A.Y. 2015/2016

POLITECNICO DI MILANO

COMPUTER SCIENCE AND ENGINEERING

SOFTWARE ENGINEERING 2

MyTaxiService

Integration Test Plan Document

Authors

SARA PISANI – 854223
LEONARDO TURCHI – 853738

21/01/2016

MyTaxiService-TestPlan.pdf · V 1.0

Table of contents

1. Introduction

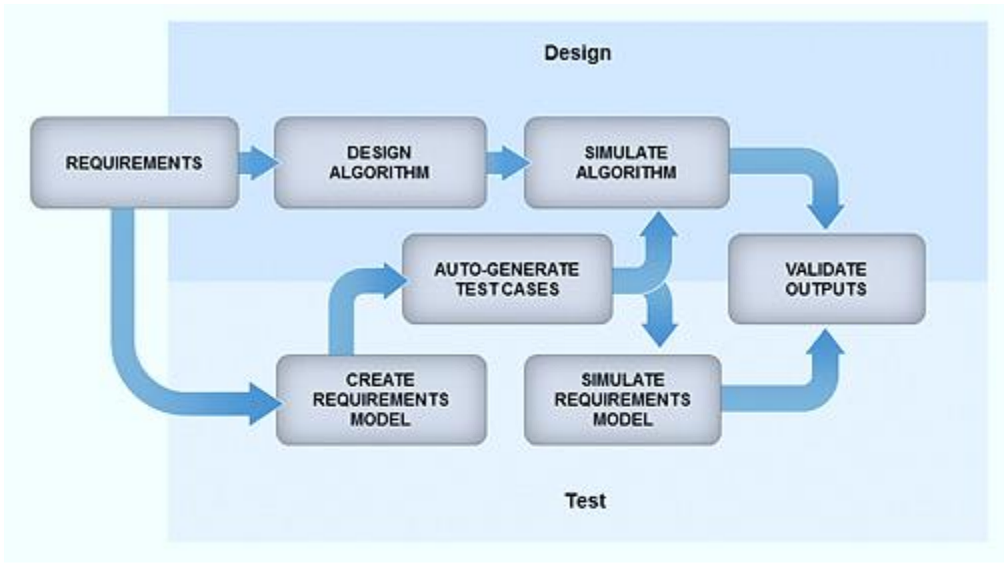
1.1 Revision History

This document is at:

Version: *1*
Revision: *1*
Status: draft (academic document – waiting for approval)

1.2 Purpose and Scope

This document aims to describe the plans for testing the integration of the created components.
The purpose is to test the component's interfaces explained in the RASD and DD attached documents. We start by giving an overview of the entire system, with an highlight of the subsystems to be integrated.



All the application/algorithm/data interfaces will be tested in accordance with the steps described in this document.
In the following, we are going to explain (to the development team) what to test and which tools are needed for testing.
This document is for the development team, to give them the directives of the stuff to be tested.

1.3 List of Definitions and Abbreviations

- ***RASD*** *– Requirements Analysis and Specification Document*
- ***DD*** *– Design Document*
- ***ID*** *– Inspection Document*
- ***ITPD*** *– Integration Test Plan Document*

- ***API*** *– Application Program Interface*
- ***JVM*** *– Java Virtual Machine*
- ***J2EE*** *– Java 2 Enterprise Edition*
- ***DBMS*** *– DataBase Management System*
- ***OS*** *– Operative System*

- ***UML*** *– Unified Modeling Language*
- ***DEV*** *– Development*
- ***IEEE*** *– Institute of Electrical and Electronics Engineers*

1.4 List of Reference Documents

List of all references:

- [*pdf*] Assignment rules and group registration
- [*pdf*] Assignment 4 - Test Plan
- [*pdf*] RASD Document
- [*pdf*] DD Document
- [*pdf*] Inspection Document

