Generic Type Constraints

```
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>;
```

The first type constraint specifies that the generic T must implement the Comparable interface, and be able to be compared to another T.

In the second method the type constraint for T is derived from U, which implements the comparable interface with itself. Therefore, U is comparable to itself and T is also comparable to U.

Exercise 1.

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.

Nouns	Verbs
version	want
control	record
system	can
change	recall
File	should
time	may
version	contain
system	want
work	be
kind	revert
source code	selected
configuration data	revert
diagram	compare
binarie	see
use	modified
system	might
state	be
project	causing
something	introduced
problem	

issue	
set	

Exercise 1.

1.1: The nouns and verbs are of the problem domain. They all describe problems and wishes that can be solved and fulfilled using software. These are the descriptions of which the solution will be applied to.

1.2: The solution domain to the problem domain can vary wildly, and many different implementations can satisfy the wishes presented in the problem domain. Depending on the software, file structure, classes, etc. can be very different. An OOP language might create objects for each noun, whereas a non-OOP would not create objects at all.

Exercise 2.

In class we discussed so far the Coronapas App and Git as cases for software systems.

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

- Coronapas App
  - System of systems
  - Stand-alone App
  - Interactive transaction-based applications

- GIT
  - Stand-alone App
  - Batch processing systems

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

- Coronapas App

  The Coronapas app fits into multiple types of applications. Most of all the Cooranpas app is a System of systems because it depends on a lot of different systems. It depends on one system for operations like authentication, and another system for things like producing a corona certification. The Coronapas app also carries some of the specifications as a standalone app, but only when the initial setup is done. After the initial setup, the app works without a network connection for showing a person's corona certificate. The Coronapas app also has some of the specifications for "Interactive transaction-based applications". This is Because a lot of the operations are done on a remote computer, and the apps then display the outputs of these operations.

- GIT

  GIT is a generic application incorporated into a lot of other systems like GitHub. Git is a stand-alone application that works on a computer without any connection to other computers or systems. Git can also be categorized as a "Batch processing systems" because it is build to handle a big batches of data.

Exercise 3.

Sommerville describes that there are two kinds of software products. Describe for the Coronapas App and Git what kind of software product they are and provide arguments for your believes.

- Coronapas App

  The Coronapas app is a "Customized (or bespoke) software". The app is built for the danish healthcare system. The app is made to meet some specific requirements set by the healthcare system.

- GIT

  GIT is a generic application incorporated into a lot of other systems like GitHub. GIT is built as a stand-alone system initially created in 2005. Git is an open-source system and can be used on a computer with no connection to the internet or other systems.

Exercise 4.

- *Sommerville discusses quality of professional software, non-functional quality attributes, or product characteristics. Compare the Coronapas App, Git, and the Insulin pump control system (see SE chap 1.3.1) with respect to the quality attributes dependability, security, efficiency, and maintainability. Do they all share the same characteristics with regards to these quality attributes or are these of varying importance to the three systems? Give examples for each of the three systems with regards to each if the quality attributes above.*

All three systems need to think of all these attributes but to a varying degree.

in the case of acceptability, they all need to make sure that their software is acceptable to the type of users, but in different ways. Git is made for developers, so they know their software will be used by users with a somewhat technical understanding, this affects the way the software is explained to the user. Differently the insulin pump control system must be used by a bigger variety of users and its use has to be easily understandable.

When talking about the dependability and security concerns of the three systems, they all must think about this in different aspects. In the case of the insulin pump, it is a life and death situation if the system performs differently than intended, so it needs to be extremely dependable. The coronapas app works with its users' personal information, so the app must be careful with its users' data security.

All the systems have to be as efficient as possible. GIT must be fast when saving different versions for the user or to find the change in files when a user wants to merge their work. The corona pas must make a fast search when looking for one user's approved certificate in millions of other users, and it also has to be memory efficient in saving all of this data. And the Insulin pump probably must make some calculations efficiently when giving its user the correct dose of medication. So, Efficiency means a great deal for all of the applications.

Maintenance means a great deal for all software applications both for their competitiveness and because the greatest cost of an app is Maintenance. This means that the developers should be able to make changes, fix bugs, and implement new features in an easy way.

Exercise 5.

Inspect the implementations Gitlet and Git a bit more thoroughly than in class.

5.1. Explain why is there likely no architecture for Gitlet.

Gitlet is made to explain how Git works and give examples of how Git's method could be implemented. Gitlet is only used for this and it isn't a real software system, and that's probably why the no architecture for Gitlet.

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?

If you wanted to attempt to figure out the architecture of a program you had no other knowledge of, other than the source code, you could look for a starting point. When the program is activated, what happens? Is there a "main" function, if so what does it start and where does it lead? Furthermore, looking at the file structure of the program would give you some pointers as to how it works.

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 is made so that its source code is easily understandable for the person who reads the source code. This means that the implementations of all the methods could have been implemented more efficient than it is (it probably is more efficient in Git) but it is meant to be easy to read and not efficient when running the program.

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

The biggest difference in how Git and Gitlet systems quality attributes come from there target customers, for example, Gitlet code is made to be easily understandable for the reader of the source code, and Git is made to be as efficient for the person using their software. So the 2 systems have completely different quality goals for their system.

Exercise 6

Look at the following two cases of issues with health care software systems:

- *"Softwareproblemer skadede mere end 100 patienter på amerikansk hospital"*
- *"Kodefejl i Sundhedsplatformen: Fem patienter har fået forkert dosis medicin"*

6.1. Based on the articles, describe the reasons that caused the respective issues.

In the first article, the problem is that the system lacks in its “Acceptability” quality, the users don’t have a good understanding of how to use it. The problem could come from a lack of testing with the users of the program. This problem can occur even though the system is thoroughly tested by the developers, because it's clear that the system behaves as intended, but is it not tested that the user understands this crucial behavior of the program.

In the second article, the problem is harder to identify, but think it originates in the doctor's trust in the program, so the system fails in its “Dependability” quality. The problem can come from either not a good enough testing of the system and its calculations, or the education of its user's interaction with the program.

6.2. Describe potential solutions to the problem.

A solution to the problem from the first article could be an adjustment to the program's design. so instead of accepting the users' inputs straight away, and putting the wrong input into an unknown queue, implement a checker that checks the input and remove the feature of the unknown queue.

A solution for the problem in the second article could be further testing their calculations for medicine ordination. The solution could also be to implement a pop-up reminder for the doctor, to remind them to check the calculation made on behalf of their patient.

6.3. Compare your solutions to the proposed solutions in the articles, which one would you as a software engineer recommend? Argue for your recommendations.

For the first article, we would recommend my own solution, to the problem. This is because we think it is bad design when a program accepted a user's input without checking them first or at least alerting them to the wrong inputs.

In the second article, we would recommend them to use the solutions mentioned in the article and also implement the pop-up for the doctor to check the calculated information, this is because with an infinite amount of testing you’re never able to assure that a program always works, and with something as important as medicine ordination the doctor should always check the information.

6.4. Discuss ethical dilemmas in case you were developing either of these health care systems.

There are a lot of ethical dilemmas when designing software for health care systems because it handles patients’ well-being and their personal information, and the software can have a big impact on people's lives. One of these ethical dilemmas could be the security of the user's data, how do we assure that our user’s personal data is safe from cyberattacks? Another dilemma could be the need for creating an unbiased algorithm in the software, this meaning that the developer keeps their own opinion out of the algorithm's design.