POLITECNICO
DI MILANO

Politecnico di Milano
A.A. 2015-2016
Software Engineering 2
"myTaxiService"

Integration Test Plan Document (ITPD)

version 1.0

Annalisa Rotondaro (mat. 854268)

Lorenzo Federico Porro (mat. 859093)

Release date: 21 jenuary 2016

Contents

1 Introduction

Software systems are built with components that must interoperate. Integration testing is a search of component faults that cause intercomponent failures. An integration strategy must answer three questions:

- Which components are the focus of the integration test?
- In what sequence will component interfaces be exercised?
- Which test design technique should be used to exercise each interface?

Most interoperability faults are not revealed by testing a component in isolation (i.e. unit testing) , so integration testing is necessary. The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e., assemblages (or groups of units), are exercised through their interfaces using black box testing, success and error cases being simulated via appropriate parameter and data inputs. Test cases are constructed to test whether all the components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e., unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.

1.1 Revision History

Version 1.0

1.2 Purpose and Scope

The purpose of the Integration Test Plan Document (ITPD) is describing how is the plan to accomplish the integration test. This document is supposed to be written before the integration test really happens. This document needs to explain to the development team what to test, in which sequence, which tools are needed for testing (if any), which stubs/ drivers/ oracles need to be developed. ITPD takes the architectural description of the software system as a starting point.

The scope of the “TaxiService” project is to create both a web “myTaxiService.com” and a mobile “myTaxiServiceApp” applications for denizens and taxi drivers, as well as an efficient management system for taxi queues.

The application is aimed both to denizens of large cities and taxi drivers. The former will have a major quality-of-life improvement in the way they benefit of the city’s taxi service, by having an intuitive and immediate interface to make taxi calls and to quicken whole process of taxi calls. The latter will have a fair queue system and management and also a fast and efficient way to start their everyday job through the mobile app.

Software Integration Testing is performed according to the Software Development Life Cycle (SDLC) after module and functional tests. The cross-dependencies for software integration testing are: schedule for integration testing, strategy and selection of the tools used for integration, define the cyclomatical complexity of the software and software architecture, reusability of modules and life-cycle / versioning management.

1.3 List of Definitions and Abbreviations

1.3.1 Definitions

- **Stub**: is a partial implementation of a component;
- **Driver**: is a class, main program, or external software system that applies test cases to the component under test;
- **Web app:** The site accesible by any browser;
- **Mobile app:** The application from which a user or a driver is able to access myTaxiService services;
- **Junit:** The most popular testing framework for Java programming language;
- **DbUnit:** Junit extension that can be used to initialize the database into a known state before each integration test;

1.3.2 Abbreviations

- **DB:** DataBase;

1.3.3 Acronyms

- **JVM:** Java Virtual Machine;
- **DBMS**: Database Management System;
- **RASD**: Requirements Analysis and Specification Document;
- **DD**: Design Document;
- **ITPD**: Integration Test Plan Document;

1.4 List of Reference Documents

- RASD;
- DD;
- Assignment 4 - integration test plan;
- Testing Object-Oriented Systems Models, Patterns, and Tools Robert V. Binder

2 Integration Strategy

2.1 Entry Criteria

In order to run the integration test process described in this document the following preconditions must be matched:

1. The JVM used to run the tests is stable;
2. The testing suite used to write the tests is stable;
3. The following components (see DD for additional informations) have been already individually tested in order to check the behaviour of the functionalities described in the RASD and DD and, as a result, these components are guaranteed to work properly:

- System logic
- Call
- Reservation
- Notifications system
- Web app
- Mobile app
- DBMS
- Database
- Queue manager
- Queue

2.2 Elements to be Integrated

The integration tests described in this document aim to check the interactions between the following set of components (see DD for additional informations):

1. web app - call;
2. webapp - reservation;
3. webapp - login;
4. webapp - registration;
5. mobile - call (users), answer (drivers);
6. mobile - reservation (users), answer (drivers);

7. mobile - login;
8. mobile - registration;
9. mobile - queue manager (drivers);
10. notification - webapp;
11. notification - mobile app;
12. system (login, registration) - DBMS.
13. call - notification
14. reservation - notification

2.3 Integration Testing Strategy

Some different types of integration testing are big bang, top-down, and bottom-up, mixed (sandwich) and risky - hardest. Other Integration Patterns are: Collaboration Integration, Backbone Integration, Layer Integration, Client/Server Integration, Distributed Services Integration and High-frequency Integration.

