M.Sc. in Computer Science and Engineering

Software Engineering 2

Integration Test Plan Document (ITPD)

Version 1.0
Released on: 15th January 2017

Authors:
Marilena Coluccia
Burcu Cesur
Mustafa Çeçe

Reference Professor: Elisabetta Di Nitto

INDEX

1 INTRODUCTION

1.1 Revision History

Release No.	Date	Revision Description
Rev. 0	26/12/2016	Integration Test Plan Template and Checklist
Rev. 0.1	29/12/2016	Strategy individualization
Rev. 0.2	05/01/2017	Plan description
Rev. 1.0	15/01/2017	Final revision and internal approval

Table 1.1- Revision History

1.2 Purpose and Scope

This Integration Test Plan document describes and defines the approach to conducting system integration testing for the PowerEnJoy project.

The Integration Test Plan identifies and documents the strategy and approach to system integration testing, detailing information such as scope, objective, resources and schedule for the testing effort, control gate criteria (entrance and exit criteria), and test artifacts (test evidence) and test deliverables. It addresses specific methods that are used (or followed) during this stage of testing to show that all aspects of the solution are adequately tested and that the system can be successfully transitioned to the next test stage.

The scope of the Integration Test Plan is to document the test planning activities and overall test strategy for the above-mentioned project.

1.3 List of Definitions and Abbreviations

Acknowledgment: signal passed between communicating processes or computers to signify acknowledgement, or receipt of response, as part of a communications protocols. It can contain a note.

API: Application Programming Interface is used for interacting and communicating with other existing systems.

Customer: A person that, after registering, has an account. Customer register to the system with name, surname, mobile number, credit card number and drive license then an access to the system is provided with a password that by sending an SMS to the mobile number used in registration.

DD: *Design Document*

Drivers module: considered as the form of dummy modules which are always distinguished as "calling programs", that is handled in bottom up integration testing, it is only used when main programs are under construction.

GPS: The Global Positioning System is a satellite-based navigation system that transmits radio signals. It is used to provide geolocalization and time information. The GPS has a specific role in the system, as it both receives and provides the coordinates of the cars and customers.

Guest: A possible customer who hasn't logged in or registered yet. Guests can have access to the system functionalities only after making a registration and logging in.

JDBC: Java Database communication

RASD: *Requirements Analysis and Specifications Document*

Reservation: It can be made for a single car for at most one hour. Once the customer has reserved a car, the reservation is created immediately and expires after one hour.

SMS: Short Message Service is a form of text-messaging communication based on phones and mobile phones.

Trip: It starts when the customer gets in the cars and starts the engine.

1.4 List of Reference Documents

The following references were used to specify this document:

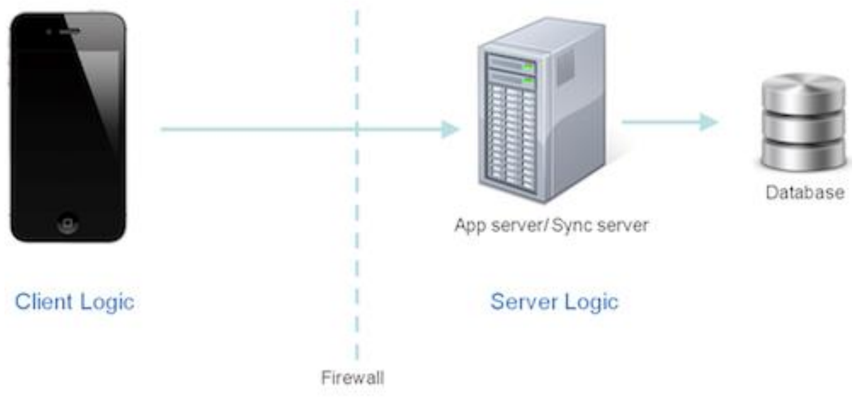
- Requirements Analysis and Specifications Document;
- Design Document;
- Verification and validation, part I;
- Verification and validation, part II;
- Verification tools;
- Software engineering II project assignment 2016/17;
- JDBC Oracle documentation;
- Arquillian web site;
- Arquillian Testing Guide;
- JUnit web site;
- JMeter web site.
- JUniversal web site

