Integration Test Plan Document

Version 1.0

Alberto Mario Pirovano Alessandro Vetere

21/01/2016

POLITECNICO
MILANO 1863

Contents

1 Introduction

This section contains a brief introduction to the **I**ntegration **T**est **P**lan **D**ocument.

1.1 Purpose and Scope

This document is mainly based on the **D**esign **D**ocument. In fact the purpose of the **I**ntegration **T**est **P**lan **D**ocument is to clearly state the order in which the software components identified in the **Component View** of the **DD** have to be integrated one with each other in order to guarantee a well tested final software. Following the exposed procedure ensures that all the software components will communicate and cooperate in the proper way.

1.2 List of Definitions and Abbreviations

In the document are often used some technical terms whose definitions are here reported:

- **Integration Test Case:** An atomic procedure done to test the integration of a component on the top of another one.
- **Integration Test Suite:** A collection of **Integration Test Cases**.
- See the correspondent section in the **RASD** and the **DD** for more definitions.

For sake of brevity, some acronyms and abbreviations are used:

- **ITPD:** Integration Test Plan Document.
- **In:** Integration Test Suite number n.
- **InTm:** Integration Test Case number m of the Integration Test Suite number n.
- **JS:** JavaScript.
- **UI:** User Interface.
- See the correspondent section in the **RASD** and the **DD** for more acronyms and abbreviations.

1.3 List of Reference Documents

- ***myTaxiService*** **RASD v1.3:** Requirements Analysis and Specification Document
- ***myTaxiService*** **DD v1.1:** Design Document
- **Assignments 4 - Test Plan:** Integration testing assignment specification document

2 Integration Strategy

This section explains the integration strategy selected to integrate all the software elements of *myTaxiService*.

2.1 Entry Criteria

Before starting the integration testing of any software component that has been designed for *myTaxiService* system, the internal functions of the considered component (i.e. public or protected methods that are exposed within the package of the component but are not part of any external public interface) must be unit tested using an appropriate framework.

2.2 Elements to be Integrated

Every software component in the Component View section of the Design Document has to be integrated. The interested components are reported below for readers' convenience:

- Passenger View
  - PS Application View
  - PS Web View
  - PS Request Creator
  - PS Receiver
- Taxi Driver View
  - TD Application View
  - TD Request Creator
  - TD Receiver
  - TD Locator
  - GPS Data Source
- Administrator View
- Controller
  - RESTful Service
  - Dispatcher
  - Location Manager
  - Session Manager
  - Profile Manager
  - Taxi Driver Manager
  - Taxi Ride Manager

  - Queue Manager
  - Taxi Sharing Manager
  - Query Manager
- Model
  - Model Query Service
  - Passenger DB Adapter
  - Taxi Ride DB Adapter
  - Zone DB Adapter
  - Queue DB Adapter
  - Database Driver Adapter

Moreover we suppose that Google Maps API are well tested by Google and thus we can use them without testing any further. For what concerns the other external software elements, we assume that the GPS Data Source module in the Taxi Driver View uses the GPS Drivers of the underlying operating system that are already tested, and the same is assumed for the Database Driver Adapter in the Model referring to Database Drivers.

2.3 Integration Testing Strategy

The bottom-up integration testing approach has been chosen, because for a medium sized project like *myTaxiService*, it is best to proceed step by step in a careful yet coherent integration strategy. The usage of the selected approach will forge a robust application with efforts concentrated in testing the **Server** parts before all. Therefore, a more stable and faster application will be distributed, instead of a maybe nicer one.

2.4 Sequence of Component/Function Integration

Here is presented how the bottom-up integration testing approach is going to be concretely used in *myTaxiService* system integration, first by analyzing each subsystem in detail and then giving a higher level overview on the subsystems integration process. A precise convention has been adopted for the semantic of diagrams:

- **Block:**
  - **Yellow:** This block is not dependent on any lower level component in *myTaxiService* and therefore it is integrated as a starting point in the current diagram.
  - **Blue:** This block is going to be fully integrated on the top of its parents.

