grp-12-assignment-01

adjr, olfw

September 2022

1 C-sharp

1.1 Generics

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

The generic type T of the first GreaterCount-method must be a type that implements the interface IComparable - meaning that T can be compared with itself.

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

The generic type U of the second GreaterCount-method must be a type that implements the interface IComparable - the generic type T must inherit U.

2 Software Engineering

2.1 Exercise 1

Noun/verb quick analysis:

Nouns: (version control system), changes, set, version, files, source code, configuration data, diagrams, binaries, state, project, user, changes, ... problem, issue

Verbs: records, recall, work with different parameters, revert, selected, compare, user data, modify, causing, introduced

Initially everything will be in the problem-domain, considering a solution hasn't been implemented yet. Later on the nouns are used to design the classes of the system. A single noun doesn't necessarily correspond to a single class (e.g. it might not be useful to have a 'Diagram'-class). Most of the verbs will have to be implemented as methods in one way or another though, since almost

all of them are related to ways of handling files or data, either by modifying it or accessing former versions.

Question 2:
The 'file' and 'state' abstractions are ones that exist in the problem-domain - this does not mean that they also need to exist in the solution-domain, other abstractions and implementations can be used instead.

2.2 Exercise 2

Coronapas app: The coronapas app is arguably a mix between an interactive transaction-based application and a system of systems. The app relies on centralised databases for information, which the user accesses when requesting their corona passport. The app itself takes care of the actual data-requests. But this means it accesses several different systems and retrieves data from them. Doing so requires the user to login with a personalized id (nem-id), which can be viewed as a user accessing the system through a client.

Git: Git doesn't fit any of Sommervilles types of applications very well. The closest one would also be the transaction-based application, in the sense that the user uses git to transfer information and data between their local repository and the online git-repository for the given project.

2.3 Exercise 3

Based on Sommerville's definition of software products it is reasonable to define the coronapas-app as a customize software product and Git as a generic software product.

The coronapas-app was commissioned by the Danish government, hence it was developed for a specific customer. It was developed under contract by Netcompany and Trifork. Since the app was developed specifically for the danish government it would fit under Sommervilles definition - *"These are systems that are commissioned by and developed for a particular customer"*. That being said the app is meant for use by the general public, which means the final product is very similar to a generic software product in terms of how it is viewed and used by the general populace.

Git is an open-source software that was developed by Linus Torvalds after he lost access to their former version-control software BitKeeper. It was developed primarily based on a needs basis, and hence it wasn't commissioned by a company/person etc. Unless one counts Torvalds decision that he needed one as a commission in itself. Even if Git is free to use it still fits under Sommerville's definition of a generic software product - *"These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them"*. In this case any customer is able to buy

the product since it is "sold" for free, but this does not change that Git fits the overall definition.

2.4 Exercise 4

Dependability: Based on personal experience the coronapas app has had a fairly high degree of dependability. It has as far as memory serves, been working when required to, without any major hickups. That being said such experience is anecdoctal, and from a realistic point of view, it probably has had the occasional downtime for different reasons, considering it consists of not only the app, but also has to communicate with several different databases. An insulin pump control system would have to effectively have a dependability of a 100% to be considered properly safe to use. Since any possible failure to work as intended could have potential lethal consequences for it's user, it would be safe to assume that such a program would be very thoroughly tested, to ensure the control system does not haphazardly break down and on the rare occasion that it does, such a breakdown should not result in any harm to the user. It would therefor also be safe to assume that in it is more important for the insulin-pump control system to have an even higher degree of dependability than the coronapas app, due to the vital function it serves.

Security: In terms of security one would assume the insulin-pump control system primary consists of a closed network in the sense that it should not need to retrieve nor send data to a database or system outside of the system itself. The coronapas-app on the other hand needs to access several databases containing private patient data. It can be assumed that the app because of that, has been given several security measures to ensure that the data remains private when retrieved from said databases. Since the data is protected by the european GDPR-laws it has to be handled according to such, which means the level of security should be rather high, due to the high potential cost of any data breach. Overall security probably/should be more of a focus point for the coronapas-app than the insulin-pump control system.

