POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
M. Sc. in Computer Science and Engineering
Dipartimento di Elettronica, Informatica e Bioingegneria

Power EnJoy

Software Engineering 2 - Project

Integration Test Plan Document

Version 1.0

Authors:
Andrea BATTISTELLO Matr: 873795
William DI LUIGI Matr: 864165

Academic Year 2016 - 2017

1 Introduction

1.1 Purpose

The purpose of the Integration Test Plan is to lay out a detailed account of the procedures and choices that we will employ in order to validate our system using automatic tests. This includes showing the test cases for the single components, and showing how we will test those component as a whole (integration test).

1.2 Scope

Duckburg is a very large city, with a lot of people. Over the years, the number of people with a driving license has increased. This caused a lot of traffic, boosted air pollution, and made it really hard to find a parking spot.

The purpose of Power EnJoy is to provide an alternative to this. We want to create a commuting service that is highly available and reliable. In fact, the reason why people drive instead of using public transport is usually related to the fact that they don't want to wait for a bus / train (low availability) and they don't want to risk not getting home if there's no public transport available (low reliability) because of a strike, vehicle malfunctions, or just because it's late in the night.

This system will allow users to rent a car for a period of time. PE will let anyone see the available cars (by showing in a map those cars that are closer to the user's location). Moreover, PE will let activated users actually book those cars, and will charge them according to the duration and usage (discounts will apply in specific cases). The system will also have an administrative interface, which will not be accessible by standard users.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

- **Payment information** refers to either a bank account or a credit card number.
- **Device** indicates either a mobile phone / tablet running PEM or a browser that is using PWA.
- **Safe area** refers to any legal parking spot in the city, where any driver can leave the car without interfering with the traffic.
- **PE Safe area** is a safe area that can be used exclusively by PE cars.
- **Web server** is a system that receives HTTP requests and provides web-based contents.
- **Application server** is a system that contains all the business logic. Exposes some interfaces that are used by the web server, mobile applications and cars. It uses containers to balance the requests load.

- **Component** is a piece of software that encapsulates some functionalities and can be reused
- **Subsystem** is a logical and physical subdivision of the system (e.g. the Application server, the Android app, the iOS app, the car app and the admin web app are all different subsystems of the same system).
- **Container** is a piece of software that manages the components and offers an abstraction for parallelism, transactions and many others.
- **Entity** is a class that maps directly into the database. An example is given by JPA.

1.3.2 Acronyms

- **PE** Power EnJoy system
- **PEM** Power EnJoy Mobile application
- **PEW** Power EnJoy Web application
- **PEA** Power EnJoy Administrator application
- **PEC** Power EnJoy Car application
- **SMS** Short Message Service
- **GPS** Global Positioning System
- **RASD** Requirement analysis and specification document
- **DD** Design document
- **UI** User interface
- **UX** User experience design
- **MVC** Model view controller
- **JDBC** Java DataBase Connectivity
- **JPA** Java Persistence API
- **JEE** Java Enterprise Edition
- **EJB** Enterprise Java Bean

1.3.3 Abbreviations

- **[Gn]** n-th goal
- **[Dn]** n-th domain assumption
- **[Rn.m]** m-th requirement related to goal [Gn]
- **[Sn]** n-th subsystem
- **[SnIm]** m-th integration for the n-th subsystem
- **[SnIm-Tk]** k-th test of the integration above
- **[SIn]** n-th integration between subsystems
- **[SIn-Tk]** k-th test for the integration above

1.4 Reference documents

- [1] IEEE Software Engineering Standards Committee, “29148-2011 - Systems and software engineering — Life cycle processes — Requirements engineering”, 2011.
- [2] Payment Card Industry (PCI) Data Security Standard, v3.2, PCI Security Standards Council, LLC.
- [3] Response Times: The 3 Important Limits, JAKOB NIELSEN, 1993.
- [4] Software Engineering 2 course slides.

