BDSA - Assignment 1

September 15, 2022

Introduction

This .pdf contains Mandatory Assignment-01 for Analysis, Design and Software Architecture made by Amalie Dyrberg Holm (amdh), Frederik Raisa (frai) & Weihao Chen Nyholm-Andersen (weny). The document is split in two parts, in which the first part contains C# and second part contains the Software Engineering exercises.

Git link

https://github.com/mfoman/assignment-01

1 C#

1.1 Generics

The first method in (Fig. 1) takes two generic types upon execution, with the single constraint that T implements whatever functions the interface declares. The second method however, has two constraints: the first being that T matches or derives from the generic type U. The second constraint says that U will then have to implement the interface, IComparable<U>. By extension, T will also implement IComparable in the second method.

In conclusion, this means that the core difference in the two methods is that, in the second one T have to be a child or the same type of U.

```
int GreaterCount<T, U>(IEnumerable<T> items, T x)
    where T : IComparable<T>;

int GreaterCount<T, U>(IEnumerable<T> items, T x)
    where T : U
    where U : IComparable<U>;
```

Figure 1: Generic Code

1.2 Iterators

https://github.com/mfoman/assignment-01/blob/main/Assignment1/Iterators.cs

1.3 Regular Expressions

https://github.com/mfoman/assignment-01/blob/main/Assignment1/RegExpr.cs

2 Software Engineering

Natural language specification of a version control system:

> I want a version control system that records changes to a file or set of files over time so that I can recall specific versions later. This system should work on any kind of files may they contain source code, configuration data, diagrams, binaries, etc.
>
> I want to use such a system to be able to revert selected files back to a previous state, revert the entire project back to a previous state, to compare changes over time, to see who last modified something that might be causing a problem, who introduced an issue and when, etc.

Yellow: Nouns or entities
Blue: Verbs or actions
Green: Extension or of importance

2.1 Exercise 1

2.1.1 Explain in which domain nouns and verbs that you identified are located.

Nouns	Verbs
I [user]	recall(specific versions) revert(selected files, entire project) compare(changes over time)
version control system [system]	records(changes)
changes [changes over time]	[whoModified(user)]
file	
set of files	
any kind of files [source code, configuration data, diagrams, binaries]	contain(any kind of files)
specific versions	
system	
source code	
configuration data	
diagrams	
binaries	
selected files	[revertTo(previous state)]
previous state	
entire project	[revertTo(previous state)]
changes over time	
who [user]	modified(something) introduced(an issue)
something	causing(a problem)
a problem	
an issue	
when	

The grayed out cells means that the entity or action is either not important or will be grouped or is a implicit child of another entity. The use of square brackets ‘[]‘ is for renaming the entity or for grouping other entities as related or children. The use of parenthesis ‘()‘ is for defining a target of an action.

All the nouns and verbs is a part of the problem domain. It’s what defines, shapes and constrains the space where the solution domain can be engineered. It defines the goals which the owner wish to achieve and the context in which the solution will evolve.

2.1.2 The implementation in *libgit2sharp* does neither contain a class FILE nor a class STATE. Explain how that can be when *libgit2sharp* is an implementation of Git which is certainly a version control system as described above.

Libgit2sharp does not have file or state class but does reference filePaths, fileStates, etc. but they're all kept in their binary format as a blob - just like Git.

Git also does not have a entity called file nor state, as they have likely had a different mission statement or concluded that the terms file and state was not the abstract enough so that they wanted to use it for naming-conventions in their problem- or solution-domain. When a file is committed, the system creates a TreeDefinition as a new or existing definition, the file is segmented and a helper class constructs a binary format(blob) which is added to the tree. The trees are saved in the Object Database also referenced as Odb in the code.

2.2 Exercise 2

2.2.1 Categorize each of the two systems into Sommervilles types of applications. Note, the systems may not fall cleanly into a single category.

The application Coronapas App has features of a couple of systems, but from a typical user it is an Interactive transaction-based application and Standalone app. In reality it is built on a System of Systems. The reason for that is that it is not limited to the app, because it has cross-country system capabilities and all these systems needs to be able to talk.

2.2.2 Argue for why you choose certain categories for each system.

The application Git is part standalone-application, because it is independent from its remote counterpart and can work by itself. However it is also a transaction-based application, because it needs to communicate and save the data on a remote server.

2.3 Exercise 3

The Coronapas-App is a specialized product given the circumstances, that many countries needed a solution to monitor, track and control the spread of the ongoing virus. This is developed with a particular global business-problem.

Git is a generic product, because it is a standalone open-market application, though mostly for tech-firms and developers. The core application is open-source and can be extended or switched out for another alternative. It tries to support most developer- and coding needs.

2.4 Exercise 4

Coronapas-App, Git, and the Insulin pump control system will have variance in their balance of their product characteristics. When we talk about the quality of software or product characteristics, we talk about:

- Acceptability: Understandable, usable and compatible with other systems users use.
- Dependability/security: Reliability, security and safety. No physical or economic damage. And safe from malicious intent.
- Efficiency: Not make wasteful use of resources. Should also have responsiveness, fast processing time and optimal resource utilization.
- Maintainability: It should be able to evolve to meet changing requirements and environment.

With the Coronpas-App it should be usable by consumer and vendors as well as interact with other systems than its own. It should be reliant and resilient in the other paths of the system regarding to the amount of incoming and outgoing data from users, vendors and cross-country sources.

It should be safe in terms of personal data - it should not be able for an attacker to claim a consumer- or vendors data. In terms of safety and reliability for the app, it does not do much harm if the app goes down or unable to talk to the system, as the data itself is backed up and protected. At most you just would not be able to eat in a restaurant and etc.

