Politecnico di Milano
A.A. 2015-2016
Software Engineering 2: "myTaxiService"

Test Plan Document

Luiza Bentivoglio, Michele Cantarutti

22January 2016

Summary

1. Introduction

1.1 REVISIONHISTORY

This is the first version of this document and no updates have been made yet. 21th January 2016.

1.2 PURPOSE AND SCOPE

The purpose of this document (ITPD) is to explain to the development team what to test, in which sequence, which tools are needed for testing, which stubs, drivers, oracles need to be developed and to describe the plans for testing the integration of the created components for the myTaxiService project. By integration, we mean the phase during which software modules are combined and tested as a group, and in fact, at this stage of our project we assume that all the modules of our project have been unit tested and therefore each one of them is working correctly individually. Therefore, the scope of this document is not focused on single modules but revolves around the system in the whole, considering all the modules together, and mostly testing whether all the components within assemblages interact correctly.

1.3 LIST OF DEFINITIONS AND ABBREVIATIONS

PASSENGER: a component on the client side, which represents a registered user, who is a customer of our service.

TAXI DRIVER: a component on the client side, which representsa taxi driver,which is a person who interacts with the system with the purpose of serving users.

CLIENT/USER:these two terms will often be used interchangeably and they're used to describe whoever uses the web application or the mobile application from the client's side, thus passengers and taxi drivers.

REQUEST: a component that represents requests, which have been defined in our RASD and DD document.

RESERVATION: a component that represents reservations, which have been defined in our RASD and DD document.

DRIVER: a main program that accepts test data, passes such data to the component and prints relevant results.

ITPD: Integration Test Plan Document

RASD: Requirement Analysis and Specification Document

DD: Design Document

1.4 LIST OF REFERENCE DOCUMENTS

As references for this document we used:

- The project description, that has been provided to us
- Our RASD document
- Our Design document
- Integration test plan example, SpinGRID

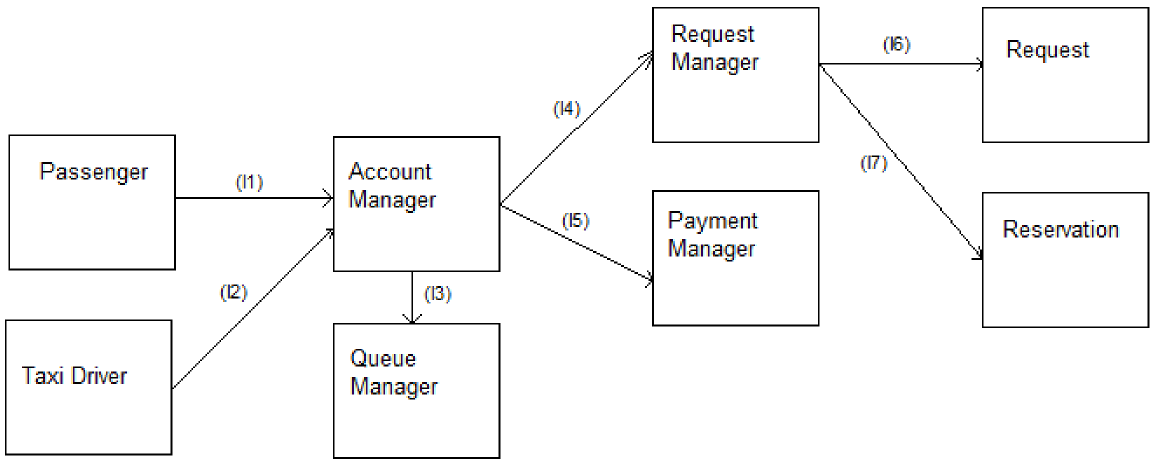
2. Integration Strategy

2.1 ENTRY CRITERIA

Before the integration testing can start, we must verify (or assume, in this document) that all the modules have undergone unit-tests successfully. Moreover, we will assume that each module is code-complete, which means there are no missing features.Likewise, we will assume that the developers have yielded a complete documentation for each module, so as to provide support in understanding any possible problems that might arise during the testing of the interaction between multiple components.

2.2 ELEMENTS TO BE INTEGRATED

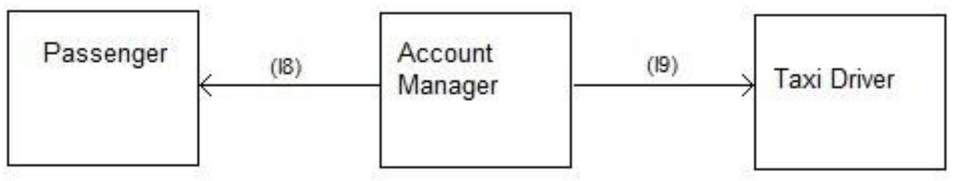
The figure below shows the main components that form the myTaxiService system. The arrows represent the order of integration. i.e. integration testing.