- [5] The assignment of *Power EnJoy*
- [6] RASD (Requirement specification and Specification document) of *PowerEnjoy*
- [7] DD (Design document) of *PowerEnjoy*

1.5 Document structure

This document is composed of five sections and an appendix:

- The **introduction** section, this one, is intended to find the goal of a design document, clarifies the definitions and acronyms used throughout the document and gives a general idea of the main functionalities of the system to be developed.
- The **integration strategy** section exposes the methods chosen for the integration of the components and subsystems, also with a brief description of every integration test
- The third section is about **individual steps and test description**, whose aim is to give a detailed description of every integration test, giving an idea of what the input and output should be like.
- The fourth section deals with the **tools and test equipment** required for testing, giving a brief description of how they are used and the rationale behind this decision
- Finally, the last section sums up all the **stubs and drivers** needed for the above defined integration tests, along with a description of special **test requirements**

2 Integration strategy

2.1 Entry criteria

Before integration testing of specific elements may begin, the following criteria must be met:

- Every component involved in the integration test should be at least feature-complete up to the level required by the test itself. Some components may be 100% complete while others may be 70% complete, as long as all the features required by the integration test are completed.
- **Documentation** should be written for all implemented features of each component involved. This is critical because, e.g., the integration test might be assigned to a different developer/team than the one who authored the single components.
- Each component involved in the integration test should be equipped with **unit tests** that cover at least 90% of the codebase. Attention should be paid to avoid falling below that threshold when (during or after the integration test) the components will be further extended to reach feature-completeness.
- Each component should be subjected to **code inspection** with automated tools and with manual checks by third-party experts (e.g. by employing former hackers).

2.2 Elements to be integrated

The PowerEnjoy system is made up of many elements. As such, this system needs to be tested in a way that guarantees that every functionality is checked. We also need to make sure these elements work correctly when used in conjunction. To do that, we will use integration tests.

We are now going to list all the elements that will be tested in a coupled manner (integrated) using the specific test set called integration tests.

2.2.1 Subsystems

The subsystems that we have identified are:

- PEM: the mobile application used by customers
- PEA: the administrator application used by call center operators and administrators
- PEC: the Android application that controls the car behaviour
- Application server: the core of PE application
- Web server: the component that receives HTTP requests from PEW, queries the application server with RESTful calls and encapsulates the result in a HTML page.
- DBMS: database management system

2.2.2 Software

Application server

The application server contains the following components to be integrated (as shown in the Design Document):

- Customer component
- Reservation manager
- Ride manager
- Account manager
- Car finder
- Input validator
- Payment manager
- Notification manager
- Car component
- Admin component
- DBManager

PE Car

PE Car contains the following components to be integrated (as shown in the Design Document):

- User input manager
- Car controller
- Car diagnostics
- Presenter
- Car communicator

2.3 Integration testing strategy

Considered the different needs and characteristics of the two levels of abstractions we think a proper solution should adopt different integration testing strategies.

- **Functional grouping** strategy to test the application server components. We chose this method for these reasons:
    - each component can be tested in parallel, thus decreasing the development time
    - requires fewer stubs and drivers than bottom-up or top-down approach.

- **Bottom-up** strategy to integrate the subsystems together. The components in the bottom layer are tested first using a suitable driver. Then the driver will be substituted with the component itself and integrated with other drivers, until the root of the hierarchy is reached.
  This approach has few advantages for subsystem integration:
    - no need for test stubs
    - since we start from the bottom of the hierarchy the critical components are tested first, thus reducing the risk of critical failures.

