POLITECNICO
MILANO 1863

POWER ENJOY

Integration Test Plan Document

Lorenzo Casalino - 877421

Tommaso Castagna

Document version 1.0

Contents

1 Introduction

1.1 Revision History

The history of document revisions is here recorded in tabular format, mapping the document version with the major changes brought to.

The current version of the document is highlighted by the version number in bold format.

Version	Revision
1.0	Document first final version.

1.2 Purpose and Scope

1.2.1 Purpose

The *Integration Test Plan Document*, also referred to with the acronym of *ITDP*, aims to provide to the development team the path to follow for the integration testing process of the software system through a complete description of the elements of the system to test, the integration strategy to adopt, the integration sequence forecasted and stubs/drivers and tools needed to accomplish the integration test phase.

1.2.2 Scope

The hereby Integration Test Plan Document presents a detailed description of the integration testing plan that the development team should follow to accomplish correctly the integration testing process.

This document contains a detailed description of the integration testing plan that the development team should follow to successfully complete the integration testing process.

The scope of this document covers four main topics regarding the integration phase. Here, the scope is presented reflecting the main structure of the ITPD, allowing the reader to better understand what is discussed in the following sections.

- **Integration Strategy:** the entry criteria, both general and specific, that must be met before the integration test take place, what are the subsystems to integrate, which integration strategies are adopted and the integration sequence to follow are deeply described and discussed, pointing out the rationale of each choice.

- **Individual Steps and Test Description:** the type of test performed in each step of the integration process and applied to each component and subsystem, along with the expected results, are presented.
- **Tools and Test Equipment:** the needed tools to perform the integration are presented, and briefly described their usage during the integration process.
- **Program Stubs and Test Data Required:** possible stubs/drivers or special data required to proceed in each step of the integration process are presented and described.

1.3 Definitions, Acronyms and Abbreviations

1.3.1 Definitions

- **Software Component:** atomic piece of software that compose the software system.
- **Design Document**: document describing the software system architecture, the design choices and their rationale.
- **Integration Test Phase:** project development phase where the components and subsystems of the software system designed are integrated and tested in order to verify the correctness of their implementation and presence of design flaws.
- **Integration Test Plan Document:** document used to guide the integration test phase.
- **Requirement Analysis and Specification Document:** document regarding the analysis of the goals of the project stakeholder and the functional and non-functional requirements of the software system to develop.
- **Software System:** the software system that is currently under development.
- **Subsystem:** part of the software system composed by two ore more software components.

1.3.2 Acronyms

- **DD**: Design Document.
- **ITPD**: Integration Test Plan Document.
- **RASD**: Requirement Analysis and Specification Document.

1.3.3 Abbreviations

- **Integration phase:** integration test phase
- **Integration process:** integration test process
- **System:** software system.

2 Integration Strategy

2.1 Entry Criteria

Before the integration testing phase of specific components may take place, it is fundamental that the following criteria (or conditions) here described are satisfied.

The verification of these conditions is really important in order to have as outputs of the integration test phase meaningful results, useful to assess the quality of the software system designed and, possibly, improve it.

It's worthful point out that some of the criteria presented are strictly tied to the kind of strategy chosen to perform the integration testing of the system's components, while others are more general and act as pre-condition for the whole integration test process.

For informations about the integration testing strategy picked out for this software system, please refer to the *Integration Testing Strategy* section.

GENERAL CRITERIA

- **RASD complete draw up:** the RASD has the main function of thoroughly document the functionalities and requirements of the software system. Because of its purpose, it is the main source of informations and comparison that the development team has to refer to check the results obtained after the integration test of components and subsystems.
- **RASD positive assessment**: since the development team refers to the RASD to veirify the results of the integration tests, it's fundamental that the RASD content reflects the goals of the stakeholders involved into the project. Documented functionalities diverging from the stakeholders' desires lead to wrong implementations, regardless of the integration test results.
- **DD complete draw up:** the design document has the purpose to describe the system's architecture and the design choices taken. These informations are essential in the development phase, where the implementation of the described software architecture take place. Furthermore, the verification of the result obtained from the integration test of subsystems relies on the content described in the DD.
- **DD positive assessment:** development and integration phase rely heavily on the informations contained in the Design Document, therefore the positive assessment of its content is an essential criterion for the correct development and integration of system's components.
- **ITPD complete draw up:** the ITPD describes all the aspects regarding the integration test phase. Without it, the integration process cannot take

place.

- **ITPD positive assessment:** the integration test choices and the path planned must comply and being consistent with the RASD and DD documents. A divergence between what is stated in the ITPD document and what is stated in the RASD and/or in the DD compromises the entire integration and development phases.

