News Agency Management System
Analysis and Design Document

Student: Varadi Robert
Group: 30432

Table of Contents

1. Requirements Analysis 3
   - 1.1 Assignment Specification 3
   - 1.2 Functional Requirements 3
   - 1.3 Non-functional Requirements 3

2. Use-Case Model 4

3. System Architectural Design 4

4. UML Sequence Diagrams 8

5. Class Design 9

6. Data Model 10

7. System Testing 10

8. Bibliography 10

1. Requirements Analysis

1.1Assignment Specification

Use Java/C# API to design and implement a client-server application for a news agency. The application has 2 types of users: the readers and the writers. The **readers** can view a list of articles, read an article and do not need to login in order the use the application. The **writers** need to authenticate in order to create, update or delete articles. So, the writer accounts are preset by the application developer and cannot be altered.

An article has the following components:

- Title
- Abstract
- Author
- Body

The application must support multiple concurrent users. If a writer posts a new article, the readers must see it in the list of articles in real time, without performing any refresh operation

1.2Functional Requirements

The functional requirements for the system are the following ones:

- The reader may perform operations without being logged into the application, and they cannot see the articles deleted by the writers.
- The writer must log in before doing any kind of operation, he/she can create a new article if and only if the article does not already exist, can update or delete only an existent article.

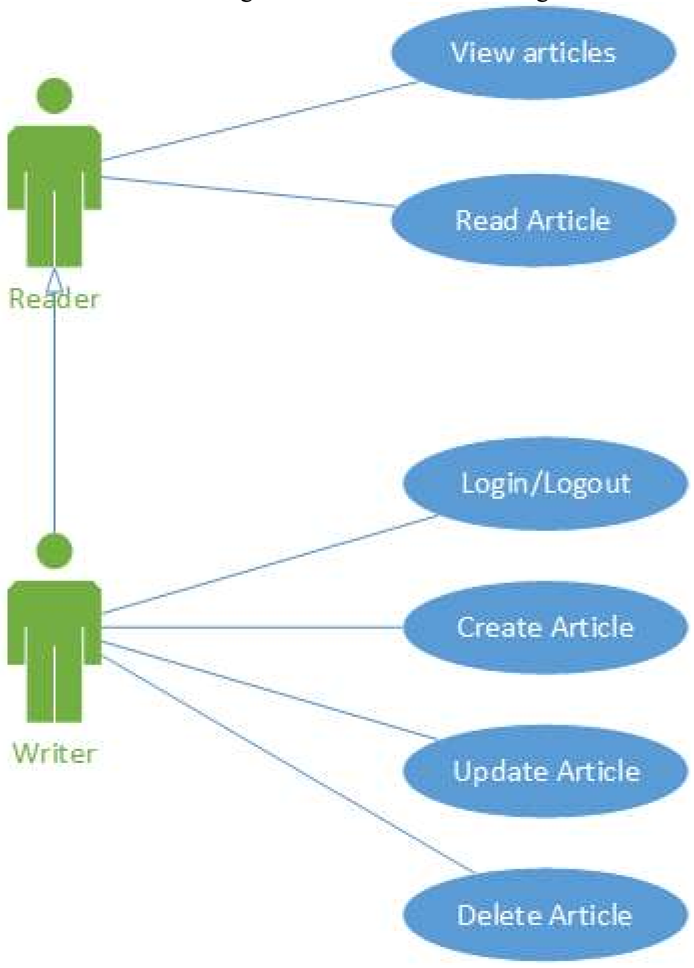
1.3Non-functional Requirements

The non-functional requirements for the system that we will take into account are the following ones:

- Availability: The system must be available 100% of the time, except the periods of planned maintenance, when the user cannot access the system.
- Performance: The system must do each operation in less than 1s 95% of the time and under 2s 5% of the time.
- Reusability: The components of the system must be developed in such a way that they can be reused by other systems if necessary.
- Testability: The system’s components must be written in such a way that they will be easy to test.
- Usability: The system must be easy to use, so that even people who are not very skilled with computers should be able to use it.

2. Use-Case Model

From the specifications of the application one can deduce the presence of the following actors: the Reader, who is able to view the articles and read a selected article and the Writer, who is able to login/logout, create an article, update an existing article and delete an existing article. The use – case diagram will be the following one:



Use Case Diagram

Use case: Create a new article
Level: user-goal level
Primary actor: Writer
Main success scenario: The new article is created and it is available for the readers who want to read it.
Extensions: The server fails and the article cannot be saved.

