# Integration Test Plan

Beretta Carolina 852650
Brizzolari Cecilia 852399

January 21, 2016

# Contents

# 1 Introduction

This section briefly illustrates the general idea of the integration test plan document.

## 1.1 Revision history

In this subsection are recorded all changes made to the document and the time in which such versions were made public. As of right now, no revision has been made.

## 1.2 Purpose

The purpose of this document is to explain to the development team everything they need to know in order to proceed with the actual integration testing: what are the subjects, in which order they need to be integrated, which software to use and if there are any stubs or drivers that have to be developed.

## 1.3 Scope

After the unit testing run on low-level components, the system needs to have all its modules integrated, while testing their functionalities looking for bugs or inadequacies of the program.

## 1.4 Glossary

This subsection contains a list of widely used concepts and abbreviations.

**RASD**: acronym for Requirement Analysis and Specification Document

**DD**: acronym for Design Document

**ITP**: acronym for Integration Test Plan

**MTS**: acronym for myTaxiService

**Guest**: generic unregistered person

**User**: generic registered person, who is logged into the system

**Passenger**: user registered as a passenger, who will be able to book a taxi through both the mobile application and the website

**Driver**: user registered as a taxi driver, who will be able to use the system functions only through the mobile application

**Call**: immediate booking, the request must be fulfilled as soon as possible

**Reservation**: advanced booking

**System**: the software system to be developed comprehensive of its all modules and function, it is the provider of the actual services the user can access through mobile applications and website

**Zone**: part of the city in which the customer and the taxi is situated. The city is defined as the disjoint union of all the zones

**Waiting Queue**: a list of available taxis in a specific zone, ordered from the one who has been available the longest to the one one who has just finished carrying a customer around

## 1.5 Reference documents

The documents used for the creation of this document are:

- *Assignment 4 - integration test plan* written and provided by the professor
- *RASD - myTaxiService*, written previously[1]
- *DD - myTaxiService*, written previously[2]
- *Integration slides*, from the university of Ottawa[3]

[1] https://github.com/rebark/SE2_BerettaBrizzolari/blob/master/Deliveries/RASD%20final%20version.pdf
[2] https://github.com/rebark/SE2_BerettaBrizzolari/blob/master/Deliveries/DD.pdf
[3] http://www.site.uottawa.ca/~ssome/Cours/CSI5118/Integration.pdf

# 2 Integration strategy

## 2.1 Entry criteria

The following documents must be delivered before the integration test may begin:

1. The Design Document
2. The Integration Test Plan Document

Moreover before the specific integration test may begin, these additional criteria must be met:

1. All components have been coded and unit tested on their basic functionalities
2. All high prioritized bugs have been fixed and closed
3. Required Test Environment has been set up for Integration testing, such as input data for the tests

## 2.2 Elements to be integrated

The elements extracted from the component diagram in the DD have been divided into subsystems, to aid the integration process between them. The division has been made according to the layer each component belongs to.

The subsystems individuated, thus, are:

- the Data Layer, which contains the *Database Manager* module
- the Business Layer, which contains the *User Connection*, *Profile Manager*, *Reservation Manager*, *Call Manager*, *Driver Manager* and *Queue Manager* modules
- the Presentation Layer, which contains the *Passenger Application*, *Driver Application* and *Notify* modules.

As can be noted, the components *Sign Up Manager*, *Log In Manager* and *Connection* have all been merged into one module for the integration test plan, because it is expected that they all work correctly with each other after the unit testing.

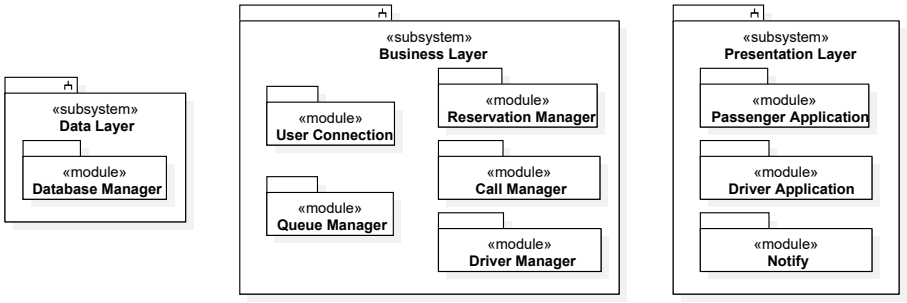


Figure 1: Subsystem identification

## 2.3 Integration testing strategy

The integration will follow the bottom-up approach, both in the integration within subsystems and in the internal integration of each subsystem. The integration will be built up starting from lower levels components and gradually moving up to high levels one, until all the modules are integrated and the entire application is tested as a single unit.

