Assignment-01-GR09

nicha mhsi brml

September 2022

Generics

In the first method the generic `U` doesn't have any constraints, whereas the generic `T` has to be combarable to itself, meaning it has to implement the interface `IComparable` itself. This also means that `T` and `U` not neccessarily are comparable.

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

The generic `T` has to have a superclass which implements the `IComparable` interface. If the generic `T` itself implements the `IComparable` interface, then we presume that `T` uses its own implementation if the amount of super classes is less than the amount of super classes between `T` and the implemenation of `IComparable`.

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

Exercise 1

Noun/verbs

- Verbs
- Nouns

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.

1. Which domains

Problem domain	Solution domain
Want	Version Control System
Records Changes	File or Set of Files
Recall	Versions
system	Source Code
work	Files
Use	Configuration data
Previous State	Diagrams
Revert	Binaries
Entire Project	Compare
Changes	
Modified	
Causing	
Problem	
Introduced	
Issue	

2. libgit2sharp

Libgit2sharp is a native git implementation that doesn't have the file class and state class since its already implemented in git. It only alters changes to the state and records the files and states controlled by git. (Just a guess)

It could also be that it stores state using text-files with hashes to reference the different git objects like for example branches, and staged files which should be committed. This is also in a very broad sense how git is implemented.

Exercise 2

1. Which type of application

The Coronapas App is mainly a combination of Sommervilles categories 1 (standalone applications) and 8(System of systems).
Git is a standalone application(1) from Sommervilles categories.

2. Argumentation

The reason that the Coronapas App mainly is categories 1 and 8 is because it usually just handles one input and returns a single output. The only connection to other services are to the health data system for each specific country. This is also the argument for why it is of category 8. The app has to handle data from all different countries, each with their own system for health data.

The reason that Git is a standalone application is because it mainly handles files from the computer, either when altering files or executing software. It then relies on other application's services when e.g accessing the cloud.

Exercise 3

Coronapas App

The Coronapas App is a *Customized software product* as Sommerville describes it. This is because the app was requested by an organization, which then paid a company to make the app for them. The app therefore was made for a specific paying customer and was requested for a specific purpose.

Git

The git application is a *generic product* because it has been produced for a wide range of consumers. It is open-source and free which makes it even more accessible for a lot of different costumers.

Exercise 4

Coronapas App

All of the categories for this app are important, although some categories might be more important than others.

One of the things this app is very concerned with, is *security*. Since the system deals with personal data of the population of Denmark, it has to be certain not to leak any of this due to any security problem. The app should also have measures to prevent people from cheating and perhaps presenting a false qr code showing that they aren't sick, although they perhaps are.
Given that it is a population wide application, it should also be *dependable*. During corona a lot of businesses like restaurants were dependent upon the application to work.
Of course *efficiency* is probably important for this app, given that a lot of people would be annoyed there was a high response time. Also a lot of data is handled which is why it could become a problem if the implemented systems aren't fast enough and use too many unnecessary resources.
A lot of resources went in to *maintaining* this application. A lot of people using different types of phones are using it and there are a lot of different systems (including old systems) that has to speak with each other. Therefor problems will inevitably arise, and the system should be implemented in a way that enables future changes and fixes.
Of all the important categories, security is perhaps the most important, because the handling of a lot of personal data.

Git

Dependability is of high importance to Git, since the purpose of the system is to secure older versions of a file or project. If Git fails often at this, it wouldn't

serve the purpose it was made for.
Since Git isn't connected to the internet and isn't build to handle personal sensitive data, security isn't a high priority for Git.
Efficiency is also a quite high priority for Git, as the project it handles can become quite large. This means that Git has to efficiently handle how it stores previous versions of files, projects etc, in order to be useful to the customer.
Since Git is quite a simple system and doesn't need to change significantly, maintainability isn't a high priority. The only real maintenance happening to Git is updates to support new platforms, etc. meaning that Git itself doesn't change significantly.

Insulin-Pump Control System

Dependability is the most important quality attribute since its a critical system that can cause temporary brain to the use if errors appear in the system.

Security also comes close in importance to the dependability because it could potentially threaten the well being of the user if there is exists a security breach.

Efficiency is needed this system to make the right calculations in the short time intervals between insulin injections.