3. System Architectural Design

3.1 Architectural Pattern Description

For this system, the main architectural pattern that will be used is the Client – Server architectural pattern, which divides the system into two tiers, the Client which interacts with the user and sends some message (request) to the other tier, the server, who is responsible for the information processing and for sending the response back to the client so that the response can be displayed in some form for the user of the system. Since we are going to use a database as well, the application will be a client – server application also from the point of view of the application back-end and the database. From this perspective, the server will be the database and the client will be the back-end of the application. Such an architecture is also called a three-tier architecture.

The other architectural pattern which will be used is the layers pattern, which divides the application into 4 layers: the data access layer, responsible for accessing data stored in data sources, in this case the database itself, or for accessing external services, provided by other applications; the service layer which contains the business logic of the application, namely the operations that can be performed on the entities present in the domain model of the system; the presentation layer, which is the layer responsible for what the user of the application sees, being the main interaction mean between the user and the application itself; the controller which is responsible for taking the commands from the presentation layer and for sending them to be displayed by the view.

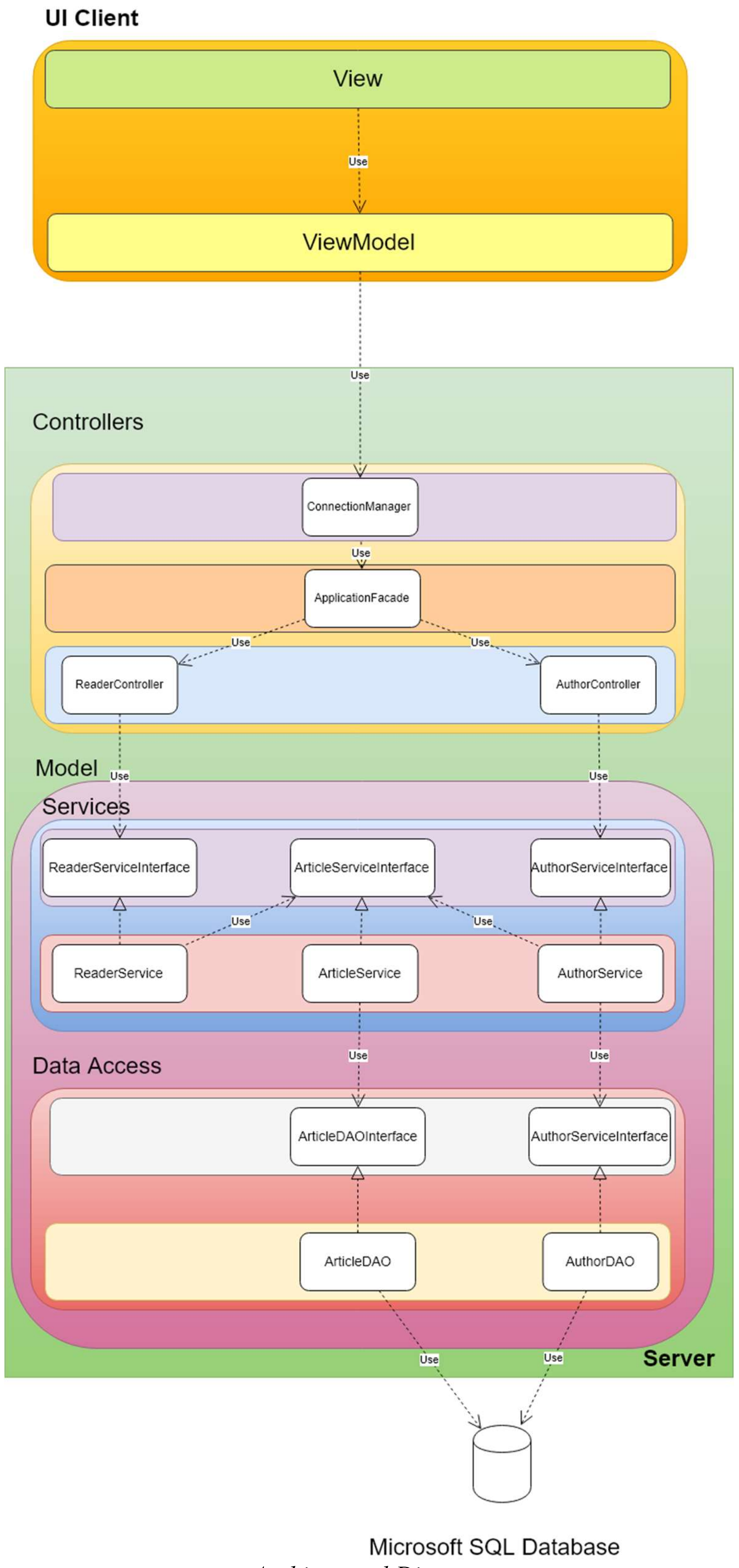
The last architectural pattern which will be used in our application is the MVVM (Model View ViewModel), which consists of 3 main components: the model, which contains the domain model entities as well as the operations that can be performed on them, the view which is the interface the system provides so that the users can use it and the ViewModel which will contain the current state of the view, namely what should the view display at a given moment in time.