2 INTEGRATION STRATEGY

2.1 Entry Criteria

To start the phase of integration testing of the system components it is necessary that the following criteria are met:

- The database structure must be complete as well as the access utilities;
- The client and the server logic should be fulfilled;
- The client interface can be basic but should let the main system property;



It is appropriate that all components have been decided during the drafting of the above documents, as mentioned in *paragraph 1.4 (List of Reference Documents)*, which are useful to establish the most suitable plan

2.2 Elements to be Integrated

The first testing phases is unit tests, after the unit tests are done, the unit tested modules are need to be combined to start doing integration testing. Regarding this, the components which are low level are integrated to create subsystems which are high level.

The main system construction is based on the high level components and their interactions, as mentioned in PowerEnJoy *Design Document*

Since subsystems consist of low level components, this architecture leads integration process.

Integration phase begins with low level module integration for obtaining subsystem and lastly integration of subsystems. The elements are integrated regarding to their functionality dependencies to provide higher level functionalities of PowerEnJoy system.

Some components, such as Mapping Service and Notification System, have crucial roles on the integration plan and it was assumed in the *Design Document* and *Requirements Analysis and Specifications Document* of PowerEnJoy that these modules were complete and well-functioning. It should be denoted once again that, according to the assumption which relying on *Design Document*, these components are working without an issue individually.

Even though integration tests are aimed to verify multiple components working together, it can be also to verify smaller pieces of functionality that are difficult to test at unit level such as data access code. This evolves developing Data Access Utilities which is considered as an upper component of Database Management System.

Customer Management subsystem is combination of components that do not depend on each other, however their functionalities rely on the same subsystem. At this integration phase part, it is basically focused on subcomponents' functionalities collection belonging to the subsystem. Customer Management subsystem consists of following components; Registration, Login, Profile Managing, Password Retrieval. In addition to these, in a different way, Mapping Service is a subcomponent of this subsystem.

Car Management subsystem subcomponents are Car System, Mapping Service and Data Access Utilities. It should be denoted that Car System is assumed in *Design Document* that it is a well working system.

Reservation Management which is a high-level subsystem is an integration of Car Management and Customer Management subsystems and has individual functionalities.

Component and subsystems that are developed for PowerEnJoy system are Customer Management, Car Management, Reservation Management subsystems and Data Access Utilities component.

2.3 Integration Testing Strategy

The integration process is designed with bottom-up approach which involves low level component testing. The structure begins from the lowest modules and gradually progresses towards the upper modules. This integration logic follows all the integrated modules till the customer application is tested as a single unit.

Bottom-up approach offers advantages and adjusts with the system. It allows to start performing integration tests during the development process. If the main module is not developed yet a temporary program called "drivers" is used to simulate the main module. Another advantage of it is if major fault exists at the lowest unit of the program it is much easier to detect it and actions can be hold to correct it. Also, it provides higher accuracy at the granular level.

Using bottom-up approach, fundamental components are started integrating with each other. Fundamental components are components that doesn't have subcomponent, lower level interaction. In the integration testing sequence, structure flow is based on integrating lower level components to provide subsystems which are constituting the system.

Moreover, the components that have already exist by inheriting their sources which are Database Management System, Mapping Service, and Notification System are suitable to use it like fundamental components for bottom-up approach.

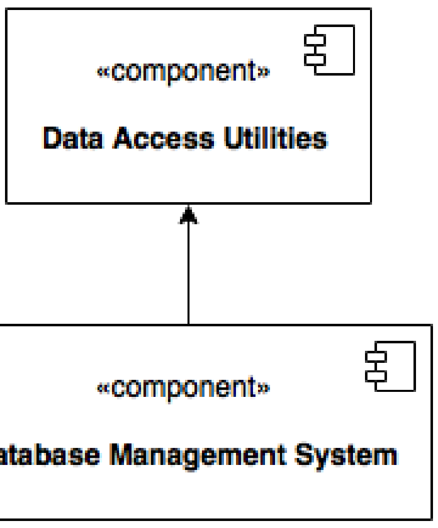
2.4 Sequence of Component/Function Integration

2.4.1 <u>Software Integration Sequence</u>

Regarding to the bottom-up approach, how the various subcomponents are integrated together to create higher level subsystems is demonstrated. Following is specified in parts of subsystems and subcomponent.

