Integration Test Plan

ANDREA MAIOLI

Entry Criteria

- All the classes and functions must be well documented using JavaDoc.
- All the classes and functions must be tested using JUnit-tests.
- Code Inspection must be performed on all the code.
- All the bugs found must be fixed.
- RASD and DD must be updated and delivered.

Elements to be Integrated
Client Tier

- Mobile Application
- Web Browser

Elements to be Integrated
Web Server Tier

- Web Server Controller
- Mobile API (with the help of JAX-RS, provided by the Web Server Controller)
- Website Interface (with the help of JSF)

Elements to be Integrated Application Server Tier

- Request Manager
- Queue Manager
- Account Manager
- Location Manager
- Taxi Manager
- Entity Beans: Queue, Area, Driver, Passenger, Operator, Request and User

Elements to be Integrated Database Server Tier

- DBMS

Integration Testing Strategy

- The integration process can be divided into two different phases:
  - 1) Integration between components that compose the same subsystem.
  - 2) Integration of different subsystems

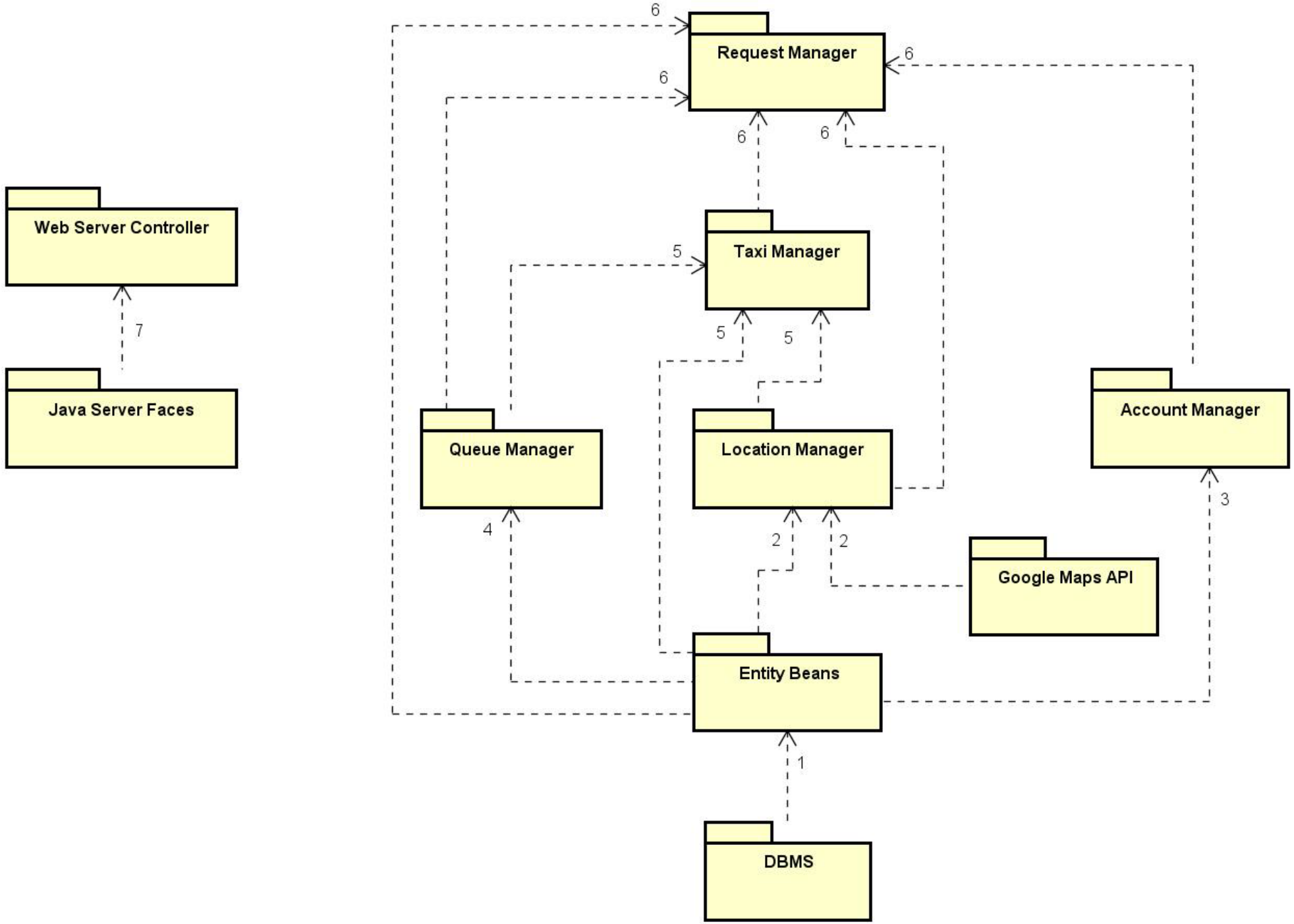
- The Integration Testing Strategy will follow the bottom-up approach.
- Why?
  - Nature of the system: different components relies on other
  - No useless stubs
  - Each component of each subsystem is already tested