  - **Green:** This block is not going to be fully integrated within the current diagram but needs further integration testing in subsequent diagrams.
  - **Red:** This block represents a stub component, that replaces the real component mocking its functionalities. That is a trick made necessary by the fact that some components are going to be integrated before some other needed components have been integrated, and therefore they require a stub in order to work. The component represented by a red block is going to be integrated afterwards.

- **Arrow:** It is a **precedence** symbol. It helps the tester to follow the right order in the whole integration process. It starts from a block and ends into another block. The block from which it starts is called **parent** and the other one **child**. In particular it means that the child block can be integrated only if its parents are already integrated. Moreover if a block is pointed by several arrows, its integration process can begin only when **all the parent blocks are integrated**.

2.4.1 Subsystem Integration Sequence

We adopted a **bottom-up** testing strategy; this strategy starts from the testing of the **Model** features.

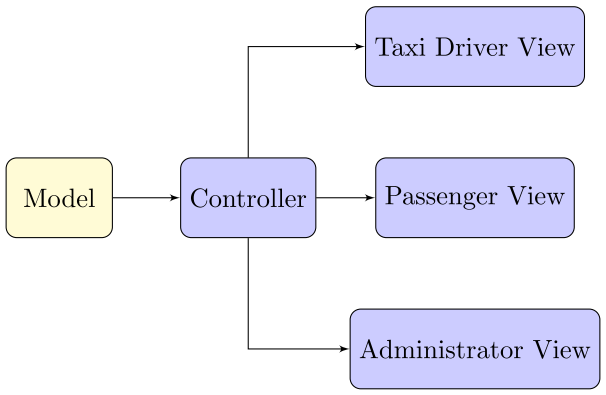

Figure 1: Software Integration Sequence Diagram

We considered **Model**, **Controller** and **Views** as **Subsystems**. First the *tester* has to test *functionalities* of the components contained in the **Model**, then, in order to simulate the behavior of the interacting software elements, he has to represent the **Controller** with a dedicated **Driver** in order to test the relevant **Model** features. In this way, the **Model** will be completely integration tested by simulating all the possible actions that the **Controller** can do on it.

Then the testing procedure passes to the **Controller** and the test sequence adopts the same strategy used to test the **Model**. The only difference from a higher point of view, is that in this case the **Controller** is tested using the already tested **Model** and using a **Driver** in order to simulate the **Views** actions.

The last part of the procedure is dedicated to the **Views**. The **Taxi Driver View, Passenger View and Administrator View** are tested using the already tested **Controller** and **Model** and no component behavior is simulated using **Stubs** or **Drivers**.

2.4.2 Software Integration Sequence

We provided six **Software Integration Sequence Diagram**, one for each main part of *myTaxiService*.

Referring to the diagram semantic explanation given in section 2.4, the reader can understand the meaning of each diagram. Further details about each testing step (represented as arrows), are then given in the following section.