Integration Tests of the dispatcher-software

ID	Integration Test	Paragraphs	
I1	Passenger -> Account Manager	2.4.1	3.1
I2	Taxi Driver -> Account Manager	2.4.2	3.1
I3	Account Manager -> Queue Manager	2.4.3	3.2
I4	Account Manager -> Request Manager	2.4.4	3.2
I5	Account Manager -> Payment Manager	2.4.5	3.2
I6	Request Manager -> Request	2.4.6	3.2
I7	Request Manager -> Reservation	2.4.7	3.2

The picture below shows the interaction and the communication between the Account manager on the server and the client. By communication we mean the notifications that the server can potentially send to the users to notify them about something that has happened.



ID	**Integration Test**	**Paragraphs**
I1	Account Manager ->Passenger	2.4.8 3.1
I2	Account Manager ->Taxi Driver	2.4.9 3.1

2.3 INTEGRATION TESTING STRATEGY

We intend to test the modules that make up our system by using a bottom-up approach. This means that the testing will start at the bottom level (so that we can grant that the basic parts of our system are working from the beginning) and then the integration testing will follow the unit testing (which we assume has already been carried out).

2.4 SEQUENCE OF COMPONENT/FUNCTION INTEGRATION

2.4.1 Integration test case I1

Test Case Identifier	I1T1
Test Item(s)	Passenger -> Account manager
Input Specifications	Create typical Passenger input
Output Specifications	Check if the correct functions are called in the Account manager

Environmental Needs	Passenger driver

2.4.2 Integration test case I2

Test Case Identifier	I1T2
Test Item(s)	Taxi driver -> Account manager
Input Specifications	Create typical Taxi driver input
Output Specifications	Check if the correct functions are called in the Account manager
Environmental Needs	Taxi driver driver

2.4.3 Integration test case I3

Test Case Identifier	I2T1
Test Item(s)	Account Manager -> Queue Manager
InputSpecifications	Create typical Account Manager input
Output Specifications	Check if the correct functions are called in the Queue Manager
Environmental Needs	I1-I2 succeeded

2.4.4 Integration test case I4

Test Case Identifier	I3T1
Test Item(s)	Account Manager -> Request Manager
InputSpecifications	Create typical Account Manager input
Output Specifications	Check if the correct functions are called in the Request Manager
Environmental Needs	I1-I2 succeeded

2.4.5 Integration test case I5

Test Case Identifier	I4T1
Test Item(s)	Account Manager -> Payment Manager
InputSpecifications	Create typical Account Manager input
Output Specifications	Check if the correct functions are called in the Payment Manager
Environmental Needs	I3-I4 succeeded

2.4.6 Integration test case I6

Test Case Identifier	I5T1
Test Item(s)	Request Manager -> Request
InputSpecifications	Create typical Request Manager input
Output Specifications	Check if the correct functions are called in the Request Object
Environmental Needs	I4 succeeded

2.4.7 Integration test case I7

Test Case Identifier	I6T1
Test Item(s)	Request Manager -> Reservation
InputSpecifications	Create typical Request Manager input
Output Specifications	Check if the correct functions are called in the Reservation Object
Environmental Needs	I4 succeeded

2.4.8Integration test case I8

Test Case Identifier	I7T1
Test Item(s)	Account Manager -> Passenger
Input Specifications	Create typical Account Manager input
Output Specifications	Check if the correct functions are called in the Passenger Object
Environmental Needs	N/A

2.4.9 Integration test case I9

Test Case Identifier	I8T1
Test Item(s)	Account Manager -> Taxi Driver
Input Specifications	Create typical Account Manager input
Output Specifications	Check if the correct functions are called in the Taxi Driver Object
Environmental Needs	N/A

3. Individual steps and test description

3.1 Integration test procedure TP1

Test Procedure Identifier	**TP1**
Purpose	This test procedure verifies whether the main software: • can handle command-line input • can handle passenger input • can handle taxi driver input • can output requested information to a passenger • can output requested information to a taxi driver
Procedure Steps	Execute I6-I7

3.2 Integration test procedure TP2

Test Procedure Identifier	**TP2**
Purpose	This test procedure verifies whether the notification software: • can handle account manager input • can output notifications to the passenger • can output notifications to the taxi driver
Procedure Steps	Execute I8-19

4. Tools and test equipment required

These are the tools we deem necessary for the testing:

-Mockito, an open source testing framework for Java. The framework allows the creation of test double objects (mock objects) in automated unit tests;

-Jmeter, which can be used as a load testing tool for analyzing and measuring the performance of a variety of services;

-Arquillian;A 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.

5. Program stubs and test data required

Depending on the situation, different elements will be needed throughout the testing. For example, the first integration tests are carried out on the communication between the client side and the account manager on the server, and these tests will require a driver for each type of client (passenger and taxi driver) to simulate the possible behaviors of the user. By driver we mean a main program that accepts test data and passes this test to the component to be tested and prints relevant results.
In other situations, for example those regarding the Queue Manager and the Request Manager, we will need test data to work on, thus a simulation of the database. The simulation of the database must represent the possible instances that might occur, for example there should be several passengers and taxi drivers to see how well the system can manage the queues and the requests that are being made.
In the case of the testing on the notification system, we will only need two stubs, one representing the server side and one representing the client side, so that we can test whether the notification functionalities work correctly while sending messages to the clients.