Politecnico di Milano

Scuola di Ingegneria dell'Informazione

Corso di Laurea Magistrale di Computer Science and Engineering

Software Engineering 2 Project

myTaxiService

Part IV : ITPD (Integration Test Plan Document)

Principal Adviser:
Prof. Raffaela Mirandola

Authors:

Turconi Andrea ID n. 853589

Raimondi Lorenzo ID n. 859001

Accademic year 2015-2016

Contents

1 Introduction

1.1 Purpose and scope

This document describes the plans for testing the integration of the created components.The purpose of this document is to test the interfaces between the components as described in Design Document in chapter 2.6. The software to test is myTaxiService, an application aimed to satisfy citizens' taxi requests. The system will help users to call a taxi on a desired time and location, and will help taxi driver to get a more sharp and suited service. Users can book a taxi in a specific location, either for an immediate request or for a reservation booking. All the communications about useful informations like requests, response, time, are forwarded from and to the users by a notification system.

1.2 Definitions, acronyms and abbreviations

1.2.1 Definitions

- Request a taxi: send a request to the system of any type of taxi service, both reservations and demand
- Demand a taxi: send a request to the system of an immediate taxi service
- Reservation of a taxi: send a request to the system of a taxi service for a certain time, specified in the form
- Future reservation: a reservation that has not yet been executed
- Pending request: a request that has been sent by the use but not managed yet by the system.
- Accepted request: a request received and managed by the system, which has already accepted it, but has not yet selected a driver for fulfilling it.
- Ongoing request: a request that is being accomplished right at this moment; this time span goes from the driver selection instant to the user's arrival in the desired destination.
- Completed request: a past request, that has been completely performed.

1.2.2 Acronyms

- RASD: Requirement Analysis and Specification Document
- DD: Design Document

- GPS: Global Positioning System
- API: Application Programming Interface
- SOA: Service Oriented Architecture

1.2.3 Abbreviations

- [Gn] n-goal
- [Rn] n-functional requirement

1.3 References

- Specification document: assignment 1 and 2.pdf
- Requirement Analysis and Specifications Document v1
- Design Document v1
- Assignments 4 – Integration Test plan.pdf

2 Integration Strategy

2.1 Entry Criteria

In way to proceed to Integration Testing, project should achieve several requirements. The code has to be completed, without any missing component or function. Prevoius documents needs to be updated to the current project status and completed with following inserted features. Finally the code must be completely unit tested, having fixed all priority bugs.

2.2 Elements to be integrated

For what concerns the elements target of integration, they are the once already identified during the project design phase and described in the project Design Document:

- **User Interface Component** Responsible for the graphical interface interaction with the system.
- **Communication Manager** In charge of handle communication and messages between user and system.
- **Account Manager** Responsible for account logging and registation.
- **Data Manager** Grants managing and access to the database.
- **Request Manager** Fullfills all the request forwarded to the system.
- **Queue Manager** Responsible for taxi queues management.
- **Location Manager** In charge of retrieve and compute Google Maps data.
- **Taxi Manager** Grants a fair management of drivers availability status.
- **Service Provider** Grants access to system services for external applications.