It should be able for a vendor to validate a user within a reasonable time at critical times e.g. at an airport or a concert, such that queues in these venues keep moving. The app and its backed systems would have to be very maintainable due to the cross-communication between many systems and ever changing interfaces that they supply.

For Git, it is very compatible with other systems, extendable and, with it being open-source, very secure and reliant as everybody can contribute with bug fixes and security holes. If you are not a programmer, then it is not the most understandable or even usable of systems. It is easy to use, but hard to master. Git is safe in terms of data, nothing gets deleted and it is always possible to rollback a breaking or even critical change/bug. It is efficient because the object tree points to the same file(blob) and rarely duplicates data. It is also very fast, but not really built for using with large data sets.

While it is trying to stay very maintainable, it is mainly built in the programming-language C, which requires a healthy understanding of the low-level system capabilities and features, so it is very easy to introduce a bug. Though it tries to stay maintainable by using healthy

design-patterns for commands and using rigours pull-request reviews and testing.

The Insulin pump control needs to be designed to the upmost standard, due to the fact that it keeps the user alive and have multiple interconnecting systems. It does not have to be usable by the consumer, in fact it would be best if the user was unable to alter it in anyway as it just needs to be able to monitor and keep them alive.
It has to be the most reliable, secure and safe version it can be - no physical harm can come to the user. It has to be somewhat efficient, meaning that it does not matter, that it can not respond in five seconds or injects 5 seconds too late, but it may not clock drift due to a bug or malfunctioning hardware - if the insulin check gets more and more irregular, at some point it will be too late for the user. Also the pump has limited memory and embedded system needs to efficiently operate with the sensors.
For maintainability it does not matter in terms of insulin dosages, they probably would not change much. They are programmed to mimic the pancreas. However, it should be able to switch out the system or parts of the embedded devices without much effort, as the user would probably outlive the embedded device.

2.5 Exercise 5

2.5.1 Explain why is there likely no architecture for Gitlet

Gitlet.js has a very simple architecture. It is one big single file with the basic functionality of Git. This means that it is not extensible with new commands without editing the main file. This is by design, as it is built with the goal to teach people how git works without reading the source code of Git itself.
The code is heavily documented and typed in a very readable way - almost like pseudo code.

2.5.2 How could you infer the architecture of Git that was depicted in class without any more documentation, i.e., only the available source code?

From the source code you can see that git uses the command pattern, which is a behavioural pattern that allows for easy extension of the program and adding commands without modifications of existing entities. The program is a standalone application which is usable both on client and server side of the organization's version control system.

2.5.3 Gitlet has a particular design that mimics the architecture of Git but that is implemented differently. Can you describe it in words?

Gitlet tries to mimic Git in the way, that it stores objects on the file system and uses the same processes/commands. Other than that Gitlet does not have the same architecture as Git.

2.5.4 Git and Gitlet are designed with respect that different quality attributes (product characteristics). Name some of the most prominent quality attributed that inference the design of each of the two systems.

If we watch Gitlet for what it is - an implementation to teach users about Git, then it has to favor acceptability, which includes being understandable, usable and compatible above all else. It does not favor the other characteristics as much as it is not trying to be production-ready nor maintainable.

On the other hand, Git is entrusted by millions of users and needs to favor maintainability and dependability/security, because if people have a critical bug or breaking change they would need to be able to hotfix it before losing customers and/or money.

2.6 Exercise 6

2.6.1 Kodefejl i Sundhedsplatformen: Fem patienter har fået forkert dosis medicin

1. There was a software update that changed the dosage for patients in another integrated system.

2. To combat software- and human errors, the engineers have to plan and design a reliant system ahead of time and make sure, that there is rigorous and aggressive testing both on the client end of the integrations at build-stage and at the server end at runtime.

3. The proposed solution is to postpone the use of the system until it's proven that there is data integrity across all integrations. It's our recommendation that in this downtime, engineers and specialists are brought together to craft a document or process with testing specifications that ensure the data is not only not breaking but also correct. This ensures that there are multiple protocols before releasing an important software that could cost someone their health or even result in death.

4. The ethical dilemmas regarding engineering software in the medical-sector is more predominant as it influences peoples health. As a generic approach we would abide my the ethical standards set forward by IEEE Code of Ethics.

2.6.2 Softwareproblemer skadede mere end 100 patienter på amerikansk hospital

1. If a doctor did not fill out the form for an examination correctly, then instead of going in a queue, it goes in "unknown queue". What should have happened was that the doctor should have received an error message and the form should not have been sent in the "unknown queue".

2. One solution would be error-handling. So that the doctor gets an error message if the form is not filled out correctly. Another solution would be to inform the doctors/users of what would happen if the form was not filled out correctly, and even suggest what needs to be filled before submitting.

3. According to the article, it was never looked at as an error but as a function that was not informed about. The solution that they use now is to check the "unknown queue" manually. This is a bad idea, because it costs extra to check it manually and is impractical. A better solution would be to handle the error and show an error message so no forms would end up in the "unknown queue". This would ensure that nobody was forgotten in the "unknown queue" and that everything ran naturally with the queues.

4. You risk some people fall way behind in the queue, because let us say that the "unknown queue" only gets checked every other week. And if the examination is critical then there could be risks connected to fall behind in the queue or be completely forgotten about. Plus they have to teach all the new employees that they have to check the queue manually. In worse case scenario a patient could get sicker or die if the system does not work correctly.