

Assignment 01

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

September 14, 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 constraint 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 from 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 3

The Coronapas App, was a software product, that the danish government wanted to get made. They wrote a public offering, containing the requirements 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 requirements.

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 requirements it needs to uphold. It is also available for all to get, hence it is generic.

5 Exercise 4

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

5.1 Dependability

If the Coronapas App were to go down or fail it would cause some economic damage, to shops and buisnesses 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 access to git would make it impossible to track changes to sourcecode. Existing code would not disappear, and development could still occur 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 important that the control system does not fail, and is dependable.

5.2 Security

Clearly all of the systems need to be secure. For the Coronapas App, missing security could allow malicious actors to get hold of people's health data, or allow a malicious actor to have a fake clearance. 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 interference. 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

5.3 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 on 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 concern to the business, since even if it hogs resources, they will be released quickly.

Git has high requirements for efficiency, because of its usecase. Since git was originally built 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 sensitive 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.

5.4 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 requirements 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 requirements, 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 git's business or devs has placed a high priority on maintainability