Politecnico di Milano

Software Engineering 2 Project
a.y. 2015-16

MyTaxiService
Integration Test Plan Document
Version 1.0

Casati Fabrizio, 853195
Castelli Valerio, 853992

Referent professor: Di Nitto Elisabetta

January 17, 2016

Contents

Chapter 1

Introduction

1.1 Revision History

Version	Date	Author(s)	Summary
1.0	21/01/16	Valerio Castelli & Fabrizio Casati	Initial release

1.2 Purpose and Scope

This document represents the Integration Testing Plan Document for myTaxiService. Integration testing is a key activity to guarantee that all the different subsystems composing myTaxiService interoperate consistently with the requirements they are supposed to fulfill and without exhibiting unexpected behaviors. The purpose of this document is to outline, in a clear and comprehensive way, the main aspects concerning the organization of the integration testing activity for all the components that make up the system. In the following sections we're going to provide:

- A list of the subsystems and their subcomponents involved in the integration activity that will have to be tested
- The criteria that must be met by the project status before integration testing of the outlined elements may begin
- A description of the integration testing approach and the rationale behind it
- The sequence in which components and subsystems will be integrated
- A description of the planned testing activities for each integration step, including their input data and the expected output

- A list of all the tools that will have to be employed during the testing activities, together with a description of the operational environment in which the tests will be executed

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Subcomponent
- SubsystemTBD tbd

1.3.2 Acronyms

- SDD: Software Design Description.
- DD: Design Document. Used as a synonym of SDD.
- DBMS: Database Management System.
- API: Application Programming Interface.
- RASD: Requirement Analysis and Specification Document.
- SRS: Software Requirements Specifications. Synonym of RASD.
- ETA: Estimated Time of Arrival.
- UI: User Interface.
- GPS: Global Positioning System.

1.3.3 Abbreviations

- Req. as for Requirement.
- WebApp as for Web Application.

1.4 Reference Documents

- The project description document: Assignments 1 and 2 (RASD and DD).pdf
- Assignment document: Assignment 4 - integration test plan.pdf
- myTaxiService Requirement Analysis and Specification Document: RASD.pdf
- myTaxiService Design Document: DD.pdf

- The Integration Test Plan Example document: Integration Test Plan Example.pdf

Chapter 2

Integration Strategy

2.1 Entry Criteria

In order for the integration testing to be possible and to produce meaningful results, there are a a number of conditions on the progress of the project that have to be met.

First of all, the **Requirements Analysis and Specification Document** and the **Design Document** must have been fully written. This is a required step in order to have a complete picture of the interaction between the different components of the system and of their required functionalities.

Secondly, the integration process should start only when the estimated percentage of completion of every component with respect to its functionalities is:

- **100%** for the **Data Access Utilities** component
- At least **90%** for the **Taxi Management System** subsystem
- At least **70%** for the **System Administration** and **Account Management** subsystems
- At least **50%** for the **client applications**

It should be noted that these percentages refer to the status of the project at the beginning of the integration testing phase and they do not represent the minimum completion percentage necessary to consider a component for integration, which must be at least **90%**. The choice of having different completion percentages for the different components has been made to reflect their order of integration and to take into account the required time to fully perform integration testing.

2.2 Elements to be Integrated

In the following paragraph we're going to provide a list of all the components that need to be integrated together.

As specified in myTaxiService's Design Document, the system is built upon the interactions of many high-level components, each one implementing a specific set of functionalities. For the sake of modularity, each subsystem is further obtained by the combination of several lower-level components. Because of this software architecture, the integration phase will involve the integration of components at two different levels of abstraction.

At the lowest level, we'll integrate together those components that depend strongly on one another to offer the higher level functionalities of myTaxiService. In our specific case, this involves the integration of the **Reservation Management**, **Request Management**, **Location Management** and **Taxi Management** subcomponents in order to obtain the **Taxi Management System** subsystem.

For what concerns the building of the **System Administration** and **Account Management** subsystems, the integration activity is actually quite limited; in fact, they simply represent a collection of functionalities belonging to the same area which however are not dependent on one another. As a result of this, their subcomponents don't really interact with each other, and the integration phase will be limited to the task of ensuring that the set of functionalities of each subcomponent is properly exposed by the subsystem. The components involved in this phase are:

- The **API Permissions Management**, **Zone Division Management**, **Taxi Driver Management**, **Service Statistics** and **Plugin Management** subcomponents in order to obtain the **System Administration** subsystem.
- The **Passenger Registration**, **Login**, **Password Retrieval** and **Settings Management** subcomponents in order to obtain the **Account Management** subsystem.

