Integration Test Plan Document

Jia Hongyan 10470662

Lang Shuangqing 10517902

Contents

Chapter 1

Introduction

1.1 Revision History

Version of this document: 1.3
Last update: 15/01/2017

1.2 Purpose and Scope

The PowerEnjoy, a car-sharing service, is to offer a better car-sharing service for the users by using PowerEnjoy application and a better car-management system for company.

The scope of this document is to explain what to test, in which sequence, which tools are needed for testing and which stubs/drivers/oracles need to be developed.

1.3 List of Definitions and Abbreviations

Name	**Definition**
RASD	Requirements Analysis and Specification Document
DD	Design Document
ITPD	Integration Test Plan Document
DB	Database
DBMS	Data Base Management System
SOAP	Simple Object Access Protocol
APP	Mobile Application, here implement on IOS, Android system

Table 1.1 List of Definitions and Abbreviations

1.4 List of Reference Documents

This document refers to the following documents:

- *AA 2016---2017 Project goal, schedule, and rules* [1]
- *RASD document of PowerEnjoy project* [2]
- *DD document of PowerEnjoy project* [3]

- *Software Engineering 2 -Assignment 3: ITPD document* [4]
- *Software Engineering: Principles and Practice* [5]

Chapter 2

Integration Strategy

2.1 Entry Criteria

There are some rules that are supposed to be completed already before the integration testing process. And those conditions will be described as following:

First, all the classes and methods must pass through unit tests which should reasonably discover major issues in the structure of the classes or in the implementation of the algorithms. Unit tests should have a minimum coverage of 90% of the lines of code and should be run automatically at each build using JUnit. Unit testing is not in the scope of this document and will not be specified in further detail.

Second, code inspection has to be performed on all the code in order to make sure maintainability. Code inspection must be performed using automated tools as much as possible: manual testing should be reserved for the most difficult features to test.

Finally, the documentations of all classes and functions (seeing [2];[3]), written using JavaDoc, has to be complete and up-to- date in order to be used as a reference for integration testing development. In particular, the public interfaces of each class and module should be well specified.

2.2 Elements to be Integrated

The main high-level components are structured in 4 layers in Design Document [3]. A more accurate way to refer layers, would be subsystems in this ITPD document.

Database Subsystem

DBMS of our system; it is not part of the software to be developed, but has

to be integrated.

Business Subsystem
This subsystem implements all the application logic and communicates with the front-ends.

Web server Subsystem
The web tier implements the web interface and communicates with the business tier and the client browsers.

Client Subsystem
This layer contains two distinguished types of client: Mobile APP Client and Browser Client.

2.3 Integration Testing Strategy

The integration strategy choice is the **bottom-up approach**.

We assume we already have the unit tests for the smallest components, so we can proceed from the bottom.

Moreover, the higher-level subsystems in section 2.2 are well separated and loosely coupled since they correspond to different tiers; they also communicate through well-defined interfaces (SOAP API, HTTP), so they will not be hard to integrate at a later time. In this way, it will be possible also to limit the stubs needed in order to accomplish the integration, because the specific components do not use the general ones, so they do not require stubs.

The **integration process** of our software is performed on two levels.
1. integration of the different components (classes, Java Beans) inside the same subsystem;
2. integration of different subsystems.

The first step needs to be performed only for the component which contains the pieces of software that we are going to develop, namely the business tier, the mobile application in the client tier and part of the web tier (which,

as stated in the DD, uses JSF to implement the web application).

2.4 Sequence of Component/Function Integration

2.4.1 Software Integration Sequence

The integration sequence of the components is described in Table 2.1 and in Figure 2.1.

The components give the opportunity to avoid the implementation of useless stubs, because when less independent components are tested, those components which they rely on have already been integrated. The components are integrated within their classes in order to create an integrated subsystem which will be ready for integration process between subsystems.