2.4.2.1 Model Integration Sequence

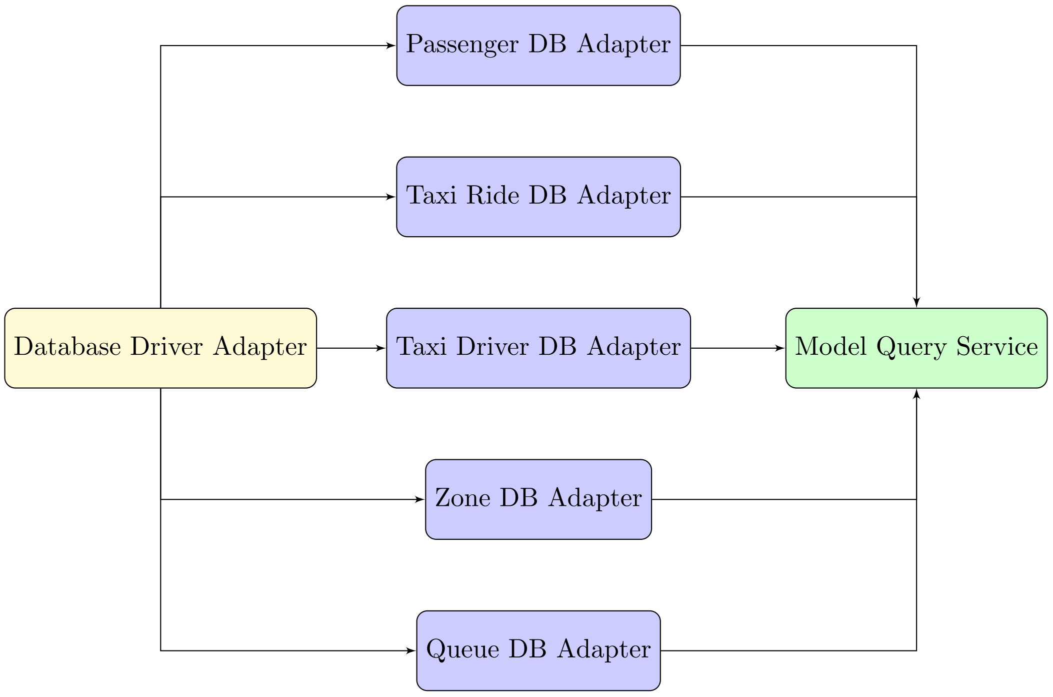

Figure 2: Software Integration Sequence Diagram - Model

2.4.2.2 Controller Business Components Integration Sequence

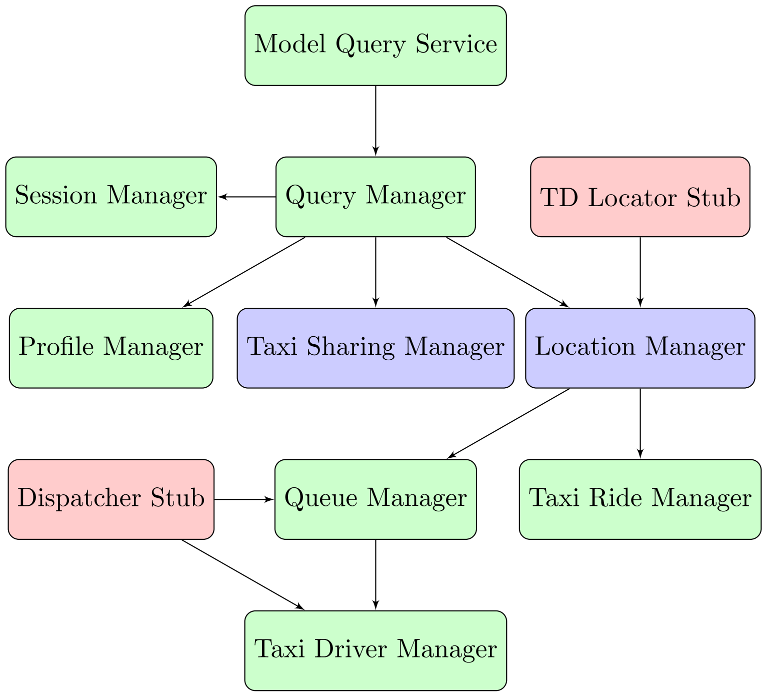

Figure 3: Software Integration Sequence Diagram - Controller Business Components

2.4.2.3 Controller Networking Components Integration Sequence

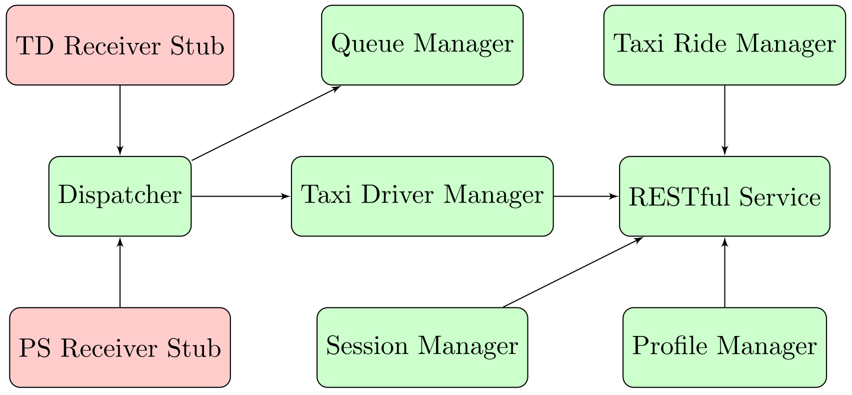

Figure 4: Software Integration Sequence Diagram - Controller Networking Components

