Power EnJoy

Integration Test Plan Document

Version 1.0.0

Redaelli Marco 877622 Zanoli Francesco 877471

15/01/2017

Contents

Figure Contents

Chapter 1

Introduction

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

1.0.1 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 the order in which the software components identified in the **Component View** of the **DD** have to be integrated one with each other. It is also used to guarantee a well tested final software. Following the exposed procedure ensures that all the software components explained in the **DD** will communicate and cooperate in the proper way.

1.1 List of definitions and abbreviations

1.1.0.0.1 Definitions In the document are often used some specific terms whose definitions are reported bellow:

- Server Database: data layer
- Server Application: application layer
- Client: client layer
- Mobile App: *PowerEnJoy* mobile application, in Client
- System: the union of software and hardware to be developed and implemented
- 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.

1.1.0.0.2 Acronyms

- **RASD:** Requirements Analysis and Specification Document
- **DD:** Design Document
- **API:** Application Programming Interface
- **DBMS:** DataBase Management System
- **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.

1.2 List of reference documents

- Software Engineering 2 Project AA 2016/2017: Project Description And Rules and Assignment 4 - integration test plan
- PowerEnJoy's Requirement Analysis and Specification Document (RASD)
- PowerEnJoy's Design Document (DD)

Chapter 2

Integration Strategy

2.1 Entry Criteria

Before starting the integration testing of any software component that has been designed for *PowerEnJoy* 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

PowerEnJoy as shown in the **DD** is a three-tier system composed by:

- **DBMS:** The memory of *PowerEnJoy* entire system
- **PowerEnJoy System:** The main server and the main core composed by:
  1. Data Manager
  2. Account Manager
  3. Ride Manager
  4. Bill Manager
  5. Map Services
  6. Notification
  7. Car Manager
  8. Zone Manager
  9. Problem Manager
- **Client Application:** Subdivided in **Car System** and **Mobile Application**

Moreover we assume that that Google Maps API and Paypal are well tested by their owner and thus we can use them without testing any further.

2.3 Integration testing strategy

The integration testing strategy, conducted in this project, is a **bottom-up** approach. This strategy tests the lower level components and start testing a way upwards to higher level components. The advantage of this strategy is that allow us to maintain the code easier, smaller modules have unit tests and there is a clearer structure of how to do things. The disadvantage is that when releasing a prototype it's impossible to see a working prototype until nearly all the program has been completed so that may take a long time before this happens. In early development, testing tools as Mockito and Arquillian (described in Chapter 4) allow us to test components which depend on incomplete ones through stubs and drivers (Chapter 5). The usage of the selected approach will create a robust application with efforts concentrated in testing the Server parts before all.

2.4 Sequence of component/function integration

2.4.1 Software integration sequence

The following diagram illustrates the integration sequence of the various components, following the integration testing strategy described above. This means that in each subsystem, components are integrated starting from the most independent to the less independent, in order to prompt the chosen approach and improving modularity.

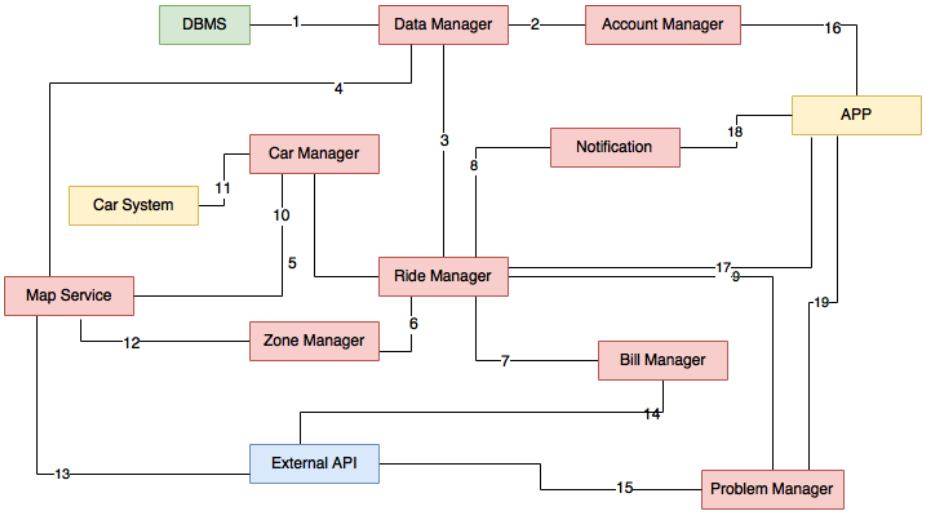

Figure 2.1: Software Integration Diagram

2.4.2 Subsystem integration sequence

The following diagram illustrates the integration sequence of the various subsystems, following the integration testing strategy described above. In particular, the Server Database is integrated before the Client, because the former does not need an actual functioning system in order to be tested efficiently, contrary to the latter.