Incremental integration is the most effective technique: add components a few at a time and then test their interoperability. Trying to integrate most of all the components at the same time is usually problematic, for example debugging is difficult because the bug may be in any interface. In contrast, incremental testing has several advantages:

- Interfaces are systematically exercised and shown to be stable before unproven interfaces are exercised.
- Observed failures are most likely to come from the most recently added component, so debugging is more efficient.

So we don't choose Big Bang Integration but we choose Functional groupings Integration because, given the project development method(see section 2.4 of DD), this strategy is the most appropriate and easy to implement.

2.4 Sequence of Component/Function Integration

The following set of functionalities has been identify for the purpose of testing. The procedures described in the tables below cover all the components that contribute to implement such functionalities:

- login;
- registration;
- call;
- reservation;

- taxi answers;
- queue operations (add in queue, remove from queue);
- notification;
- data management.

The following tables describes the procedures that must be executed in order to test all the functionalities and the components described above. The procedures are identified by progressive numbers, the first one identify the specific procedure, while the second number (if present) identify the same procedure on different platforms (Web or Mobile, e.g.: P1.1 and P1.2 tests the same functionalities on different components, the first on the web app and the second on the mobile app).

The specific test referred in the procedures tables are specified in chapter 3.

ID	P1.1
Tested functionalities	call, notification, taxi answer
Component integration sequence	user → web app → call → notification → web app/mobile app → user/taxi driver
Test execution sequence	call1 → not 3 → not1/not2 → ans
Platform	Web
Additional informations	Notification has to be integrated with both the web and the mobile apps, after the notification step the web app and the mobile app have to be integrated respectively with the user and the taxi driver.

ID	P1.2
Tested functionalities	call, notification, taxi answer
Component integration sequence	user → mobile app → call → notification → mobile app → user/taxi driver
Test execution sequence	call2 → not3 → not2 → ans
Platform	Mobile
Additional informations	Notification has to be integrated with the mobile app for both user and taxi driver

ID	P2.1
Tested functionalities	reservation, notification, taxi answer
Component integration sequence	user → web app → reservation → notification → web app/mobile app → user/driver
Test execution sequence	res1 → not4 → not1/not2 → ans
Platform	Web
Additional informations	Notification has to be integrated with both the web and the mobile apps, after the notification step the web app and the mobile app have to be integrated respectively with the user and the taxi driver.

ID	P2.2
Tested functionalities	reservation, notification, taxi answer
Component integration sequence	user → mobile app → reservation → notification → mobile app → user/driver
Test execution sequence	res2 → not4 → not2 → ans
Platform	Mobile
Additional informations	Notification has to be integrated with the mobile app for both user and taxi driver

ID	P3.1
Tested functionalities	login, data management
Component integration sequence	user → web app → login → DBMS → DB
Test execution sequence	log1 → data1
Platform	Web
Additional informations	-

ID	P3.2
Tested functionalities	login, data management
Component integration sequence	user & taxi driver → mobile app → login → DBMS → DB
Test execution sequence	log2 → data1
Platform	Mobile
Additional informations	This procedure has to be executed twice: once for the user and once for the taxi driver.

ID	P4.1
Tested functionalities	registration, data management
Component integration sequence	user → web app → registration → DBMS → DB
Test execution sequence	log1 → data2
Platform	Web
Additional informations	-

ID	P4.2
Tested functionalities	registration, data management
Component integration sequence	user & taxi driver → mobile app → registration → DBMS → DB
Test execution sequence	log2 → data2
Platform	Mobile
Additional informations	This procedure has to be executed twice: once for the user and once for the taxi driver.

ID	P5
Tested functionalities	add operation, remove operation
Component integration sequence	driver → mobile app → queue manager → queue
Test execution sequence	que
Platform	Mobile
Additional informations	This procedure has to be executed twice: once for the add operation and once for the removal operation.