2.4.2.4 Passenger View Integration Sequence

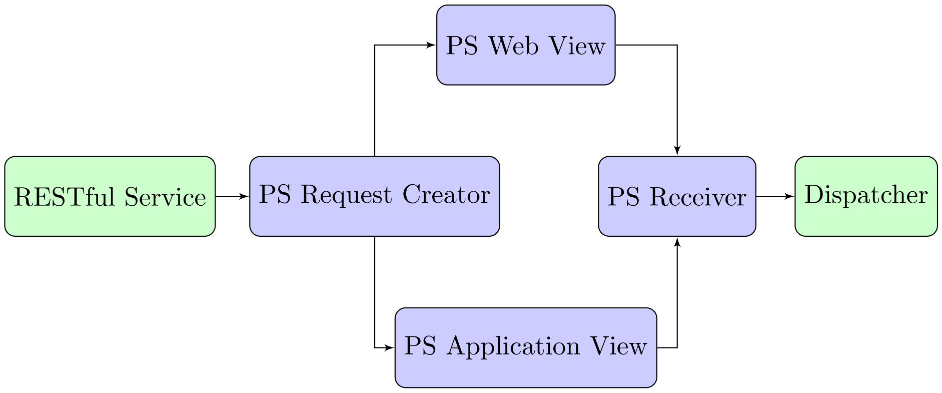

Figure 5: Software Integration Sequence Diagram - Passenger View

2.4.2.5 Taxi Driver View Integration Sequence

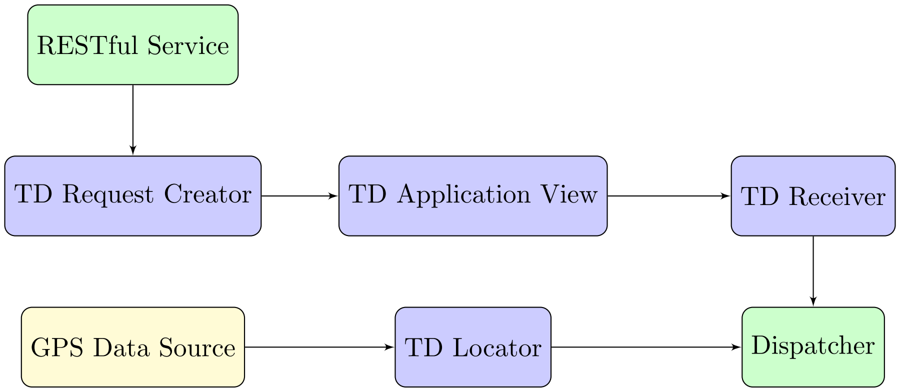

Figure 6: Software Integration Sequence Diagram - Taxi Driver View

2.4.2.6 Administrator View Integration Sequence

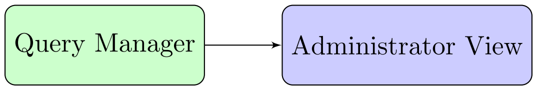

Figure 7: Software Integration Sequence Diagram - Administrator View

3 Individual Steps and Test Description

This section is divided into six parts, each one containing the relevant tests of a different part of *myTaxiService*. For each part we defined many **Integration Test Suites**. The title we assigned to each **Integration Test Suite** contains the name of the software component on which top the integration of other components is done. Furthermore, for each **Integration Test Suite** we define one or more specific **Integration Test Cases**. Each **Integration Test Case** allows the integration of another software component on the top of the one mentioned in the **Integration Test Suite** title.

Refer to section 5 to check which kind of sample data must be pre-inserted in *myTaxiService* database in order to permit the whole testing procedure.

3.1 Model Integration Test Cases

3.1.1 I1 - Database Driver Adapter

3.1.1.1 Test Case T1

Test Item(s)	Passenger DB Adapter → Database Driver Adapter
Input Specification	Create typical input for Passenger DB Adapter
Output Specification	Check if correct methods are called in Database Driver Adapter
Environmental Needs	Sample data present in the DB
Target	Verify that Passenger DB Adapter works

3.1.1.2 Test Case T2

Test Item(s)	Taxi Ride DB Adapter → Database Driver Adapter
Input Specification	Create typical input for Taxi Ride DB Adapter
Output Specification	Check if correct methods are called in Database Driver Adapter
Environmental Needs	Sample data present in the DB
Target	Verify that Taxi Ride DB Adapter works

3.1.1.3 Test Case T3

Test Item(s)	Taxi Driver DB Adapter → Database Driver Adapter
Input Specification	Create typical input for Taxi Driver DB Adapter