SUBSYSTEMS AND COMPONENTS TEST CRITERIA

- **Creation of required drivers:** the integration of some components may need the presence of components not already developed. To overcome this obstacle, the scaffolding process comes in play, designing the drivers needed to emulate the required components.
- **Complete component static and dynamic analysis:** the supporting functionalities provided by the components to be integrated must thoroughly inspected through static analysis methods, such as code inspection, and dynamic analysis methods, i.e. unit testing. This step is important because allows to discover possible fault in the implementation, easing and speeding up the integration testing process.
- **Regression testing:** before starting the integration testing of a certain functionality through new test cases, the old test cases should be run in order to verify the compatibility and the correcteness of the new integration. This process should be executed before the new functionality testing because if the old tests show an incompatibility, the components integration must be reviewed.

2.2 Elements to be Integrated

In the following, the system's elements to be integrated are spotted and described. With *elements* here we mean the subsystems composing the system's design. The description of a subsystem is recursive, meaning that if a subsystem is composed by other subsystems, even these are described. Description of atomic components is avoided since are well described into the DD.

The main *elements* composing our system are:

- **Data Tier:** This element represents the DBMS. Even though we are not developing the DBMS itself, it is still part of the system, hence it has to be integrated.
- **Business Logic:** This element contains all the application logic present in our system. The main components of this element are: the **Authentication Manager**, the **Account Manager**, the **Maintenance Manager**, the **Vehicle Manager** and the **Reservation Manager**.

- **Web Server:** This is the element responsible for the communications between the business logic tier and the clients. It exposes a RESTful implementation of the communication interfaces.
- **Client:** This element contains the three different kind of clients that can access our application, namely the **mobile client**, the **web application client** and the **car central system**

Every component that forms the different *elements* has aledy been tested individually, the tests to be performed are applied to the interfaces that connect a component with another one.

2.3 Integration Testing Strategy

The architecture designed for the system software under development [DD 2.2] [DD 2.3] is characterized by a **small extension**, namely a restricted number of components, and by a **loose coupling factor**, due to the presence of several indipendent components.

Making reference to the integration test strategies present in software engineering literature, the category of **structural integration strategies** results to be the most indicated for small and medium projects. Other integration categories, like the functional-oriented strategies, require a planning effort that overcomes the output of the process, making them useless.

Among the strategies belonging to the structural category, the **Bottom-up strategy** proves to be the most appropriated approach, exploiting at the best the simplicity of the system's architecture, avoiding complex and unnecessary scaffolding, and the presence of low level indipendent components.

Starting from the bottom of the architecture's hierarchy, the process integrates step by step the indipendent components into small subsystems, shifting afterwards to integrate the small subsystems obtained from the previous steps, climbing up the hierarchy until all the components are integrated and tested.

Each step of the integration testing activity is supported by the implementation of *drivers*, which emulate the behaviour and the calls performed by one higher-level components towards the lower components to which it is linked, allowing the integration and testing of the architecture. Once one of the small subsystems is thoroughly tested and integrated, the driver can be substituted by the real implementation of the component emulated.

2.4 Sequence of Component/Function Integration

2.4.1 Software Integration Sequence

In this section the integration sequence of the system's components will be described. Due to the fact that we rely on the **Java Persistence API (JPA)** the integration with the database is checked every time a component is tested.

The component to the left of the arrow is needed to the component to the right in order for it to function.

ID	Integration Test		
I1	**Area Manager**	→	**Map Manager**
I2	**Vehicle Manager**	→	**Map Manager**
I3	**Vehicle Manager**	→	**Maintenance Manager**
I4	**Area Manager**	→	**Maintenance Manager**
I5	**Payment Manager**	→	**Account Manager**
I6	**Driving License Manager**	→	**Account Manager**
I7	**Fares and Policies Manager**	→	**Reservation Manager**
I8	**Payment Manager**	→	**Reservation Manager**
I9	**Map Manager**	→	**Reservation Manager**
I10	**Vehicle Manager**	→	**Reservation Manager**
I11	**Maintenance Manager**	→	**Authentication Manager**
I12	**Vehicle Manager**	→	**Authentication Manager**
I13	**Fares and Policies Manager**	→	**Authentication Manager**
I14	**Area Manager**	→	**Authentication Manager**
I15	**Reservation Manager**	→	**Authentication Manager**
I16	**Account Manager**	→	**Authentication Manager**

2.4.2 Subsystem Integration Sequence

After all the subsystems are individually tested we can start the integration between them using the following order:

Web Server → Business Logic
Client → Web Server

3 Individual Steps and Test Description

3.1 Area Manager → Map Manager

Retrive the parking areas around a location	
Parameter(s)	Location, Radius
Pre-condition(s)	**Post-condition(s)**
Location or radius parameters are not encoded correctly.	Returns a specific error.
No areas are found in the specified location.	Returns a message prompting the user to change the parameters because there is not an area that meets the specified parameters.
One or more parking areas meet the specified parameters.	Returns a list of all the areas found in the specified location with the respective information.

