BDSA - Assignment01

ehel, bath, jakst

September 2022

Link to GitHub: https://github.com/Emilia-Victoria/Assignment01

Generics

The two classes, here referred to as GreaterCount and GreaterCount2, have a few differences, due to type constraints. First off, GreaterCount has only one type constraint, which is "where T : IComparable¡T¿". This means that type T, must either be or implement the interface IComparable¡T¿, where T is its own type. This means it must have a compareTo() method, where it can be compared to other objects of the same class. For our method "GreaterCount¡T, U¿(IEnumerable¡T¿ items, T x)", this means that the objects in our Enumerable list "items" must be of, and be able to be compared to, the given type T. Since our x must also be of, and be able to be compared to, the given type T, we can now safely implement the compareTo method in our execution, as shown in the code. The type U has no bearing on anything, as it is not used elsewhere in the method signature. Therefore, we can pass any object type, including the slim Generic type. And as seen in the tests it changes nothing. In our GreaterCount2 method, we have two type constraints, which are "where T : U" and "where U : IComparable¡U¿". First off, "where T : U" means that T must either be, or be a subclass of, type U. So whatever type U we give, T must be a subclass of it, or the same type. Second, "where U : IComparable¡U¿" means the same thing as "where T : IComparable¡T¿", but for the type U. Since T must be of type, or be a subclass of, U, and subclasses must implement all interfaces of the parent, this means that T must also implement IComparable¡U¿, and therefore be able to be compared to type U. In practice, this means that the given type T must be comparable to the given type U, which must either be the same type as T, or a parent of T. Therefore, T must be comparable to itself, and possibly any parent of T we might give as U. All in all, we can still safely use our compareTo method in our execution.

Exercise 1

The nouns and the verbs are located within the problem domain. The noun/verb analysis works solely within the problem domain, as it only analyses the descrip-

Nouns	Verbs
File	Records (changes)
Set (of files)	Revert (file/project back)
Version	Compare(changes)
Source code	See(who made modifications)
Configuration data	Recall (versions)
Binary	
Diagram	
State	
Project	
Change	
Problem	
Change	
Issue	

tion of what the system or particular task should do.

When it comes to libgit2sharp, there are many ways to design software structure. With something as simple as a differently worded problem description you could end up with a different set of verbs and nouns.
Additionally the problem domain is separate from the solution domain - in this case, the problem domain specifies that there must be files in a certain state. It is possible to design this without classes that are expressly based on the noun/verb analysis. This is simply dependent on the implementation developed in the solution domain. It could be that the responsibilities of the class File could be split between several differently named classes for instance, or that everything is contained within one singular script.

Exercise 2

Git is an interactive transaction-based application, as it works as a cloud-based service that stores data that can can be accessed and updated through git-commands. However, since some git-commands are usable without internet access, such as commit, it could be argued it is at least in some small way a stand-alone application.

The Coronapas App is an interactive transaction-based application, since it needs to retrieve information from remote databases in order to function logically.

Exercise 3

Git is a Generic software as it was not commissioned by anyone, and is freely usable by everyone.

The Coronapas app is a customized software, as it was specifically commissioned by the Danish government, and then developed specifically for Danish citizens. However, since the app is made for a general population, this is a special case with some Generic software elements.

Exercise 4

Dependability: Determining how important dependability is for a system can be done by viewing the consequences of the system not working. For Git, this would mean that software developers around the world would be unable to share their work with others, perhaps leading to them being unable to work, or use less efficient methods, which can lead to additional and unnecessary costs. A system failure for the Coronapass App would mean that people couldn't partake in various activities under the pandemic, such as going to work. This leads to general inconvenience and could be costly. A system failure for the insulin pump can mean death or at least severe health risks. Therefore, a focus on dependability is arguably the most important for insulin pumps.

Security: Security is not as important for the insulin pump, as the software is an embedded system, and thus can't be remotely accessed. Safety, on the other hand, is extremely important, as a system failure can lead to critical health conditions. For the Coronapas app, there is a lot of personal data involved, which means that security must be prioritized. For Git, being able to mess with someones-, or some companies' repository could cause massive headaches and delays. However, problems like these could likely be overcome via hard copies on a local machine. So very important for Git, but for Coronapas, security holes would be critical.

Efficiency: Efficiency is extremely important for the insulin pump, as the sensor and the pump controller needs immediate response times and precise information to function properly. Efficiency for Git is also an important factor. The users would try to find faster alternative version control systems if Git had unacceptably slow response times. Furthermore, because Git is a popular system with many users, efficient resource management is key to managing the resources of millions of users. The Coronapas App has only one real function - to show whether or not the coronapass is valid. This includes only a few tasks with a more flexible standard for acceptable response times. That being said, since the coronapass had a huge societal importance, fast response times would be preferable. At the same time, important information based upon millions of users needs good data management.

Maintainability: The insulin pump is a finished product and should not require software updates but future iterations of insulin pumps could be built upon the preexisting software if it is maintainable. As for the Coronapas App

and Git, maintainability is important. They both need to be able to evolve to changing needs and an expansive user base, in both cases numbering in the millions. Furthermore, Git sometimes introduces new relevant features, and the development of these features should be smooth and efficient.

Exercise 5

1: The entirety of Gitlet seems to only consist of one singular script. An architecture requires communication between different parts of the project, and this quality seems to be downplayed here.

2: By inspecting the directories and checking which files communicate with which - where are objects stored and how? Which kinds of files are created? This will be evident by close inspection and looking at when certain functions are called.

3: Everything that Gitlet shares with Git has been centralized into a single script. The object of Gitlet is to explore Git functions in a more readable way. Every feature/module in Git has been implemented as a function in Gitlet.

4: Gitlet has explicitly been designed with readability and the acceptability product characteristic in mind. The Git repository is very difficult to parse but Gitlet is not.
The most important product characteristic for Git is dependability and security, as that is the prime responsibility for a version control system. However, since Git is a fully fledged version control system that is used by millions, every product characteristic is important in some way. Gitlet, on the other hand, is simply an exploration of how Git works, and have less need for efficiency or security.

Exercise 6

1: In the article about the American hospital, the issue is caused due to a poor design choice regarding a feature that none of its users were made aware of.

The issue in the second article is caused due to changes made to the system not being integrated properly with other parts of the system.

2 and 3: To fix the first issue the unknown queue could be removed and instead the users should not be able to submit forms if they are not filled out properly. Compared with manually observing the queue this would be a better option as the user would be made aware whenever a formula is not filled out properly and would have to handle it before submitting it.

As for the issue with Sundhedsplatformen, making sure to test the system extensively before integrating changes would prevent a lot of issues. For better

maintainability the system should be documented properly. Structuring the program so that it has a low coupling would also make it easier to change things in the program without affecting the rest of the system too much. Temporarily implementing a more strict work process can be good, however, this must lead to a better attitude regarding testing and upholding GDPR in the long run.
4: An important ethical dilemma when it comes to developing a health care system is whether you have the means to provide a good enough product as flaws in the system could be fatal. Constraints on budget and time may rise many dilemmas regarding prioritization and push the line for when something is "good enough".