Assignment 01

Benjamin Edon Hagelin, bhag
Emil Boesgaard Nørbjerg, emno
Frederik Høegh Christensen, frhc

September 15, 2022

1 Github Link

https://github.com/Emilfranord/BDSA-assignment-01

2 Generics

First method:

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

What does the type constraint mean for the first method: The first method only have one constraint which is where T : IComparable<T>;. This given constraint defines and sets up a requirement That for GreaterCount it will be using the IComparable Interface. For both methods, we are able to receive two different types, T and U. These two types could be any type, fx integer and float, where it is required to give two types which can be the same. The constraint then adds that T must be comparable with itself. This means that two seperate instances of T needs to be comparable with eachtoher because of the IComparable interface constraint.

Second method:

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

What does the type constraint mean for the second method: The second method have two constraints. The first is where T : U and the second is where U : IComparable<U>;. The two constrainst are in a way connected with eachother. Meaning that T needs to be comparable to U and U needs to be comparable with itself, since U is implementing the IComparable<U> interface. U is required by the constraint to be comparable to itself and since we also have a constraint stating that T implements U, then T is required to be comparable with U.

first: T comparable with T
second: T comparable with U and U comparable with U

3 Exercise 1

Nouns and verbs The following lists are of nouns and verbs, that appear in the specification.

Nouns:

1. version control system
2. changes
3. files
4. set of files
5. time
6. specific versions
7. system
8. source code
9. configuration data
10. diagrams
11. binaries
12. state
13. project
14. changes
15. problem
16. issue

Verbs:

1. want
2. records
3. can recall
4. work
5. revert
6. compare
7. see
8. modify
9. introduce

Domain The majority of the words belong to the technical domain. Several of the nouns only exist in the IT technical domain, such as 'source code', and 'binaries'. Other words come form the technical world in a less direct way.

Some of the verbs that are used, relate to the concept of time. Especially 'revert' and 'recall' belong to the time domain.

All together the words belong to the domain of version control systems.

libgit2sharp While libgit2sharp does not have classes representing 'File' or 'State', it still functions as a version control system. This is possible since the architecture of the program does not demand it. Instead of files, this implementation tracks 'GitObject' instances, be they files or something else. That way libgit2sharp, can still track the changes to files, and other data that needs to be tracked. This design was probably used to abstract the implementation away from specific files and data structures.

In this design there is not a 'state', because it is an emergent property of the system. That is, the state of the system at any given time is defined by the state of the objects, not a single object. This was there is no need for a class to handle the state of files, commits or similar, since that is handled inside the object instances themselves.

4 Exercise 2

First we categorize the git application. We see git to be both a Stand-alone applicatiton and an interactive transaction-based application. The reasoning for these choices is that git can be both depending on how the user will use the program. A user could create a repository with git, only work offline with commits and most of its features, making it stand-alone. A user could technically have a remote section locally on a different part of its computer still making it stand-alone. Seperately a user could use git remotely with a server that it could connect to from multiple devices, making it interactive transaction-based. This would also be used in relations to schools, companies or groups working together where they all need to access the same repository on different devices. Therefore, depending on how and where the user need to interact with the platform, the system can switch between two different application types.

Secondly, the Coronapas app is only seen as interactive transaction-based. A user is unable to properly use the Coronapas app if not connected to the Coronapas app servers in one way or the other. if a user is not connected they are not able to load the qr code to show they are valid or not valid. All processing of data, loading if a user is valid or not etc. is hosted Seperately through database systems and servers.

5 Exercise 3

The Coronapas App, was a software product, that the danish goverment wanted to get made. They wrote a public offering, contining the requrements for the app, and then companies could submit their bids. Therefore the app is a Customized software product, because the purchaser is in charge of the requrements.

On the other hand, Git is a Generic product. This is the case because the actors who use git, have no say in what it does, and what requrements it needs to uphold. It is also avalible for all to get, hence it is generic.

6 Exercise 4

Three systems will be compared, namely the Coronapas App, Git, and a Insulin pump control system.

6.1 Dependability

If the Coronapas App were to go down or fail it would cause some economic damage, to shops and buisnisses that rely on the app. There are other ways of validating ones Coronapas, and as such it would mostly be an inconvenience for the end user. Dependability is of import, but not critical.

Loosing acess to git would make it impossible to track changes to sourcecode. Existing code would not dissapear, and development could still occour during the downtime. Hence, a short time failure or breakdown, would not be problematic but a permanent loss would be. Therefore, short term dependability is not that important for git.

Failure of the Insulin pump control system can be categorised in two broad groups. It can either give too much, or too little insulin. Both types of failure will cause physical damage to the end user, both on the long and short term. Hence it is really importat that the controll system does not fail, and is dependable.