2. Integration Strategy

2.1 Entry Criteria

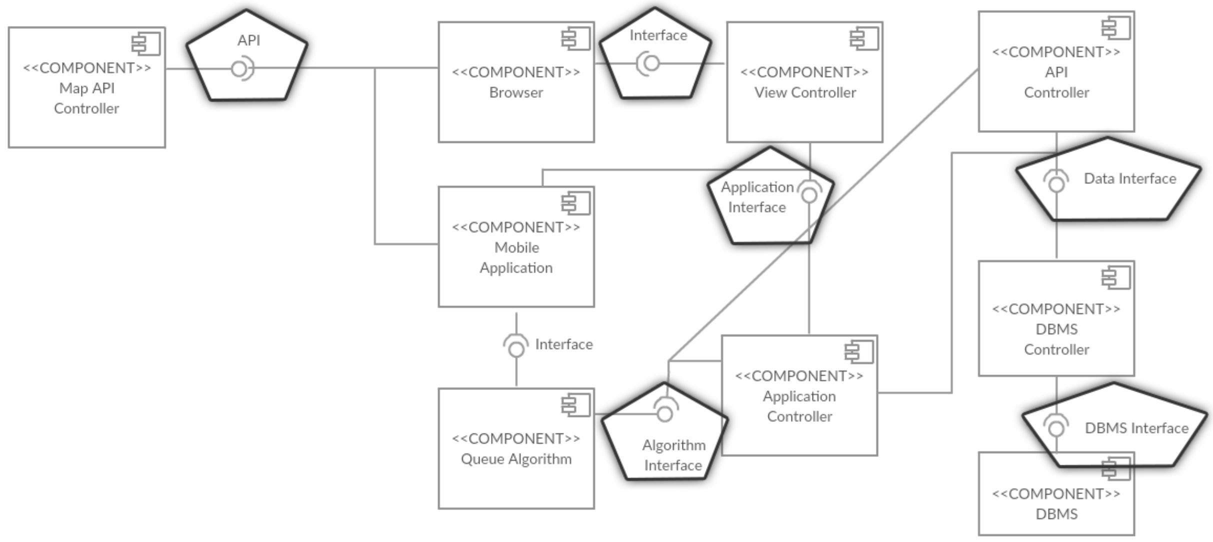
The criteria that must be met, before integration testing of specific elements may begin, are:

- *RASD* and *DD* documents are complete and revised.
- Most of *MyTaxiService* modules must be complete.
  Is not necessary that all code is complete because is possible to test adding code integration (Stub or Drivers, depending on the chosen strategy), but a substantial percentage of components are required to start the testing phase.
- Driver to cover the eventual lack of modules.
- The implementation must satisfy the requirements and the assumptions specified in the RASD document.
- The implementation must be done following the architecture and the design specified in the DD document.

2.2 Elements to be Integrated

The elements that have to be integrated are the ones specified in the Component View in chapter 2.3 of the DD document.
This document explain how they have to be integrated and the order of integration, according to the strategy adopted.

The following component diagram is based on the DD document and is useful to identify the interfaces that create a communication between separate units.
We have to test the highlighted interfaces indicated below:


<<COMPONENT>> Map API Controller
API
<<COMPONENT>> Browser
Interface
<<COMPONENT>> View Controller
<<COMPONENT>> API Controller
Application Interface
Data Interface
<<COMPONENT>> Mobile Application
Interface
<<COMPONENT>> DBMS Controller
<<COMPONENT>> Application Controller
<<COMPONENT>> Queue Algorithm
Algorithm Interface
DBMS Interface
<<COMPONENT>> DBMS

2.3 Integration Testing Strategy

In this case a Bottom-Up strategy should be better: in this approach testing is conducted from sub module to main module, if the main module is not developed a temporary program called Drivers is used to simulate the main module.

Advantages are:

- ✓ Advantageous if major flaws occur toward the bottom of the program.
- ✓ Test conditions are easier to create (develop drivers is simpler than develop stubs).
- ✓ Observation of test results is easier.
- ✓ Bugs are more easily found.
- ✓ Makes it easier to report testing progress in the form of a percentage.
- ✓ Helps to determine the levels of software developed.

Even if driver components have to be created starting from zero, is wrong to consider them as throwaway code, because Drivers can become automated test cases.