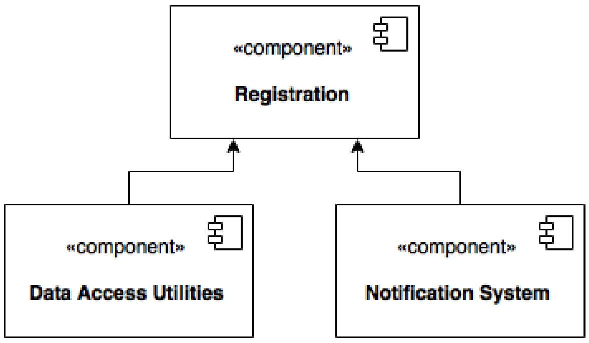
Data Access Utilities

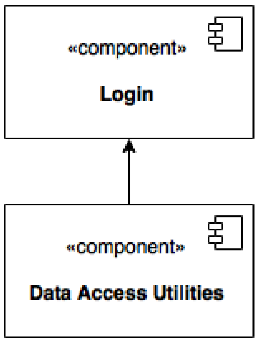
Database Management System is integrated with Data Access Utilities. Data Access Utilities component is crucial since it meets upper components in performing data queries.

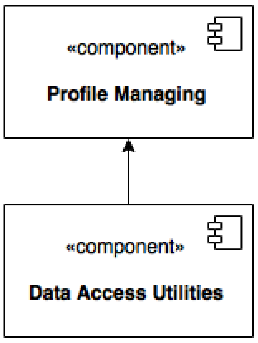


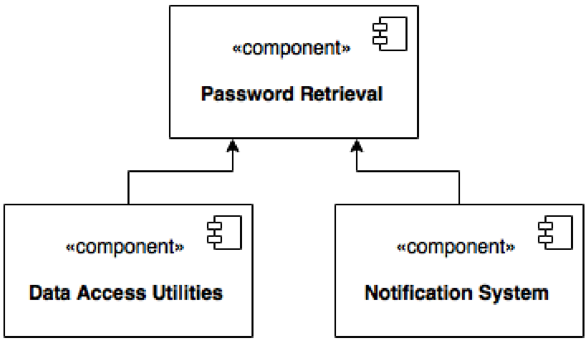
Customer Management

Integration sequence follows with the implementation of the Customer Management subsystem by involving subcomponents. The subcomponents of Customer Management consist of operations on customer account. In the following diagram, it is demonstrated how these subcomponents are interact with other components by the bottom-up approach.

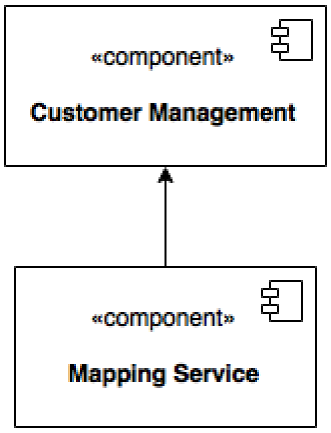

«component» Registration
«component» Data Access Utilities
«component» Notification System