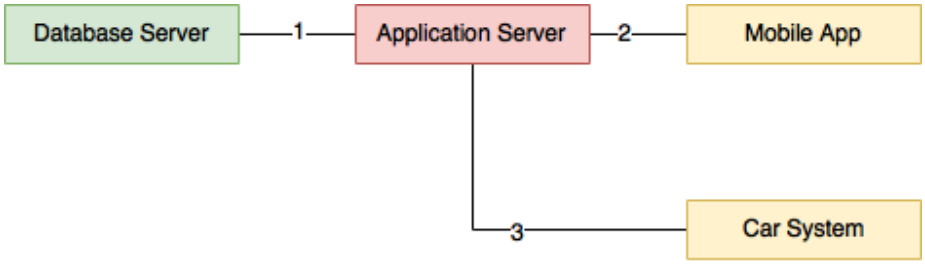

Figure 2.2: Subsystem Integration Diagram

Chapter 3

Individual steps and test description

3.1 Model Integration Test Cases

3.1.1 Software Integration Test Cases

3.1.1.0.1 Test Case I1

Test Item(s)	Data Manager $\longleftrightarrow$ DBMS
Input Specification	Queries on the DBMS for some Table
Output Specification	The queries return the expected results
Environmental Needs	Glassfish, a test DataBase
Target	Verify that the typical query to the DB works

3.1.1.0.2 Test Case I2

Test Item(s)	Data Manager $\longleftrightarrow$ Account Manager
Input Specification	A set of methods calls on Data Manager to retrieve user information
Output Specification	Check if the user's information are correct
Environmental Needs	Glassfish Server, a test Database, I1 successful
Target	Verify that the user information are retrieved from the Data Manager

3.1.1.0.3 Test Case I3

Test Item(s)	Data Manager ⟷ Ride Manager
Input Specification	A set of methods calls on Data Manager to retrieve reserved ride information
Output Specification	Check if the user's information are correct
Environmental Needs	Glassfish Server, a test Database, I1 successful
Target	Verify that the reserved ride info are retrieved from the Data Manager

3.1.1.0.4 Test Case I4

Test Item(s)	Data Manager ⟷ Map Service
Input Specification	A set of methods calls on Data Manager to retrieve informations about coordinates
Output Specification	Check if the user's information are correct
Environmental Needs	Glassfish Server, a test Database, I1 successful
Target	Verify that the info of coordinates are retrieved from the Data Manager

3.1.1.0.5 Test Case I5

Test Item(s)	Ride Manager ⟷ Car Manager
Input Specification	A set of methods calls on Car Manager
Output Specification	Check if an available Car is returned
Environmental Needs	I8 and I2 successful
Target	Retrieve an available Car

3.1.1.0.6 Test Case I6

Test Item(s)	Ride Manager ⟷ Zone Manager
Input Specification	A set of methods calls on Zone Manager
Output Specification	Verify that the returned zone is the correct one
Environmental Needs	I13 and I5 successful
Target	Find the position in a certain zone

3.1.1.0.7 Test Case I7

Test Item(s)	Ride Manager ⟷ Bill Manager
Input Specification	A set of methods calls on Bill Manager
Output Specification	Verify that the total amount is correct
Environmental Needs	I13, I11 and I2 successful
Target	Verify that the Bill Manager calculate the correct amount of the ride

3.1.1.0.8 Test Case I8

Test Item(s)	Ride Manager ⟷ Notification
Input Specification	A set of methods calls in order to create a Notification
Output Specification	Check if the correct notification is created
Environmental Needs	I2 and I5 successful
Target	Verify that the Notification Manager creates the notification from the Ride Manager

3.1.1.0.9 Test Case I9

Test Item(s)	Ride Manager ⟷ Problem Manager
Input Specification	A set of methods calls on Problem Manager
Output Specification	Verify that the problem belong to the provided list
Environmental Needs	I2 and I18 successful
Target	Check if Problem Manager can handle correctly the calls, returning an element belong to the provided listbox

3.1.1.0.10 Test Case I10

Test Item(s)	Car Manager ⟷ Map Service
Input Specification	A set of methods calls on Map Services Manager
Output Specification	Verify that the position of the car is correct
Environmental Needs	I2 and I13 succesful
Target	Retrieve the position of a car in a certain position on the map

3.1.1.0.11 Test Case I11

Test Item(s)	Car Manager ⟷ Car System
Input Specification	A set of methods calls Car Manager
Output Specification	Verify if the state of the car is correct
Environmental Needs	Car with GPS and Data Connection enabled
Target	Verify that the information exchanged between car and system works

3.1.1.0.12 Test Case I12

Test Item(s)	Zone Manager ⟷ Map Service
Input Specification	A set of methods calls on Map Service
Output Specification	Verify if the coordinates of the Zone are contained into the map
Environmental Needs	I13 and I4 successful
Target	Verify that the coordinates of a certain zone are valid