Output Specification	Check if correct methods are called in Database Driver Adapter
Environmental Needs	Sample data present in the DB
Target	Verify that Taxi Driver DB Adapter works

3.1.1.4 Test Case T4

Test Item(s)	Zone DB Adapter → Database Driver Adapter
Input Specification	Create typical input for Zone DB Adapter
Output Specification	Check if correct methods are called in Database Driver Adapter
Environmental Needs	Sample data present in the DB
Target	Verify that Zone DB Adapter works

3.1.1.5 Test Case T5

Test Item(s)	Queue DB Adapter → Database Driver Adapter
Input Specification	Create typical input for Queue DB Adapter
Output Specification	Check if correct methods are called in Database Driver Adapter
Environmental Needs	Sample data present in the DB
Target	Verify that Queue DB Adapter works

3.1.2 I2 - Passenger DB Adapter

3.1.2.1 Test Case T1

Test Item(s)	Model Query Service → Passenger DB Adapter
Input Specification	Create Passenger related input for Model Query Service
Output Specification	Check if correct methods are called in Passenger DB Adapter
Environmental Needs	I1T1
Target	Verify that Model Query Service works

3.1.3 I3 - Taxi Ride DB Adapter

3.1.3.1 Test Case T1

Test Item(s)	Model Query Service → Passenger DB Adapter
Input Specification	Create Taxi Ride related input for Model Query Service
Output Specification	Check if correct methods are called in Passenger DB Adapter
Environmental Needs	I1T2
Target	Verify that Model Query Service works

3.1.4 I4 - Taxi Driver DB Adapter

3.1.4.1 Test Case T1

Test Item(s)	Model Query Service → Taxi Driver DB Adapter
Input Specification	Create Taxi Driver related input for Model Query Service
Output Specification	Check if correct methods are called in Taxi Driver DB Adapter
Environmental Needs	I1T3
Target	Verify that Model Query Service works

3.1.5 I5 - Zone DB Adapter

3.1.5.1 Test Case T1

Test Item(s)	Model Query Service → Zone DB Adapter
Input Specification	Create Zone related input for Model Query Service
Output Specification	Check if correct methods are called in Zone DB Adapter
Environmental Needs	I1T4
Target	Verify that Model Query Service works