For each component of the architecture we will assign a package and for each sub – component a sub – package will be assigned.

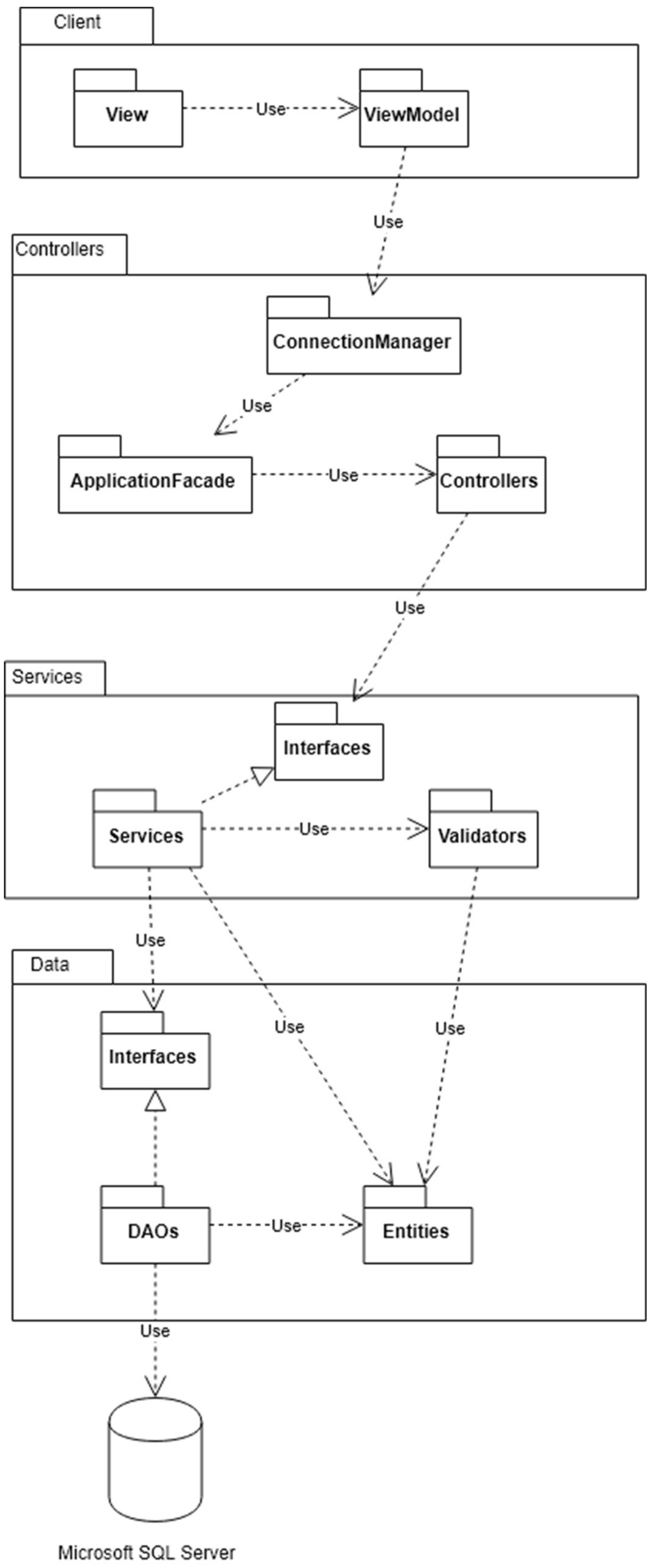
For the component diagram, we have 3 tiers, the database which provides an interface for the back-end of the application, the back – end which provides an interface for the UI client and the UI itself which directly interacts with the user.

