Assignment 1

Clara Lyngeraa, Sigurd Hermund Holm og Mikkel Bistrup Andersen

September 2022

Contents

1 Generics

1.1 Explain in your own words what the type constraints mean for both methods.

The first method does not have any type constraints for U, essentially meaning that U can be any type. T however must be a type that implements $ICompara-ble¡T¿$, meaning the T must be comparable to itself. The second option is more constrained because it sort of binds two generics together. In the statement *where T : U*, the type argument for the constraint on T must derive from the constraint on U

Which can result in compile time error, if T tries to implement something that U implements.

2 Exercise 1

Use the noun/verb technique from "Objects First with Java: A Practical Introduction Using BlueJ" chapter 15, to analyze the given description.

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 control system	*records* (changes)
	recall (versions)
	revert (files)
	revert (project)
	compare (changes)
file	*contains* (source code, configuration data, diagram, etc.)
	has (states)
project	*has* (states)
state	
version	
change	

2.1 Explain in which domain nouns and verbs that you identified are located.

The domain of the identified nouns and verbs is an organisation engaged in collaborative file editing in one or more projects, possibly a software development team. The need for recording and comparing changes as well as being able to revert files or an entire project to a previous state, suggests that the projects this organisation is engaged with span over an extended period of time.

2.2 The implementation in libgit2sharp does neither contain a class *File* nor a class *State*. Explain how that can be when *libgit2sharp* is an implementation of Git which is certainly a version control system as described above.

The intention of the noun/verb technique is not to give a one-to-one correspondence between classes and nouns or methods and verbs. Rather, it aims

to give a rough understanding of the structure of the program, by revealing the *initial classes*. The noun/verb technique takes place in the *analysis* phase of developing software which should put the developer(s) on the path towards implementation, though not necessarily reveal the actual future implemented classes. Some of the classes of the final program may not be present in in the description and vice versa. Perhaps in the development of *libgit2sharp* maybe later in the analysis phase or in the design phase, the *File* class seemed superfluous as the files that were being kept track of, were already being handled by the file-system and all the program needed was the path to these files. And possibly the *State* class was proved too large and complex and was thus broken into several classes.

3 Exercise 2

3.1 Coronapas App

The Coronapas app closely resembles a *Stand-alone application*. The Coronapas app functions mostly on it's own being able to display, scan and generate a Coronapas once initialized, but this also brings up an important part where the Coronapas app does not resemble a stand-alone application. A session has to be initialized for it to work as a stand-alone application, this is because it need to authenticate a user to get their vaccination status, but once this has been done the app works as a stand-alone application being able to run completely of the network.

3.2 Git

Git is hard to place into one of Sommerville's eight application types, but it resembles *Interactive transaction-based applications* the most. Git is a Version Control System (VCS) and functions almost entirely in the cloud. Git is basically an application that stores your programs and keeps track of the multiple versions, branches etc. and it can be accessed either by command-line or by a web-based application like Github. This closely resembles other interactive applications since it manages and incorporate large amounts of data trough an interactive access point.

4 Exercise 3

4.1 Coronapas App

The Coronapas app is a type of *Customized Software* since is was a public tender by the Danish government. They stated their requirements and then the software companies who "won" the public tender built an application customized to meet those requirements. This is almost the textbook description of customized software.

4.2 Git

Git is a *Generic Product* since Git is a free open-source system and not developed for any specific company or person. Git was specifically designed for version control and that is all it does just like other generic applications such as accounting systems or library systems.

5 Exercise 4

Correction: We have assumed that there is an error in the assignment since the actual quality attributes according to Sommerville are Acceptability, Dependability & Security, Efficiency and Maintainability. We have solved the exercise with the quality attributes from the book.

5.1 Coronapas App

The Coronapas App clearly follows the quality attribute of acceptability. It does exactly as promised and nothing more making is incredibly user-friendly and easy to understand. When is comes to dependability and security the Coronapas App is not as straight forward. The app is not very dependable, but luckily it is not the only way to attain a valid coronapas and the app is somewhat usable without a network connection, when it comes to security the app is extremely secure piggybacking on the very secure NemID/MitID system making it almost impossible to gain access to temper with the app. The app also generates one off QR-codes to avoid fakes and has a features that prevents screenshots of QR-codes.

The Coronapas App is very efficient, this is mostly because of the very simple function it serves. It uses very few resources and requires little time to execute it's function. Though the same thing cannot be said for Maintainability as the Danish government did not see it necessary to maintain the app as coronavirus was dying down. The App was only maintained to the extend that it could function, so only critical errors in security and functionality were fixed.

5.2 Git

Git is a very complex system used by a lot of different people so saying whether or not it follow the attribute of acceptability is hard, but personally we think it does live up to this attribute. Git is a version control system and it does this really well especially in conjunction with some visual tool like Github. It allows users to work on the same system and prevents them from just pushing code to the newest version without considering the effect (merge conflicts). Git promises to do nothing more and it doesn't.

