BDSA 2022 - Assignment 1

Johan Richardt-Bunke, Laurits Brok Pedersen og Laurits Kure

September 14, 2022

Link to Github repo: https://github.com/lauritsbrok/assignment-01

1 C Sharp

COMPARE:

```
   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 upper GreaterCount only allows for key comparisons. This is stated by the constraint statement. Here T must implement IComparable¡T¿, while there are no constraints on U.

The lower allows for value comparisons, which in turn allows for key comparisons as well. This is also stated by the constraint statements. Here U must implement IComparable¡U¿, while T must implement whatever is implemented by U.

2 Software engineering

2.1 Exercise 1

Nouns	**Verbs**
Version Control System	records (changes to a file / set of files)
file	
source code	
configuration data	
diagrams	
binaries	
system	recall (specific versions)
	revert (selected files)
	revert (entire project)
	compare (changes)
	see (who last modified something)
state	

1. They are located in the solution domain, since the purpose of the analysis is to extract solution design.
2. The noun/verb analysis provides a framework for designing system architecture, but not a blueprint. Not all nouns become classes, and not all verbs become methods. libgit2sharp implements a branch class, and a commit class. These are responsible solving the above problems. Having a file class is not an apparent necessity, since files are provided by the user. The system just needs to handle these files.

2.2 Exercise 2

2.2.1 Coronapas App:

Interactive Transaction-based Application.
It is a bigger system, often administered on a server, with lots of connecting devices. In this example it's a cloud-based solution.

2.2.2 Git:

It fits multiple categories:
- System of Systems: It connects multiple seperate products, IDEs, unifying the code.
- Data Collection and Analysis System: It collects a lot of data (code) and organizes it. Furthermore, it can be a cloud based system, which is a requirement.
- Stand Alone Application Cloud-Based is not necessary to use git. Git can run locally on a computer and still provide version control.

2.3 Exercise 3

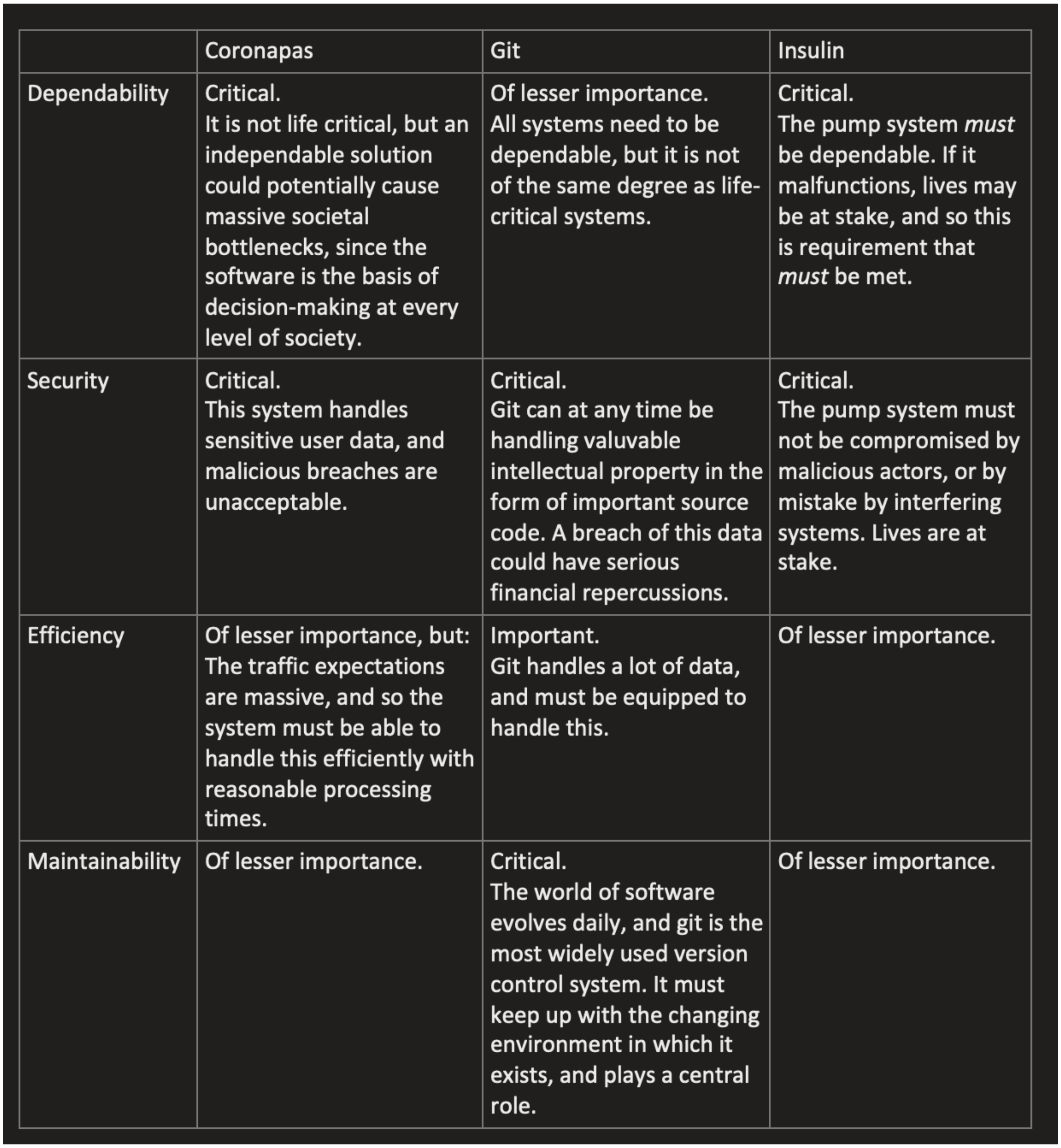
2.3.1 Generic products: Git

Git is a stand-alone system designed and produced to be used and or sold on the open market, to any costumer or user interested in the software.

2.3.2 Customized (or bespoke) software: Coronapas App

This app was commisioned by a particular customer, and designed to specification.

2.4 Exercise 4



	Coronapas	Git	Insulin
Dependability	Critical. It is not life critical, but an independable solution could potentially cause massive societal bottlenecks, since the software is the basis of decision-making at every level of society.	Of lesser importance. All systems need to be dependable, but it is not of the same degree as life-critical systems.	Critical. The pump system *must* be dependable. If it malfunctions, lives may be at stake, and so this is requirement that *must* be met.
Security	Critical. This system handles sensitive user data, and malicious breaches are unacceptable.	Critical. Git can at any time be handling valuvable intellectual property in the form of important source code. A breach of this data could have serious financial repercussions.	Critical. The pump system must not be compromised by malicious actors, or by mistake by interfering systems. Lives are at stake.
Efficiency	Of lesser importance, but: The traffic expectations are massive, and so the system must be able to handle this efficiently with reasonable processing times.	Important. Git handles a lot of data, and must be equipped to handle this.	Of lesser importance.
Maintainability	Of lesser importance.	Critical. The world of software evolves daily, and git is the most widely used version control system. It must keep up with the changing environment in which it exists, and plays a central role.	Of lesser importance.

Figure 1: Exercise 4

2.5 Exercise 5

1. Gitlet is simply documentation for Git. It helps the user understand what happens under the hood when executing Git commands. Even though documentation is important and need to be well written, it normally doesn't require it's own architecture. For Gitlet to need a architecture, it would

need to be it's own system.

2. When checking out the Git repository, it has a folder named "Documentation" full of well written documentation for all classes and features. It also contains how to's to user who are in need of help using Git. Without the documentation, you would have to look inside each classes and make sense of the structure, which is nearly impossible when looking at complex systems like Git. Git succeeds in being so well documented, that no further documentation is needed.