«component» Profile Managing
«component» Data Access Utilities

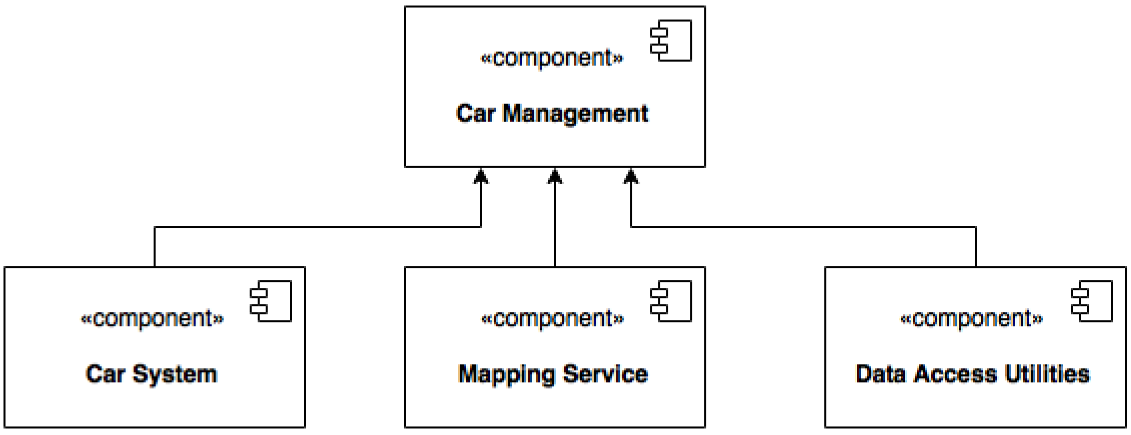

«component» Password Retrieval
«component» Data Access Utilities
«component» Notification System

The Customer Management subsystem which is explicitly denoted with the following diagram comprises of the above-mentioned components. Overall, this subsystem is a package of components methods and an individual component interacts with it, which is Mapping Service.



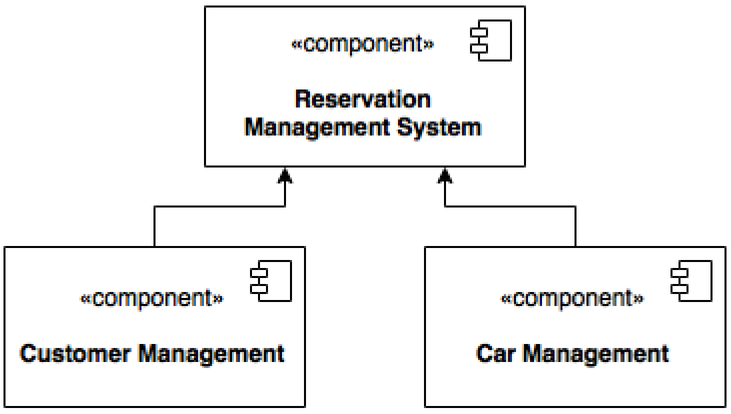
Car Management

Integration process follows with the implementation of the Car Management subsystem by integrating subcomponents. Bottom-up is the integration testing approach. Since Car System is accepted as a well working system and well connected with the subsystem, according to the *Requirements Analysis and Specifications Document* and the *Design Document*, in the following diagram their connection is demonstrated. Mapping Service subcomponent is again having the same above mentioned logic of connection and obligation to be tested for Car Management subsystem. Data Access Utilities must also be tested for below the subsystem to determine success status of all data processes on DB.