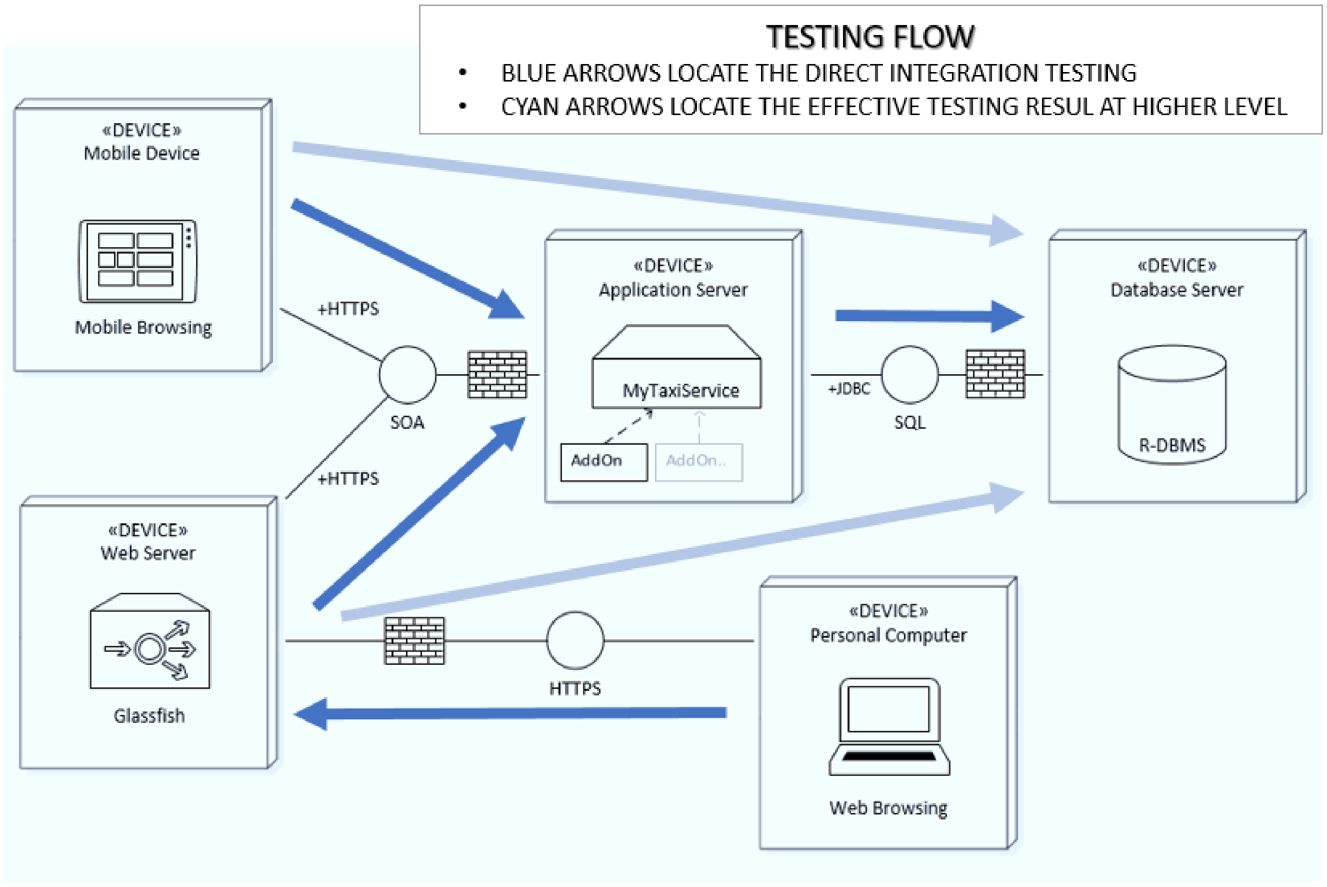
2.4 Sequence of Component/Function Integration

In this section we will identify all the integration sequences of the various modules.
An arrow (A → B) will be used to explain the integration that goes from a component A to a component B, while A *USES* the interface of the component B.

