# 

**POLITECNICO DI MILANO**

# SOFTWARE ENGINEERING 2 PROJECT

# A.Y. 2016-17

**PowerEnJoy**

Integration Test Plan Document

Enrico Migliorini, Alessandro Paglialonga, Simone Perriello

TODO Data

# **Contents**

1. **Introduction** 
   1. Revision History
   2. Purpose and Scope
   3. Definitions, Acronyms and Abbreviations
   4. Reference Documents
2. **Integration Strategy**
   1. Entry Criteria
   2. Elements to be Integrated
   3. Integration Testing Strategy
   4. Sequence of Component/Function Integration
      1. Software Integration Sequence
      2. Subsystem Integration Sequence

Chapter 1

**Introduction**

## Revision History

TODO TABELLA

## Purpose and Scope

This Document represents the Integration Test Plan Document for the ***PowerEnJoy***project.

This kind of testing is performed to find defects which are generally related to inter-process communication or parameter and data inputs. Two units may have passed unit testing and work well individually, but fail to communicate vital information with one another. By testing the units in aggregate, it’s easier to identify the source of the issues found.

The purpose of this document is to outline, in a clear and comprehensive way, the main aspects concerning the organization of the integration testing activity for all the components that make the system up.

The aim of the ***PowerEnJoy***project is to provide a *Car-Sharing Service* which implements electric-powered cars only. This system will have to interface the Cars, Charging Areas, allowing Users to reserve, unlock, drive and park Cars, finally they will be charged the cost of the ride.

In the following sections will be provided :

* A list of the subsystems and their subcomponents involved in the integration activity that will have to be tested.
* The entry criteria that must be met before the integration testing of the specified elements may begin.
* The reasoning process which led to the decision of the integration testing approach and the description of the latter. This includes the order of sequence in which components will be integrated.
* A list of all the tools employed during the testing activities.

## Definitions, Acronyms and Abbreviations

DBMS

## Reference Documents

Chapter 2

**Integration Strategy**

## Entry Criteria

Before this document is written, and for the steps written here to be meaningful, the **Requirements Analysis and Specification Document** and the **Design Document** of the *PowerEnJoy* projectmust have been fully written.

The second required step before the integration test is performed is the Unit Testing of each component.

Unit Testing verifies the correct functioning of each module and its methods according to their specification; it is used to test a specific unit (class) in the application testing all the methods in the class including all exception paths in the methods. Unit Tests only cover testing of each module in the application, hence we can’t test the functional requirements of the application or scenarios.

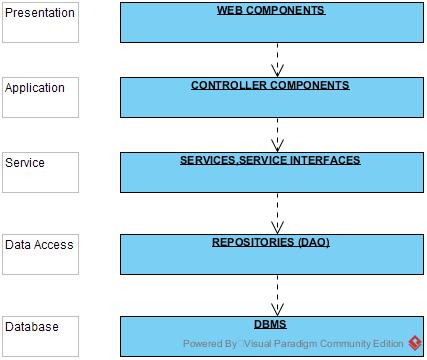
Unit testing of the DBMS, Mapping System, Banking System have already been done by their respective software producers.

## Elements to be Integrated

In this paragraph we report the list of the components to be integrated.

Starting from the bottom layer of the architecture to go along with our Integration Testing Strategy (see paragraph 2.3) we’ll integrate starting from Server-Side : **Repositories** with the **DBMS, Repositories** with **Services, Services** with **Service Interfaces, Service Interfaces** with **External Systems, Services** with **Controllers**, **Controllers** with the **Dispatcher, Dispatcher** with the **MobileApp/WebClient.**

**PowerEnJoy Application Layered Architecture Diagram**



## Integration Testing Strategy

During the integration testing strategy selection several popular choices were evaluated by the team. Different strategies exist for the Integration Testing which all have benefits and drawbacks depending on the project to implement, Top Down, Sandwich, Bottom Up, Critical modules first and Big Bang; there’s no strategy which is clearly better than the others as each approach is perfectly suited for a specific type of architecture of components and communication between components. Consequently, the team has evaluated the different methodologies looking at the big picture of this project.

The choice which seemed to best fulfill our requirements was the Bottom Up approach. During the evaluation some of the advantages which led our choice were :

* The absence of urgency for any kind of program stubs, which is needed in top-down approach instead, as we always start developing and testing with absolute modules
* Starting from the bottom of the hierarchy means that the critical modules are generally built and tested first and therefore any errors or mistakes in these forms of modules are find out early in the process.
* Fault localization is easier and no time is wasted waiting for all modules to be developed unlike Big-Bang approach.

During the evaluation some of the drawbacks which we accepted existing for our project were :

* We always have to form the test drivers which are the simulated environment in which the component being tested is to integrated. In addition, drivers have to be tested themselves and so more time and efforts are needed to accomplish this complicated task.
* The application as an entity does not exist until the last module is added.