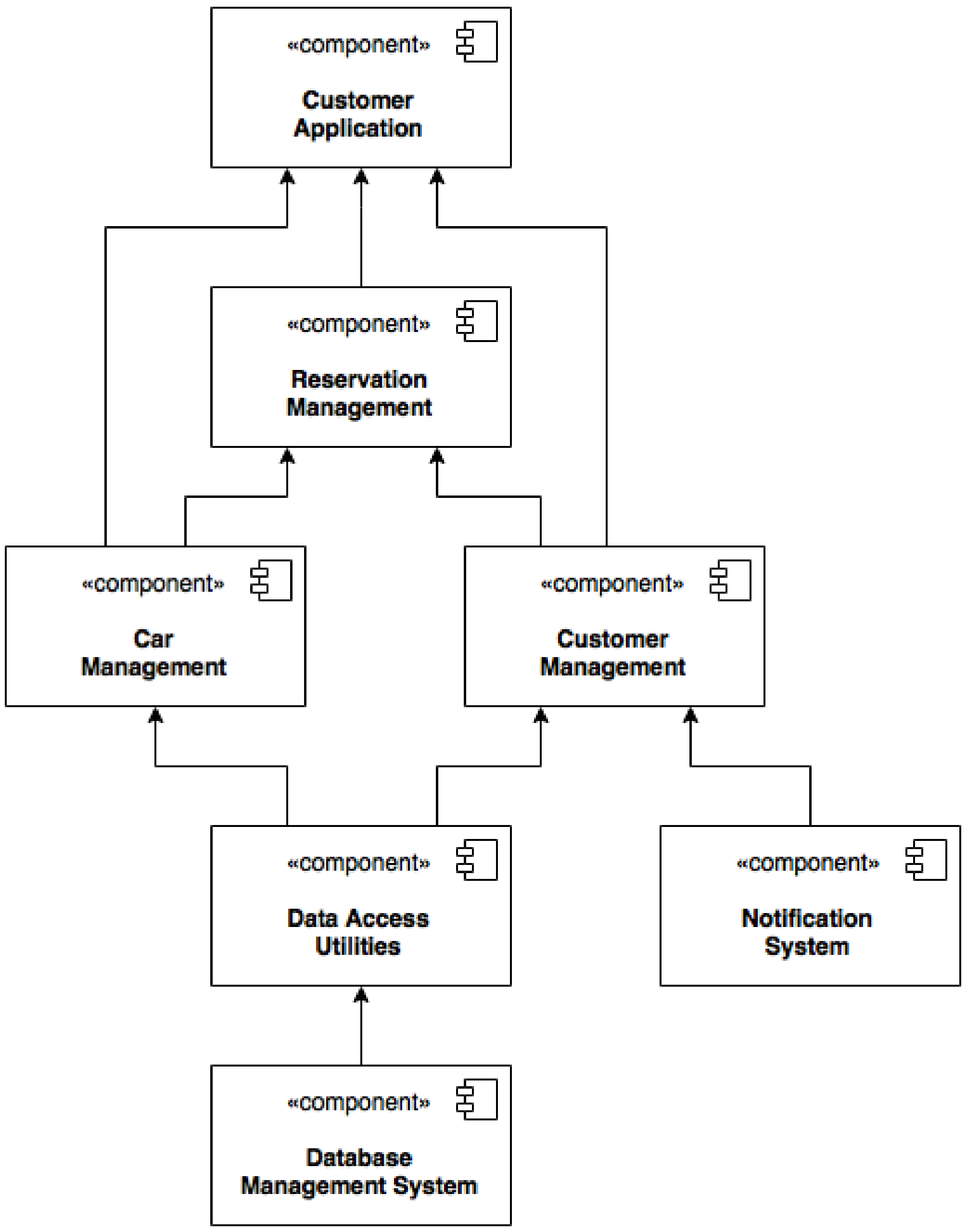
Reservation Management

Integration process follows with the implementation of the Reservation Management subsystem by integrating subsystems. This integration is again approached with bottom-up testing. The subsystem obtains all pre-required tests by these subcomponents, therefore no it has other subcomponents.



2.4.2 Subsystem Integration Sequence

In the following diagram a general overview provided in terms of how the various high-level subsystems are integrated together to create the full PowerEnJoy infrastructure.



3 INDIVIDUAL STEPS AND TEST DESCRIPTION

As already mentioned above, it was decided to use a bottom-up test plan approach, which implies the knowledge of all the systems and subsystems used. The technique that is deemed more useful in this type of approach consists of *Unit testing*, *Integration testing* and *Regression testing*.

3.1 Unit testing

Unit testing is a software testing method by which individual units of source code, sets of one or will more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determinate whether they are fit for use.

3.2 Integration testing

Integration testing is the phase in which individual software modules are combined and tested as a group. The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. To perform this step, if it becomes necessary, they will use the driver modules. The integration tests, regarding the above system, are provided below.