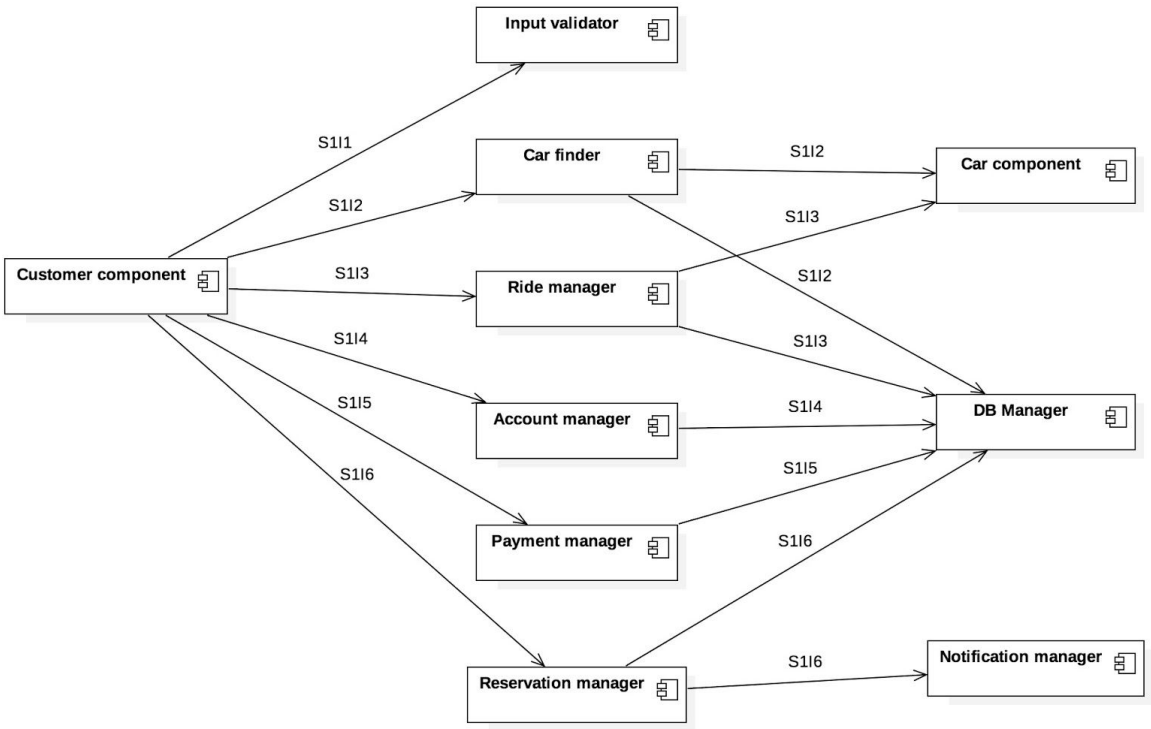
2.4 Sequence of Component/Function Integration

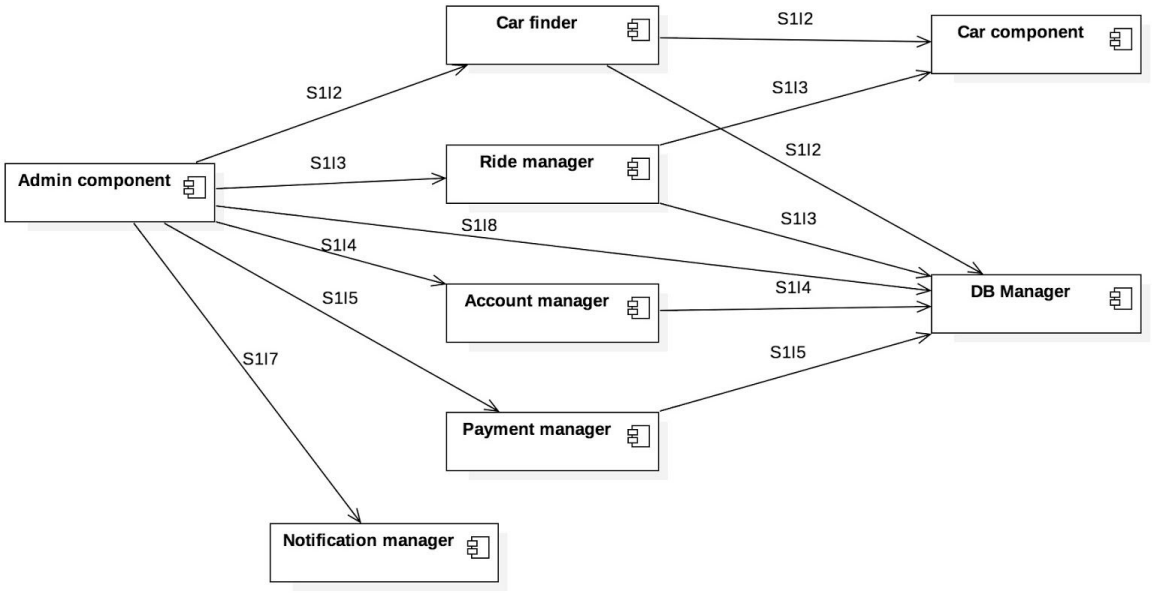
2.4.1 Software Integration Sequence.

