Content

1. Introduction

1. Purpose and Scope

In our project, the Integration Testing Plan Document is an important part, which can guarantee all the subcomponents be integrated consistently.

The main purpose of this document is to make up the system when the subcomponents need to be revised according to the user's requirements or other unexpected reasons. The Integration Testing Plan Document can give us a clear and simple way to organize all the testing of subcomponents. We will provide these sections as follows: - To introduce the integration testing subsystems and their subcomponents involved in the integration activity. - The Elements to be Integrated and their entry criteria. - A description of the Integration Testing Strategy. - The Sequence of Components needs to be integrated, including Software Integration Sequence and Subsystem Integration Sequence. - Individual steps and test description and their input data and the expected output. - A description of performance analysis. - All the tools used in testing, and a description of the operating environment that perform all the tests.

2. Definitions and Abbreviations

1. Definitions

- Subsystem: a single, pre-defined work environment and a high-level functional unit of the system.
- Subcomponent: Implementation of subsystem functions

2. Abbreviations

- RASD: Requirements Analysis and Specification Document
- DD: Design Document
- ITPD: Integration Test Plan Document
- API: Application Programming Interface
- GUI: Graphical User Interface
- DBMS: Database Management System
- GPS: Global Positioning System
- SDK: Software Development Kit
- Req: Requirement
- App: Application

3. Reference Documents

- Requirements Analysis and Specification Document
- Design Document

- Assignments AA 2016-2017
- Integration testing example document
- IEEE standard on requirement engineering

2. Integration Strategy

1. Entry Criteria

For doing this part we should ensure that all units of project work can commence to achieve the corresponding unit test quality objectives and output the corresponding test report.

Besides, the following documents should have been completed, reviewed, approved to do unit testing phase, for instance, the Requirements Analysis and Specification Document and the Design Document. W.R.T integration testing phase, we assume that the project has already being code-complete. And there are no missing features or media elements. The product satisfies the performance and memory requirements specified by the Functional Spec. All priority bugs have been fixed and closed. Internal documentation has been updated to reflect the current state of the product.

Also, we should as possible as keeping the percentage of completion of every component with respect to its functionalities as:

- 100% for the Data Persistence component
- At least 90% for the **Car Management** subsystem
- At least 90% for the **Reservation and Billing Management** subsystem
- At least 80% for the **System Administration and Account Management** subsystems
- At least 70% for the **Client Application**

2. Elements to be Integrated

As we presented before in the design document our system is composed by many components and units. So in this section, we will list all the main components which will be integrated in this phase. And we concern the integration phase as two levels of abstraction: - High-level components integration testing - Lower-level components integration testing

For high-level integration testing, as we introduced 3-tier structure to build our system(PowerEnjoy) in Design Document so we will follow the structure to process the testing, 1. presentation tier (mobile client, web client, web server), 2. logic tier (application server), 3. persistence tier (DB server).

At lower-level integration testing, we decided to integrate those components which are highly depending on one another to offer the higher level functionalities of PowerEnjoy. In this case, these components will be involved : Ride

controller, Bill controller, Reservation controller, Economic controller. And we assume that the car controller and charge station and the interaction between car and charge station will be well integration tested by third part.

3. Integration Testing Strategy

The goal of integration testing is to uncover errors that may arise when two or more components are integrated together. As stated before, unit tests are performed and validated before conducting the integration tests. The latter provides assurance that bugs arise from the integrations and not from the components (units) themselves.

We opted for a bottom-up strategy because of many reasons. First, we are using many of already-developed or commercially available components that interact directly with our low level components. Since those components are already tested, they represent reliable building blocks to integrate from. Second, using a bottom-up approach removes the need to develop stubs that are needed in a top-down approach. In addition to that, our main datasources (database and car parameters) are already tested which removes the need for stubs. Furthermore, the low-level modules are the most critical in our application (model, localization and car communication) and almost all other components depend on them. So the bottom-up approach ensures that these components are integrated first making sure that they are bug-free earlier. In case errors are uncovered, they can be fixed early in the process.

4. Sequence of Component/Function Integration

Given the choice of a bottom-up approach, we should put on the first stage the components that do not have any dependency and then integrate on them. The components in question are: database system, notification helper and Google localization API.

In order to determine the sequence of the integrations, we will prioritize the components that many other components depends on. In other words, the most critical components will be integrated first.

3. Individual Steps and Test Description

1. External System

1. DBMS, Model

2. Google Maps Component, Localization

insertReservation(reservation)	
Input	*Effect*
A null parameter	A NullArgumentException is raised.
A reservation with an id already existent in the database	An InvalidArgumentValueException is raised.
lksqnflkdjflkds	lkfjsdflkjsdlkfjsd

3. CarComponents, Model

2. Car Management System

1. Localization, Model

2. Car Component, Model

3. Car Component, Bill Component

4. Car Component, Localization Component

3. Operations Management System

1. Reservation Component, Model

2. Bill Component, Model

3. Ride Component, Model

4. Reservation Component, Car Component

5. Reservation Component, Bill Component

6. Reservation Component, Ride Component

7. Bill Component, Ride Component

8. Ride Component, Localization Component

9. Localization Component, Economic Component

4. User Management System

1. User Component, Model

2. User Component, Car Component

3. User Component, Bill Component

4. User Component, Reservation Component

4. Tools and Test Equipment Required

5. Program Stubs and Test Data Required

6. Effort Spent

Reda Aissaoui

- 08/01/2017 2h
- 11/01/2017 3h