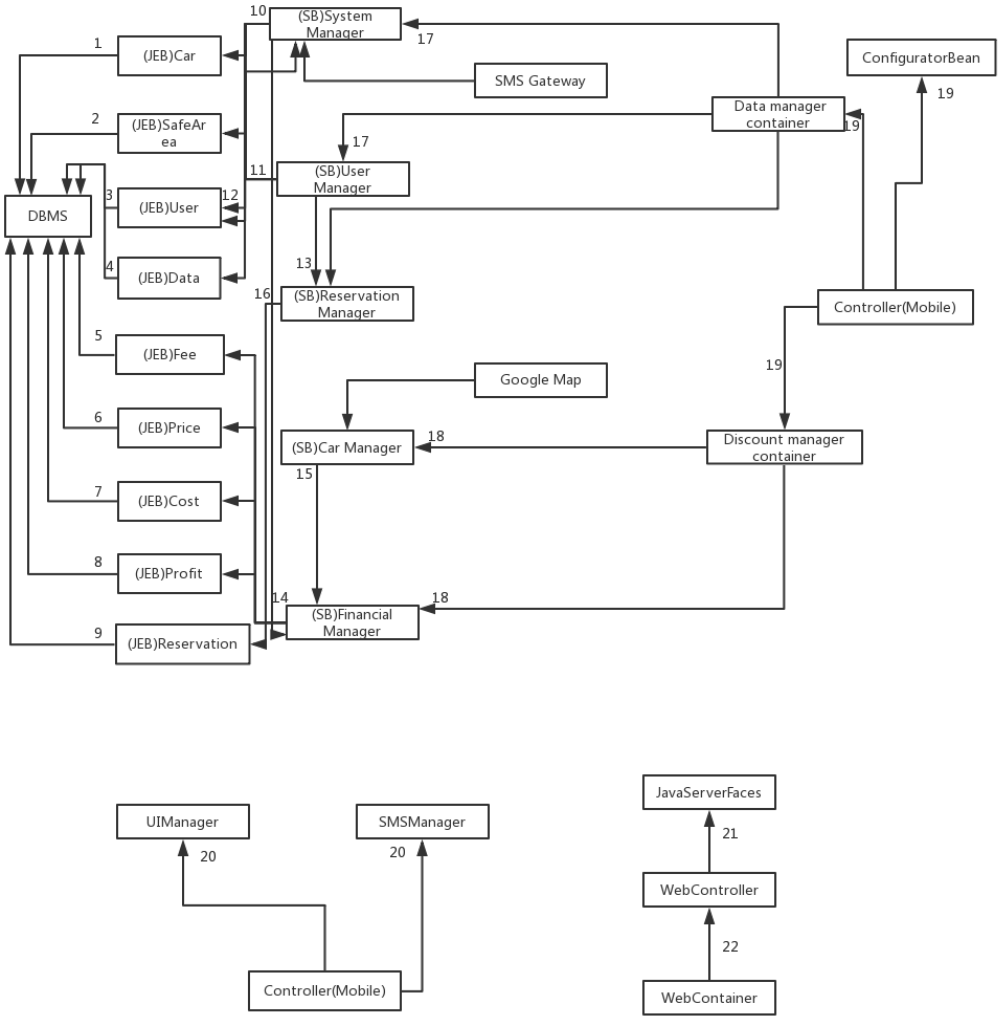

Figure 2.1 Diagram of the components integration.

N	Subsystem	Componenet	Integrates with
I1	Database,Business	(JEB)Car	DBMS
I2	Database,Business	(JEB)SafeArea	DBMS
I3	Database,Business	(JEB)User	DBMS
I4	Database,Business	(JEB)Data	DBMS
I5	Database,Business	(JEB)Fee	DBMS
I6	Database,Business	(JEB)Price	DBMS
I7	Database,Business	(JEB)Cost	DBMS
I8	Database,Business	(JEB)Profit	DBMS
I9	Database,Business	(JEB)Reservation	DBMS
I10	Business	(SB)System Manager	SMS Gateway Car SafeArea User Data
I11	Business	(SB)User Manager	Financial manager
I12	Business	(SB)User Manager	System manager User
I13	Business	(SB)User Manager	Reservation manager
I14	Business	(SB)Reservation Manager	Reservation
I15	Business	(SB)Car Manager	Google Map Financial manager
I16	Business	(SB)Financial Manager	Fee Price Cost Profit
I17	Business	(EJB) Data manager container	System Manager User Manager Reservation manager
I18	Business	(EJB)Discount manager container	Car Manager Financial Manager
I19	Business	Controller	Data manager container Discount manager container ConfiguratorBean
I20	Mobile	Controller	UIManager GPSManager SMSManager
I21	Web	WebController	JavaSeverFaces
I22	Web	WebContainer	WebController

Table 2.1 Integration of the system component.