3.2.1 *Data Access Utilities and Database management system*

As already specified in the *Design Document* will be used JDBC to access our database. All notions concerning it are in the documentation that Oracle provides, among which is also present how to properly test for communications with different databases(link to the web page)

3.2.2 *Customer Management system and Data Access Utilities*

newCustomer(Customer)

Goal	Insertion of a new customer information in the DB
From	Customer Management
To	Data Access Utilities
Input	Customer instance with personal information received from the registration form (i.e. Name, Surname, Phone number, etc.)
Expected output	---
Possible errors	Empty instance Invalid input type

updateCustomer(Customer)

Goal	update customer information in the DB
From	Customer Management
To	Data Access Utilities
Input	Customer instance with personal information received from the account management
Expected output	---
Possible errors	Empty instance Invalid input type Missing data

deleteCustomer(Customer)

Goal	delete customer information from the DB
From	Customer Management
To	Data Access Utilities
Input	Customer instance
Expected output	---
Possible errors	Empty instance Invalid input type Missing data

getInfo(telephone)

Goal	request customer information at the DB
From	Customer Management
To	Data Access Utilities
Input	Telephone
Expected output	Customer instance
Possible errors	Empty input Invalid input type Missing data

3.2.3 *Customer Management system and Notification system*

sendSms(Customer)

Goal	Request sending customer credentials (password) via Sms to the telephone number used for the registration
From	Customer Management
To	Notification system
Input	Customer instance
Expected output	---
Possible errors	Empty instance Invalid input type Missing data

3.2.4 *Customer Management system and Mapping service*

getLocation(Customer)

Goal	Request customer location
From	Customer Management
To	Mapping service
Input	Customer
Expected output	GPS coordinates
Possible errors	Connection error Empty instance Invalid input type

sendLocation (GPS coordinates)

Goal	Send customer location
From	Mapping service
To	Customer Management
Input	GPS coordinates
Expected output	---
Possible errors	Empty instance Invalid input type

3.2.5 *Car management and Data Access Utilities*

updateCar(Car)

Goal	update car information in the DB
From	Car Management
To	Data Access Utilities
Input	Car instance
Expected output	---
Possible errors	Empty instance Invalid input type Missing data

getInfo(carId)

Goal	request car information at the DB
From	Car Management
To	Data Access Utilities
Input	CarId
Expected output	Car instance
Possible errors	Empty input Invalid input type Missing data

3.2.6 *Car Management system and Mapping service*

getLocation(Car)

Goal	Request car location
From	Car Management
To	Mapping service
Input	Car
Expected output	GPS coordinates
Possible errors	Connection error Empty instance Invalid input type

sendLocation (GPS coordinates)

Goal	Send car location
From	Mapping service
To	Car Management
Input	GPS coordinates
Expected output	---
Possible errors	Empty instance Invalid input type

3.2.7 *Car Management system and Car system*

As mentioned in the *Design Document* assumptions, it is hypothesized that the machine's system is well-functioning much as his communications with the above system.

Regarding to that, arranging the integration testing of these systems will not be necessary.

3.2.8 *Reservation Management system and Customer Management system*

sendReservation (CarReservation, CustomerId)

Goal	Send reservation
From	Reservation Management
To	Customer Management
Input	CarReservation and CustomerId
Expected output	---
Possible errors	Empty instance Invalid input type Missing data

requestReservation(reservationId)

Goal	request reservation to work on it (i.e. end the reservation, calculate the fee to charge, etc.)
From	Reservation Management
To	Customer Management
Input	reservationId
Expected output	CarReservation
Possible errors	Invalid input type Missing data

requestCustomer(CustomerId)

Goal	request customer instance that match with the customerId
From	Reservation Management
To	Customer Management
Input	CustomerId
Expected output	Customer
Possible errors	Invalid input type Missing data

3.2.9 *Reservation Management system and Car Management system*

requestCar(carId)

Goal	request car instance that match with the carId
From	Reservation Management
To	Car Management
Input	carId
Expected output	Car
Possible errors	Invalid input type Missing data

occupyCar(CarId)