For the deployment diagram we have 3 nodes, the database node, the application back-end node and the UI client node. The database node's artifact will be a Microsoft SQL Server database, the application back-end node's artifact is the application back-end written in Java and the client is a JavaFX application which interacts with the user and sends requests to the server in JSON format.

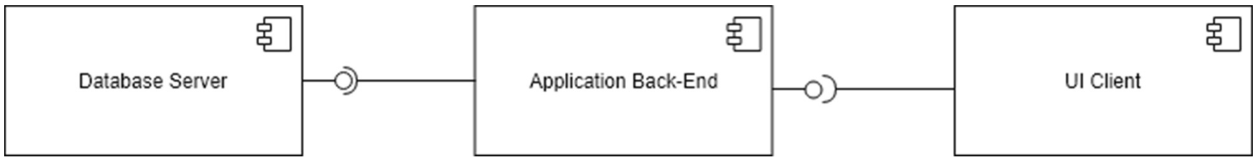
3.2 Diagrams



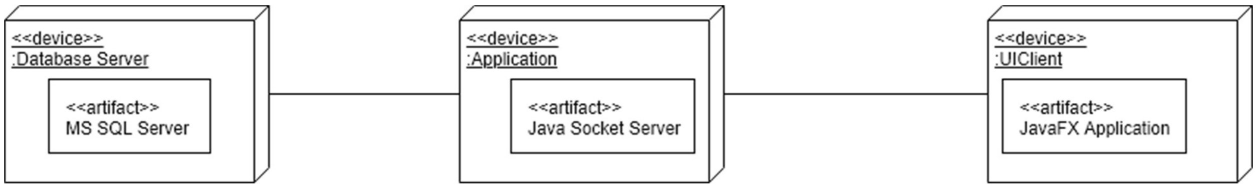
Architectural Diagram



Package Diagram



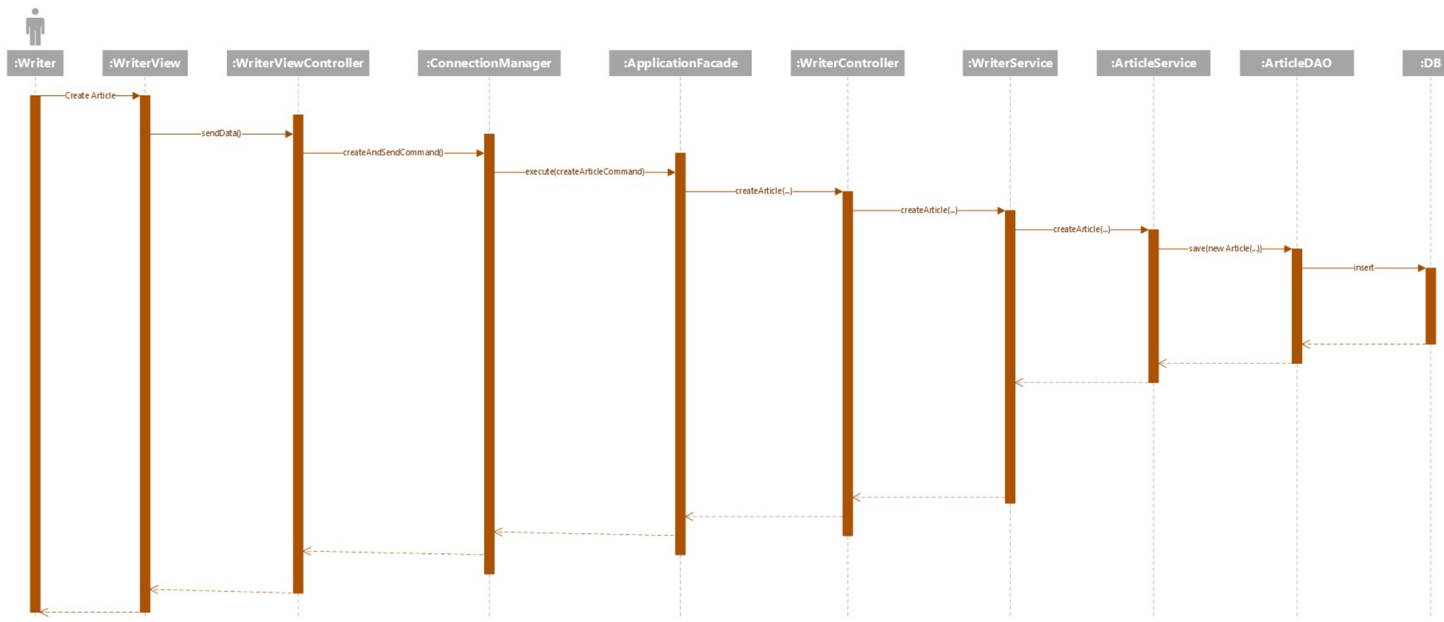
Component Diagram



Deployment Diagram

4. UML Sequence Diagrams

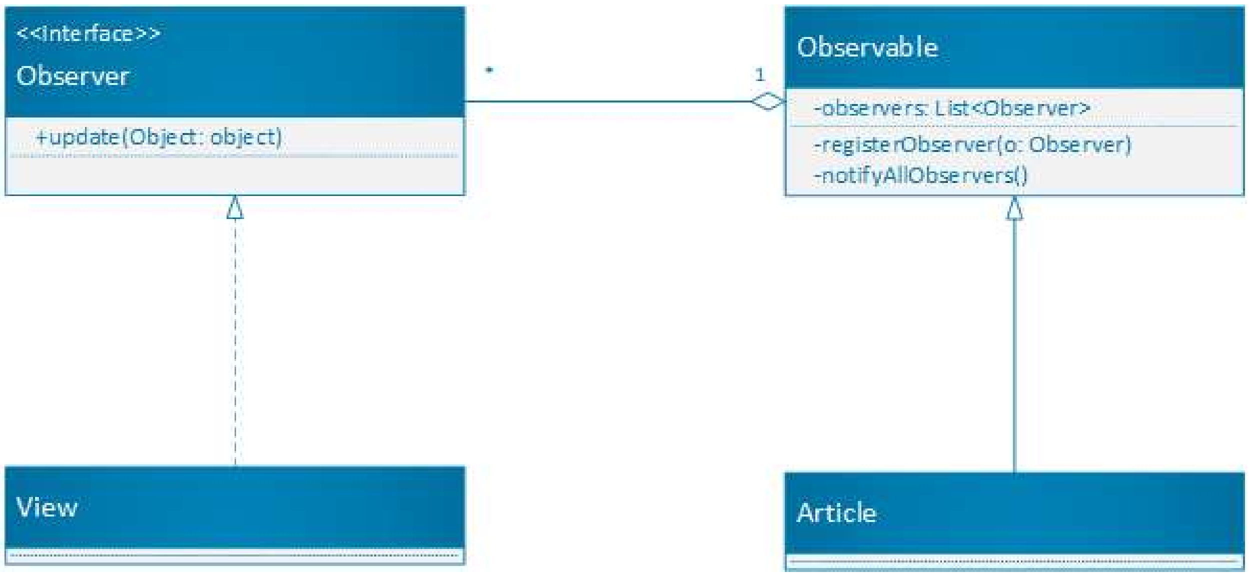
In this section we will describe the "Create Article" use case using a sequence diagram. The actor for this use case is the Writer, who, after entering the data regarding the article, presses the "Create Article" button, action which is intercepted by the ViewModel component of the application, responsible for connecting to the server, creating a command and sending it to the socket server. The socket server's connection manager listens for new connections and once a new connection arrives, the command is sent to the ApplicationFacade which will know which Controller is responsible for executing the command. In this case the WriterController is the one responsible for such an action. The controller calls the WriterService's method for creating a new article. The call is delegated to the ArticleService, which creates and stores the article object using its ArticleDAO.