The advantage of this approach is that, if a major fault exists at the lowest unit of the program, it is easier to detect it, and corrective measures can be taken. In fact each unit is tested for its correctness before going for further integration, improving the robustness of the whole system. This strategy has been preferred to the top-down approach due to the amount of work that needs to be put into creating supporting elements: as seen in the literature, *stubs* are more complex and thus require more effort, time and creativity to be built, as opposed to the *drivers*' simplicity.

## 2.4 Sequence of component/function integration

As stated in section 2.3, the strategy chosen for the integration of each component is the bottom-up strategy, executed starting from the integration of each component within the lower subsystem: the plan is to ensure that each subsystem works before integrating it with the one "above".

The drivers used for the integration steps were called like the interfaces present in the component diagram for convenience, since those drivers need to examine every function those interfaces ask of the component that implements them.

### 2.4.1 Modules integration sequence

The first test to be made is if the database works correctly and gives the correct information to all calls made by the *DataManagement* interface: it is important that this is the very first task the development team executes because without it the testing of the Business Layer can't be done exhaustively.

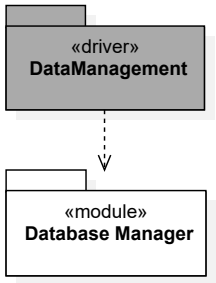


Figure 2: Data Layer modules integration

After the Database Manager has been tested, the driver used can be substituted by the modules that actually call for the methods offered by the *DataManagement* interface, that are the modules *User Connection*, *Profile Manager*, *Reservation Manager*, *Call Manager* and *Queue Manager*.

Now the development team can proceed to the integration of the components within the Business Layer.

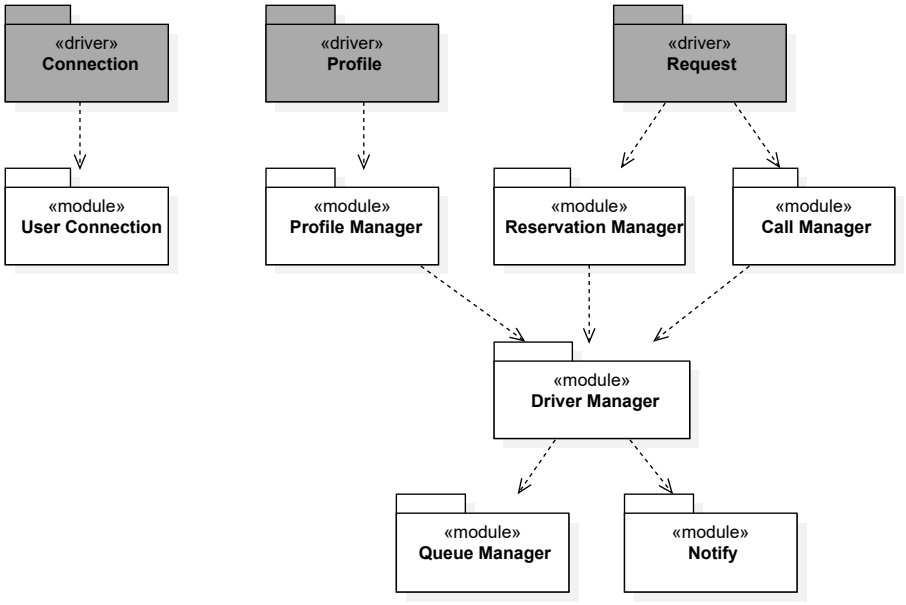


Figure 3: Business Layer modules integration

This layer clearly has two distinct functional groups, so the order of integration between the two doesn't matter.