2.4.2 Subsystem Integration Sequence

The integration sequence of the subsystems is described in Figure 2.2 and in Table 2.2.

The integration is processed from the server side towards the client applications, integrating the mobile app before the web tier. The reason to do so is that in order to have a functioning client we need to have a working business tier. The business tier, instead, can be tested without any client, by making API calls also in an automated fashion.

By integrating the mobile application before the web tier, we aim to obtain a fully operational client-server system as soon as possible.

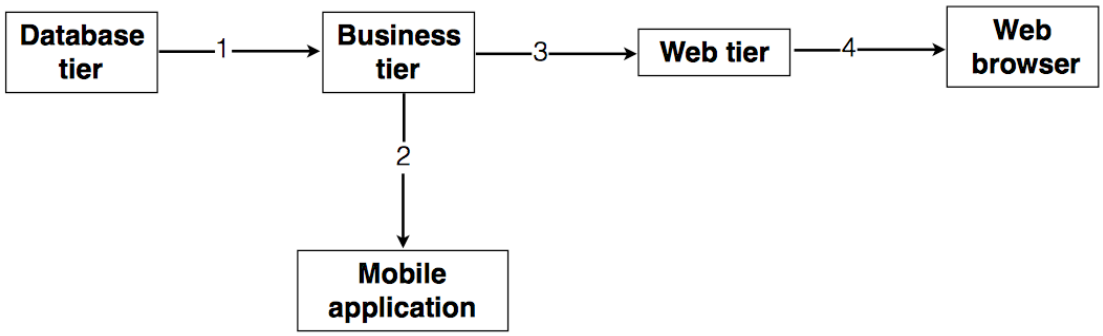

Figure 2.2 Subsystems with order of integration.

N.	Subsystem	Integrates with
SI1	Business tier	Database tier
SI2	Mobile application	Business tier
SI3	Web tier	Business tier
SI4	Client browser	Web tier

Table 2.2 Integration order of the subsystems.

Chapter 3

Individual Steps and Test Description

3.1 Integration test case SI1

Test Case Identifier	SI1T1
Test Item(s)	Business tier → Database Tier
Input Specification	Typical calls to the methods of the JPA Entities, mapped with tables in the Database tier.
Output Specification	The Database tier shall respond by doing the correct queries on the test database. It must also react in the right way both if the requests are made correctly and if they come from unauthorized sources that are trying to access the data.
Environmental Needs	Complete implementation of the Java Entity Beans, Java Persistence API, Test Database, driver that calls the Java Entity Beans.
Test Description	The response will be compared with the expected output of the queries.
Testing Method	Automated with JUnit.

3.2 Integration test case SI2

Test Case Identifier	SI2T1
Test Item(s)	Mobile application → Business Tier
Input Specification	Typical API calls (both correct and intentionally invalid ones) to the business tier (SOAP API).
Output Specification	The business tier shall respond accordingly to the API specification. Also, it must react correctly if the requests are malformed or maliciously crafted.
Environmental Needs	Complete implementation of the Business tier; SOAP API client (driver) that mocks the actual mobile client.
Test Description	The clients should make typical API calls to the business tier; the responses are then evaluated and checked against the expected output. The driver of this test is a standard SOAP API client that runs on Java.
Testing Method	Automated with JUnit.

Test Case Identifier	SI2T2
Test Item(s)	Mobile application → Business Tier
Input Specification	Multiple concurrent (typical, correct) requests to the SOAP API of the business tier.
Output Specification	The business tier must answer the requests in a reasonable time with the applied load.
Environmental Needs	GlassFish Server, fully developed business tier, Apache JMeter.
Test Description	This test case assesses whether the business tier fulfills the performance requirement stated in the RASD (Performance requirements).In particular, the system has to support at least 1000 connected passengers at once, 95% of requests shall be processed in less than 5 s and 100% of requests shall be processed in less than 10 s.
Testing Method	Automated with JUnit.

3.3 Integration test case SI3

Test Case Identifier	SI3T1
Test Item(s)	Web tier → Business tier
Input Specification	Requests for services offered by the business tier, also invalid ones.
Output Specification	The web tier must call the proper SOAP APIs or report an error.
Environmental Needs	GlassFish Server, Web tier.
Test Description	This test has to ensure the right translation from HTTPS requests into SOAP APIs calls, reporting errors when needed.
Testing Method	Automated with JUnit.

