Politecnico di Milano
Academic Year 2015/2016
Software Engineering 2: “myTaxiService”
Test Plan

Massimo Schiavo, Marco Edoardo Cittar

21st January 2016

Contents

1 Introduction

1.1 Purpose and scope

This document describes how to test the integration of the components of the system. Its purpose is to test if the components are correctly integrated between each other, as described in the Design Document.

The scope of the document is to check that the application functionalities are correctly implemented, starting from the login process, to the ride requests to be made by the users and the correct handling of all this through the application logic.

1.2 List of definitions and abbreviations

- Capitalized names, if not already explicitly stated, refer to the functionalities and components of the system.
- RASD: Requirements Analysis and Specification Document
- DD: Design Document

1.3 List of reference documents

- RASD of the application (RASD_myTaxiService_Schiavo_Cittar.pdf)
- DD of the application (DD_myTaxiService_Schiavo_Cittar.pdf)
- The assignment document (Assignment 4 - integration test plan.pdf)

2 Integration strategy

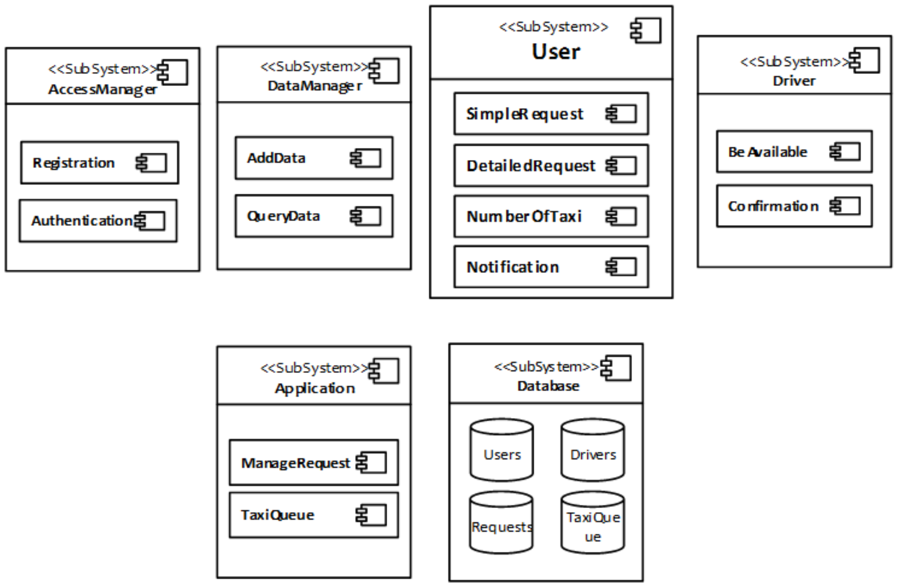
2.1 Entry Criteria

Before starting the integration testing of the subsystems, we recommend to perform a unit test on the "Application" sub-system in order to check the algorithm correctness, the functional specifications and the structural coverage. This is the major class that contains the logic of myTaxiService. In this class will be implemented, according to the document design already provided, the algorithm of management of the queues of taxis and the algorithm responsible for the forwarding of the requests arrived from the user to the taxi drivers and that's why it's important to perform a unit test on this class before proceeding to the integration test.

The documentation about what is needed to perform this kind of test is in the fifth paragraph of this document.

