Software Engineering

Assignment 4: Software Architecture & Quality Control

3 Ba INF 2018-2019

Benjamin Vandersmissen
Francuiscus Fekkes

December 3, 2018

1 Software Engineering

1.1 Oefening 1

The **cohesion** of the *System* class is rather low, it implements a lot of functions that are executed on its attribute members. An example of a function that could be implemented somewhere else and be much better is *System.login(user_id, username, password)*. It makes no sense to put this function in the *System* class, it would be better in the *User* class. Another reason for the low cohesion are the add/removeItem... functions, they should better be implemented in the class *Cart*.

The **coupling** of this class is rather high, the only functions that depend on methods defined by other objects are the add/removeItem functions. As said in the previous paragraph, these functions shouldn't be defined in the *System* class, but should rather be used in conjunction with a (not implemented) *System.getCart(user_id)* method. This will ensure that changing the add/removeItem... method in the class *Cart* will not have any effects on the methods in class *System*.

The **Model-View-Controller pattern** should be used to separate the front-end and the back-end cornensponding to the view, resp. model. The layout and the design of the page shouldn't be dependent of the back-end and a redesign of the front-end should mean that we wouldn't have to edit anything in the back-end.

1.2 Oefening 2

The **Client-Server pattern** divides the functionality in two parts, a **Client** and a **Server** part. The server is always reachable and is reactive, this means that to execute serverside code, the server needs to have an external stimulus. The client is a process that requests information from a server. A server can have multiple clients, who each can request different data and/or functionality from the same server.

If we were to implement this pattern in our system, we should have all components running the Client-Server protocol. Almost all of the code should be serverside, such that the client can't manipulate data in his favor. The only code that the client should be able to run is the visualisation code, which should include the getters.

A disadvantage of this pattern is that there is a single point of failure. This means that if the server were to fail for any number of reasons, the clients aren't able to reach the server and they have almost no functionality. Another problem could be the connection between the client and the server. It is possible that the connection is insecure and the clients data is intercepted or the server data is modified before it arrives to the client.

To solve the single point of failure problem, we could work with microservices and P2P networks, where we subdivide the functionality on the server in multiple cohesive, loosely coupled chunks and run them all as microservices on different computers. This will ensure that even when the user service fails for example, the clients can still browse the catalog and order anonymously.

2 Quality Control

2.1 Oefening 1: Summary FTR

The **Formal Technical Review** method can be used to check the quality of the software. This method will be used before the testing phase of the development. It uses formally predefined checklists and forms to review code. This reviewing is done by peers or seniors of the author of said code. The reviewers wil note issues and the importance of said issue. During a meeting between the reviewers and the authors feedback wil be given so the authors can fix the problems. Issues can range from small bugs or style differences to system crash inducing problems.
There are several advantages to this system. Although the reviewing and meetings take time, it saves al lot in this long run as wel. As more people are looking at the same problem allows for more and better pre-emptive bug fixing. The more people look at each others code also improves the overall insight to the

entire program. As a final positive, it also allows for a permanent learning experience for the authors.

2.2 Oefening 2: Applying FTR

Zie map bijlage/checklists en bijlage/forms.
Er zijn voorbeeld moderators,authors en reviewers genomen.