The first stage of integration is carried out between the components within each subsystem. In this way the integration tests can be done in parallel, thus reducing the development time.

[S1] Application server

The application server is composed by several components so, in order to improve readability we divided the system in two parts. The first part deals with the user functionalities while the second part concerns the admin functionalities. Note that all the other components are exactly the same.
The integration tests of this subsystem will follow the functional integration approach, that is, every integration test will group all the components required to fully test a specific set of functionalities



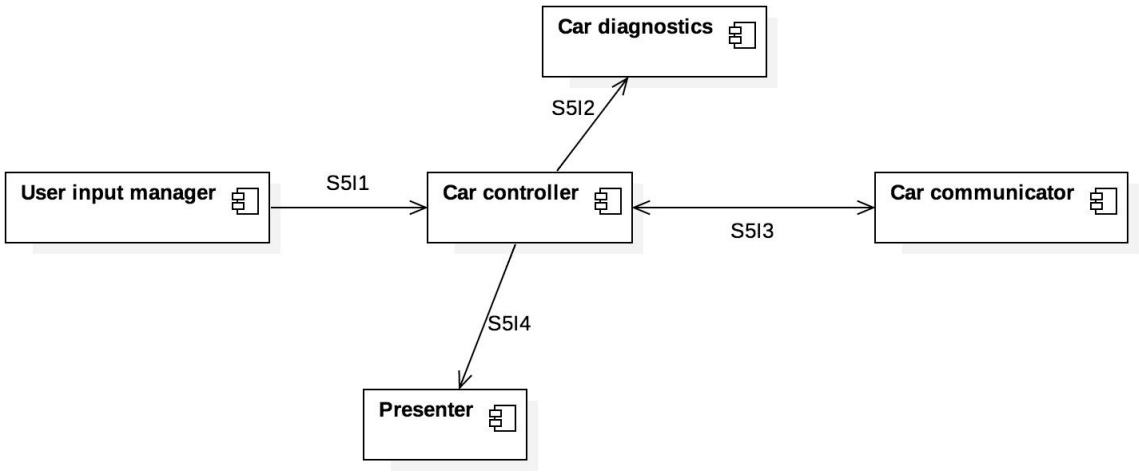


The integration tests that we have identified are:

Test	Components	Description
S1I1	Customer comp.→ Input validator	Input validator: check ID, check license, ...
S1I2	Customer comp., Admin comp., Car finder → Car comp., DB Manager	Car finder: search cars
S1I3	Customer comp., Admin comp., Ride manager → Car comp., DB Manager	Ride manager: new ride, get ride info, ...
S1I4	Customer comp., Admin comp., Account manager → DB Manager	Account manager: create account, manage account, ...
S1I5	Customer comp., Admin comp., Payment manager → DB Manager	Payment manager: add payment, charge user, ...
S1I6	Customer comp., Reservation manager → DB Manager, Notification manager	Reservation manager: reserve car, view reservations, ...
S1I7	Admin comp. → Notification manager	Notification manager: send SMS, receive SMS, notify maintenance, ...
S1I8	Admin comp. → DB Manager	Admin functionalities

[S5] PE Car

We have identified the following components inside PE Car application. The integration strategy for this subsystem will be bottom-up.