The integration for the right group will follow this steps:

1. the Queue Manager and Notify are tested by using two drivers

2. the drivers are substituted by the Driver Manager, which is tested with the aid of another driver

3. this, in turn, is substituted by three modules, Profile Manager, Reservation Manager and Call Manager, that will need their own driver each. The drivers for Reservation Manager and Call Manager are merged into one because the two have the same interface in the component diagram, and thus the methods used to call them are the same.

The left group, instead, will only need one step: that the User Connection module is tested by using his own driver.

This passages leave three drivers pending: this is because they have to be substituted with the modules belonging to the Presentation Layer, *Passenger Application* and *Driver Application*.

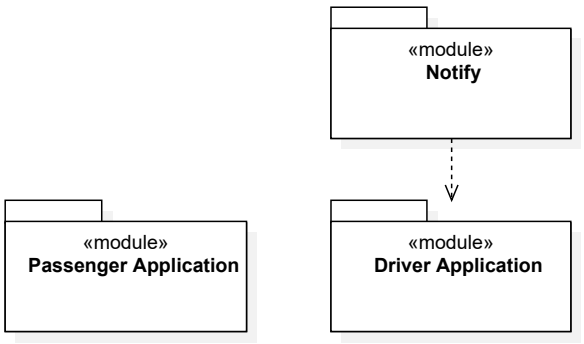


Figure 4: Presentation Layer modules integration

The integration here probably will not follow the general approach: since the Notify module is used in the Business Layer integration, the test of the connection between the Driver Application and Notify will have to be made before the actual testing of the Driver Manager. This only stands for the dependency illustrated in figure 4, so only those functionalities will be tested before; the integration of the whole layer with the other two will be made only afterwards.

### 2.4.2 Subsystem integration sequence

The subsystems, as written in section 2.4.1, will be integrated from the Data Layer to the Presentation Layer. The drivers will be substituted as pictured in figure 5, exactly how was described in the preceding paragraph.

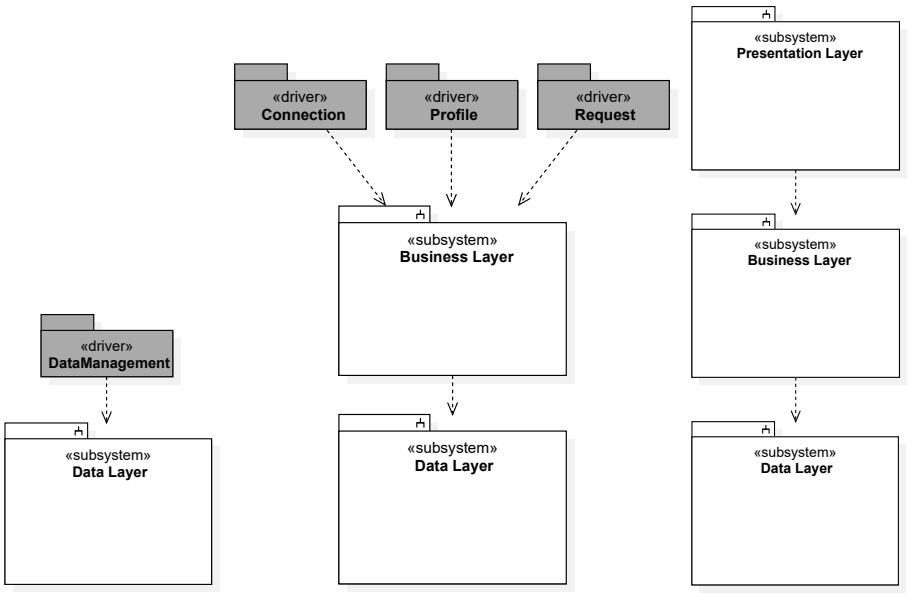


Figure 5: Integration between subsystems

The arrows indicate dependency between subsystems, so following the bottom-up approach will generate the progression shown. How the integration tests should be made will be explained in the following chapter.

# 3 Individual steps and test description

This chapter analyzes the specific test to be done at each step in the integration process, in order to guarantee that integrated elements perform as expected. For a more detailed specification of the interfaces the components use to communicate, refer to the section 2.6 (component interfaces) of the RASD.

## 3.1 Data layer and Business layer

As stated in section 1.4.1 the database will be tested using a driver that emulates all the calls that can be made from the *DataManagement* interface.