6.2 Security

Clearly all of the systems need to be secure. For the Coronapas App, missing secutity could allow malicious actors to get hold of peoples health data, or allow a malicious actor to have a fake clearence. This would be problematic for the customer, the danish government. For git, it could cause publication or changes to code repos, or other files tracked via it. The Insulin pump control system, too needs to be kept secure from outside interferance. Not doing so could cause the system to deliver too much insulin to the end user, killing them. As such, this system is safety-critical

6.3 Efficiency

It is important that all three systems operate efficiently, and use as few resources and as little time as possible.

The Coronapas App is often run of phones, that have a lot of resources available, and is only used for a short while for each use. So efficiency is less of a consern to the buisniss, since even if it hogs resources, they will be released quickly.

Git has a high requrements for efficiency, because of its usecase. Since git was originally build to facilitate the development of the linux kernel, it would have a lot of users working on the same code repo. As such many people would

need to be kept in synchronization often. Hence git needs to be fast to avoid waittimes, for its users.

In order to make the Insulin pump control system affordable, it needs to be as simple and cheap as possible. Cheaper hardware often has fewer resources to work with. Hence it needs to be efficient with the resources it has. On the other hand, insulin is not a time sentitive drug in the order of seconds. That is to say, it does not matter if the control system takes, 10^{-9} or 10^{-7} seconds to calculate something. It matters that pump is controlled, not how fast.

6.4 Maintainability

Maintainability is only relevant, if there are changing needs of customers. For the Coronapas App, it was a shortlived, approximately 2 years, and had a defined set of requrements from the beginning. Hence, there was not a significant need for maintainability within the app.

Similarly, the Insulin pump control system does not have not a significant need for maintainability. That is, once the pump works and operates correctly, the needs of customers should not change too much. It is not that there will be no changes to the requremnts, but they will be few and far between.

Contrary to those examples, git often has changes to its needs. The source code repo for git has over 67000 commits, and as such it gets changed really often. In principal, there does not need to be a change to the needs of git's customers, but there are. Hence gits business or devs has placed a high prioriry on maintainability

7 Exercise 5

Gitlet is build up in a straight forward way. The architecture of Gitlet is not obvious. All the methods are listed one after another in the source code. There are not multiple classes or any other object orinted structures. Considering the architecture of Git, it's a rahter different. Git is a system build with a bunch of modules that work together for different parts of the program. There are a different files and folders with different parts of git inside of them, connecting the parts together to a fully build system. The design of Gitlet seems to be a boiled down smaller and simpler version of git. A lot of the same features are used, but there is also a lot that got removed on purpose since the developers and the business didn't find them as necessary. Regarding the different quality attributes, the most notisible are efficiency and maintainability. You are quickly able to run a few commands though your terminal to run through several task, that could take a lot of time otherwise. The maintainability in the sense that they always make sure the program is always the same, easy to use and it's never really down or having problems since they don't make all new big updates crashing the program or making it stop working. Gitlet seems to be strongly focused on efficiency and easiness of use of the program. Gitlet too, seems to have a bigger focus on a smaller piece of software with only the most important features, so that it's fast and easy to use.

8 Exercise-6

“Kodefejl i Sundhedsplatformen: Fem patienter har fået forkert dosis medicin”

1. This article describes the reasons for the issue was due to codechanges that lead to unintentional changes in the already implemented system FMK.

2. In order to accomodate the issue described above, it would seem obvious to initiate some procedures that ensures that code changes does not interfere with the dataintegrity.
Practically this would include, but not restricted to: Thorough testing of the changes Peer-Rewieving Maybe running a beta version on a representive test-group

3. The solutions presented above coincides with the somewhat vague solutions described in the article. The article proposed that a procces is developed that ensures that no changes in ”Sundhedsplatformen” can be deployed without ensuring that it does not interfere with other
already integrated systems. In conclusion a combination of solutions described in 2 and 3 would seem appropriate since it accomodates the need described in the article.

“Softwareproblemer skadede mere end 100 patienter på amerikansk hospital”
1. This article pictures does not really describe an issue. It moreso argues that the implementation of the system, specifically the ”Unknown Queue” has been done without much thought on the domain in where its used.

2. In order so solve the issue above the system developers should definately remove the possibillity of anything going into the ”unknown queue”. When any form is filled incorrectly it should be prompted to the user so they can fill it properly. A form should be submittable unless it is filled correctly.

3. The article presents a temporary solution which consists of manually managing the ”unknown queue” which seems to solve the issue for now, but is definately not a long-term solution.
The article presents the long term solution to be more notifications and better automatisation. More notifications coincide with the solution described above. However the concept of the ”unknown queue” seems to be able to cause major faults which is why we, as software engineers, think it should be removed. In conclusion the approach we would recommend would be to work closely with an expert of the field domain in order to understand it properly. In addition the possibillity of forms ending in the unknown queue should be removed.

4. When developing systems in the healthcare systems some dilemmas come to light. It is of the essence to think about flaws in the software as being potentially life threatening.
If you develop a tool for a firm to help them track employer performance, if anything goes wrong, noone dies as a direct result of that. However in these two cases it might just have been that someone had suffered servere physical nuisance.
When developing software for the healtcare system, you should always be aware that consequences of flaws might not only result in loss of money.