2.4.1 Software Integration Sequence

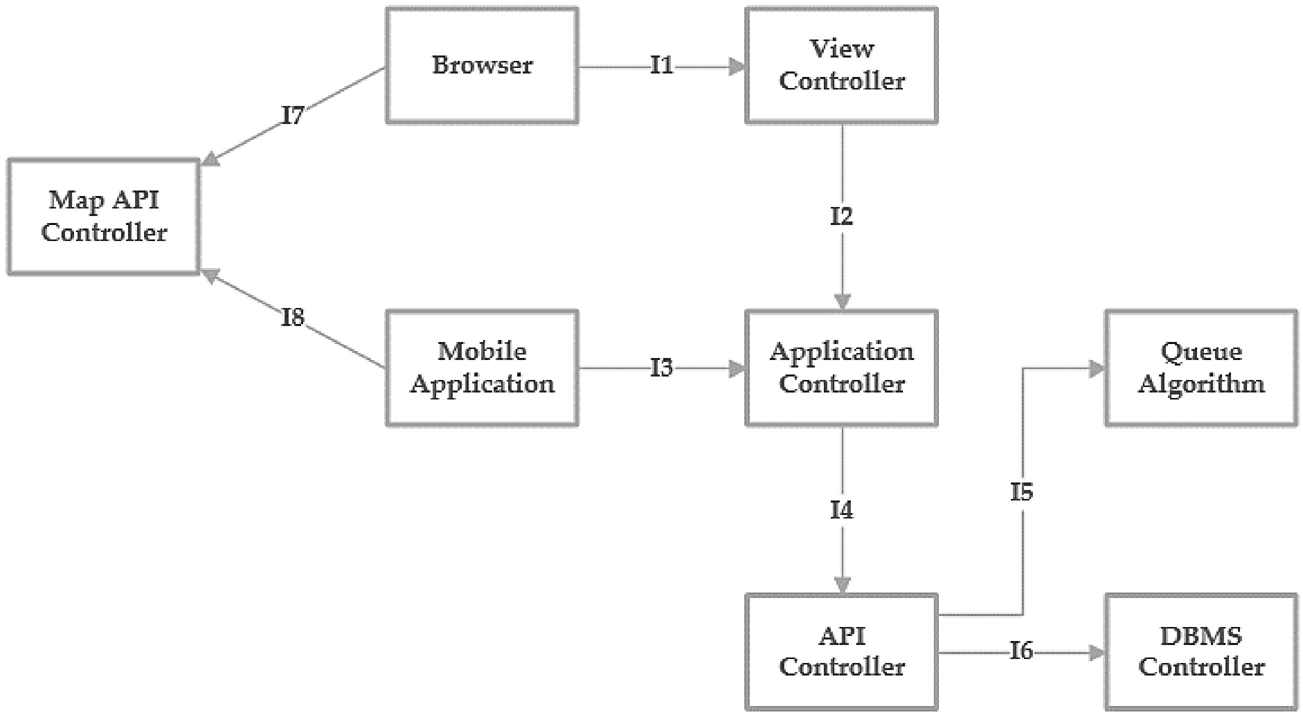
Note*: the system is unique and does not have subsystems; this chapter will be focused on the main system integration*.

In line with the 2.2, we have the highlighted interfaces to test, so we can determine the testing flow.



These interfaces make all the components working through a data communication.
The picture below shows the interaction and the communication between the components, and the integration testing that will be done.

In this case, we recognized 8 test cases, as you can see in the following picture:



ID	Integration Test	Ref
I1	Browser → View Controller	*3.1.1*
I2	View Controller → Application Controller	*3.1.2*
I3	Mobile Application → Application Controller	*3.1.3*
I4	Application Controller → API Controller	*3.1.4*
I5	API Controller → Queue Algorithm	*3.1.5*
I6	API Controller → DBMS Controller	*3.1.6*
I7	Browser → Map API Controller	*3.1.7*
I8	Mobile Application → Map API Controller	*3.1.8*

3. Individual Steps and Test Description

3.1 Integration Test Cases

3.1.1 Integration Test – Case I1

Test case identifier	**I1**
Test items	Browser → View Controller
Input specification	login simulation, authentication data, page navigation, input buttons, page details view, 'back' functionality (all the requests coming from the browser) [RASD 3.5.3]
Output specification	Correct response for each type of request. No browser's visualization issue. No connection error (GET requests check).
Environmental needs	I2 succeeded.