Retrive a specific parking area	
Parameters	Area identificator
Pre-condition(s)	**Post-condition(s)**
The parameter doesn't match the area identificator design pattern.	Returns in an encoded form a specific error.
There is no area corresponding to the provided identificator.	Returns in an encoded form a specific error.
There is an area corresponding to the specified identificator.	Returns in an encoded form the information regarding the parking area.

3.2 Vehicle Manager → Map Manager

Retrive the cars around a location	
Parameters	Location, Radius
Pre-condition(s)	**Post-condition(s)**
Location or radius parameters are not encoded correctly.	Returns a specific error.
No cars are found in the specified location.	Returns a message prompting the user to change the parameters because there is not a car that meets the specified parameters.
One or more cars meet the specified parameters.	Returns a list of all the cars found in the specified location with the relative information.

Retrive a specific car	
Parameters	Car identificator
Pre-condition(s)	**Post-condition(s)**
The parameter doesn't match the car identificator design pattern.	Returns a specific error.
There is no car associated to the provided identificator.	Returns a specific error.
There is a car corresponding to the specified identificator.	Returns the information regarding the car.

3.3 Payment Manager → Account Manager

Check the payment information of a registering user	
Parameter(s)	Payment information.
Pre-condition(s)	**Post-condition(s)**
The payment information does not respect the designed pattern.	Returns a specific error. Nothing is changed in the system.
There's no payment account relatad to the specified information.	Returns a notification prompting the user to change the payment information provided in the registration form.
There is a valid payment account related to the specified information.	Returns a positive notification and let the system proceed with the registration function.

Change the payment information of a registered user	
Parameter(s)	User information, Payment information.
Pre-condition(s)	**Post-condition(s)**
The payment information or the user information do not respect the designed pattern.	Returns a specific error. Nothing is changed in the system.
There's no user account relatad to the specified information.	Returns a specific error. Nothing is changed in the system.
There's no payment account relatad to the specified information.	Returns a notification prompting the user to change the payment information provided.
There is a valid payment account related to the specified information.	Returns a positive notification and changes the payment information of the specified user.

Retrieve the payment information of a registered user

Parameter(s)	User information, Payment information.
Pre-condition(s)	**Post-condition(s)**
The payment information or the user information do not respect the designed pattern.	Returns a specific error. Nothing is changed in the system.
There's no user account relatad to the specified information.	Returns a specific error. Nothing is changed in the system.
There's no payment account relatad to the specified information.	Returns a specific error. Nothing is changed in the system.
There is a user that meets the specified information.	Returns a positive notification and the information relative to the specified payment account.

3.4 Driving License Manager → Account Manager

Check the driving license information of a registering user	
Parameter(s)	Driving license information.
Pre-condition(s)	**Post-condition(s)**
The driving license information does not respect the designed pattern.	Returns a specific error. Nothing is changed in the system.
There's no driving license relatad to the specified information.	Returns a notification prompting the user to change the driving license information provided in the registration form.
There is a valid driving license related to the specified information.	Returns a positive notification and let the system proceed with the registration function.

Change the driving license information of a registered user	
Parameter(s)	User information, Driving license information.
Pre-condition(s)	**Post-condition(s)**
The driving license information or the user information do not respect the designed pattern.	Returns a specific error. Nothing is changed in the system.
There's no user account relatad to the specified information.	Returns a specific error. Nothing is changed in the system.
There's no driving license relatad to the specified information.	Returns a notification prompting the user to change the driving license information provided.

There is a valid driving license related to the specified information.	Returns a positive notification and changes the driving license information of the specified user.

Retrieve the payment information of a registered user	
Parameter(s)	User information, Driving license information.
Pre-condition(s)	**Post-condition(s)**
The driving license information or the user information do not respect the designed pattern.	Returns a specific error. Nothing is changed in the system.
There's no user account relatad to the specified information.	Returns a specific error. Nothing is changed in the system.
There's no driving license relatad to the specified information.	Returns a specific error. Nothing is changed in the system.
There is a user that meets the specified information.	Returns a positive notification and the information relative to the specified driving license.