Efficiency: Since we do not have access to the source code for neither the coronapas app or the insulinpump control system, it would be difficult to make any concrete comparison of their efficiency. That being said it would be reasonable to assume that both programs have gone through some degree of optimization, to avoid excessive resource usage.

Maintainability: The software of an insulin-pump control system is an embedded system which means it runs as a part of the insulin-pump automation. Depending on how it is implemented it might be difficult to update, without changing the pump itself, but assuming that it is possible in this case, the maintainability of it would highly depend on what needs to be updated. Since the underlying logic and functionality of the system should be based on medical science, it should not need as high a degree of maintainablity as an app, such as the

coronapas app. The app on the other hand needs to be implemented in such a way, that it can quickly be adapted to newer phones/tables, as the software and hardware for these are constantly updated and improved on a monthly/yearly basis.

2.5 Exercise 5

Question 1
Gitlet is essentially just a long script containing a hashmap of keywords to functions. This means it doesn't really have a typical architecture.

Question 2
By inspecting the sourcecode of Git, one could figure out the different classes/components and their connection to each other, and hence infer the over all architecture.

Question 3
Gitlet consists of a long script with a number of functions. These functions maintain a similar functionally to Git, hence mimicking the architecture without actually implementing it.

Question 4
Git was arguably developed with especially acceptability and efficiency in mind. On the other hand the product characteristic that fit Gitlet the most is dependability, since it's essentially a script that one would use locally. This makes it easy to solve potential issues, compared to Git.

2.6 Exercise 6

American article: The issue with the Oracle Cerner-system in the hospital in Washington, was a mix between "nonintuitive" implementation of an unknown queue, along with lack of proper communication that said queue existed. By nonintuitive it is meant, that most people would assume that a program would give them an error message, if they gave it input it could not handle as usual. Instead the program added the input, in this case typically consisting of orders of medical checkups from doctors, would be added to a hidden queue, and hence they would never be handled. A possible solution to this problem, would be to change the way the system handles error in input when doctors order eg. a check-up. If the system is changed to tell the doctors or other healthcare-professionals they have made a mistake in their input, and what said mistake is, they would be able to fix said mistakes and then reattempt to send the order throughout the system. The article explains that the current temporary solution is for someone to manually handle the hidden queue. Long term they the company behind the software is working a fix, that should improve the automation

of the process and implement more adversaries. While it is not entirely clear what that means, it sounds like long term solution proposed in the article isn't trying to remove the hidden queue and instead make it less hidden and easier to handle. Our own solution would simply remove the hidden-queue and just make sure the system implements proper error handling of user input instead. Both propositions can work. Ours would probably require a bit more refactoring of the system, since the way of handling error in user input is changed entirely, but it should result in making sure there are no hidden orders, that never get properly processed. That being said it also would require that the health-care professionals that use the system, got some training in how to give correct user-input, or else they risk getting stuck in the system, where as the solution in the article is based upon improving the way the system currently works, to avoid another case of a hidden queue or similar errors. We would argue that both can work, and which one is best is somewhat dependant on the way the current system is implemented/exactly what changes the they are currently planning to implement further down the line.

Danish article: Based on the article it seems like the changes that was added to Sundhedsplatformen wasn't tested thoroughly enough on how it impacted sub-parts of the system such as the FMK, which resulted in numbers being changed unintentionally. A simple solution would be to run thorough testing on the system, to find the bug that caused the unintentional changes in the numbers and then correct the error. Furthermore making sure that a similar error does not occur again, would be a good idea. This can also be done by improving testing prior to releasing updates to the system. The solution proposed in the article is very similar to ours, it also consists of improving the current testing regime. Besides that it is also suggested that more SP-consultants are hired to help healthcare professionals who might not be as it-proficient. Since the solutions proposed are overall the same, both are fine to recommend. Arguably it makes sense to solve the problem like this, since the problem arrived from a lack of proper testing in the first place.

There are numerable possible ethical dilemmas to be considered, when developing systems like these. They range from patient security, to what can reasonably be expected in terms of IT-knowhow from health-care professionals. Regardless of which of these systems one is developing, making sure that they are working as intended, with an absolute minimum of errors should be highly prioritised, since the consequences of errors can range from minor inconveniences to possible fatal outcomes.