Test Case Identifier	SI3T2
Test Item(s)	Web tier → Business tier
Input Specification	Multiple concurrent API calls to the Business tier.
Output Specification	Web requests should be served without problems when a reasonable load is applied on the Business tier.
Environmental Needs	GlassFish Server, Web tier, Apache JMeter.

Test Description	This test case assesses whether the Business tier fulfills the performance requirement stated in the RASD (Performance requirements).In particular, the system has to support at least 1000 connected passengers at once, 95% of requests shall be processed in less than 5 s and 100% of requests shall be processed in less than 10 s.
Testing Method	Automated with Apache JMeter.

3.4 Integration test case SI4

Test Case Identifier	SI4T1
Test Item(s)	Client browser → Web tier
Input Specification	Typical and well-formed HTTPS requests from client browser; incomplete, malformed and maliciously crafted requests.
Output Specification	The web tier shall display the requested pages if the requests are valid; if the requests are invalid it shall display a generic error message.
Environmental Needs	GlassFish Server, fully developed web tier, HTTP client (driver).
Test Description	This test should emulate HTTP requests from typical users of the service and also incorrect requests.
Testing Method	Automated with JUnit.

Test Case Identifier	SI4T2
Test Item(s)	Client browser → Web tier
Input Specification	Multiple concurrent requests to the web server.
Output Specification	Web pages should be served without problems when a reasonable load is applied on the web server.
Environmental Needs	GlassFish Server, fully developed web tier, Apache JMeter.
Test Description	This test case assesses whether the Web tier fulfills the performance requirement stated in the RASD (Performance requirements).In particular, the system has to support at least 1000 connected passengers at once, 95% of requests shall be processed in less than 5 s and 100% of requests shall be processed in less than 10 s.

Testing Method	Automated with Apache JMeter.

3.5 Integration test cases I01, I02, I03, I04, I05,I06,I07,I08,I09: components that integrate with the DBMS

The test cases (I01-I09) refer to the integration between the Java Entity Beans and the underlying Database tier. We group them together since they are similar.

Test Case Identifier	I01T1
Test Item(s)	Car → DBMS
Input Specification	Typical queries on table Car.

Test Case Identifier	I02T1
Test Item(s)	SafeArea → DBMS
Input Specification	Typical queries on table SafeArea.

Test Case Identifier	I03T1
Test Item(s)	User → DBMS
Input Specification	Typical queries on table User.

Test Case Identifier	I04T1
Test Item(s)	Data → DBMS
Input Specification	Typical queries on table Data.

Test Case Identifier	I05T1
Test Item(s)	Fee → DBMS
Input Specification	Typical queries on table Fee.

Test Case Identifier	I06T1
Test Item(s)	Price → DBMS
Input Specification	Typical queries on table Price.

Test Case Identifier	I07T1
Test Item(s)	Cost → DBMS
Input Specification	Typical queries on table Cost.

Test Case Identifier	I08T1
Test Item(s)	Profit → DBMS
Input Specification	Typical queries on table Profit .

Test Case Identifier	I09T1
Test Item(s)	Reservation → DBMS
Input Specification	Typical queries on table Reservation.

Output Specification	The queries return the correct results.
Environmental Needs	GlassFish server, Test Database, driver for the Java Entity Beans.
Test Description	The purpose of these tests is to check that the correct methods of the Entity Beans are called, and that they execute the correct queries to the DBMS.
Testing Method	Automated with JUnit.

3.6 Integration test case I10

Test Case Identifier	I10T1
Test Item(s)	System Manager → SMS Gateway, Car, SafeArea, User, Data
Input Specification	Methods call from System Manager to the SMS Gateway in order to guarantee a right SMS authentication process.
Output Specification	The SMS authentication process must be correctly handled.
Environmental Needs	GlassFish Server
Test Description	Assure that a user can properly verity password from SMS in order to start accessing the system. In order to do that, a mock SMS Gateway which simulates the user behavior is needed.
Testing Method	Automated with JUnit and Mockito.

3.7 Integration test case I11

Test Case Identifier	I11T1
Test Item(s)	User Manager → Financial Manager
Input Specification	Methods call from User Manager to Financial Manager and System Manager, to manage and update the financial information and fee about the reservation.
Output Specification	The estimated fee of the reservation has to be computed correctly.
Environmental Needs	GlassFish Server
Test Description	The test aims to verify that the User requests are correctly satisfied by Financial Manager
Testing Method	Automated with JUnit.