3.5 Fares And Policies Manager → Reservation Manager

Retrieve rule	
Parameter(s)	Rule identificator.
Pre-condition(s)	**Post-condition(s)**
The rule identificator does not respect the designed input patter.	Returns a specific error. Nothing is changed in the system.
No rule is matched by the rule identificator.	Returns a specific error. Nothing is changed in the system.
A rule is matched by the rule identificator.	Returns a positive notification. The encoded rule mapped by the rule identificator is returned.

3.6 Payment Manager → Reservation Manager

Deduct the ride cost from the payment account of the user	
Parameter(s)	User's payment information, Ride cost.
Pre-condition(s)	**Post-condition(s)**
The user's payment information or the ride cost does not respect the designed input pattern.	Returns a specific error. Nothing is changed in the system.

The user's payment information don't match a payment account in the payment manager.	Returns a specific error. The corresponding user is marked as insolvent, this prevents the user from reserving another car.
The user's payment account doesn't have sufficent money to pay the ride.	Returns a specific error. The payment account is drained of the remaining money and the corresponding user is marked as insolvent, this prevents the user from reserving another car.
The user's payment account have sufficent money to pay the ride.	Returns a positive notification and the ride's cost is deducted from the payment account.

3.7 Map Manager → Reservation Manager

Retrive the available cars and all the parking areas around a location	
Parameters	Location, Radius
Pre-condition(s)	**Post-condition(s)**
Location or radius parameters are not encoded correctly.	Returns a specific error.
No cars that meet the specified requirements are found in the map cache maintained by the map manager.	The map manager forwards the request to the vehicle manager and to the area manager and awaits the response from those components. If the response is positive the map manager returns a list of all the cars and areas found in the specified location with the relative information, whereas if the response is negative the map manager returns a message prompting the user to change the parameters of the search.
One or more cars and zero or more parking areas that meet the specified parameters are found in the map cache.	Returns a list of all the cars and areas found in the specified location with the relative information.

Retrive a specific car	
Parameters	Car identificator
Pre-condition(s)	**Post-condition(s)**
The parameter doesn't match the car identificator design pattern.	Returns a specific error.

The parameter match the car identificator design pattern.	The map manager forwards the request to the vehicle manager and awaits for the response. If the response is positive the map manager returns all the information regarding the specified car, if the response is negative the map manager returns a message that prompts the user to change the parameters of the search.

3.8 Vehicle manager → Reservation Manager

Enable the unlocking mechanism of a reserved car	
Parameters	User identificator, Car identificator
Pre-condition(s)	**Post-condition(s)**
The parameters do not match the design pattern.	Returns a specific error and nothing changes in the system.
The user identificator doesn't match any user in the system.	Returns a specific error and nothing changes in the system.
The car identificator doesn't match any car in the system.	Returns a specific error and nothing changes in the system.
The specified user hasn't reserved the specified car.	Returns a specific error and nothing changes in the system.
The specified car is alredy unlocked.	Returns a specific error and nothing changes in the system.
The specified car's unlocking system has alredy been enabled.	Returns a specific error and nothing changes in the system.
The specified user has an active reservation on the specified car, the car's unlocking system is not enabled and the car is locked.	Returns a positive notification, enables the unlocking system and sets the relative timer.

Unlock a reserved car	
Parameters	User identificator, Car identificator, Unlocking code
Pre-condition(s)	**Post-condition(s)**
The parameters do not match the design pattern.	Returns a specific error and nothing changes in the system.
The user identificator doesn't match any user in the system.	Returns a specific error and nothing changes in the system.
The car identificator doesn't match any car in the system.	Returns a specific error and nothing changes in the system.

The specified user hasn't reserved the specified car.	Returns a specific error and nothing changes in the system.
The specified car is alredy unlocked.	Returns a specific error and nothing changes in the system.
The specified car's unlocking system is not enabled.	Returns a specific error and nothing changes in the system.
The specified user has an active reservation on the specified car, the car is locked, the unlocking system is enabled but the unlocking code is wrong.	Returns a message warning the user that the code is wrong, prompting him to enter the correct code.
The specified user has an active reservation on the specified car, the car is locked, the unlocking system is enabled and the unlocking code is correct.	Returns a positive notification, unlocks the car and starts the reservation.

