PowerEnJoy

Politecnico di Milano

Industrial and Information Engineering Computer Science and Engineering

Cattaneo Davide

El Hariry Matteo

Frontino Francesco

## Integration Test Plan Document

Contents

1. ***Introduction***
   1. Purpose and scope
   2. Definitions, acronyms and abbreviations
   3. Reference documents
2. ***Integration strategy***
   1. Overview 8
   2. Elements to be integrated 9
   3. Integration testing strategy 10
   4. Integration sequence 13
3. ***Tests description***
   1. Nome del test 1 21
   2. Nome del test 2 23
4. ***Tools and equipment***
   1. Tools 27
   2. Equipment 29
5. ***Revision***
   1. Team work 34
6. ***Introduction***
   1. ***Purpose and scope***

Purpose of this document is to define an approach for testing software during the development phase. The integration and test plan document (ITPD) provides guidelines for the developers about which components of the software platform are to be tested and how to test them.

Both integration and standalone module testing methodologies are reported in this document.

* 1. ***Definitions, acronyms and abbreviations***

Here is a brief description of the most important actors and words used in our system:

* **User:** by user is meant a person already registered in the system, so that has a profile, uses the features provided by the system and performs actions accordingly. (S)He can use all the functionalities described below (see Functional Requirements).
* **Guest:** a guest is a person that probably for the first time accesses the system or that hasn’t already signed up. Guest has less power in the system; his/her actions are limited to access an introduction view and register to the service.
* **System:** is the application core. The software system which will perform all the operations and monitor interactions and be a medium between users and cars.
* **Reservation:** the allocation of a car to a user, which starts when the booking request arrives and ends either when the expiration time ends or when the car is unlocked. In this last case it triggers the start of the first travel so it initiates a ride.
* **Car:** the vehicle used by the users, which contains different sensors and an embedded computer. It has seat sensors to detect passengers, sensor to know battery level and charging actions. The computer, of course, has as main functionality to provide navigation facilities through a GPS system and to send all the relevant data to the main system server.
* **Ride:** conceptually is the use of the car, and it can be identified by the time duration of the user’s journey, from unlocking the vehicle until the final parking (having user selecting “end ride” or “end ride & charge” on the car screen) with the car locked.
* **Travel:** is considered as the ride segment and is identified by a change of the status of the car. More travels can be part of a single ride.
* **Operator:** is a flexible actor in our system. He’s part of a set of people operating under the administrator directions. Their normal tasks are to bring to charging stations cars left with less than 15% battery level, interact with users which call for help during a ride, intervene when necessary (e.g. a wheel brakes during a ride). Their exceptional task can be the case in which they have to go and get back cars taken by the police or cars involved in incidents etc.
* **Administrator:** the administrator of the system is the person allowed to manage eventual unexpected cases (like incidents and damaging situations). He is the person notified every time a problem occurs, and once analyzed the situation (s)he’ll decide how to handle it (call for support, send operators, call the police etc.).
* **Safe Area:** is a part of a set of areas considered safe for parking cars after a ride is over. Temporary stops can be everywhere, but long term parks can only occur in safe areas. They must be very spread and every neighborhood should have at least one.
* **Normal rate:** the charging rate applied when the car engine is ON.
* **Halt rate:** the charging rate applied when the engine is OFF and either the user is inside or (s)he has parked the car in temporary stop mode.
* **Board Controller:** BC is the car system, which includes all the hardware and software components interacting with the vehicle itself, with the users’ smartphone and the Central System. The Main components part of the BC are the CAN controller, the Android System and the car display. All of them are interconnected in order to guarantee an efficient flow of information from and to sensors and with the outside environment (users and central system).
* **CAN:** A Controller Area Network (CAN bus) is a [vehicle bus](https://en.wikipedia.org/wiki/Vehicle_bus) standard designed to allow [microcontrollers](https://en.wikipedia.org/wiki/Microcontroller) and devices to communicate with each other in applications without a [host computer](https://en.wikipedia.org/wiki/Host_computer). It is a [message-based protocol](https://en.wikipedia.org/wiki/Message-based_protocol), designed originally for [multiplex](https://en.wikipedia.org/wiki/Multiplexing) electrical wiring within automobiles, but is also used in many other contexts.
  1. ***Reference documents***

Specification Document:

\* Assignments 1 and 2 (RASD and DD).pdf

\* Verification and validation, part I and II

\* Verification Tools

Examples documents:  
\* Sample Integration Test Plan Document

\* Integration testing example document

1. ***Integration strategy***
   1. ***Overview***

To provide a correct, robust and resilient software, the test plan aims to ensure that at least 80% of the application code results to be covered by unity testing. At this purpose, this sections provides the major guidelines to follow during the integration and testing phase. Those reported in this document represent the minimum set of test for guaranteeing a sufficient level of correctness and robustness.