### 3.1.1 Integration test case S01

Since the Database Manager is responsible for retrieving and modify data, in order to integrate it with the Business layer, a single driver is sufficient.

| | |
|---|---|
| **ID** | S01 |
| **Test items** | Database Manager → Business layer |
| **Purpose** | Check the interface link between the Database Manager module and the Business layer |
| **Description** | The test simulates all the possible calls the Database Manager can receive from the Business layer, such as finding a Driver given the ID or returning the list of all available drivers |
| **Pass conditions** | The database manager executes the queries on the database accordingly to the request made by the Business layer. In particular read operations do not modify data, whereas update or delete operations modify the data in the database accordingly to the requested operation |
| **Environmental needs** | None |

## 3.2 Business layer

The business layer will use the previous tested Database Manager component to access data needed for testing through the *DataManagement* interface.

### 3.2.1 Integration test case S02

| | |
|---|---|
| **ID** | S02 |
| **Test items** | Queue Manager → Driver Manager |
| **Purpose** | Check the interface link between the Queue Manager and Driver Manager module |
| **Description** | The test simulates the following call conditions:<br>1. Remove a given driver from a queue<br>2. Add a given driver to a queue<br>3. Moving a given driver from a queue to another<br>4. Returning a specific driver to be assigned to a ride request |
| **Pass conditions** | The queues are updated accordingly to the required action or the right driver is returned. If a driver is returned after the algorithm for a ride allocation has been run, the following conditions must be met:<br>1. The driver's previous status was available<br>2. The driver is in no queue |
| **Environmental needs** | S01 succeeded |

#### 3.2.2 Integration test case S03

| | |
|---|---|
| **ID** | S03 |
| **Test items** | Notify → Driver Manager |
| **Purpose** | Check the interface link between the Notify and Driver Manager module |
| **Description** | The test simulates the ride request notification to a driver |
| **Pass conditions** | There are two possible exit conditions that must be tested:<br>1. Ride declined: the notify module either declined the request or triggered the timeout<br>2. Ride accepted:the notify module accepted the request before the timeout |
| **Environmental needs** | S01 succeeded |

#### 3.2.3 Integration test case S04

| | |
|---|---|
| **ID** | S04 |
| **Test items** | Driver Manager → Profile Manager |
| **Purpose** | Check the interface link between the Driver Manager and Profile Manager module |
| **Description** | The test simulates the status update of a driver form available to offline and vice-versa |
| **Pass conditions** | The driver's status is correctly updated and the update affects the queues too |
| **Environmental needs** | S01, S02 succeeded |

#### 3.2.4 Integration test case S05

| | |
|---|---|
| **ID** | S05 |
| **Test items** | Driver Manager → Reservation Manager |
| **Purpose** | Check the interface link between the Driver Manager and reservation Manager module |
| **Description** | The test simulates the request of a driver for a reservation |
| **Pass conditions** | There are two possible exit conditions that must be tested:<br>1. No taxi found: the request is canceled<br>2. Taxi found: the driver assigned to the request is set to busy<br>In both cases, if a driver has declined a request his position is changed in the queue. Moreover the algorithm must be run at least fifteen minutes before the pick-up point indicated in the request. |
| **Environmental needs** | S01, S02 succeeded |

#### 3.2.5 Integration test case S06

| | |
|---|---|
| **ID** | S06 |
| **Test items** | Driver Manager → Call Manager |
| **Purpose** | Check the interface link between the Driver Manager and Call Manager module |
| **Description** | The test simulates the request of a driver for a call |
| **Pass conditions** | There are two possible exit conditions:<br>1. No taxi found: the request is canceled<br>2. Taxi found: the driver assigned to the request is set to busy<br>In both cases, if a driver has declined a request his position is changed in the queue |
| **Environmental needs** | S01, S02 succeeded |

## 3.3 Business layer and presentation layer

Once the integration tests within the business layer has been done, the integration process can continue to the presentation layer. Although not all S02-S06 test are strictly mandatory to proceed with the execution of the following ones, they should be executed before the integration with the presentation layer.

The integration will be done using three different drivers that represent the three different interfaces used to communicate between the two layers.