Goal	Set as busy the car that match with the CarId
From	Reservation Management
To	Car Management
Input	CarId
Expected output	---
Possible errors	Invalid input type Missing data

freeCar(CarId)

Goal	Set as free the car that match with the CarId
From	Reservation Management
To	Car Management
Input	CarId
Expected output	---
Possible errors	Invalid input type Missing data

3.2.10 Customer Application and Customer Management system

Login (mobile number, password)

Goal	Let access at the user to the main system functionalities
From	Customer Application
To	Customer Management
Input	Mobile number and password
Expected output	Acknowledgment
Possible errors	Incorrect Password Invalid input type Missing data

registration (Name, Surname, CreditCard, DriverLicense, Telephone)

Goal	Send information for a Set as free the car that match with the CarId
From	Customer Application
To	Customer Management
Input	Name, Surname, CreditCard, DriverLicense, Telephone
Expected output	Acknowledgment
Possible errors	Invalid input type Missing data

profile(customerId)

Goal	Request a customer information
From	Customer Application
To	Customer Management
Input	CustomerId
Expected output	Profile Data
Possible errors	Invalid input type Missing data

requestNewPassword(customerId)

Goal	Request a new password for a customer
From	Customer Application
To	Customer Management
Input	CustomerId
Expected output	Acknowledgment
Possible errors	Invalid input type Missing data

requestPosition(customerId)

Goal	Request a customer location
From	Customer Application
To	Customer Management
Input	CustomerId
Expected output	Location
Possible errors	Invalid input type Missing data

3.2.11 *Customer Application and Car Management system*

requestCars(Location)

Goal	Request information about cars near a certain location
From	Customer Application
To	Car Management
Input	Location
Expected output	Cars set
Possible errors	Invalid input type

3.2.12 *Customer Application and Reservation Management system*

newReservation (customerId, carId)

Goal	Submit information for a reservation
From	Customer Application
To	Reservation Management
Input	CustomerId, carId
Expected output	Acknowledgment
Possible errors	Invalid input type OccupiedCar Missing data

cancelReservation(customerId)

Goal	Request the dissolution of a customer reservation
From	Customer Application
To	Reservation Management
Input	CustomerId
Expected output	Acknowledgment
Possible errors	Invalid input type Missing data

3.3 Regression testing

Regression testing is an integral part of the extreme programming software development method and is done after functional testing has concluded, to verify that the other functionalities are working. Regression testing has traditionally been performed by a software quality assurance team after the development team has completed work.

4 TOOLS AND TEST EQUIPMENT REQUIRED

4.1 Performance Analysis

The full range of performance analysis of the whole PowerEnJoy framework looks possible just in the system integration stage. However, some features of the system can be measured. Specifically, we assume that the mobile phones of the intended population have sensible CPU and main memory usages.

As indicated in the PowerEnJoy *Requirements* Analysis and Specifications Document, there are some performance requirements of the mobile applications which are:

- Loading of Main Page is not more than 2 seconds.
- Searching of car is not more than 2 seconds.
- Reservation of car is not more than 1 seconds.
- Cancelling of reservation is not more than 1 seconds.
- System RAM must be minimum 1GB.

System performance should be tested every week, to be sure there is no delays. This is essential because this kind of failures will dissatisfy the users.

Another point of interest is the capability of users supported by the system. At the beginning, 100 synchronical users will get access to the system. Its performance will be checked every day in relation with user satisfaction. This is essential for this system, in that when all users are logged to system at the same time, the system should not slow down.

The system should be stored 10 GB data a day and will be checked every day to ensure that it can overcome over 10 GB of data stored. This is useful to see how much data flows within the system.

In addition, even though no exact number is settled now, the capacity of the storage must be acceptedly little. Also, the mobile phones should support high definition images. These tests are going to be carried out utilizing the suitable performance analysis apparatus gave the SDK of each mobile phone.