3.1.2 Integration Test – Case I2

Test case identifier	**I2**
Test items	View Controller → Application Controller
Input specification	request for a page, request for a response, execution request for a login, execution request for a 'new ride' page's data, correct popup visualization dispatch (all the requests coming from the view)
Output specification	Correct response for each type of request. No error or deadlock.
Environmental needs	I4 succeeds. The view controller must be available and in correct communication with the application controller to dispatch the requests. This test case is similar to the I3.

3.1.3 Integration Test – Case I3

Test case identifier	**I3**
Test items	Mobile Application → Application Controller
Input specification	login simulation, authentication, various app section navigation, response for an input (all the requests coming from the app) [RASD 3.5.3]
Output specification	Correct response for each type of request. No communication error (http request check).
Environmental needs	I4 succeeds. The application controller must be reachable and in correct communication with the DBMS controller to dispatch all the requests. Database data available (even simulated data). This test case is similar to the I2.

3.1.4 Integration Test – Case I4

Test case identifier	**I4**
Test items	Application Controller → API Controller
Input specification	method call for data get or queue management, forwarding query, responding with data, stress test with many simultaneous requests
Output specification	Correct forwarding of requests, no error thrown. No communication error (GET/POST request check).
Environmental needs	The API controller must be available.

3.1.5 Integration Test – Case I5

Test case identifier	**I5**
Test items	API Controller → Queue Algorithm
Input specification	request for a queue management, execution of a enqueue/dequeue/migration method [DD 3.2 – 3.3]
Output specification	Correct execution of commands. Real change of the queue, without exception or overflows.
Environmental needs	I4 succeeds. The API controller must be ready. Queue data structure must exist, must be available and ready (even with simulated data).

3.1.6 Integration Test – Case I6

Test case identifier	**I6**
Test items	API Controller → DBMS Controller
Input specification	database authentication, request for data, *INSERT* - *SELECT* - *UPDATE* commands (all commands for a correct DB management)
Output specification	Correct commands execution, no exception thrown. No connection error to DBMS.
Environmental needs	I4 succeeds. The API controller must be ready. DBMS controller must be available. The DBMS core (data structure) must be available and ready to serve data (even simulated data).

3.1.7 Integration Test – Case I7

Test case identifier	**I7**
Test items	Browser → Map API Controller
Input specification	get coordinates from street or address, get map thumbnail from street, calculate approximate trip time, check correctness of an address, check if address is inside an area [DD 4.1 – 4.2]
Output specification	Correct dispatch for all requests, no error thrown. API-Key for the public service (i.e. G.Maps) are valid for many simultaneous requests. No connection errors.
Environmental needs	I1, I2 succeeds. Public Map-API service is available and reachable. This test case is similar to the I8.

3.1.8 Integration Test – Case I8

Test case identifier	**I8**
Test items	Mobile Application → Map API Controller
Input specification	check coordinates correctness, get position from coordinates, get street/zone from coordinates, get map thumbnail from coordinates, calculate approximate trip time, check correctness of an address, check if address is inside an area [DD 4.1 – 4.2]
Output specification	Correct dispatch for all requests, no error thrown. API-Key for the public service (i.e. G.Maps) are valid for many simultaneous requests. No connection errors.
Environmental needs	I3 succeeds. Public Map-API service is available and reachable. This test case is similar to the I7.

3.2 Test Procedures

3.2.1 Integration Test – Procedure TP1

Test procedure identifier	**TP1**
Purpose	User Registration and User Login [RASD 3.5 – 3.5.3] [DD 2.5]
Check that	The system can handle a registration or login process. The system promptly response to all request without loss of data or deadlock. The system generates correct data in the view.
Procedure steps	Execute I1 > I2 > I4 > I6 Execute I3 > I4 > I6

3.2.2 Integration Test – Procedure TP2

Test procedure identifier	**TP2**
Purpose	Ride Request and Ride Booking [RASD 3.5 – 3.5.3] [DD 2.5]
Check that	The system can handle a ride request or a ride booking from the user. The system correctly forward all the requests through the right components. There is a right use of the database and the queue algorithm controllers. There is a correct communication between all the components.
Procedure steps	Execute I1 > I2 Execute I3 Execute I4 > I5 > I6 > I7 > I8