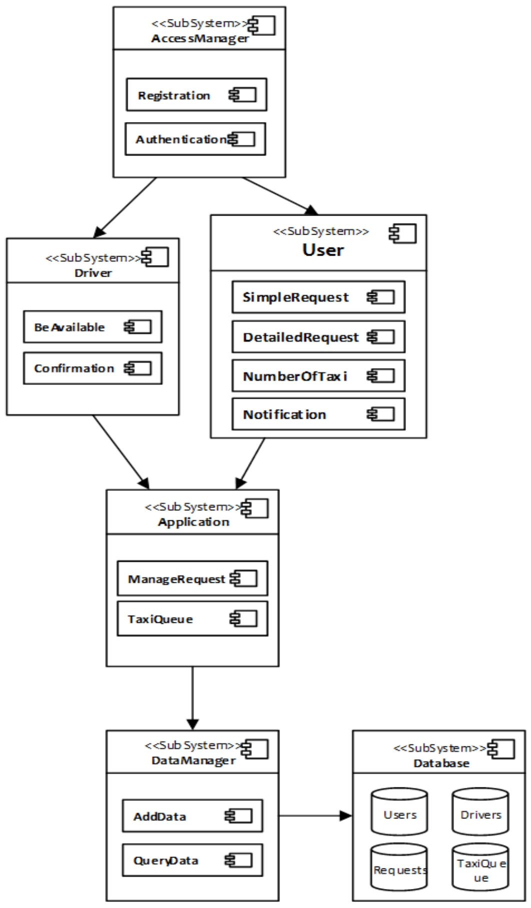
2.2 Elements to be integrated

The items to be tested consists of the integration of all the subsystem developed that form myTaxiService.



2.3 Integration testing strategy

For testing, we choose the top-down approach. In this approach testing is conducted from the high level interfaces to the core of our application. It's advantageous because the major flaws occur toward the top of the program. It's also a way to simplify the readability of the results of the test case because once the I/O functions are added, representation of test cases is easier. Due to these considerations, in the pictures below is shown through the arrows the order of integrations between the subsystems.



Regarding the User subsystem, the first software components to be integrated are SimpleRequest and DetailedRequest.

Regarding the Application subsystem, the first software component to be integrated is TaxiQueue.

Regarding all the others subsystems, each component has to be integrated with the entire subsystem.

3 Individual steps and test description

3.1 Integration test I1

Integration Test Identifier	I1T1
Test item(s)	This test procedure verifies the correct integration between AccessManager and the User and Driver components.
Purpose	The test must check if the AccessManager can correctly handle users' input.
Preconditions	A Database and a DataManager stub must be created to simulate the login process.
Procedure Steps	Some input tries will be performed, including the login of a User and a Driver. If the components are correctly integrated, the AccessManager should be able to differentiate between the two types of login and show the correct personal page.

3.2 Integration test I2

Integration Test Identifier	I2T1
Test item(s)	This test procedure verifies the correct integration between the User and the Application components.
Purpose	The test must check if the User component can correctly dispatch users' requests.
Preconditions	I1T1 must have been already correctly executed.
Procedure Steps	The two main functionalities of this component are tested, that are SimpleRequest and DetailedRequest. If the components are correctly integrated the Application should correctly receive the requests from the User and will then proceed to manage them.

Integration Test Identifier	I2T2
Test item(s)	This test procedure verifies the correct integration between the Driver and the Application components.
Purpose	The test must check if the Driver component can correctly receive users' requests and reply to them.
Preconditions	I2T1 must have been already correctly executed. The stubs described in the 5.3 paragraph must have been created.
Procedure Steps	At first it must check that the Driver, by setting himself as available, is correctly inserted in a queue. Then a simulated request from the stub is sent to the Driver which then should act properly, either if he answers the request or not.

3.3 Integration test I3

Integration Test Identifier	I3T1
Test item(s)	This test procedure verifies the correct integration between the Application and the DataManager.
Purpose	The test must check if the Application component can correctly make query and modification requests of the Database through the DataManager.
Preconditions	A Database stub is needed.
Procedure Steps	The Application should try dispatching the query and add calls to the DataManager. Some examples of these calls can be the query for the first available driver in a queue and the modification of the order of a taxi queue after the Driver's answer to a request.