3.1.6 I6 - Queue DB Adapter

3.1.6.1 Test Case T1

Test Item(s)	Model Query Service → Queue DB Adapter
Input Specification	Create Queue related input for Model Query Service
Output Specification	Check if correct methods are called in Queue DB Adapter
Environmental Needs	I1T5
Target	Verify that Model Query Service works

3.2 Controller Business Integration Test Cases

3.2.1 I7 - Model Query Service

3.2.1.1 Test Case T1

Test Item(s)	Query Manager → Model Query Service
Input Specification	Create typical input for Query Manager
Output Specification	Check if correct methods are called in Model Query Service
Environmental Needs	I2, I3, I4, I5, I6
Target	Verify that Query Manager works

3.2.2 I8 - Query Manager - 1

3.2.2.1 Test Case T1

Test Item(s)	Session Manager → Query Manager
Input Specification	Create typical input for Session Manager
Output Specification	Check if correct methods are called in Query Manager
Environmental Needs	I7
Target	Verify that Session Manager works

3.2.2.2 Test Case T2

Test Item(s)	Profile Manager → Query Manager
Input Specification	Create typical input for Profile Manager
Output Specification	Check if correct methods are called in Query Manager
Environmental Needs	I7
Target	Verify that Profile Manager works

3.2.2.3 Test Case T3

Test Item(s)	Taxi Sharing Manager → Query Manager
Input Specification	Create typical input for Taxi Sharing Manager
Output Specification	Check if correct methods are called in Query Manager
Environmental Needs	I7
Target	Verify that Taxi Sharing Manager works

3.2.2.4 Test Case T4

Test Item(s)	Location Manager → Query Manager
Input Specification	Create typical input for Location Manager
Output Specification	Check if correct methods are called in Query Manager
Environmental Needs	I7, TD Locator Stub
Target	Verify that Location Manager works

3.2.3 I9 - Location Manager

3.2.3.1 Test Case T1

Test Item(s)	Queue Manager → Location Manager
Input Specification	Create typical input for Queue Manager
Output Specification	Check if correct methods are called in Location Manager
Environmental Needs	I8, Dispatcher Stub
Target	Verify that Queue Manager works

3.2.3.2 Test Case T2

Test Item(s)	Taxi Ride Manager → Location Manager
Input Specification	Create typical input for Taxi Ride Manager
Output Specification	Check if correct methods are called in Location Manager
Environmental Needs	I8
Target	Verify that Taxi Ride Manager works

3.2.4 I10 - Queue Manager

3.2.4.1 Test Case T1

Test Item(s)	Taxi Driver Manager → Queue Manager
Input Specification	Create typical input for Taxi Driver Manager
Output Specification	Check if correct methods are called in Queue Manager
Environmental Needs	I9, Dispatcher Stub
Target	Verify that Taxi Driver Manager works

3.3 Controller Networking Integration Test Cases

3.3.1 I11 - Dispatcher

3.3.1.1 Test Case T1

Test Item(s)	Queue Manager → Dispatcher
Input Specification	Create typical input for Queue Manager
Output Specification	Check if correct methods are called in Dispatcher
Environmental Needs	I10, TD Receiver Stub, PS Receiver Stub
Target	Verify that Queue Manager works

3.3.1.2 Test Case T2

Test Item(s)	Taxi Driver Manager → Dispatcher
Input Specification	Create typical input for Taxi Driver Manager
Output Specification	Check if correct methods are called in Dispatcher
Environmental Needs	TD Receiver Stub, PS Receiver Stub
Target	Verify that Taxi Driver Manager works

3.3.2 I12 - Taxi Driver Manager

3.3.2.1 Test Case T1

Test Item(s)	RESTful Service → Taxi Driver Manager
Input Specification	Create typical input for RESTful Service
Output Specification	Check if correct methods are called in Taxi Driver Manager
Environmental Needs	I11
Target	Verify that RESTful Service works