### 3.3.1 Integration test case S07

| | |
|---|---|
| **ID** | S07 |
| **Test items** | Business layer (*Profile*) → Application |
| **Purpose** | Check the link between the Business and Application layer through the connection interface |
| **Description** | The test simulates the connection between the application layer and the business one, focusing on the functionalities offered by the *Profile* interface:<br>1. Show user information<br>2. Modify user information<br>3. Delete user account |
| **Pass conditions** | The business layer is able to fulfill all request made by the presentation layer and the outcome is visible in the database if a modification or deletion is performed. |
| **Environmental needs** | S01, S02, S03, S04 succeeded |

### 3.3.2 Integration test case S08

| | |
|---|---|
| **ID** | S08 |
| **Test items** | Business layer (*Request*) → Application |
| **Purpose** | Check the link between the Business and Application layer through the connection interface |
| **Description** | The test simulates the connection between the application layer and the business one, focusing on the functionalities offered by the *Request* interface:<br>1. Reserve a taxi<br>2. Call a taxi<br>3. Show active request |
| **Pass conditions** | The business level is able to fulfill all request made by the presentation layer and provides a feedback on the performed operation |
| **Environmental needs** | S01, S02, S03, S05, S06 succeeded |

### 3.3.3 Integration test case S09

| | |
|---|---|
| **ID** | S09 |
| **Test items** | Business layer (*Connection*) → Application |
| **Purpose** | Check the link between the Business and Application layer through the profile interface |
| **Description** | The test simulates the connection between the application layer and the business one, focusing on the functionalities offered by the *Connection* interface:<br>1. Sign up<br>2. Log In |
| **Pass conditions** | The business level is able to fulfill all request made by the presentation layer: either log in or sign up to the system |
| **Environmental needs** | S01 succeeded |

## 3.4 Presentation layer

Once all integration test S01-S09 have been executed, the whole system will be to tested as a whole. Since the presentation layer is the layer that has been tested less during the integration process, the first tests should start from this layer. These initial tests should examine how the users can navigate through the system functionalities, and that for each single selected functionality, the user is redirected to the specific page. An example is when a user log in, he/she must be redirected to the user's home page and not any other page. The relationship between pages and how to access them is specified in the DD: section 4.1.1 (guest), 4.1.2 (passengers), 4.1.3 (drivers).

# 4 Tools and test equipment required

The testing should be done in some cases manually, but in others can be computed automatically. For the latter case, the development team will need to be aided by tools able to test the program code provided. Such required tools are

- Junit, already required for unit testing,
- Arquillian.

Every test result must be reported, indicating if it has been passed; if that is not the case, the team should specify the problems that arose so they can be resolved as soon as possible.

The development team should also test, when the integration is in its last phase, the Presentation Layer, checking that the program meets all non-functional requirements specified in the RASD. This means testing the program on all required devices, such as web browsers and mobiles.

# 5 Program stubs and test data required

## 5.1 Drivers

The adopted integration strategy is bottom-up, so the test will require appropriate drivers. At each integration step illustrated in chapter 3 should correspond a specific driver, therefore the total amount of drivers to be developed is nine. Such elements will need to replicate the methods present in the interfaces specified in subsection 2.6 of the DD. In tables of chapter 3 the drivers are the modules on the right side of the arrow, in the items field.

## 5.2 Data

Each integration step will have a driver, but will also need test case input to work on. The data for the integration test should be generated prior to each specific test run. The generated data should be such that each test is able to cover all possible scenarios, from the standard ones to the extreme cases.

This means that in the database there should be enough dummy data to be able to run each scenario, much like the model verification made in appendix A of the RASD using the tool Alloy Analizer: a significant amount of Driver, Passenger, Request/Call objects and a restricted model of the city. The first integration test between the Database Manager and the Business Layer will need a quite big amount of data, in respect to the following integrations, as its main focus is to test that the system is able to handle queries on any kind of data. The next tests in the integration process will partially relay on this initial data, but it is important to generate new data at each step of the integration. In fact, variance in data testing can improve the robustness and reliability of the system and helps in detecting bugs and problems. Therefore it is important to create specific data sets for each integration steps, in addition to the already generated data.

# A Appendix

## A.1 Software and tools

1. *Lyx* to redact and format the document
2. *StarUML* for the diagrams in chapter 2

## A.2 Hours of work

- Carolina Beretta ~ 10 h
- Cecilia Brizzolari ~ 10 h