much effort.

Moreover, the possibility for payments to be electronic, hence recorded, should help decrease the amount of tax evasion as all transactions can be tracked and accounted for.

1.3 List of Definitions and Abbreviations

- ITPD: Integration Test Plan Document

1.4 List of Reference Documents

- Specification Document: myTaxiService Project AA 2015-2016.pdf
- RASD.pdf
- DD.pdf
- Assignment 4 - integration test plan.pdf
- Ian Sommerville - Software Engineering (Ninth Edition)
- Lecture slide
- Mockito reference document
- Arquillian reference document

1.5 Overall Description

The following is the structure of this integration test plan document:

- In **chapter 2** are outlined the approach and the strategies selected for the integration plan, as well as the components that will be tasted and the ways in which they interact.

- In **chapter 3** can be found an in depth description of each test.
- In **chapter 4** we present the tools and the equipment that will be used to conduct the test. A detailed explanation of each of them is provided.
- In **chapter 5** are outlined the additional program stubs and test data that will be required in order to perform the actual integration test.

2 Integration Strategy

In this chapter particular emphasis is on the preconditions that have to be met before the integration test begins, the components to be integrated, the selected approach and the exact order in which the components are supposed to be tested.

2.1 Entry Criteria

Some criteria must be met in order for the integration testing to be ready to be performed. All the components must be fully developed and unit tested and so must be all the required drivers and stubs.

2.2 Elements to be Integrated

We are performing integration testing at a component level. A full and detailed description of all the components of myTaxiService can be found in the Design Document. We have decided not to include all the components in the integration testing but only those which perform meaningful functionalities and are actually integrated with other components.

Below it is shown a list of the selected components:

- Driver manager
- Queue manager
- Request allocator
- Request manager
- Payment
- Receipt
- Account information manager
- Credential validation
- Reservation allocator

- Reservation manager
- Queue manager

2.3 Integration Testing Strategy

We have chosen to perform the integration testing of our system with a bottom-up approach.

With this strategy we start testing the lowest level components that will be used to test the higher level components.

We have chosen a bottom-up strategy because in general we don't need stubs since we start testing the less demanding components. Actually, we do need some stubs anyway in order to simulate the behavior of the users or the DataBase.

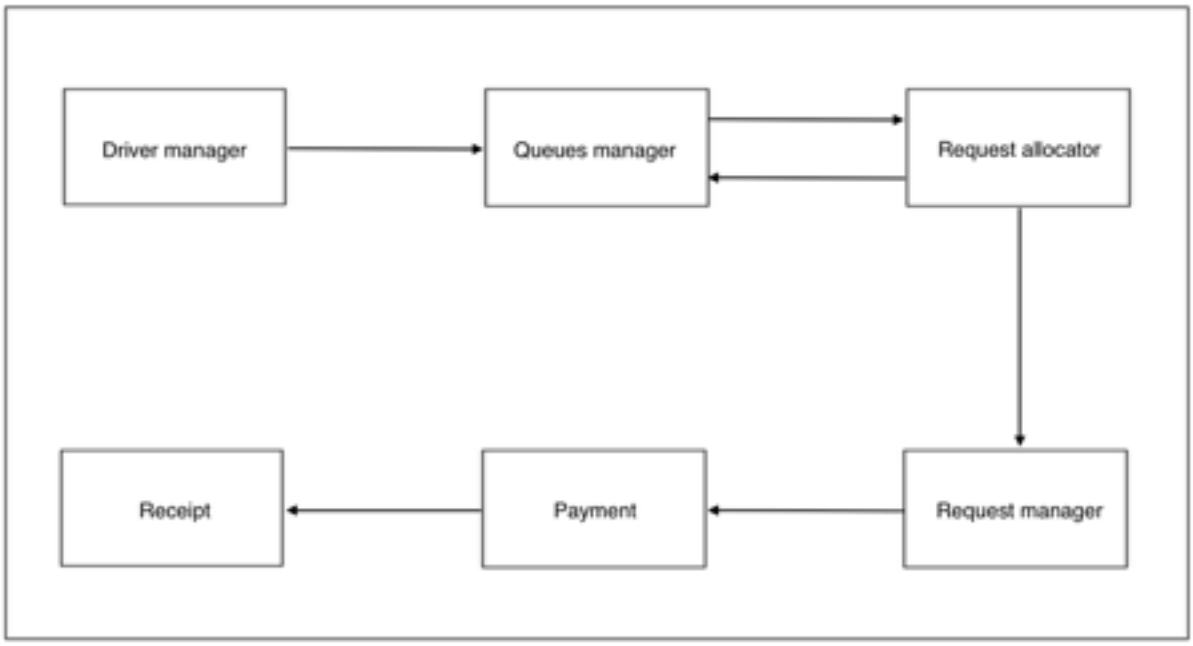
An other advantage is that starting from the bottom means that the critical modules are generally tested first an therefore any error or mistake is found out early in the process.