3.3.3 I13 - Taxi Ride Manager

3.3.3.1 Test Case T1

Test Item(s)	RESTful Service → Taxi Ride Manager
Input Specification	Create typical input for RESTful Service
Output Specification	Check if correct methods are called in Taxi Ride Manager
Environmental Needs	I9
Target	Verify that RESTful Service works

3.3.4 I14 - Session Manager

3.3.4.1 Test Case T1

Test Item(s)	RESTful Service → Session Manager
Input Specification	Create typical input for RESTful Service
Output Specification	Check if correct methods are called in Session Manager
Environmental Needs	I8
Target	Verify that RESTful Service works

3.3.5 I15 - Profile Manager

3.3.5.1 Test Case T1

Test Item(s)	RESTful Service → Profile Manager
Input Specification	Create typical input for RESTful Service
Output Specification	Check if correct methods are called in Profile Manager
Environmental Needs	I8
Target	Verify that RESTful Service works

3.4 Passenger View Integration Test Cases

3.4.1 I16 - RESTful Service - 1

3.4.1.1 Test Case T1

Test Item(s)	PS Request Creator → RESTful Service
Input Specification	Create typical input for PS Request Creator
Output Specification	Check if correct methods are called in RESTful Service
Environmental Needs	Nothing
Target	Verify that PS Request Creator works

3.4.2 I17 - PS Request Creator

3.4.2.1 Test Case T1

Test Item(s)	PS Web View → PS Request Creator
Input Specification	Create sample user interaction for PS Web View
Output Specification	Check if correct methods are called in PS Request Creator
Environmental Needs	I16
Target	Verify that PS Web View works

3.4.2.2 Test Case T2

Test Item(s)	PS Application View → PS Request Creator
Input Specification	Create sample user interaction for PS Application View
Output Specification	Check if correct methods are called in PS Request Creator
Environmental Needs	I16
Target	Verify that PS Application View works

3.4.3 I18 - PS Web View

3.4.3.1 Test Case T1

Test Item(s)	PS Receiver → PS Web View
Input Specification	Create typical input for PS Receiver
Output Specification	Check if correct methods are called in PS Web View and the UI is correctly updated
Environmental Needs	I17
Target	Verify that PS Receiver works

3.4.4 I19 - PS Application View

3.4.4.1 Test Case T1

Test Item(s)	PS Receiver → PS Application View
Input Specification	Create typical input for PS Receiver
Output Specification	Check if correct methods are called in PS Application View and the UI is correctly updated
Environmental Needs	I17
Target	Verify that PS Receiver works

3.4.5 I19 - PS Receiver

3.4.5.1 Test Case T1

Test Item(s)	Dispatcher → PS Receiver
Input Specification	Create typical input for Dispatcher
Output Specification	Check if correct methods are called in PS Receiver
Environmental Needs	I18, I19
Target	Verify that Dispatcher works

3.5 Taxi Driver View Integration Test Cases

3.5.1 I20 - RESTful Service - 2

3.5.1.1 Test Case T1

Test Item(s)	TD Request Creator → RESTful Service
Input Specification	Create typical input for TD Request Creator
Output Specification	Check if correct methods are called in RESTful Service
Environmental Needs	I12, I13, I14, I15
Target	Verify that TD Request Creator works

3.5.2 I21 - TD Request Creator

3.5.2.1 Test Case T1

Test Item(s)	TD Application View → TD Request Creator
Input Specification	Create user interaction for TD Application View
Output Specification	Check if correct methods are called in TD Request Creator
Environmental Needs	I20
Target	Verify that TD Application View works

3.5.3 I22 - TD Application View

3.5.3.1 Test Case T1

Test Item(s)	TD Receiver → TD Application View
Input Specification	Create typical input for TD Receiver
Output Specification	Check if correct methods are called in TD Application View and the UI is correctly updated
Environmental Needs	I21
Target	Verify that TD Receiver works