Some of these subcomponents also directly rely on higher level, atomic components: that is the case, for instance, of the dependency on the **Data Access Utilities** component. This dependency will be taken care of in the integration process.

Finally, we will proceed with the integration of the higher level subsystems. In particular, the integration activity will involve:

- A number of commercial, already existing components, used to achieve specific functionalities: these are the **DBMS**, **Mapping Service**, **Notification System** and **Remote Services Interface** components.
- Those components and subsystems specifically developed for myTaxiService, specifically:

- On the server side: the **Taxi Management System**, **System Administration**, **Account Management** subsystems, together with the **Data Access Utilities** component.
- On the client side: the **Administration Web Application**, **Passenger Web Application**, **Passenger Mobile Application** and **Taxi Driver Mobile Application** components.

2.3 Integration Testing Strategy

The approach we're going to use to perform integration testing is based on a mixture of the bottom-up and critical-module-first integration strategies.

Using the bottom-up approach, we will start integrating together those components that do not depend on other components to function, or that only depend on already developed components. This strategy brings a number of important advantages. First, it allows us to perform integration tests on "real" components that are almost fully developed and thus obtain more precise indications about how the system may react and fail in real world usage with respect to a top-down approach. Secondly, working bottom-up enables us to more closely follow the development process, which in our case is also proceeding using the bottom-up approach; by doing this we can start performing integration testing earlier in the development process as soon as the required components have been developed in order to maximize parallelism and efficiency.

Since subsystems are fairly independent from one another, the order in which they're integrated together to obtain the full system follows the critical-module-first approach. This strategy allows us to concentrate our testing efforts on the riskiest components first, that is those that represents the core functionalities of the whole system and whose malfunctioning could pose a very serious threat to the correct implementation of the entire myTaxiService infrastructure. By proceeding this way, we are able to discover bugs earlier in the integration progress and take the necessary measures to correct them on time.

2.4 Sequence of Component/Function Integration

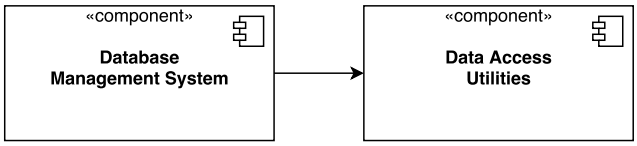
In this section we're going to describe the order of integration (and integration testing) of the various components and subsystems of myTaxiService. As a notation, an arrow going from component C1 to component C2 means that C1 is necessary for C2 to function and so it must have already been implemented.

2.4.1 Software Integration Sequence

Following the already mentioned bottom-up approach, we now describe how the various subcomponents are integrated together to create higher level subsystems.

Data Access

The first two elements to be integrated are the **Data Access Utilities** and the **Database Management System** components. We start from here because every other component relies on **Data Access Utilities** to perform queries on the underlying data structure.



Taxi Management

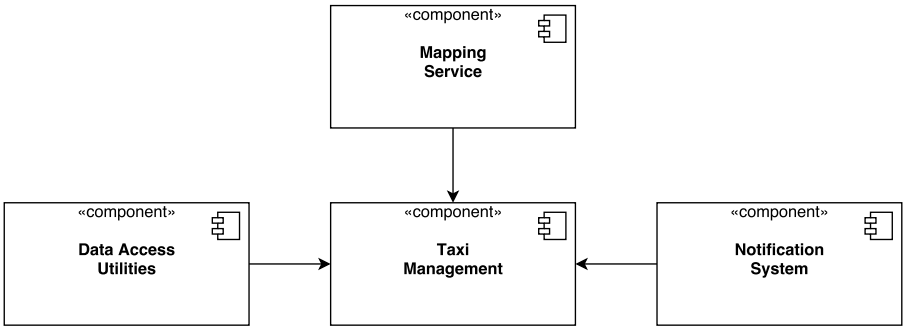
The second step in the integration process is to appropriately connect the subcomponents implementing the **Taxi Management System**. This choice comes from the critical-module-first approach, because the taxi management is the single most important functionality of myTaxiService.

In the following diagrams, we are going to show exactly which components must be integrated together in order to implement this functionality using a bottom-up approach.