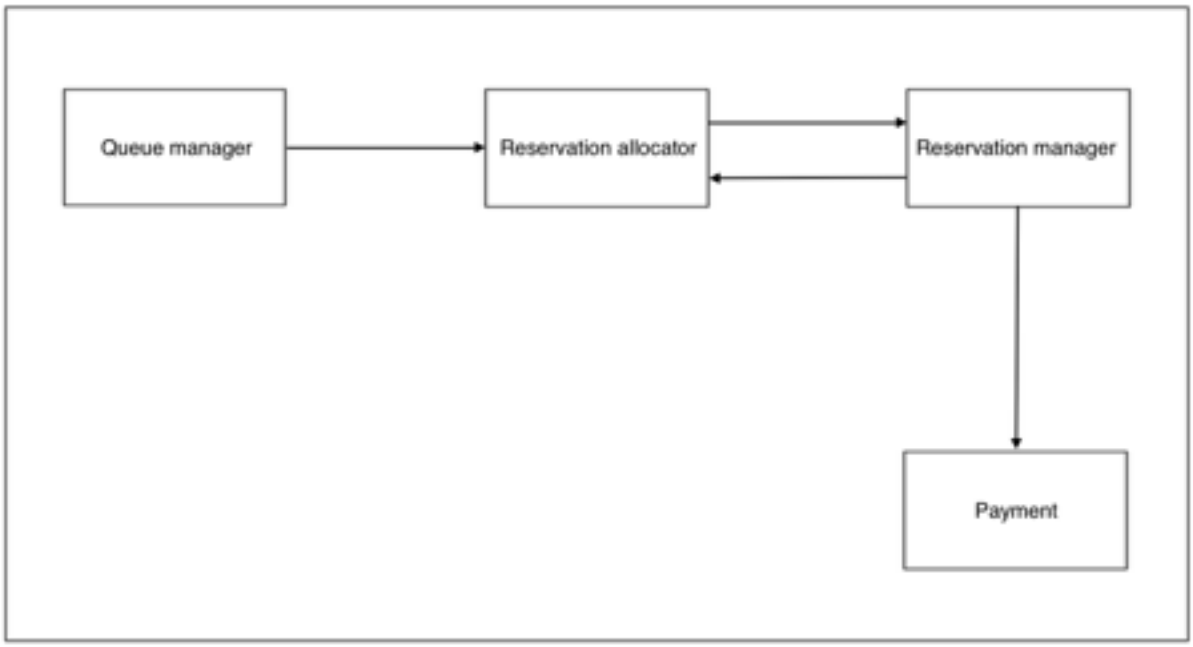
Furthermore, with a bottom-up approach test conditions are easier to create, so is the observation of test result.

2.4 Sequence of Component/Function Integration

In the design document we have grouped our components in subsystem according to their functionalities (request, reservation, profile, …). Unfortunately when it comes to integration testing such distinction is not very meaningful (a more operative distinction would have been preferred eg. I/O processor, Notification Engine, …). For the reasons above we will consider the server application as a unique subsystem.

In the figures below the sequence of integration will be shown. The arrow represent the order of integration.


Driver manager
Queues manager
Request allocator
Receipt
Payment
Request manager


Queue manager
Reservation allocator
Reservation manager
Payment

Integration order:

ID	Integration Test
I1	Driver manager —> Queues manager
I2	Queues manager —> Request allocator
I3	Request allocator —> Queues manger
I4	Request allocator —> Request manager
I5	Request manager —> Payment
I6	Payment —> Receipt
I7	Queue manager —> Reservation allocator
I8	Reservation allocator —> Queue manager
I9	Reservation allocator —> Reservation manager
I10	Reservation manager —> Payment

3 Individual Steps and Test Description

This chapter is devoted to the detailed description of each test case. A special attention is given to the preconditions and postconditions as well as to the environmental needs.