3.4 Integration test I4

Integration Test Identifier	I4T1
Test item(s)	This test procedure verifies the correct integration between the DataManager and the Database components.
Purpose	The test must check if the DataManager can correctly query and modify the Database.
Preconditions	I3T1 must have been already correctly executed.
Procedure Steps	The DataManager should be able to receive query and modification requests from the logic (Application) of the system, correctly execute them and eventually return the desired data.

4 Tools and test equipment required

In this paragraph, we suggest some tools to use to perform different kind of tests. It is only a suggestion that can be followed or not.

A manual testing that helps the activity of verification and validation is the V&V activities and software artefacts (the V model). This is an easy understandable way to fix what the developed program really does compared to what it should do.

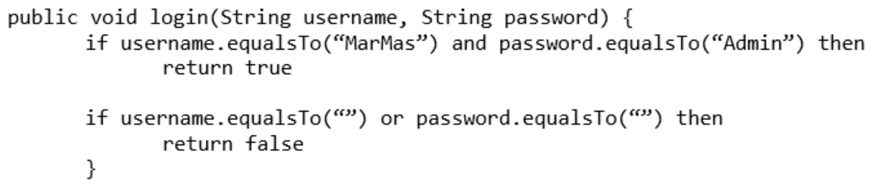
In myTaxiService application, the objects are in continuous interaction and it is useful to know what are the methods called “n” times or never called. It is also useful knowing if a controller is calling the right service. We suggest Mockito software, available at this link http://mockito.github.io , in order to perform this kind of test.

Another important tool to verify other aspects is JMeter. myTaxiService could have a very high contemporary audience especially during peak hours. With JMeter we can simulate logging into the web site and application, clicking buttons, sending requests, performs generally some action that are going to make heavy our server. It is very important to have this aspect about the performance tested in order to be safe about an application’s crash due to bigger traffic on the server.

5 Program stubs and test data required

In this paragraph we are going to highlight what it's necessary about program stubs, drivers or anything else, in order to perform the integration test following the guideline we have proposed. Since we have chosen a top-down approach, stubs are required at every single step during the integration test. A 'Stub' is a piece of software that works similar to a unit which is referenced by the Unit being tested, but it is much simpler than the actual unit. A Stub works as a 'Stand-in' for the subordinate unit and provides the minimum required behaviour for that unit. A Stub is a dummy procedure, module or unit that stands in for an unfinished portion of a system.

1. At the first step of our integration, a stub that simulates the database response correlated to a logging or registering action is required. In example:

   ```
   public void login(String username, String password) {
          if username.equalsTo("MarMas") and password.equalsTo("Admin") then
                 return true

          if username.equalsTo("") or password.equalsTo("") then
                 return false
          }
   ```

   - This is a very low level of example just only to show and make understandable what kinds of stubs we aspect the tester will perform. It's needed only to simulate a function not already implemented.

2. A stub that simulate operations on a database is also needed.

3. At the second step of the integration test different stubs are required:

   - A stub for drivers which by clicking on be_available gains the right to enter in a queue list.
   - A stub that simulate a pending request.
   - A stub that simulate the timing out of the response time with the relative consequences.

4. Finally, by integrating the remaining components there are no stubs to do, that's because now all the entire application is integrated and no more functionalities have to be implemented.

More detail on what to expect for a single test case can be found in the third paragraph of this document.

The documentation that must be provided before starting with the integration test are:

- The Design Document (DD).
- All this document, in particular the specific paragraphs concerning the strategy.

The documentations and materials that must be provided before starting a specific integration test are:

- The paragraph concerning the order of the component's integration.
- All the stubs needed for testing the components.
- The unit test report concerning the Application component.

At the end of the integration test we advise for a system test checking not only the design, but also the behaviour and even the believed expectations of the customer.

6 References

6.1 Tools and software used

- LyX: to redact and format this document.
- Microsoft Visio: to make the Component integration diagrams.

6.2 Hours of work

This is the time spent to redact this document:

- Massimo Schiavo: 10 hours.
- Marco Edoardo Cittar: 10 hours.