3 Individual Steps and Test Description

3.1 Call tests

ID	call1
tested funcionalities	call
procedure	P1.1
tested components	(1) web app → call
input	Call data sample
output	The correct method is called (data is handled properly)
preconditions	User driver, login is done

ID	call2
tested funcionalities	call
procedure	P1.2
tested components	(5) mobile app → call
input	Call data sample
output	The correct method is called (data is handled properly)
preconditions	User driver, login is done

3.2 Login tests

ID	log1
tested funcionalities	login
procedure	P3.1
tested components	(3) web app → login
input	Login data sample
output	The correct method is called (data is handled properly)
preconditions	User driver, DB stub, registration is done

ID	log2
tested funcionalities	login
procedure	P3.2
tested components	(7) mobile app → login
input	Login data sample
output	The correct method is called (data is handled properly)
preconditions	User driver, Taxi driver driver, DB stub, registration is done

3.3 Reservation tests

ID	res1
tested funcionalities	reservation
procedure	P2.1
tested components	(2) web app → reservation
input	Reservation data sample
output	The correct method is called (data is handled properly)
preconditions	User driver, login is done

ID	res2
tested funcionalities	reservation
procedure	P2.2
tested components	(6) mobile app → reservation
input	Reservation data sample
output	The correct method is called (data is handled properly)
preconditions	User driver, login is done

3.4 Registration tests

ID	reg1
tested functionalities	registration
procedure	P4.1
tested components	(4) web app → registration
input	Registration data sample
output	The correct method is called (data is handled properly)
preconditions	User driver, DB stub

ID	reg2
tested functionalities	registration
procedure	P4.2
tested components	(8) mobile app → registration
input	Registration data sample
output	The correct method is called (data is handled properly)
preconditions	User driver, Taxi driver driver, DB stub

3.5 Answer operations test

ID	ans
tested functionalities	taxi answers
procedure	P1.1, P1.2, P2.1, P2.2
tested components	(5, 6) mobile app → answer
input	Call or Reservation data sample
output	The correct method is called (data is handled properly)
preconditions	Taxi driver driver, login is done, a call or reservation is issued

3.6 Queue operations test

ID	que
tested functionalities	add to queue, remove from queue
procedure	P5
tested components	(9) mobile app → queue manager
input	Taxi driver' s selection sample
output	The correct method is called (data is handled properly)
preconditions	Taxi driver driver, Queue stub, login is done

3.7 Notification tests

ID	not1
tested functionalities	notification
procedure	P1.1, P2.1
tested components	(10) web app → notification
input	Notification data sample
output	The correct method is called (data is handled properly)
preconditions	User driver, login is done, a call or reservation is issued

ID	not2
tested functionalities	notification
procedure	P1.2, P2.2
tested components	(11) mobile app → notification
input	Notification data sample
output	The correct method is called (data is handled properly)
preconditions	User driver, Taxi driver driver, login is done, a call or reservation is issued

ID	not3
tested functionalities	notification
procedure	P1.1, P1.2
tested components	(13) call → notification
input	Call data sample
output	The correct method is called (data is handled properly)
preconditions	User driver, login is done, a call or reservation is issued

ID	not4
tested functionalities	notification
procedure	P2.1, P2.2
tested components	(14) reservation → notification
input	Reservation data sample
output	The correct method is called (data is handled properly)
preconditions	User driver, login is done, a call or reservation is issued

3.8 Data management tests

ID	data1
tested functionalities	data management
procedure	P3.1, P3.2, P4.1, P4.2
tested components	(12) login → DBMS
input	Login data sample
output	The correct method is called (data is handled properly)
preconditions	User driver, DB stub

ID	data2
tested functionalities	data management
procedure	P3.1, P3.2, P4.1, P4.2
tested components	(12) registration → DBMS
input	Registration data sample
output	The correct method is called (data is handled properly)
preconditions	User driver, DB stub

4 Tools and Test Equipment Required

- A machine with adequate hardware to run the application program and the software needed for testing;
- JVM;
- An IDE that supports Java and JEEE (e.g: netbeans, eclipse);
- Adequate test suite (JUnit);

- Mockito: integration and stub testing tool;
- Arquillian: integration testing tool;
- DbUnit;

5 Program Stubs and Test Data Required

The following stubs and drivers are required in order to run the intergration testing described in this document (see RASD and DD for additional informations on the type of data for stub and drivers):

- **User driver:** that simulates users' input;
- **Driver driver:** that simulates driver' input;
- **DB stub:** that contains sample data of logins,registrations, calls and reservations;
- **Queue stub:** that contains a sample taxi queue.

6 Supporting information

6.1 Tools used

- **LyX**: used to write this document.
- **DropBox**: used to share materials between members of the group.
- **GitHub**: used to share materials between members of the group and to deliver this document.
- **Paint**: used to build the pictures.

6.2 Workload

The time spent to redact this document is approx:
Lorenzo Federico Porro : 10 hours
Annalisa Rotondaro : 10 hours