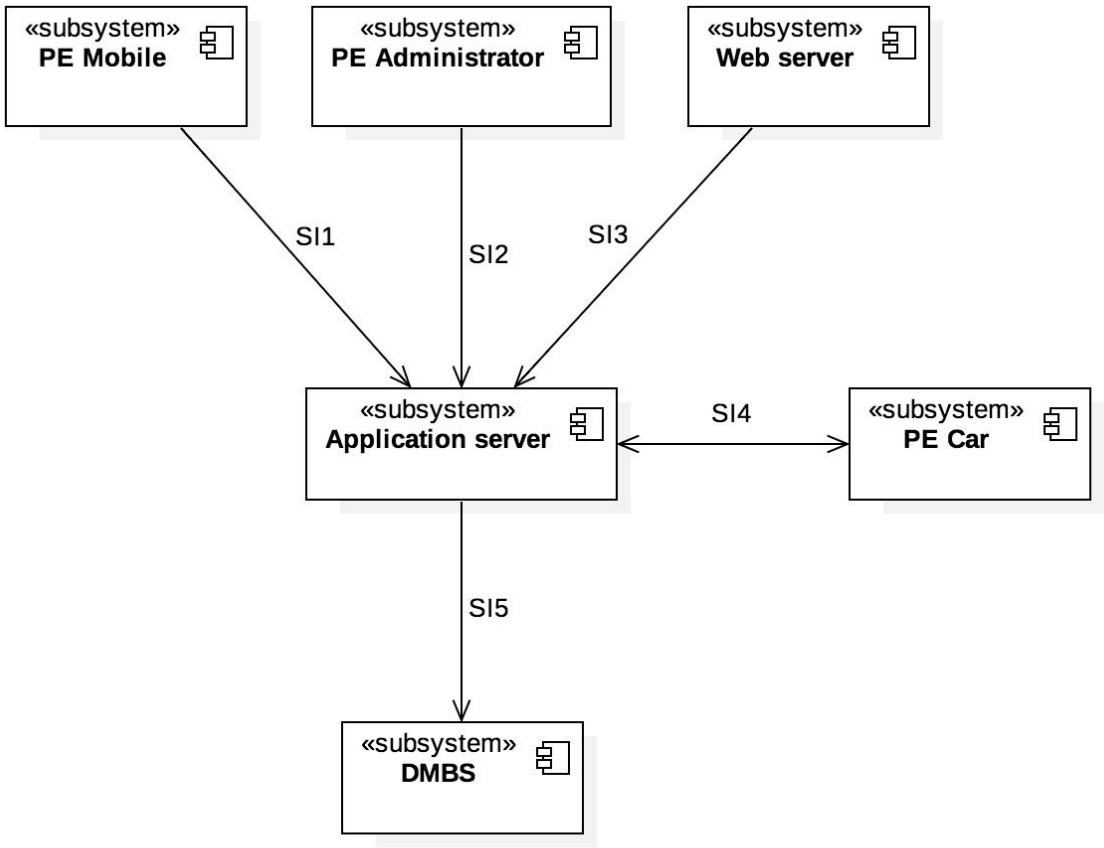
The integration tests that we have identified are:

Test	Components	Description
S5I1	User input manager → Car controller	Automatically perform operations on the input manager, using the Android UI (with UI Automator), and check that the right Car controller methods are called.
S5I2	Car controller → Car diagnostics	Check that the Car controller reacts correctly to changes in the battery status, to a malfunctioning reported by the hardware, and so on.
S5I3	Car controller → Car communicator	Test whether the controller can effectively use the communicator component to send messages to the server.
S5I4	Car controller → Presenter	Check that the correct data is rendered on screen (by using UI Automator) when the controller is in a certain software configuration.

2.4.2 Subsystem Integration Sequence.

In this second phase, we focus on how to integrate all the subsystems identified in the previous sections.

For the subsystem integration sequence we are going to adopt the bottom-up strategy. In this way, once the application server is integrated, the other subsystems can be integrated in a parallel manner.