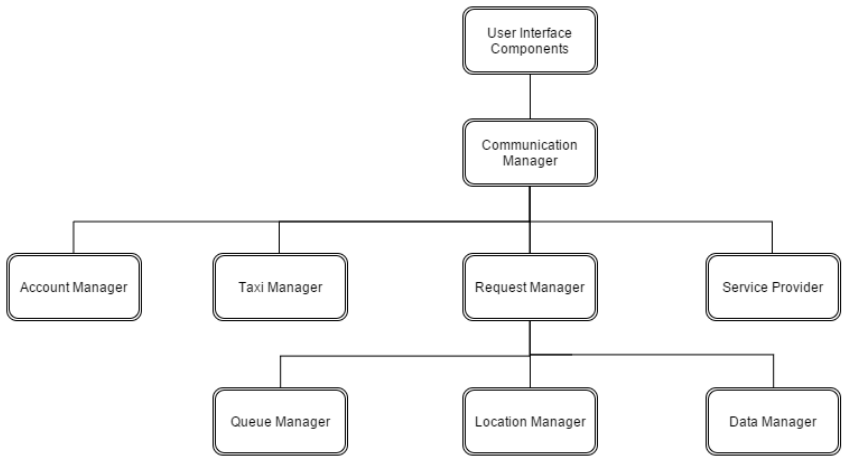
2.3 Integration Testing Strategy

Among the different approaches to integration testing we decided to adopt the Top-down one. This means that the integration starts from the upper level components, going down step by step to the lower level compononents, until the end. We reached this decision by taking into account pros and cons of the different strategies. The Top-down approach in fact allow to test firstly the integration of the top components, that are also the most important and usually complex, and so more prone to errors. In addition, starting to integrate the top components allow to build up a usable system, that becomes more and more complete during the integration process, otherwise adopting a bottom-up strategy we would have to integrate the bottom components, without having a usable system until the reach of the top ones.

2.4 Sequence of Components

In order to integrate components using the chosen Top-down strategy, we designed a tree of components, starting from presentation, business logic and then to data model. From this tree we decided to integrate the components, of course starting from the top ones, with a "depth first" method.

- UI Components
- Communication Manager
- Request Manager
- Account Manager
- Taxi Manager
- Queue Manager
- Location Manager
- Data Manager
- Service Provider



Starting from the User Interface Components it will be possible to use the application interface, that will only grant a proper interface navigation. Integrating Communication Manager the system will be capable of receive and send data from/to the user. Adding the Account Manager will implement logging and registering functions. Integrating the Taxi Manager, logging and availability editing of drivers will be achieved. Then, with the following step, the integration of the Request Manager, the system will accept and manage request. Then, with the Queue Manager, the system will be capable of handle queues of drivers and the request will choose the first available driver. Integrating Location Manager the system will be capable of locating users and compute waiting times in account of the Request Manager. Furthermore, integrating the Data Manager, the application will be able to store and get data from the database. Finally, adding the Service Provider, a backend interface is open for external applications.

3 Individual Steps and Test Description

3.1 Integration Test Cases

In all the integration tests, it also verified that all parameters have the same meaning. For example, if one component calls a method from another component, passing two parameters, one integer and one object. The component that calls the function intends integer as meters, but the function called intends integer as kilometres. The function is called in a right way, but the results will not be correct.

The test, besides verifying type of passed parameters and their order, should verify that these misunderstandings about the meaning of the parameters, does not occur.

3.1.1 Integration Test Case I1

Test Case Identifier	**I1T1**
Test Item(s)	UI Component → Communication Manager
Input Specification	Create typical Communicator input
Output Specification	Check if the system is able to send/receive message to/from a client
Environmental needs	N/A

3.1.2 Integration Test Case I2

Test Case Identifier	**I2T1**
Test Item(s)	Communication Manager → Account Manager
Input Specification	Create typical Account Manager input
Output Specification	Check if the system allows registrations and login
Environmental needs	I1T1

3.1.3 Integration Test Case I3

Test Case Identifier	**I3T1**
Test Item(s)	Communication Manager → Taxi Manager
Input Specification	Create typical Taxi Manager input
Output Specification	Check if the system changes drivers' status and manages unforeseen notices
Environmental needs	I1T1

3.1.4 Integration Test Case I4

Test Case Identifier	**I4T1**
Test Item(s)	Communication Manager → Request Manager
Input Specification	Create typical Request Manager input
Output Specification	Check if the system create, edit and update a request
Environmental needs	I1T1

3.1.5 Integration Test Case I5

Test Case Identifier	**I5T1**
Test Item(s)	Request Manager → Queue Manager
Input Specification	Create typical Queue Manager input
Output Specification	Check If the system returns first element of a queue, add/remove a driver to/from a queue
Environmental needs	I4T1