* 1. ***Elements to be integrated***

Starting from the previous releases of the project (RASD and DD) these high-level core modules can be identified in the system:

1. the Database System
2. the Central System (application core)
3. the Board Controller (car system)
4. the User App
5. the Administration System
6. the Notifier
7. the Payment Gateway

In particular, these are the high-level components associated with each module that must be tested:

1. the Database System

* DBMS

1. the Central System (application core)

* UserManager
* CarManager
* RideManager

1. the Board Controller (car system)

* RemoteController
* RideFunction
* AssistanceRequest
* MapService

1. the User App

* AccountFunctions
* RideFunctions
* RemoteCar

1. the Administration System

* MaintenanceFeautures
* RemoteCar
* AccountFunctions

1. the Notifier
2. the Payment Gateway

As a first step a low level of integration among the components of each module can be described.

1. Starting from the Database, the goal is to develop a solid and reliable platform for saving and retrieving data, such as the electric-cars being part of the sharing system, the users accounts and so on. To explain better this phase, an Entity Relationship Diagram will be provided to support the design of the database and its testing phase.
2. Now the focus must be on the Central System, which is the core of all the interactions of the PowerEnJoy reality. Here some more details on how each component is constructed to be provided:

- UserManager

- CarManager

- RideManager

1. Board Controller is the following subsystem to be tested, this in order to evaluate the efficiency of the car system components and their connectivity with the environment and the headquarter (CS).
2. Once the previous modules are exanimated the User App can be assessed by checking its working behavior.  
   In our specific case, this involves the integration of the AccountFunctions, RideFunctions, AssistanceRequest and RemoteCar subcomponents in order to obtain the UserApp subsystem.
3. For what concerns the building of the Administration System the components involved are the Maintenance Features and RemoteCar. The first is meant to allow both administrators and operators to perform tasks through the browser portal accessible with their credentials, the second is the component allowing the set of operations to interact with the vehicles.
4. (7.) Finally, two commercial, already existing components are used to achieve specific functionalities: Notification System and Payment Gateway.

All the above components and subsystems specifically developed for PowerEnJoy can be seen using the architectural client/server point of view:

– On the server side: Ride Management System, System Administration and Account Management subsystems component.

– On the client side: Administration Web Application, User Web Application, User Mobile Application and the Board Controller (Car system manager) components.

* 1. ***Integration testing strategy***

To test these modules and their interaction a bottom-up approach will be implemented.

**Bottom up testing** will be conducted from sub module to main module, if the main module is not developed a temporary program called DRIVERS is used to simulate the main module.  
  
**Advantages:**  
  
- Advantageous if major flaws occur toward the bottom of the program.  
- Test conditions are easier to create.  
- Observation of test results is easier.

The strategy for testing and integration consists of 4 phases:

1. Testing the application core
2. Testing maintenance logic
3. Test the app interaction with the system
4. Test the car interaction with the system

These 4 phases are to be considered separately.

1. ***Testing the application core:***

This first phase consists of:

1. Testing each single element reported in paragraph 2.2 (except maintenance manager) via unity testing
2. Integrating all the elements using a big bang strategy
3. Test the whole system core
4. ***Testing maintenance logic:***

The second phase consists of:

1. Test the MaintenanceManager via unity testing
2. Integrate the MaintenanceManager in the system core
3. Test maintenance requests
4. ***Test the app interaction with the system:***

The third phase consists of:

1. Test the application via unity testing
2. Test the interaction with the core system
3. ***Test the car interaction with the system:***

The fourth phase consists of:

1. Test the application via unity testing
2. Test the interaction with the system core

This strategy aims at achieving a gradual integration by first testing the main logic, simulating every possible command coming from the “world”, and then adding step by step a new element that replaces the precedent stubs to verify whether the real components act like they were meant to do.

Given that JEE is a tied framework, assumptions about the correctness of data storage and communications between layers are made. Moreover, HTTP requests for maintenance, reservation etc. are considered always working due to the robustness of the protocols.

The platform relies on two external services (payment gateway and Google APIs) that can’t be tested and are assumed to be fully working for the system purpose.

* 1. ***Integration sequence***

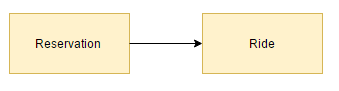
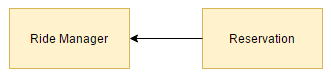
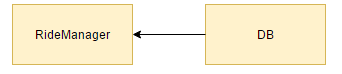
Following the already mentioned bottom-up approach, we now describe how

the various subcomponents are integrated together to create higher level

subsystems.

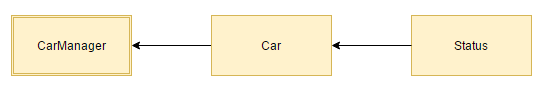
All the low level components in the system will be integrated according to the following diagrams and descriptions.