Here is a description of every integration test:

Test	Subsystems	Brief description
SI1	PE Mobile → Application server	Send/receive customer requests
SI2	PE Administrator → Application server	Test administration functionalities
SI3	Web server → Application server	Test the PEW functionalities
SI4	PE Car ↔ Application server	Test communication protocol between application server and PE cars
SI5	Application server → DBMS	Test connection with the database

Since we will use bottom-up strategy, the order of integration will be:

SI5 → SI4 → SI1, SI2, SI3

with the latter ones carried out in parallel.

3 Individual Steps and Test Description

3.1 Software integration

3.1.1 [S1] Application server

Test case identifier	**S1I1-T1**
Test item(s)	Customer comp. → Input validator
Input specification	Typical valid registration informations, that comprehend: - Driving license number - ID number - Credit card or bank account number
Output specification	Confirm validity of the provided informations
Environmental needs	Working connection with external systems (governmental system, bank) or equivalent stub systems (mocking the behavior of the actual external systems)

Test case identifier	**S1I1-T2**
Test item(s)	Customer comp. → Input validator
Input specification	All possible side cases: - Wrong driving license number - Expired driving license number - Wrong ID number - ID number of expired document - Wrong credit card or bank account number - Malformed or missing input
Output specification	Return a suitable error message.
Environmental needs	Working connection with external systems (governmental system, bank) or equivalent stub systems (mocking the behavior of the actual external systems)

Test case identifier	**S1I2-T1**
Test item(s)	Customer comp., Admin comp., Car finder → Car comp., DB manager

Input specification	Typical car search query, either via address or specific position with increasing range from 10 meters up to 50 km in suitable growing steps.
Output specification	Check if the components access the proper elements in the database and in PE Car stub.
Environmental needs	PE Car stub

Test case identifier	**S1I3-T1**
Test item(s)	Customer comp., Admin comp., Ride manager → Car comp., DB manager
Input specification	Typical car ride that comprehend: - "Start driving" message - Several "unlock" messages - "Stop driving" message
Output specification	- Check the communication with the PE Car stub - Make sure proper functions are called - All the ride information must be stored in the database
Environmental needs	PE Car stub

Test case identifier	**S1I3-T2**
Test item(s)	Customer comp., Admin comp., Ride manager → Car comp., DB manager
Input specification	Simulation of several rides at the same moment
Output specification	Check if all the tracking informations are accessible from admin component.
Environmental needs	PE Car stub

Test case identifier	**S1I4-T1**
Test item(s)	Customer comp., Admin comp., Account manager → DB manager
Input specification	Complete registration of a new account

Output specification	- Check that all the informations are stored in the database - Make sure the SMS are sent - Check that all the correct methods are called in Account Manager
Environmental needs	Active SMS service, or equivalent stub

Test case identifier	**S1I5-T1**
Test item(s)	Customer comp., Admin comp., Payment manager → DB manager
Input specification	Perform various transactions from different methods of payment (credit card and bank account)
Output specification	- Check that all the correct methods are called in Payment manager - Check that all the informations are stored in the database - Make sure that the money are actually transferred
Environmental needs	Working connection with external bank, or equivalent stub

Test case identifier	**S1I6-T1**
Test item(s)	Customer comp., Reservation manager → DB manager, Notification manager
Input specification	Typical reservation from a client
Output specification	- Check that the phone number verification via SMS is performed correctly - Check that all the informations are stored in the database
Environmental needs	Active SMS service, or equivalent stub

Test case identifier	**S1I7-T1**
Test item(s)	Admin comp. → Notification manager
Input specification	Call a phone number
Output specification	Check that a phone call is performed to the provided phone number

Environmental needs	-

Test case identifier	**S1I7-T2**
Test item(s)	Admin comp. → Notification manager
Input specification	Send email to maintenance
Output specification	Check that the email is correctly sent to the specified email address
Environmental needs	-

Test case identifier	**S1I8-T1**
Test item(s)	Admin comp. → DB manager
Input specification	Update user information
Output specification	Updates data
Environmental needs	-

3.1.2 [S5] PE Car

Test case identifier	**S5I1-T1**
Test item(s)	Car controller → User input manager
Input specification	Automatically perform operations on the input manager, using the Android UI (with UI Automator).
Output specification	Check that the right Car controller methods are called.
Environmental needs	-

Test case identifier	**S5I2-T1**
Test item(s)	Car controller → Car diagnostics
Input specification	Apply changes e.g. to the battery status, malfunctioning reported by the hardware, and so on.
Output specification	Check that the Car controller reacts correctly to the changes.