3.1.1.0.13 Test Case I13

Test Item(s)	Map Service ⟷ External API
Input Specification	Create a typical set of methods calls by Map Service on External API
Output Specification	Check if all the methods of External APIs Manager produce the expected results
Environmental Needs	N/A
Target	Verify that the External APIs Manager works with the Map Services Manager

3.1.1.0.14 Test Case I14

Test Item(s)	Bill Manager $\longleftrightarrow$ External API
Input Specification	Create a typical set of methods calls by Bill Manager on External API
Output Specification	Check if all the methods of External APIs Manager produce the expected results
Environmental Needs	N/A
Target	Verify that the External APIs Manager works with the Bill Manager

3.1.1.0.15 Test Case I15

Test Item(s)	Problem Manager $\longleftrightarrow$ External API
Input Specification	Create a typical set of methods calls by Problem Manager on External API
Output Specification	Check that all the methods of External APIs Manager produce the expected results
Environmental Needs	N/A
Target	Verify that the External APIs Manager works with the Problem Manager

3.1.1.0.16 Test Case I16

Test Item(s)	Mobile App $\longleftrightarrow$ Account Manager
Input Specification	Create a typical set of methods calls performed by Mobile App on Account Manager
Output Specification	Check if methods calls mentioned in Input Specification produce the expected results
Environmental Needs	A device that can run Mobile App
Target	Verify if Account Manager can handle correctly Mobile App methods calls

3.1.1.0.17 Test Case I17

Test Item(s)	Mobile App ⟷ Ride Manager
Input Specification	Create a typical set of methods calls performed by Mobile App on Ride Manager
Output Specification	Check if methods calls mentioned in Input Specification produce the expected results
Environmental Needs	A device that can run Mobile App
Target	Verify if Ride Manager can handle correctly Mobile App methods calls

3.1.1.0.18 Test Case I18

Test Item(s)	Mobile App ⟷ Notification
Input Specification	Create a typical set of methods calls performed by Mobile App on Notification
Output Specification	Check that methods calls mentioned in Input Specification produce the expected results
Environmental Needs	A device that can run Mobile App
Target	Verify if Notification can handle correctly Mobile App methods calls

3.1.1.0.19 Test Case I19

Test Item(s)	Mobile App ⟷ Problem Manager
Input Specification	Create a typical set of methods calls performed by Mobile App on Problem Manager
Output Specification	Check that methods calls mentioned in Input Specification produce the expected results
Environmental Needs	A device that can run Mobile App
Target	Verify if Problem Manager can handle correctly Mobile App methods calls

3.1.2 Subsystem Integration Test Cases

3.1.2.0.1 Test Case S1

Test Item(s)	Database Server ⟷ Application Server
Input Specification	Queries on the DBMS for the table Driver, Car, Ride and Zone
Output Specification	The queries return the expected results
Environmental Needs	Glassfish Server, a test Database
Target	Verify that the typical queries to the DBMS works

3.1.2.0.2 Test Case S2

Test Item(s)	Application Server ⟷ Mobile App
Input Specification	A set of methods calls on both Server Application and Mobile App
Output Specification	Check if methods calls mentioned in Input Specification produce the expected results
Environmental Needs	Glassfish, a test Database
Target	Verify the interaction between Server Application and Mobile App works

3.1.2.0.3 Test Case S3

Test Item(s)	Car System ⟷ Application Server
Input Specification	Create a typical set of methods calls performed by Mobile App on Ride Manager
Output Specification	Check if methods calls mentioned in Input Specification produce the expected results
Environmental Needs	A device that can run Mobile App
Target	Verify if Ride Manager can handle correctly Mobile App methods calls

Chapter 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 **Swift**.

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

Chapter 5

Program stubs and test data required

This section describes the specification of stubs and drivers needed to replace the part of software components that still don't exist and test the others. This is necessary to perform the integration steps. DBMS should contain sample data in order to perform proper test cases. We assume that the **Integration testing** comes after **Developing** and **Unit testing**. On the other hand we need few **Drivers** in order to make the not yet integrated components work, because we want to respect the **Bottom-Up** strategy.

To better catch the need for introducing **Drivers**, an example of usage is proposed below.

In order to integrate the **Car System** in **I11** and **I10** we need a component that mocks **Car System** functionalities in a predefined way. We have decided to introduce this because the system of the car could be developed in different time. The real **Car System** will be integrated when the integration procedure arrives to **Car System** and in **I11** some sample GPS data are needed.

Another example is in **I1**, there is the need for some sample data to be in the **Database**.

Appendix A

Appendix

A.1 Tools

- **TeXstudio:** LaTeX editor used to write the document.
- **StarUML:** To draw diagram.

A.2 Hours of work

In the following are listed the hours of work that each member of the group did:

1. Marco Redaelli: 19 *hours*
2. Francesco Zanoli: 19 *hours*

A.3 Version History

In the following are listed the differences between versions:

1. **15/01/2017:** First version