Integration Testing Strategy

- The integration will start from the components with the minimum number of dependencies. This prevents the implementation of stubs because when the integration of a component takes place, the components in which it relies on have been already integrated

Integration Sequence

Software


Web Server Controller
Java Server Faces
7
Request Manager
6
6
6
6
6
Taxi Manager
5
5
5
Queue Manager
Location Manager
Account Manager
4
2
2
3
Google Maps API
Entity Beans
1
DBMS

Integration Sequence - Software

1) Integration of the Testing DBMS with all the Entity Bean (Area, Queue, User, Driver, Passenger, Operator, Request).
2) Integration of Entity Beans (Area, Queue) with Location Manager.
3) Integration of Entity Beans (User) with Account Manager.
4) Integration of Entity Beans (Area, Queue, Driver) and Queue Manager.
5) Integration of Entity Beans (Driver, Queue, Area), Location Manager and Queue Manager with Taxi Manager.
6) Integration of Entity Beans (Area, Queue, Passenger, Operator, Driver, Request), Location Manager, Account Manager, Queue Manager and Taxi Manager with Request Manager.
7) Integration of the Java Server Faces with the Web Server Controller.

Integration Sequence - Software

- **Database Server Subsystem**

  No integration at software level.

- **Application Server Subsystem**
  - Integration starts from the Entity Beans: they have no dependecies on the other componens of the subsystem.
  - The Entity Beans must be tested with a Testing Database.

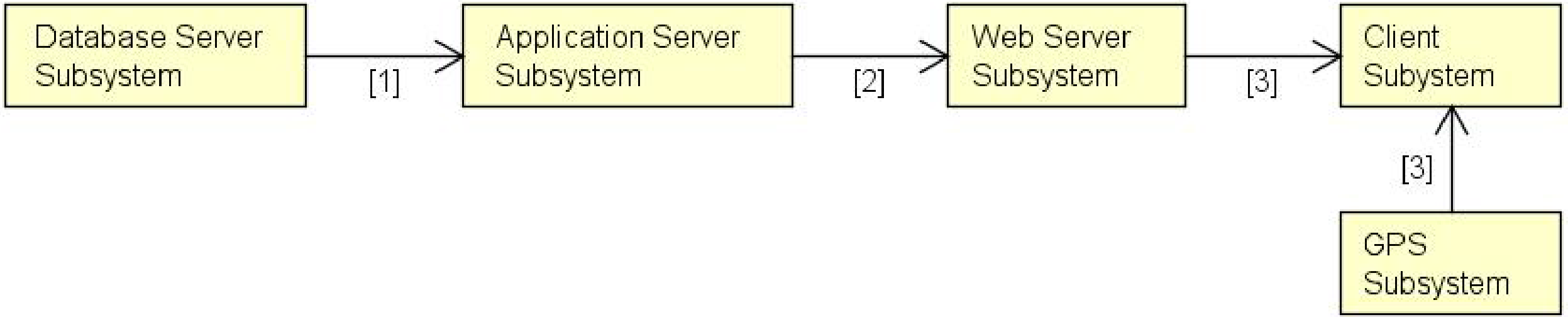
Integration Sequence - Software

- **Web Server Subsystem**
  - The only integration is between the JSF component and the Web Server Controller
- **Client Subsystem**
  - No integration at software level