3.2.3 Integration Test – Procedure TP3

Test procedure identifier	**TP3**
Purpose	Driver Area Management [RASD 3.5 – 3.5.3] [DD 2.5]
Check that	The system can handle a driver choice (e.g. a new ride confirmation). The system correctly forward all the requests through the right components. There is a right use of the database and the queue algorithm controllers. There is a correct update of the taxis position and its forwarding to the server. There is a correct update of the queue. There is a correct communication between all the components.
Procedure steps	Execute I3 > I4 > I5 > I6 > I8

4. Tools and Test Equipment Required

4.1 Mockito and Junit

Mockito is an open source testing framework for Java.

The framework allows the creation of test double objects (mock objects) in automated unit tests.
Testing code is a problem when different parts of code work together: unit testing does not want to test fragments of code that collaborate with other units (e.g. DBMS or external services).

MyTaxiService application is not composed by isolated elements: there is a dialogue between components and the DBMS or the net (API) and components have a dependance on environment.

Using mock Object is possible to simulate collaborators and allow to do "Integration testing" albeit isolating the block that must be tested, so Junit will be used with the aid of a Mock Objects framework: Mockito.

Mock Object advantages:

- ✓ They must not be written but declared
- ✓ They are refactoring sensitive
- ✓ They return values
- ✓ They launch exceptions
- ✓ They verify the invocation order
- ✓ They verify the number of calling to a method

4.2 Arquillan and Shrink Wrap

Arquillan is 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.

Arquillian brings the test to the runtime so you don't have to manage the runtime from the test (or the build). Arquillian eliminates this burden by covering all aspects of test execution, which entails:

- ✓ Managing the lifecycle of the container (or containers)
- ✓ Bundling the test case, dependent classes and resources into a ShrinkWrap archive (or archives)
- ✓ Deploying the archive (or archives) to the container (or containers)
- ✓ Enriching the test case by providing dependency injection and other declarative services
- ✓ Executing the tests inside (or against) the container
- ✓ Capturing the results and returning them to the rest runner for reporting

The aim of test archive is to isolate all the classes and resources used to make the test from the rest of classpath. unlike a normal unit test, Arquillian's test does not import all the project classpath: will be included only the part that the test needs. the archive is defined using ShrinkWrap, that is a series of Java API aimed to create Java archives (e.g. jar, war, ear). The strategy of micro-deployment allows to focus on the classes that will be tested. The test will be very light and handy.

4.3 Apache Jmeter

The Apache JMeter application is open source software, a 100% pure Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.

Apache JMeter may be used to test performance both on static and dynamic resources (Webservices (SOAP/REST), Web dynamic languages - PHP, Java, ASP.NET, Files, etc. -, Java Objects, Data Bases and Queries, FTP Servers and more). It can be used to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyze overall performance under different load types. It makes a graphical analysis of performance or test the server/script/object behavior under heavy

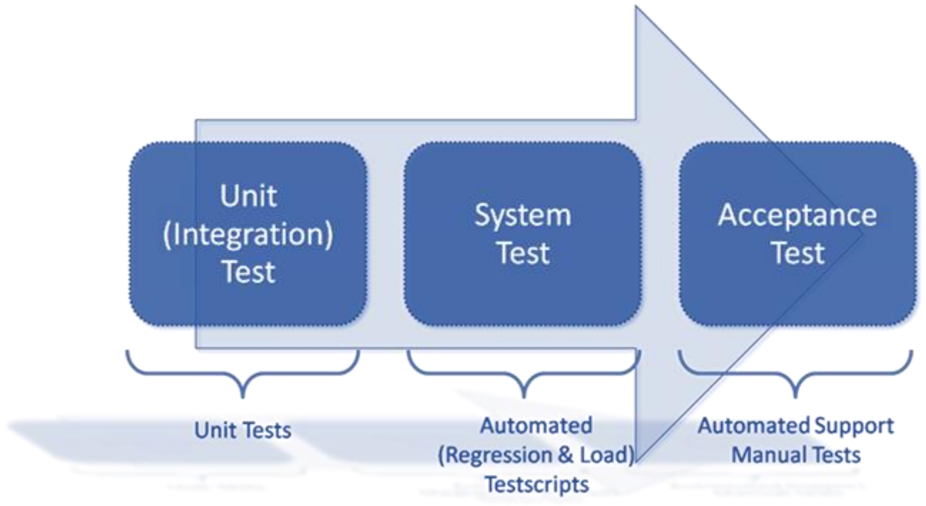
concurrent load.
Apache JMeter features include:

- ✓ Ability to load and performance test many different server/protocol types.
- ✓ Complete portability and 100% Java purity.
- ✓ Full multithreading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by separate thread groups.
- ✓ Careful GUI design allows faster Test Plan building and debugging.
- ✓ Caching and offline analysis/replaying of test results.
- ✓ Highly Extensible core (Pluggable Samplers allow unlimited testing capabilities, several load statistics may be chosen with pluggable timers, data analysis and visualization plugins allow great extensibility as well as personalization, functions can be used to provide dynamic input to a test or provide data manipulation, Scriptable Samplers).

4.4 Manual 'beta' testing

The Manual Testing includes the testing of the Software manually, without using any automated tool or any script. This technique is a preliminary testing, must be carried out prior to start automating the test cases and also needs to check the feasibility of automation testing.

In the test phase, the 100% of automation is not possible so the Manual Testing is very important.


Unit (Integration) Test
System Test
Acceptance Test
Unit Tests
Automated (Regression & Load) Testscripts
Automated Support Manual Tests

Differences between Manual Testing and Automation Testing:

Manual testing will be used when the test case only needs to runs once or twice.	Automation testing will be used when need to execute the set of test cases tests repeatedly.
Manual testing will be very useful while executing test cases first time & may or may not be powerful to catch the regression defects under frequently changing requirements.	Automation testing will be very useful to catch regressions in a timely manner when the code is frequently changes.
Simultaneously testing on different machine with different OS platform combination is not possible using manual testing. To execute such task separate testers are required.	Automation testing will be carried out simultaneously on different machine with different OS platform combination.
No programming can be done to write sophisticated tests which fetch hidden information.	Using Automation testing, Testers can program complicated tests to bring out of sight information.

4.5 Other software and tools used

- Microsoft Word: to redact this document.
- GitHub: to share the material of this project.
- VirtualBox: the open virtual machine framework.
- Eclipse Luna: to open Glassfish’s code.
- Java/Glassfish Online Documentation.

5. Program Stubs and Test Data Required

5.1 User simulator Driver

In order to carry out tests I1, I2, I3, I7 and I8, a driver able to simulate users' behavior is necessary.

- ✓ For the first 3 cases this driver must imitate, depending on the situation, a driver or a passenger and this is important because different pages will be shown to drivers and to passengers after login: the operation of the app must be tested in both cases.
- ✓ For the last two cases this driver allows the insertion of an address (via browser or via app) to make a ride request as if it were a passenger who needs a taxi.

5.2 Googe API simulator Driver

In the test cases I7 and I8 must exist a driver that provides GPS coordinates, simulating the Google API answer to the system request.

5.3 DBMS manager Stub

During the testing phase, direct operations with the DB are risky: a useful solution could be the creation of a DBMS manager stub, that can be filled with all necessary information and will answer to queries with these data.

The convenience could be found

- ✓ in the test I5: to test the functioning of the queue algorithm, a queue of taxis must be in the database. It will be built and inserted in the DBMS manager stub to avoid problems with the DB.
- ✓ In the test I6: to test the behavior of the DBMS Controller, several access to the database must be made. Diverting all these operations on the DBMS manager stub, problems with the DB will be avoided.

6. Hours of works

Here is the time spent for redact this document:

[sum of hours spent by team's members]
+2h (14/01)
+6h (15/01)
+2h (18/01)
+6h (19/01)
+6h (20/01)
+8h (21/01)

TOTAL ~ 30 hours

- Leonardo Turchi: ~ 15 hours
- Sara Pisani: ~ 15 hours