3.8 Integration test case I12

Test Case Identifier	I12T1
Test Item(s)	User Manager → System Manager, User
Input Specification	Methods call from User Manager to System Manager, to manage and update the user's profile and information.
Output Specification	The user's profile and states must be correctly updated without mistakes and duplication.
Environmental Needs	GlassFish Server
Test Description	The test aims to verify that the user's action are correctly satisfied by system Manager.
Testing Method	Automated with JUnit.

3.9 Integration test case I13

Test Case Identifier	I13T1
Test Item(s)	User Manager → Reservation Manager
Input Specification	Methods call from User Manager to Reservation Manager to manage and update the information and fee about the reservation.
Output Specification	The information of the reservation has to be updated and handled correctly.
Environmental Needs	GlassFish Server
Test Description	Assume that when user makes the reservation,

	Reservation Manager will handle it properly.
Testing Method	Automated with JUnit.

3.10 Integration test case I14

Test Case Identifier	I14T1
Test Item(s)	Reservation Manager → Reservation
Input Specification	Call from Reservation Manager to to Reservation.
Output Specification	The reservation information will be stored and updated correctly without duplication.
Environmental Needs	GlassFish Server
Test Description	Verify reservation information is handled properly.
Testing Method	Automated with JUnit.

3.11 Integration test case I15

Test Case Identifier	I15T1
Test Item(s)	Car Manager → Google Map, Financial Manager
Input Specification	Methods call from Car Manager to Financial Manager, to tell Financial manager about the the physical statues (battery/number of passenger/parking area) of the car.
Output Specification	Financial Manager computed the basic fee and penalty/ discount correctly according to the situation of the car.
Environmental Needs	GlassFish Server
Test Description	The test aims that information about the basic fee and penalty/discount is successful known by Financial Manager. In order to do that, a mock path working with Google Map which simulates the user behavior is needed.
Testing Method	Automated with JUnit and Mockito.

3.12 Integration test case I16

Test Case Identifier	I16T1
Test Item(s)	Financial Manager → Fee, Price, Cost, Profit
Input Specification	Methods call from Financial Manager to Fee, Price, Cost, Profit.
Output Specification	Fee, Price, Cost, Profit have to be computed correctly

	and updated
.Environmental Needs	GlassFish Server
Test Description	The test aims that the overall fee calculation process of each rides is computed correctly by Financial Manager.
Testing Method	Automated with JUnit.

3.13 Integration test case I17

Test Case Identifier	I17T1
Test Item(s)	Data manager container → System Manager, User Manager, Reservation manager
Input Specification	Requests for the System Manager, User Manager, Reservation manager SessionBeans.
Output Specification	The SessionBeans must be correctly assigned and the concurrency between the request must be properly managed.
Environmental Needs	GlassFish Server
Test Description	Multiple requests for the System Manager, User Manager, Reservation manager SessionBeans have to be simultaneously carried out, in order to ensure that the users have no concurrency trouble.
Testing Method	Automated with JUnit and Arquillian.

3.14 Integration test case I18

Test Case Identifier	I18T1
Test Item(s)	Discount manager container → Car Manager, Financial Manager
Input Specification	Requests for the Car Manager and Financial SessionBeans.
Output Specification	The SessionBeans must be correctly assigned and the concurrency between the request must be properly managed.
Environmental Needs	GlassFish Server
Test Description	Multiple requests for the Car Manager, Financial Manager have to be simultaneously carried out, in order to en- sure that the users have no concurrency trouble.

Testing Method	Automated with JUnit and Arquillian.

3.15 Integration test case I19

Test Case Identifier	I19T1
Test Item(s)	Controller → Data manager container, Discount manager container
Input Specification	Requests from Controller to the containers for the functionalities offered by SessionBeans within containers.
Output Specification	The controller has to be able to provide the right functionality carrying out the proper request to the containers.
Environmental Needs	GlassFish Server
Test Description	Ensure that the controller is able to provide the functionalities of the system offered by the containers.
Testing Method	Automated with JUnit and Arquillian.