Integration Sequence

Subsystems

Integration Sequence - Subsystems



Integration Sequence - Subsystems

1) Integration of the Database Server Subsystem with the Application Server Subsystem.
2) Integration of the Application Server Subsystem with the Web Server Subsystem.
3) Integration of the Web Server Subsystem and the GPS Subsystem with the Client Subsystem.

Individual Steps and Test Description

Test Case Identifier	I1T1
Test Item(s)	DBMS → Entity Bean "Area"
Input Specification	Typical query on the table "Area".
Output Specification	All the requested operations are made on the table and all the expected data is returned from the query.
Environment Needs	Testing Database, Glassfish Server, Driver for the Entity Bean
Purpose	This test check if the called methods of the Entity Bean "Area" execute the expected query on the DBMS.

Individual Steps and Test Description

Test Case Identifier	I2T1
Test Item(s)	Google Maps API → Location Manager
Input Specification	Request from Location Manager to the Google Maps API.
Output Specification	The request returns the expected information.
Environment Needs	Google Maps API, Glassfish Server, Driver for the Location Manager
Purpose	This test checks if the correct information about a given location (provided in form of latitude and longitude) is retrieved from the Google Maps API.

Individual Steps and Test Description

Test Case Identifier	I5T4
Test Item(s)	Location Manager → Taxi Manager
Input Specification	Methods call from the Taxi Manager to Location Manager.
Output Specification	Check if the Taxi Manager calls the correct methods of Location Manager.
Environment Needs	Testing Database, Glassfish Server, Driver for the Taxi Manager
Purpose	This test checks that the Taxi Manager is able to properly manage the location information associated to a Taxi, update the Taxi associated area and compute the estimated time to get to a request pick-up point.

Individual Steps and Test Description

Test Case Identifier	I5T5
Test Item(s)	Queue Manager → Taxi Manager
Input Specification	Methods call from the Taxi Manager to Queue Manager.
Output Specification	Check if the Taxi Manager calls the correct methods of Queue Manager.
Environment Needs	Testing Database, Glassfish Server, Driver for the Taxi Manager
Purpose	This test checks that the Taxi Manager is able to add or remove the considered taxi from a Queue, in front of an update of the taxi status.

Individual Steps and Test Description

Test Case Identifier	I6T10
Test Item(s)	Taxi Manager → Request Manager
Input Specification	Methods call from the Request Manager to Taxi Manager.
Output Specification	Check if the Request Manager calls the correct methods of Taxi Manager.
Environment Needs	Testing Database, Glassfish Server, Driver for the Request Manager
Purpose	This test checks that the Request Manager is able to change the status of the taxi associated to the request.

Tools and Test Equipment Required

- **Mockito**: is a testing framework that allows to abstract dependencies, generating mock objects, drivers and stubs.
- **Arquillian**: is a testing framework that allows to execute test cases inside a java container.
- **JMeter**: is a software designed to load test functional behavior and measure performance. In the integration testing phase can be used to automate the integration test between the Web Server and the Client.

Program Stubs and Test Data Required

- **Testing Database**: all the testing environment must include a DBMS configured in the same way of the production DBMS, but with a less number of instances. This will prevent the waste of resources and time in the integration testing phase, but will grant to work with the same data.
- **Stubs of the Application Server Subsystem**: this stub will provide a small set of data used for the Web Server Subsystem testing, without having the Application Server ready.
- **Tiny API Client**: this driver will be used for the testing of the RESTful API provided by the Web Server Subsystem, without having a client application ready.
- **Drivers for each component**: this is needed for simulating a call for the component methods and verify the result on the integrated components.

Thank You

ANDREA MAIOLI