D7032E

D7032E Software Engineering

Assignment 6 – Home Exam

Group 4 - Entente Github project CGeo
Malin Christoph, icoami-5@student.ltu.se

Date: 29.10.2015

Table of Contents

1. The Software Engineering context of Software Architecture

I would like to create an online application. The application shall act as a calendar with the possibility to load also other calendars from other websites e.g. google calendar. I would like to use Scrum for Software development.

At the beginning feature requests from team members (developers, testers, ...), customers and executives will be written down to scrum user stories e.g. As a (role), I want (feature), so that (benefit). The stories help to find out the non-functional requirements e.g. reliability, maintainability, etc. of the system. Next step is do decide how software architecture will be described. I would like to use UML as Architecture description language. I think class diagrams, module diagrams or sequence diagrams are perfect for that. For the hole project I would like to have a pattern based architecture design. As a Software architect I know most of the design patterns and so I can use the Model View Controller Pattern as basic architecture. The advantage of using patterns is that they provide good solutions for common problems and that a lot of programmers are very familiar with patterns.

With the knowledge of the different patterns and the following design principles in mind I can create my architecture:

- Single responsibility principle, don’t repeat yourself, principle of least knowledg, separation of concerns, minimize upfront design

The knowledge from previous projects is always part of design considerations. Output of all the steps is a word document containing different UML diagrams and descriptions to the diagrams.

2. System Architecture

I will use the Model-view-controller pattern. Since I would like to implement the photo-album application as an online application the MVC pattern looks perfect for me.

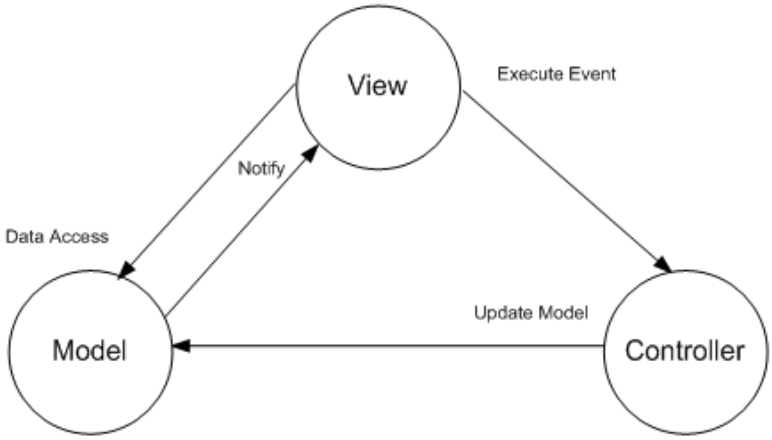

Figure 4: Simplistic MVC

As you can see in the diagram the application is split up into three interconnected parts. This allows to split up the work between the three teams fairly easily. On team can focus on the development of the image recognition and face detection feature. Meanwhile one team can design and program the view while the other team can word on the model (database) and some parts of the controller. Since the attributes of pictures e.g. size, friends in the photo are stored in the database it is very easily to filter very fast. Some SQL statements can do the job. For the communication with the social media platforms extra packages with interfaces will be created that we have a realy weak coupling.

Advantages

- Application is split up into model, view and controller → This allows to easily replace one part without the need to change the other parts e.g. replacing one database with another
- Parallel development by the three different teams → each time can focus on one part of the MVC
- Multiple view support → It is easy to generate different templates for the users. When the view get's changed, it is not always necessary to touch the model for example.

Disadvantages

- Complexity → it is not the right pattern for smaller applications
- closely coupling between view and controller → when you modify one of it, it may affect also the other

3. Architecture design / Software maintenance

High cohesion means that a class is focused on what it should do. The class has only methods related to the intention of the class. **Coupling** is about how related classes are to each other. Low coupling means that you can make a major change in one class and the other class is not affected from it.

I think **modifiability** is an quality attribute where high cohesion and low coupling is very important. Modifiability in general is about change, how expensive a change is and the risk of making a change. With high cohesion and low coupling it is getting easier to implement a change. Maybe you only need to touch one module (low coupling) and inside the module only two classes. Cause of the high cohesion all the related functions stay close together.

Low coupling and high cohesion is also important for **testability**. Writing the test cases gets easier because the amount of modules which are part of the unit testing can be reduced to a minimum. When it is easy to write test cases you save time and also money.

High Cohesion and low coupling is always good, but sometimes hard to achieve. It increases readability because all the related functions stay close to each other. It is also good for maintainability because when you need to debug the program you may stay in a single module.

4. Testing

Many companies found out that a good testing strategy can save a lot of money. The later a bug gets discovered in the lifecycle of a software the more expensive it gets. It is easy to fix a bug when you start developing but very expensive when your software is already distributed on million devices where you don't have access to make updates. Depending on the type of software a different code coverage must be provided from the testing software. To ensure that there are nearly no bugs within the software all different code parts must be tested with different input values. When the test data is bigger you can assume that your code will work better.

When you create test cases for all functions in the system it is much easier to find a bug automatically. For example: You write test cases for every function in your project. You implemented the function and also the tests were successful. Days later you add more features and change some things. You check in your code. All the test cases are automatically executed after the check in. The test case who worked fine before now failed. Now it's easily to see that your change effected other code and you can fix it. If you don't have a good testing you always need to test it maybe manually by people. And people are normaly the most expensive part of every project. It is cheap to write automatic test cases but very expensive to pay 200 people testing a program every time when you made some changes.

5. Implementation

It always depends on the functional, non-functional requirements of the software and also how much time and money you have to implement the software.

It may be appropriate to start just coding if the programm will not be so big and you are able to implement a decent solution which fulfills his purpose and the code is not to messy. If it is very urgent you may not have the time to start with designing an architecture.

In nearly all other cases where you have enough time and resources just coding may be the wrong way. A good design has so many advantages that it is most of the time cheaper in total to spend some time thinking about design than just start with coding. With a good design you get a better modularity, reusability, modifiability, testability. It will be much easier to implement new features in the future or to change specific things. If you just start coding it may become very difficult to add an extra feature and the whole program may become very convoluted.

Resources

http://www.liaolin.com/Courses/architecture02.pdf

http://ehrscience.com/2012/02/29/software-architecture-and-design-first-steps/

https://en.wikipedia.org/wiki/Software_architecture

https://en.wikipedia.org/wiki/Software_design

https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

http://www.careerride.com/MVC-benefits.aspx

http://stackoverflow.com/questions/3085285/cohesion-coupling

https://500internalservererror.wordpress.com/2009/02/23/what-do-low-coupling-and-high-cohesion-mean-what-does-the-principle-of-encapsulation-mean/

http://thebojan.ninja/2015/04/08/high-cohesion-loose-coupling/