4.2 Tools

It will be used some automated testing tools for testing the different segments of PowerEnJoy mobile application, saving time and money. Now, two tools will be used in the Java Enterprise Edition runtime environment.

Arquillian is a testing framework for Java that leverages JUnit and TestNG to execute test cases against a Java container. The Arquillian framework is broken up into three major sections: test runners (JUnit or TestNG), containers (Weld, OpenWebBeans, Tomcat, Glassfish, and so on), and test enrichers (integration of your test case into the container that your code is running in). Arquillian will be used to check that the correct components are infused when reliance infusion is indicated, that the associations with the database are legitimately overseen and comparative container-level tests.

Another tool is the *JUnit* framework which is a Regression Testing Framework used by developers to implement unit testing in Java, and accelerate programming speed and increase the quality of code. JUnit Framework can be easily integrated with either of the following Eclipse, Ant, and Maven. JUnit test framework provides the following important features which are fixtures, test suites, test runners, and JUnit classes.

A specific performance analysis tool will be utilized to ensure the application routine: *JMeter*. Thanks to that, the overload capacity can be tested.

To let the cross-platform usage, a code translator like *JUniversal* will be used and the results must be tested according to this document.

To sum up, it ought to be noticed that nevertheless the use of computerized testing tools, a portion of the arranged testing activities will likewise require a lot of manual operations, particularly to set up the fitting arrangement of testing data.

4.3 Test Equipment

In this paragraph, test equipment for testing activities will be mentioned. PowerEnJoy application will be used as mobile application and/or desktop application. Therefore, the following devices are necessary for testing:

- a smartphone or a virtual machine that virtualize mobiles operative systems;[1]
- data storage to collect information about test results.

A responsive user interface will be used to ensure a suitable user experience on all devices.

[1] As specified in the RASD, Microsoft Azure can be used to run virtual machine easily.

5 PROGRAM STUBS AND TEST DATA REQUIRED

5.1 Program Stubs and Drivers

As mentioned in the paragraph 2.2 (*Integration Testing Strategy*) the bottom-up approach will be used in this tests plan. This methodology requires ***drivers modules*** to test components in an appropriate way, if those are under implementation. Unlike top-down approach, the bottom-up approach does not require the use of any type of stub. They could be managed to simulate the existence of clients until they are absolutely improved.

In the following sections the drivers and their features are presented.

5.1.1 Data Access Driver

This testing module is going to summon the methods exposed by the Data Access Utilities component by delivering the end goal to test its interaction with the DBMS.

5.1.2 Customer Management Driver

This testing module is going to summon the methods exposed by the Customer Management subcomponent, including those with package-level visibility, to test its interaction with the Mapping Service.

5.1.3 Car Management Driver

This testing module is going to summon the methods exposed by the Car Management subcomponent, including those with package-level visibility, to test its interaction with the Car System, the mapping Service, and the Data Access Utilities.

5.1.4 Reservation Management Driver

This testing module is going to summon the methods exposed by the Reservation Management subcomponent, including those with package-level visibility, in order to test its interaction with the Customer Management, and the Car Management.

5.1.5 Registration Driver and Password Retrieval Driver

These two-testing module is going to summon the methods exposed by its correspondent component to test its interaction with the Data Access Utilities and the Notification System.

5.1.6 Profile Managing Driver and Login Driver

These two-testing module is going to summon the methods exposed by its correspondent component to test its interaction with the Data Access Utilities.

5.2 Test Data

To manage in a consistent and correct system operation, also the exceptions set out in paragraph 3.2 (*Integration testing*) should be tested for each test. The analysis of these tests could lead to the modification of components so far considered or, at worst, the review of previous project documents.

6 EFFORT

Name & Surname	*26.12.16*	*29.12.16*	*05.01.17*	*15.01.17*	*Outside of Group*	*Total*
Marilena COLUCCIA	2h	2h	2h	2h	30 h	38 hour
Burcu CESUR	2h	2h	2h	2h	28 h	36 hour
Mustafa ÇEÇE	2h	2h	2h	2h	11 h	19 hour