Test Case Identifier	**I5T2**
Test Item(s)	Taxi Manager → Queue Manager
Input Specification	Create typical Queue Manager input
Output Specification	Check if the system manages queues after a driver's status update
Environmental needs	I3T1

3.1.6 Integration Test Case I6

Test Case Identifier	**I6T1**
Test Item(s)	Request Manager → Location Manager
Input Specification	Create typical Location Manager input
Output Specification	Check if the system updates queues and calculates travel time
Environmental needs	I4T1

Test Case Identifier	I6T2
Test Item(s)	Queue Manager → Location Manager
Input Specification	Create typical Location Manager input
Output Specification	Check if the system gets position of a driver or a client
Environmental needs	I5T1 and I5T2

3.1.7 Integration Test Case I7

Test Case Identifier	I7T1
Test Item(s)	Request Manager → Data Manager
Input Specification	Create typical Data Manager input
Output Specification	Check if the system saves data on the database
Environmental needs	I4T1

Test Case Identifier	I7T2
Test Item(s)	Taxi Manager → Data Manager
Input Specification	Create typical Data Manager input
Output Specification	Check if the system gets data from database
Environmental needs	I3T1

Test Case Identifier	I7T3
Test Item(s)	Account Manager → Data Manager
Input Specification	Create typical Data Manager input
Output Specification	Check if the system creates a new client in the database and checks data
Environmental needs	I2T1

Test Case Identifier	I7T4
Test Item(s)	Queue Manager → Data Manager
Input Specification	Create typical Data Manager input
Output Specification	Check if the system gets queues data from the database
Environmental needs	I5T1 and I5T2

3.1.8 Integration Test Case I8

Test Case Identifier	**I8T1**
Test Item(s)	Communication Manager → Service Provider
Input Specification	Create typical Service Provider input
Output Specification	Check if the system guarantees all the services to external application
Environmental needs	All previous tests

3.2 Integration Test Procedures

3.2.1 Integration Test Procedure TP1

Test Procedure Identifier	**TP1**
Purpose	This test procedures verifies whether: • Client could send a demand message • System receives a demand message • System checks if all information are correct inserted • System gets client's position • System selects an appropriate taxi driver • System notifies the client • System saves a new demand in database
Procedure Steps	Execute I5T1-I6-I7 after I1/I4

3.2.2 Integration Test Procedure TP2

Test Procedure Identifier	**TP2**
Purpose	This test procedures verifies whether: • Client could send a registration/login message • System receives a registration/login message • System checks if all information are corrects • System creates a new account in database
Procedure Steps	Execute I7T3 after I2

3.2.3 Integration Test Procedure TP3

Test Procedure Identifier	**TP3**
Purpose	This test procedures verifies whether: • Taxi driver could send a changing-state message • System receives a changing-state message • System manages queues in a proper way • System saves new queues • System check the state of the taxi driver
Procedure Steps	Execute I5T2-I7T2-I7T4 after I3

4 Tools and Test Equipment Required

Integration Testing is not trivial at all, because unlike Unit Testing the testing procedure gets more and more difficult and complex with the addition of every component and requires to consider several differents conditions and interactions. Having said that we think the integration process have to be accompanied using a testing framework. So the most proper choice for this aim is the use of Arquillan, that offers also a useful integration with Unit Testing tools, like JUnit, with the possibility of a coupled tests. Furthermore, as the integration process requires the use of component stubs, we consider a good choice to adopt Mockito Framework to handle and test them, in way to avoid problems related to stubbing and not with integration.

5 Program Stubs

While proceding with integration are written and used several different stubs, useful to simulate functions and behaviours of the lower levels component before proceding with the proper integration. At each integration step are so used sequent stubs:

- Communication
- Taxi
- Request
- Account
- Location
- Queue
- Data