Environmental needs	-

Test case identifier	**S5I3-T1**
Test item(s)	Car controller → Car communicator
Input specification	Test whether the controller can effectively use the communicator component to send messages to the server.
Output specification	Check that the methods are correctly called.
Environmental needs	-

Test case identifier	**S5I4-T1**
Test item(s)	Car controller → Presenter
Input specification	Apply a certain software configuration on the controller.
Output specification	Check that the correct data is rendered on screen (by using UI Automator) depending on the configuration.
Environmental needs	-

3.2 Subsystem integration

3.2.1 [S2] PE Mobile

Test case identifier	**SI2-T1**
Test item(s)	PE mobile, Application server
Input specification	Automatically register a new account using the Android UI (with *UI Automator* black-box testing).
Output specification	A user is successfully registered in the system.
Environmental needs	A working PE application server (connected to a DB) SI5 tests passed.

Test case identifier	**SI2-T2**

Test item(s)	PE mobile, Application server
Input specification	Automatically register a new account using the Apple iOS UI (with *UI Automation* black-box testing).
Output specification	A user is successfully registered in the system.
Environmental needs	A working PE application server (connected to a DB) SI5 tests passed.

3.2.2 [S3] PE Administrator

Test case identifier	**SI3-T1**
Test item(s)	PE administrator, Application server
Input specification	Automatically search for a car using the web app (with *Selenium* white-box testing).
Output specification	The car position is returned.
Environmental needs	A working PE application server (connected to a DB) SI5 tests passed.

Test case identifier	**SI3-T2**
Test item(s)	PE mobile, Application server
Input specification	Automatically run the “maintenance call” procedure (with *Selenium* white-box testing).
Output specification	An email is received on the maintenance team’s end.
Environmental needs	A working PE application server (connected to a DB and mail server) SI5 tests passed.

3.2.3 [S4] Web server

Test case identifier	**SI4-T1**
Test item(s)	PE web server, Application server
Input specification	Check that the HTML/CSS generated is valid by using the W3C validator.
Output specification	The HTML/CSS should be determined as valid.
Environmental needs	A working PE application server (connected to a DB) SI5 tests passed.

3.2.4 [S5] PE Car

Test case identifier	**SI5-T1**
Test item(s)	PE car, Application server
Input specification	Use white-box testing (with *Espresso*) to automatically run the following procedure: - "Start" driving - Simulate movement of the car - "Stop" driving
Output specification	The car should be reported as not available and then again as available, and the Application server should report the car movement as they happen.
Environmental needs	A working PE application server (connected to a DB) SI5 tests passed.

4 Tools and Test Equipment Required

4.1 Tools

For our testing purpose, we will employ a number of automated tools that provide us with industry-standard style checks, code inspection, unit testing and integration testing.

The tools that we will use are:

- **Selenium**, which is a suite of tools useful to automate browsers across different systems and platforms. With Selenium, we can automate tasks such as filling and sending a registration form, logging in, reserving a car and so on, all of this with only a few lines of code. We will use this mainly for front-end testing, e.g. to check that a certain action takes place when clicking some button.
- **Espresso** and **UI Automator**, which are Android tools useful to write *white box*-style and *black box*-style UI tests, respectively (that is: one has access to implementation details of the UI while the other just "pushes buttons" ignoring how the UI is implemented). We will use those to test that our mobile interfaces work correctly.
- Online HTML/CSS validators. There are a number of online services that can be used to validate HTML and CSS, e.g. to check that there are no unused CSS rules, or that some HTML tags haven't been closed. One of these online validators can be found at the URL: *http://validator.w3.org*
- **JUnit** is the de-facto standard for unit testing in the Java programming language. We will use it to write tests for each component of our system.
- **Mockito**, as suggested during lectures, is a mocking framework which can be very useful when testing a component that doesn't actually exists yet (e.g. even if there are no cars yet, we can already test components that depend on them by *mocking the behavior* of a car, i.e., it should expose a battery level, some coordinates, and so on). We will use Mockito mainly when developing integration tests.