Lock a reserved car	
Parameters	User identificator, Car identificator
Pre-condition(s)	**Post-condition(s)**
The parameters do not match the design pattern.	Returns a specific error and nothing changes in the system.
The user identificator doesn't match any user in the system.	Returns a specific error and nothing changes in the system.
The car identificator doesn't match any car in the system.	Returns a specific error and nothing changes in the system.
The specified user hasn't reserved the specified car.	Returns a specific error and nothing changes in the system.
The specified car is alredy locked.	Returns a specific error and nothing changes in the system.
The specified car presernts at least one of the following features: • The engine is turned on. • The key is not inserted in the car's panel. • There's a passenger inside the car. • The car has a door open. thus can't be unlocked.	Returns a specific error and nothing changes in the system.
The specified user has an active reservation on the specified car and the car can be unlocked (doesen't present any of the features listed above).	Returns a positive notification and locks the car.

4 Tools and Test Equipment Required

In this section are described what tools and equipment are used to aid the integration and testing activity and how they insert in it, easing the overall process.

Some of the tools here presented refer to architectures implemented through the Java Enterprise Edition technology, one of the most spread technology, but is possible to find equivalent tools for other technologies as well, i.e. Nunit for unit testing on C#-based architectures.

4.1 Tools

- **Mockito:** java framework developed to support the integration and the testing activities through the creation of so-called *Mock components*. In this projects, it supports the bottom-up strategy, providing at each step of the integration activity the required drivers.
- **JMeter:** software used to execute performance and stress analysis against the subsystems through the creation of custom data loads. It's used to test the integration between the webserver and the business logic, simulating different and several requests forwarded to the former.
- **Junit:** java testing framework developed to support the unit testing activity. It's used to perform unit testing over indipendent and integrated components.
- **Code inspection:** manual testing tool used as a preliminary analysis of the individual components implementation.

4.2 Equipment

- **Kali Linux:** operating system based on linux containing a complete suite for penetration testing. It's used to perform penetration testing against various aspects of the software system, such as communications reliability and authentication mechanism.
- **Mobile performance analysis suites:** software suites developed from different mobile's vendor used. It is used to execute performance analysis against the mobile application developed for the car's sharing system.
- **Vehicle prototype:** prototype of the final vehicle that the customers will drive. It's used to test all the functionalities and the interaction between the vehicle and the business logic of the software system.
- **Cloud computing platform**: the cloud platform where the software system will be deployed and hosted. Performance analysis designed to

verify the compliance to the performance requirements individuated are performed.

- **IOS, Windows and Android mobiles set:** set of different mobile devices developed by different vendors. Each device must have different feature in order to test the mobile application within different hardware and software environment.
- **Most widespread web browsers:** to ensure the highest level of usability and performance to the largest set of users, the most popular means to access the web are used to test these qualities.

5 Program Stubs and Test Data Required

In this section we are presenting the drivers and stubs that have to be created in order to test our system. Since we are using a bottom-up approach we will be developing different drivers that allow us to simulate parts of the system that are not yet implemented.

A populated test database: The database has to contain a small set of all the entities needed to the system to work properly.

Map Manager Driver: This driver uses the functionalities exposed by the **Area Manager** and the **Vehicle manager**. It should be developed in two different steps, in the first step will be developed and tested the part related to the vehicle manager, while in the second step the functions relative to the area manager will be tested.

Maintenace Manager Driver: The development of this driver can also be separated into two different parts, the first one regarding the **Vehicle Manager**, and the second one regarding the **Area Manager**.

Account Manager Driver: This is the driver used while testing the **Driving License Manager** and the **Payment Manager** components that are used during the *registration* of a user.

Reservation Manager Driver: This driver that is the one used to simulate the action of a user reserving a car. It is used to perform the integration test on the **Fares and Policies Manager** component, the **Vehicle Manager** component, the **Payment Manager** component and on the **Map Manager** component.

Authentication Manager Driver: This is the driver situated at the highest level of the bottom-up hierarchy, it is used to test the full functionalities of system. This driver exploits the **Abbount Manager** component, the **Reservation Manager** component and the **Maintenance Manager** component.

6 Effort Spent

The effort spent by each member of the group in terms of hours is shown in the following:

Lorenzo Casalino

- 27 December 2016 - 1h 10m
- 28 December 2016 - 1h
- 29 December 2016 - 1h 50m
- 30 December 2016 - 1h
- 02 January 2017 - 1h
- 03 January 2017 - 1h
- 05 January 2017 - 2h 10m
- 07 January 2017 - 2h 40m
- 09 January 2017 - 2h
- 11 January 2017 - 3h
- 12 January 2017 - 2h
- 14 January 2017 - 1h 40m
- 15 January 2017 - 1h 45m

Tommaso Castagna

- 29/12/16 - 1h30m
- 30/12/16 - 1h
- 02/01/17 - 1h30m
- 04/01/17 - 1h
- 05/01/17 - 1h
- 06/01/17 - 2h