3.16 Integration test case I20

Test Case Identifier	I20T1
Test Item(s)	Controller → UIManager, GPSManager, SMSManager
Input Specification	Requests from Controller to UIManager, GPSManager, SMSManager.
Output Specification	The location data shall be returned from GPSMan- ager in a suitable format, or an exception shall be raised if the location data is not available.]SMS Manager should provide the required resource without errors.
Environmental Needs	Xcode, iOS Simulator, Android Emulator.
Test Description	Ensure that the managers provided correct information and work well.
Testing Method	Automated (Android and iOS testing suites).

3.17 Integration test case I21

Test Case Identifier	I21T1
Test Item(s)	WebController → JavaServerFaces
Input Specification	WebController is given the typical output to be dis-

	played on the web page.
Output Specification	JavaServerFaces shall display the required output in a correct way.
Environmental Needs	GlassFish Server, Stub of the Business Tier to provide the output data.
Test Description	The purpose of this test case is to check if JSF can communicate correctly with the WebController bean.
Testing Method	Automated with JUnit.

3.18 Integration test case I22

Test Case Identifier	I22T1
Test Item(s)	WebContainer → WebController
Input Specification	Run the web application.
Output Specification	WebContainer injects the WebController bean, using JSF.
Environmental Needs	GlassFish Server.
Test Description	This test verifies if the correct component is injected into JSF.
Testing Method	Automated with JUnit.

Chapter 4
Tools and Test Equipment Required

JUnit
JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.
We use it for unit test, but together with Mockito and Arquillian, it is able to manages our other needs for testing.

Mockito
Mockito is a mocking framework, JAVA-based library that is used for effective unit testing of JAVA applications. Mockito is used to mock interfaces so that a dummy functionality can be added to a mock interface that can be used in unit testing.
We use it in section 3.6, section 3.11.

Arquillian
Arquillian is a test framework which can also manage the test of the containers and their integration with JavaBeans (dependency injection).
We use in in section 3.13, section 3.14, and section 3.15.

Apache JMeter
Apache JMeter is a tool used to test performance both on static and dynamic resources, Web dynamic applications. It can be used to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyze overall performance under different load types.
We use it for a heavy load on the SOAP in business tire and web tire, in section 3.2, section 3.3 and section 3.4.

Chapter 5

Program Stubs and Test Data Required

With bottom-up approach used in our testing, we avoid the implementation of most useless stubs. But in order to perform integration testing without having developed the entire system, we still need to use some test data, stubs and drivers.

Test DB: the testing environment must include a configured DBMS configured. The test data contained in this database includes a reduced set of instances of all needed entities.

Light API client: a simple API client which interacts with the business tier by simple HTTP requests is needed, so that SOAP API of the business tier without the actual client application can be implied. The driver needs to be scriptable for automated testing.

Drivers for the Java Entity Beans: used to test the Java Entity Beans when the Business Tier is not fully developed. They call the relevant methods of the EJBs to test the correctness of the queries.

Stub of the Business Tier: used to provide a minimum set of data to test the web tier when the business tier is not fully developed.

Mock SMS sender and receiver: used to automate the testing of the SMS confirmation process.

Mock Google Map Path Inquire: used to automate the testing of the financial calculation process.

Chapter 6

Effort spent (hour of work)

Jia HongYan	10 hours
Lang Shuangqing	10 hours

Chapter 7

Used tools

Google drive: a tool for version controller and group cooperation.
Word: a tool to write down the document.
Github: a tool for submit this document.
Draw.io: a tool for diagrams.

Chapter 8

Bibliography

[1] AA 2016---2017 Software Engineering 2- Project goal, schedule, and rules
[2] *Li Xiaoxu, Lang Shuangqing, Jia Hongyan,* Software Engineering 2: PowerEnjoy- Requirements Analysis and Specification Document
[3] *Jia Hongyan, Lang Shuangqing,* Software Engineering 2: PowerEnjoy- Design Document
[4] *Chitti Eleonora, De Nicolao Pietro, Delbono Alex,* Software Engineering 2: myTaxiService- Integration Test Plan Document
[5] *Van Vliet, H., Van Vliet, H., & Van Vliet, J. C. (1993)*. Software engineering: principles and practice (Vol. 3).

Chapter 9

Change log

V1.1
-Modified the Table 2.1 testing method part.
-Re-drew Figure 2.1.

V1.2
-Modified Diagram of the components integration.
-Modified Integration test case I19.
-Added reference documents in list of section 1.4.

V1.3
-Modified Integration test case I19.
-Fixed the overall format.