3.5.4 I23 - TD Receiver

3.5.4.1 Test Case T1

Test Item(s)	Dispatcher → TD Receiver
Input Specification	Create typical input for Dispatcher
Output Specification	Check if correct methods are called in TD Receiver
Environmental Needs	I22
Target	Verify that Dispatcher works

3.5.5 I24 - GPS Data Source

3.5.5.1 Test Case T1

Test Item(s)	TD Locator → GPS Data Source
Input Specification	Create typical input for TD Locator
Output Specification	Check if correct methods are called in GPS Data Source
Environmental Needs	Sample GPS Data
Target	Verify that TD Locator works

3.5.6 I24 - TD Locator

3.5.6.1 Test Case T1

Test Item(s)	Dispatcher → TD Locator
Input Specification	Create typical input for Dispatcher
Output Specification	Check if correct methods are called in TD Locator
Environmental Needs	I24
Target	Verify that Dispatcher works

3.6 Administrator View Integration Test Cases

3.6.1 I25 - Query Manager - 2

3.6.1.1 Test Case T1

Test Item(s)	Administrator View → Query Manager
Input Specification	Create typical input for Administrator View
Output Specification	Check if correct methods are called in Query Manager
Environmental Needs	I7
Target	Verify that Administrator View works

4 Tools and Test Equipment Required

The following part of the document contains a set of recommended software that can be used to implement the concrete procedure of testing. Moreover, because the high-level architecture proposed in the **DD** is designed using a **Java-based** style, the programming language that better adapts to this style is **Java**, but a lot of other emerging languages can be used in order to build a proper software, like **JavaScript**.

If it is decided to use **Java**, the proposed and well-known tools are:

- **JUnit:** Unit testing framework.
  - http://junit.org/
- **Mockito:** Another unit testing framework.
  - http://site.mockito.org/
- **Arquillian:** Integration testing framework.
  - http://arquillian.org/
- **Espresso:** Android UI testing automation.
  - http://developer.android.com/training/testing/ui-testing/espresso-testing.html

Furthermore, if **JavaScript** is used, these tools can be useful:

- **Karma JS:** A simple tool that allows you to execute JavaScript code in multiple real browsers.
  - https://karma-runner.github.io/0.13/index.html
- **Mocha JS:** A flexible, fun JavaScript test framework for node.js and the browser.
  - https://mochajs.org/

5 Program Stubs and Test Data Required

We assume that the **Integration testing** comes after **Developing** and **Unit testing**. In this way we don't need any **Driver** because the software components are already developed.

On the other hand we need few **Stubs** in order to make the not yet integrated components work, because we want to respect the **Bottom-Up** strategy.

We decided to use **Stubs** in these **Integration Test Cases**:

- **I8T4:** TD Locator Stub
- **I9T1:** Dispatcher Stub
- **I10T1:** Dispatcher Stub
- **I11T1, I11T2:** TD Receiver Stub, PS Receiver Stub

To better catch the need for introducing **Stubs**, an example of a specific **Stub** usage is proposed below.

In order to integrate the **Location Manager** in **I8T4** we need a component that mocks **TD Locator** functionalities in a predefined way. To respect the **Bottom-Up** strategy, given the fact that the **TD Locator** is a component of the **TD View**, we have decided to introduce its **Stub**. The real **TD Locator** will be integrated when the integration procedure arrives to **TD View**.

In **I1** there is the need for some sample data to be in the **Database**, and in **I24** some sample GPS data are needed.

6 Appendix

6.1 Tools

- **TeXstudio:** LaTeXeditor used to write this document.
- **SourceTree:** Used to allow team work and synchronize GitHub repository.

6.2 Hours of Work

- **Alessandro:** 15
- **Alberto Mario:** 15

6.3 Revision History

Version Number	***Release Date***	***Changelog***
1.0	21/01/2016	Initial Release.