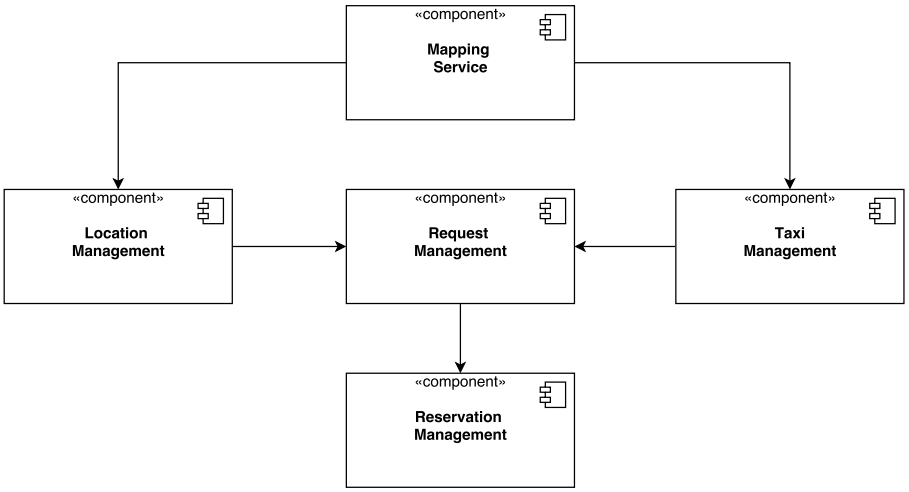
First, we procede by integrating together the **Request Management** subcomponent with the **Data Access Utilities** and the **Notification System** components.

The same activity is performed between the **Reservation Management** subcomponent and the **Data Access Utilities** and the **Notification System** components.

Finally, we integrate together the **Taxi Management** component with the **Data Access Utilities**, the **Notification System** and the **Mapping Service** components.



At this point, the four sub-components of **Taxi Management System** are ready to be integrated together.

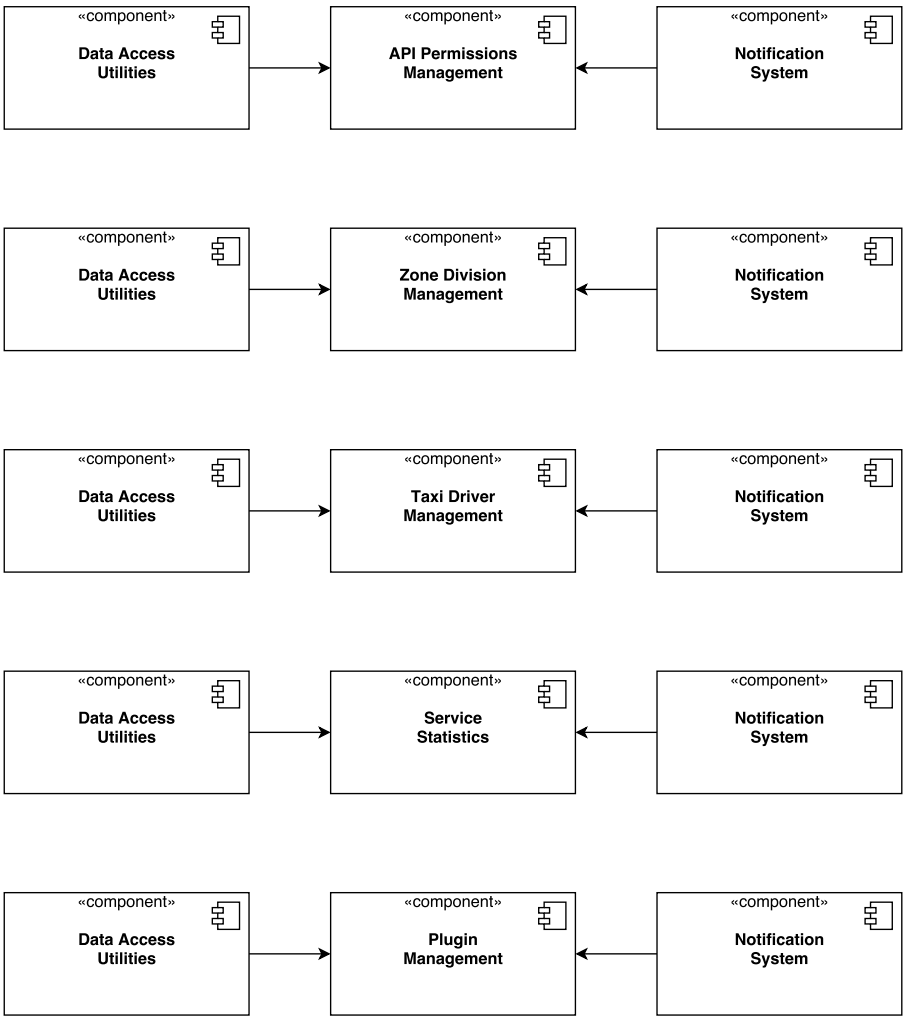


System Administration

The third step in the integration process is to appropriately connect the subcomponents implementing the **System Administration** subsystem. This choice comes from the critical-module-first approach, because the system administration is the second most important functionality of myTaxiService. Once it has been integrated and tested, we can use this functionality to more easily populate the database for the following integration tests.