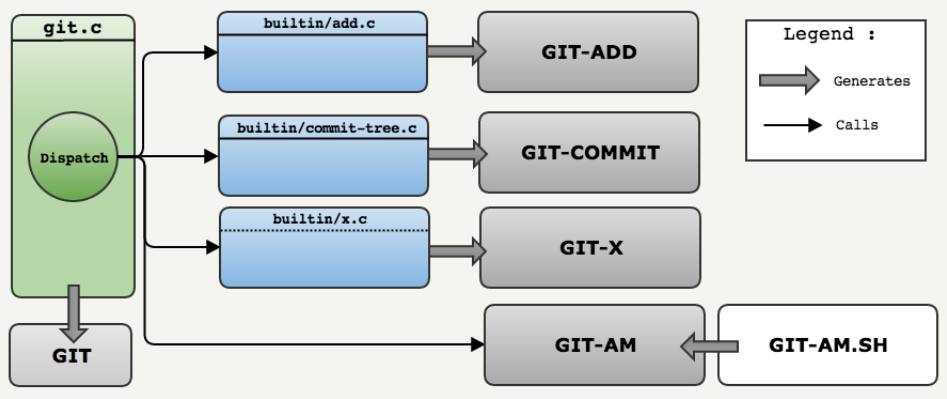
Lastly maintainability would be the least important attribute compared to the others, in the sense that the technology doesn't need to work with other system and only it's own functions are used.

Exercise 5

1. Gitlet's missing architecture

The purpose of Gitlet was simply to show how Git works and there therefore isn't a need for a large, thoroughly thought through architecture since Gitlet likely won't be updated and isn't meant for commercial use. Gitlet is therefore simply a teaching tool to show how Git works, without having to understand the advanced software architecture that Git has.

2. The architecture of Git



We saw a diagram of the architecture that depicts how git generates e.g add.c and commit-trees and shows how those instances call on their respective functions.
By looking at the different folders and the files within, we can infer some of the architecture. For example the folder 'builtin' suggests that the functions withing are the builtin functions in git and that there can be added more functionality by other programs. In the builtin folder, there are functions dealing with committing and cloning.

3. Gitlets design and the difference between Git and Gitlet

Gitlet is used to learn the software architecture of Git and therefore isn't as well designed and doesn't have as advanced software architecture, since there isn't a need for maintenance. Gitlet has different functions for the different commands that can be used. For example a merge, status and clone function.

4. Quality attributes of Git and Gitlet

In terms of quality attributes, they both don't need much security since they both run locally on the computers system.
Acceptability is most important for Gitlet because it functions as a manual for using Git and has to be easy to use for transitioning into Git.
Maintainability is especially not needed for Gitlet since it is only used as a introductory for Git and rarely changes.
Dependability is important for all version control systems, since their purpose is securing files/projects with older versions of them.
Efficiency is also an important quality attribute for all version control systems, since they have to work on large projects.

Exercise 6

1. Reasons for the issues

The cause for the issue in the American hospital's system was because a feature only was half implemented and the medical staff therefore didn't know how to use the system properly, because of bad communication. Furthermore the feature with the "unknown queue" wasn't fully implemented, meaning that it couldn't be used automatically and they therefore had to check it manually.

The cause of the second article was due to lack of personnel who had knowledge of their health system, where the problem persisted for a longer period. Contrary to the first problem, this problem was an unintended bug in the code.

2. Potential solutions

For the issue from the American hospital's software system, it could prompt the user with a message, whenever something is put into the unknown queue, therefore making it clear, when it is used.

The second problem could have been noticed and fixed using unit tests, and more tests of the system before making it available for people to use.

3. Our solutions against theirs

The first article writes about some of the same solutions as us, where an error message would be sent if a patients data was incomplete and got sent to the unknown queue. As software engineers it would be best if that unknown queue got more known, perhaps named as a incomplete queue, where the actors of the system could edit their data and make it easier to discover.

In the second article their answer was to educate the employees regarding their health system, but the better solution would be to create more tests that minimize the chances for errors and lowering costs of mistakes like mentioned in the article.

4. Ethical dilemmas in developing health systems

An obvious ethical dilemma involved in developing such system is setting the limit, for the allowed number of errors. Another dilemma is who to blame, when mistakes eventually happens. Is it the programmer implementing the feature or the project manager not thoroughly controlling everyone involved? Or maybe even the person hiring the programmers not making sure they are competent enough for implementing such a system?