3. Gitlet shows the overall structure of executing Git commands. Where Gitlet has it's difference is documentation being very readable and shown right beside each comment. If Git decided to include all documentation directly inside the code, it would be very difficult to make sense of what methods does. It would also be take a long time to change comments in case code get's recaptured. This is why documentation has it's own section in the repository.

4. Gitlet is created to help new users understand how Git works under the hood. Git is focused on being very specific and easy to use. After getting used to Git commands, you don't need an explanation each time. Git is there designed for fast and efficient daily use. Gitlet is designed to help new users understand git further.

2.6 Exercise 6

2.6.1 Softwareproblemer skadede mere end 100 patienter på amerikansk hospital

1. In the case of the American hospital, the problem occurred because of a feature called the "Unknown queue". This feature was meant as a place for forms that were not filled out correctly, to be stores, so that presumably it could get looked over again. However the problem was two-fold;
   Despite the fact that this was a feature that was ordered by the customer, it was not something that the day-to-day users was taught how to use .
   And if the users made a mistake filling out the form, or purposely left out important fields unfilled. The form would go to this "Unknown queue", but to user themselves were not warned/told by the program that this was where the form was going, and they therefore assumed it just went to where it was supposed to go.

2. A solution would be to simply not have an "unknown queue" and instead give an error when filling out the form, if it is not filled out correctly. This would solve the issue, but it seems likely that there are times where they would want to not fill out the form correctly, which was why the feature was requested in the first place.
   So instead another solution that still keeps the "unknown queue" would that a pop-up/warning-message was shown to the user, when a form is about to be sent to the unknown queue instead of the intended replicant. But to avoid that this queue is still ignored, there should be some kind of notification, to the relevant user, that there are unfilled forms in this "Unknown queue", so that they are not forgotten.

3. The proposed solution involved that the users would be better trained in this feature.
   This isn't terrible, but since the "Unknown queue" is a feature that would be used irregularly, it seems very likely that the forms in this queue would still be forgotten or at least filled out slower than they should be. So without some kind of persistent notification, problems will still occur.
   Another problem with this solution is that it is a human dependent solution, that requires everyone that uses to system, and everyone who will in the future use this system, to be trained in how to use

this unintuitive feature. This is of course very time consuming, which has the effect that it is also expensive for the customer. A software solution like the one we proposed would be a way better, for this reason.

2.6.2 Kodefejl i Sundhedsplatformen: Fem patienter har fået forkert dosis medicin

1. Changes to the code of “Sundhedsplatformen”, had unforeseen problems with it’s integrated system FMK. These changes had the unfortune effect that patients got prescribed the incorrect doses of their medicine.

2. Ideally this sort of thing shouldn’t be allowed to happen when working with something as important as peoples medical records, so in a perfect world one would assume that what was needed was more thorough testing before any changes to the program is released. And maybe that would’ve solved this particular case. But it’s practically impossible to prove that a system is infallible, so even if they spent 10 times as many resources on making sure everything is working as intended, how sure can you really be that bugs aren’t going to spill through the cracks?

3. The way they will avoid incidents like this in the future, is to make it illegal to make changes to the systems functionality or data foundation, without making sure that these changes won’t cause known problems with the integrated systems used.
   This is a weak solution, as it really isn’t a solution. If it is found that the systems developers intentionally launched a version that compromised peoples personal information, then that obviously has to change. Maybe by incentivising spending more time and manpower on testing, like we suggested. But it still just boils down to, “please discover and fix all bugs before launch” which still doesn’t really account for the possibility of bugs that will only be found after release.

4. Since we are creating systems for healthcare professionals, that will have a direct impact on saving lives, making sure that the system is intuitive, idiot-proof, and bug-proof, are more important than ever, as a mistake here would result in not only financial damage, but also human damage.