It should be noted that the subcomponents of **System Administration** are loosely coupled together as they cover different aspects of the system administration activity. Because of this, they can be integrated with the other components of the system independently from one another.

In the following diagrams, we are going to show exactly how these subcomponents interact with the other components using a bottom-up approach. The **System Administration** subsystem, which here is not explicitly represented, is simply a wrapper for the methods of these subcomponents that have to be exposed to the other parts of the system and performs additional preprocessing to ensure these methods are properly called. It will be discussed more in depth in the **Subsystem Integration Sequence** section.



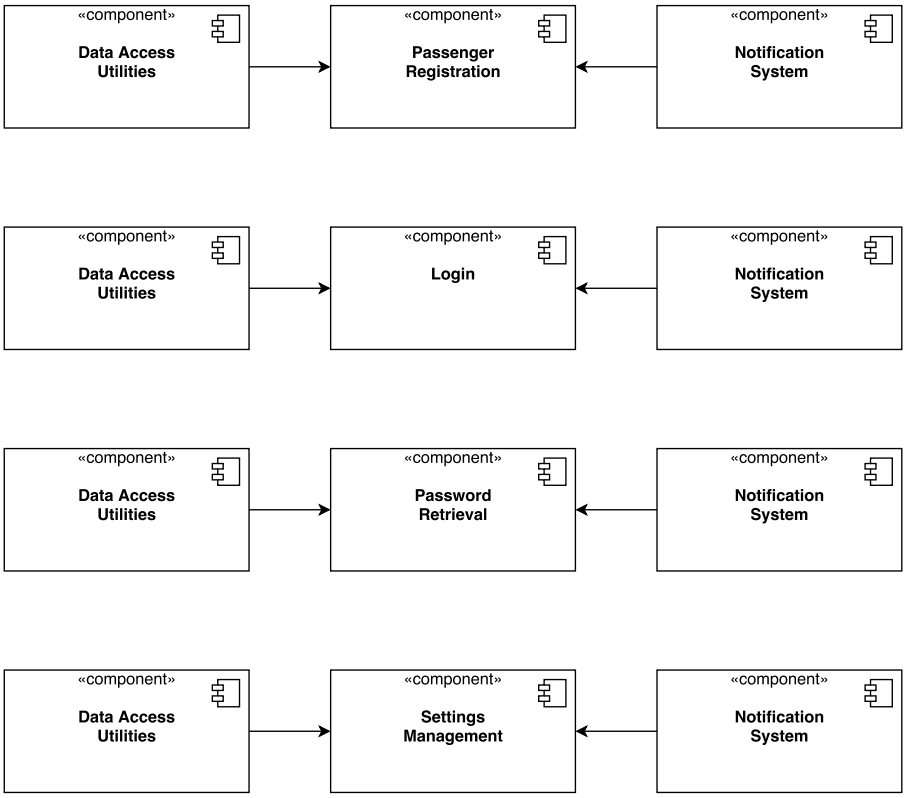
Account Management

The fourth step in the integration process is to appropriately connect the subcomponents implementing the **Account Management** subsystem. This choice is dictated by the bottom-up approach that we are following, because

account management is the last functionality that can be implemented without depending on anything but already implemented components.

It should be noted that the subcomponents of **Account Management** are loosely coupled together as they cover different operations that can be performed on accounts. Because of this, they can be integrated with the other components of the system independently from one another.

In the following diagrams, we are going to show exactly how these subcomponents interact with the other components using a bottom-up approach. The **Account Management** subsystem, which here is not explicitly represented, is simply a wrapper for the methods of these subcomponents that have to be exposed to the other parts of the system and performs additional preprocessing to ensure these methods are properly called. It will be discussed more in depth in the **Subsystem Integration Sequence** section.



2.4.2 Subsystem Integration Sequence

Chapter 3

Individual Steps and Test Description

Chapter 4

Tools and Test Equipment Required

Chapter 5

Program Stubs and Test Data Required

Appendix A

Hours of work

To redact this document, we spent 20 hours per person.