5. Class Design

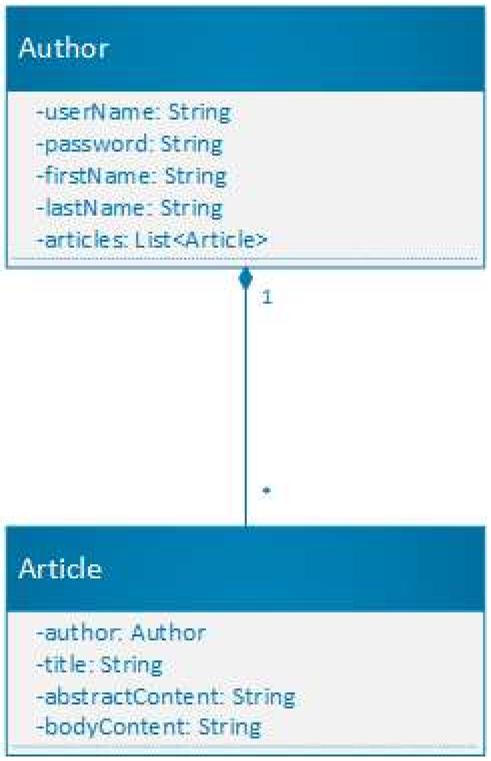
5.1 Design Patterns Description

The design pattern which will be used is the Observer pattern, which consists of two main components, an Observable class which will be the subject of observation for the Observer, which is an interface, which has an update method which is being called anytime the Observable updates. For our case, the changes will be made on the server, which should update all its observers, which are the clients, whenever an article is modified.



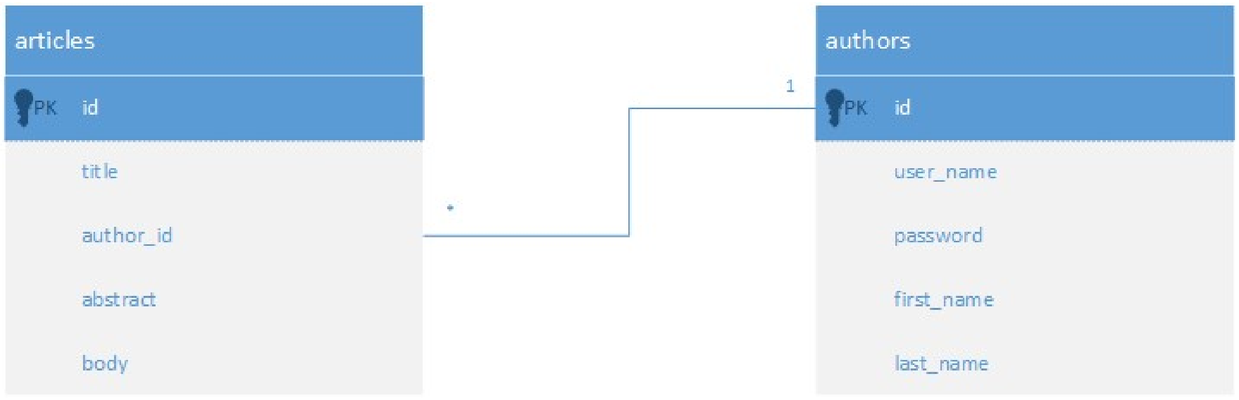
5.2 UML Class Diagram

The UML class diagram of the application is very simple, it only has to contain the Author and Article classes, which have a one – to – many composition relation among them since an article should not exist without an author for that article.



6. Data Model

The data model for the application is very similar to the class diagram, the only difference is that instead of references we use foreign keys and each entity will have a primary key.



7. System Testing

In order to test the application, we will use unit testing which consists of writing the methods to be tested into a JUnit file, in which each method is a test case and can be used to test some functionalities of the system. The test consists of providing some input data to the method, perform some operations and compare the result with the result expected by us, through an assertion. If the assertion fails, the unit test failed, so there could be a problem with our implementation.

The testing method will be testing based on partitioning in which we try to split input domain into disjoint subsets, whose union gives the initial input domain (partitions). This partition has the property that for all its elements, the method will behave in the same way (for example if, they are all invalid input data, the method should throw an exception). For testing the whole input domain, we could select want representative from each partition and run the test for that particular value. Together with this testing method, we should employ also the boundary analysis which means to test not only for the values from the particular partitions, but also for the boundaries between two partitions. This way we might detect inconsistencies in the program. For example, let us suppose that the input domain is the set of real numbers and the program should behave differently for positive real numbers and for negative real numbers. In this case we take a negative number, a positive number and test the output of the method, but it would be also useful to test the output for the boundary, that is, zero in this case, because the method might not behave as expected for boundary values.

8. Bibliography