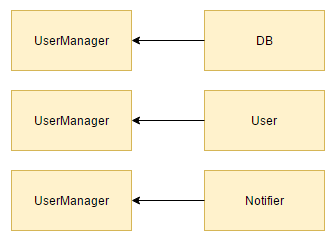
* Central System subsystems integration:
* RideManager component:
* By integrating the reservation with the ride it can be checked if the two instances communicate properly (e.g. an allocation of a ride instance must start once a reservation is completed).
* The same operation is done with ride manager and reservation.
* Finally, the ride manager is integrated with the database, checking that all the operations needed to be saved will correctly update the database.

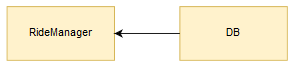
  
   
 

* CarManager component:

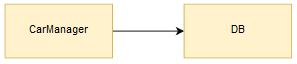
The Car Manager instance is the core of this subsystem and it needs the integration with the car and with the status subcomponents.



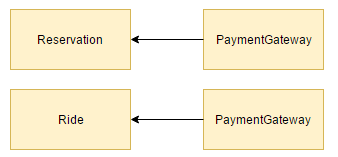
* UserManager component:  
    
  
* Now, all these three subsystems must be integrated singularly with the database:



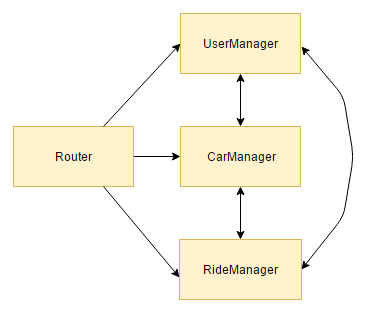




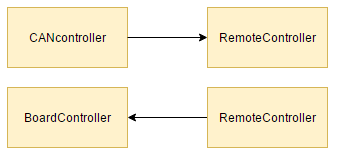
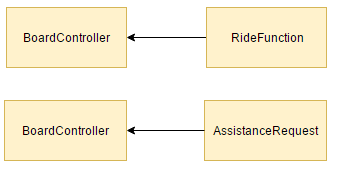
* An integration of the ride and reservation modules must be performed with the payment gateway

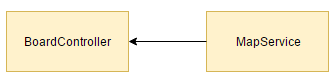


* The main module is now developed and a temporary program called DRIVER is used to simulate it as follows.  
  Through the Router accesses the right connectivity and exchange of data between the three main subsystems with the outside environment can be checked. Also the correct workflow between the inner components themselves is tested here.

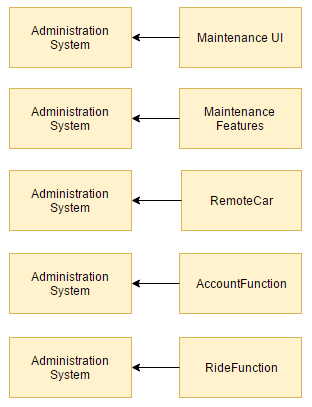


* Board Controller subsystems integration:

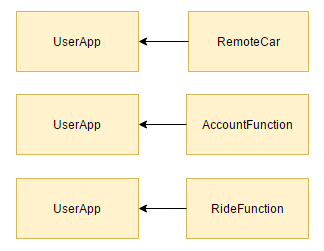
 



* Administration System components integration:



* User App components integration:



1. ***Tests description***
   1. ***Integration test case I***
2. ***Tools and equipment***
   1. ***Tools***

Testing procedures are to be executed using one of the following operating systems:

* Windows 10
* MacOS 10.12
* Linux 4.8

due to guarantee the application to be working in the most modern environments.

Both the car application and the system core must be developed using the latest Eclipse IDE version and must be based on Java 7 APIs. The framework used to develop the business logic must be JEE 7.

Junit is the only framework suggested for testing the applications. The use of other frameworks aiming at improving the quality of testing is accepted.

To develop the web site, is mandatory to use HTML 5 / CSS 3 / JS primitives using Bootstrap as a base layer. No peculiar IDE nor editor is suggested in this case.

* 1. ***Equipment***

To achieve a full testing, the following equipment is needed:

* Android smartphone (O.S. v4.4 or greater)
* An iOS smartphone (O.S. v7.0 or greater)
* A Windows Mobile smartphone (O.S. v10.0 or greater)
* An ARM mobile development board provided with Bluetooth, mobile connection, GPS and NFC modules.

For each smartphone is necessary to test the mobile application and the web site.

The ARM development platform used for testing the car application must be compliant with the specifications previously given (RASD).

No peculiar hardware is suggested for each element of the equipment but the following minimum requirements must be granted:

* Mobile devices:
  + 1.8 GHz CPU or above
  + 4G connection
  + Bluetooth 4.2 or above
* Development board:
  + 1.8 GHz CPU or above
  + High precision GPS

Due to the web site to run on desktop / laptop machines too, it is enough for it to be fully usable at least on two of the machines used for the development. No peculiar hardware is needed for the PCs.

1. ***Revision***
   * 1. ***Team work***

Here is reported a compact table showing how the work was brought on by all the members of the group.

Document work finished on the xth January 2017.