Furthermore, on Android we will use testing and profiling tools provided by the Android Studio IDE. On iOS we will use testing and profiling tools provided by the Xcode IDE.

As for manual testing, we will delegate this task to a designated team: a **QA team**. This set of people will fulfill the task of making sure that the system works correctly and consistently, especially after enduring modifications. Moreover, to every ticket regarding the UI opened by the client, we will assign a member of the QA team: in this way he/she will triage the issue and, when the issue is fixed, will also confirm the ticket resolution.

4.2 Equipment

In order to perform the testing activities described, we will need to acquire special equipment. The reason for this is that the same software might look or feel differently under different hardware configurations (e.g. screen size).

This is a list of the equipment that we will require in order to perform the tests:

- A development-specific machine, where the latest version of the PowerEnjoy system is installed.
- At least one device for each supported client platform (Android, iOS, Windows Phone), possibly more. The exact number and variety will be decided by analyzing the market share of each brand/model/resolution.
- A machine with a number of different browsers installed: Internet Explorer 9+, Chrome [latest version - 3], Firefox [latest version - 3]. Additionally, on this machine Selenium will be installed.

5 Program Stubs and Test Data Required

5.1 Software integration

In order to test every component within every subsystem, we will need the following drivers and stubs:

Test(s)	Driver / Stub	Description of the driver / stub
S1I1	Gov Stub	Emulates a governmental system returning the validity of IDs and driving license
S1I1, S1I5	Bank Stub	Emulates a bank returning the validity of a payment method.
S1I2, S1I3	Car Stub	Emulates the response to application server requests
S1I4, S1I6	SMS_Phone Stub	Emulates the SMS and phone calls services
S5I1	Car Input Driver	Emulates user input for PE Car application
S5I2, S5I3, S5I4	Car Controller Driver	Emulates the controller of PE Car.

5.2 Subsystem integration

At the level of subsystem integration, we use bottom-up approach, so we will need the following drivers:

Test	Subsystems	Driver / Stub	Description of the driver / stub
SI1	PE Mobile → Application server	MobileDriver	Sends RESTful requests to Application server emulating a normal Mobile application
SI2	PE Administrator → Application server	AdminDriver	Sends requests to Application server emulating the typical operations of a call center operator and an administrator
SI3	Web server → Application server	WebServerDriver	Sends requests to Application server emulating the PEW application
SI4	PE Car → Application server	CarDriver	Emulates the typical requests done by a PE Car.

SI4*	Application server → PE Car	Car Stub	Emulates the response to application server requests
SI5	Application server → DBMS	ApplicationServer Driver	Sends queries to DBMS

*Since the application server and PE Car are dependent each other we need either a car stub and a car driver to integrate.

6 Appendix

6.1 Used tools

- Google Docs (http://www.docs.google.com) for this document
- Star UML (http://staruml.io/) for the UML modelling, in particular use case, statechart, activity, class and sequence diagrams
- Alloy Model 4.2 (http://alloy.mit.edu/alloy) to generate the world and proving its consistency
- Balsamiq Mockup (http://balsamiq.com/products/mockups/) for the sketch on the user interfaces of this software system

6.2 Hours of work

The following are the total number of hours spent by each member of the group:

- Andrea Battistello: 15 h
- William di Luigi: 15 h

6.3 Revision history

Version	*Date*	*Revision description*	*Revision notes*
1.0	14-01-2017	First release	-