3.1 Integration test case I1

Test case identifier	I1T1
Test item(s)	Driver manager —> Queues manager
Input specification	Typical information about an available taxi driver
Output specification	Check if data are valid and if the taxi driver is properly enqueued in the right queue.
Environmental needs	Client device (mobile App)

3.2 Integration test case I2

Test case identifier	I2T1
Test item(s)	Queues manager —> Request allocator
Input specification	Typical information about the first taxi driver in the queue
Output specification	Check if the request is correctly created and the notification is properly sent to the taxi driver
Environmental needs	I1, Client device (mobile App)

3.3 Integration test case I3

Test case identifier	I3T1
Test item(s)	Request allocator —> Queues manger
Input specification	Typical information about a taxi driver
Output specification	Check if the taxi driver is sent to the tail of the queue
Environmental needs	I2

3.4 Integration test case I4

Test case identifier	I4T1
Test item(s)	Request allocator —> Request manager
Input specification	Typical request data
Output specification	Check if the request is correctly handled and if the notifications are correctly sent to both passenger and the driver
Environmental needs	I2, Client devices (mobile App and Web application)

3.5 Integration test case I5

Test case identifier	I5T1
Test item(s)	Request manager —> Payment
Input specification	Typical request data
Output specification	Check if the price is correctly evaluated
Environmental needs	I4

3.6 Integration test case I6

Test case identifier	I6T1
Test item(s)	Payment —> Receipt
Input specification	Typical data about a payment
Output specification	Check if the receipt is correctly created
Environmental needs	I5

3.7 Integration test case I7

Test case identifier	I7T1
Test item(s)	Queue manager —> Reservation allocator
Input specification	Typical information about the first taxi driver in the queue
Output specification	Check if the reservation is correctly created and the notification is properly sent to the taxi driver
Environmental needs	Client device (mobile App)

3.8 Integration test case I8

Test case identifier	I8T1
Test item(s)	Reservation allocator —> Queue manager
Input specification	Typical information about a taxi driver
Output specification	Check if the taxi driver is sent to the tail of the queue
Environmental needs	I7

3.9 Integration test case I9

Test case identifier	I9T1
Test item(s)	Reservation allocator —> Reservation manager
Input specification	Typical reservation data
Output specification	Check if the reservation is correctly handled and if the notifications are correctly sent to both passenger and the driver
Environmental needs	I7, Client devices (mobile App and Web application)

3.10 Integration test case I10

Test case identifier	I10T1
Test item(s)	Reservation manager —> Payment
Input specification	Typical reservation data
Output specification	Check if the price is correctly evaluated
Environmental needs	I9

4 Tools and Test Equipment Required

In this chapter we will introduce some tools that could be very useful to the testing team during the integration testing to the testing team.

The first tool is Mockito (http://site.mockito.org/).

Mockito is an open source testing framework that allows the creation of mockups for unit testing.

Mocking is particularly useful since allows you to abstract dependencies and have predictable results. A very useful test-wise feature is that it is possible to check that the interaction between the component under testing and the mock is correct.

The second tool that we present is Arquillian (http://arquillian.org).

Arquillian is a testing platform that enables developers to easily create automated integration, functional and acceptance tests. The core of Arquillian are containers that, unlike mocks, brings the test to the runtime giving you meaningful feedback and insight about how the code really works.

Finally, besides this isn't an "actual" tool, we want to remind that sometimes the best way to perform testing is by manual testing. It's often faster, easier and more convenient.

5 Program Stubs and Test Data Required

In order to make the integration testing possible there is the need to develop some stubs. A stub is a piece of code used to stand in for some other programming functionality. A stub may simulate the behavior of existing code or be a temporary substitute for yet-to-be-developed code.

A **Passenger Simulator** able to "act" as a real passenger.

This stub performs the following features:

- register;
- login;
- submit a request;
- submit a reservation;
- interact with the system during a ride.

A **Driver Simulator** able to "act" as a real taxi driver.

This stub performs the following features:

- login;
- set the availability;
- accept a request;
- accept a reservation;
- interact with the system during a ride.

6 Appendix

6.1 Hours of work

- Azzalini Davide: 7
- Azzalini Fabio: 7