Git is not very secure because it was not built for security but for collaboration. Therefore Git offers very little in the form of security if any at all, the security of Git highly depends on which Git service you use (Github etc.). But when it comes to dependability Git is in a class of it's own. Git is decentralized so loss of data is extremely unlikely and Git can do nothing without human commands so it is much more likely that Git fails do to human error rather than an error in Git itself. Git is also quite old and this means it has had a lot of time to grow and handle different errors, therefore Git is incredibly resilient.

Git is a very smart system and that means it tries to cut corners wherever possible to be as efficient as possible. This means that Git stores a whole repo under a single hash and creates diffs for older versions this makes Git very efficient since it doesn't store the same three over and over, furthermore Git also checks if files are clanged or not, if for example you push a repo where you only changed a single file Git doesn't store all the other files it just stores the changed files an points to previous version of the unchanged files. This makes Git very efficient because it always tries to be barebones and only store what is needed.

Git is a open source and free system and is maintained by a lot of different people. This makes Git really versatile when it comes to maintaining the system because lots of people can maintain different parts of the systems and can get feedback from lot of different users. A lot of the people making Git are Git-users themselves and this mean they have a deep insight into what works and what doesn't when it comes to Git. The major disadvantage of this kind of maintenance is that there is no guarantee that a part of the system will be maintained by the same person and new people may not be probably introduced to the system before they have to maintain it.

5.3 Insulin Pump

The Insulin pump follows the attribute of acceptability, especially because the alternative is worse and the insulin pump keeps track of a lot of information and displays is on simple displays. It gives the user access to all the information they need in a nice simple way. The same goes for dependability the insulin pump has three fail-safes should it fail. It logs the amount of insulin given, it checks if the pump actually gives the correct amount of insulin and it sounds an alarm if it fails to do these check letting the user know. This makes the system very dependable since it checks itself and warns the user if something goes wrong. The insulin pump is also very safe, this is mostly because of the fact that it is a *embedded control system* and not connected to a network, the system is totally self-reliant. The insulin pump is also quite effective doing only what is minimally required of the system and nothing more. The simplicity of the system also makes it very efficient. Although the system follows some the quality attributes well the same cannot be said for maintainability. The insulin pump can only be maintained by physically interacting with it since it has no network connection and being such an important system for the lives of millions the system must be maintained by professionals, this makes it very hard to effectively maintain an insulin pump and that is why they are simplistic and build to last. To make the requirements for maintainability less.

5.4 Comparison

The three different systems are all quite different they are all in a different application type. This makes comparing them very difficult, but what we can do is use the different quality attributes to make educated guesses on what the

different systems prioritises. For example it becomes apparent that while Git prioritises maintenance the Insulin pump and Corona app does not. The opposite can be said for acceptability the Corona App and Insulin pump both come with easy-to-use displays while Git needs a third party tool to have a user-friendly interface, even though they all do what is required of them, they prioritize differently when it comes to the user interface. The last big comparasion we can make is that of security, Git has none while the Coronapas App and Insulin pump both prioritize security heavily for different reasons just like Git has its own reason to have a low level of security.

6 Exercise 5

Gitlet likely has to architecture because unlike Git is doesn't really store repo's as blobs, tree and commits. It uses linked list and doesn't use hashing like Git. Gitlet does it all by tracking the path of a linked list.

Without documentation, inferring the architecture of a piece of software as large Git just from the source code is an endeavour that requires time and focus. Fundementally, what we want to conceive is a map of how the different files and parts of the program relate to each other, that is, which parts *refers* to which - like a rough UML-diagram. Thereby it is possible to create a sort of hierarchical structure, where those most referenced must constitute a larger part of the overall architecture of the program. One approach to this task is to start by finding what one might consider to be one of the more important (that is, more referenced) files of the program (ex. *git.c*) and follow from there the different references to other files.

Git can different Gitobjects like trees that store locations (like directories) and blob that store content of files it also has commit that can be made to blobs or trees and trees and reference subtrees. Gitlet mimics this behavior by using a linked list (trees) of files (blobs) to do version control the head of the linked list is the commit. Gitlet also does something really smart by being able to split a linked list by having two different elements point to the same element effectively creating a new branch, Gitlet can then merge two branches by keeping track of the HEAD of each branch and when a merge is done it tells the user the changes needed to merge the current HEAD to the other branch's HEAD. This very closely mimics the architecture of Git, but using no objects and no extra layers in the architecture.

Gitlet was designed to be extremely simple and efficient therefore it is clear that Gitlet focuses a lot on the acceptability and efficiency attributes. Git of course almost completely ignores acceptability focusing almost entirely on dependability and efficiency. Git also requires a lot of maintenance whereas Gitlet requires almost none. This also makes a lot of sense considering that Gitlet was made to be a simpler and more understandable version of Git.

7 Exercise 6

Both of the problems described in the articles where caused by bad communication and contracts between the customer and supplier. We are generally a fan of the sentence: "Make software for what you have instead of what you don't have". In both of these cases software was made not for the doctors and nurses who actually used the systems but for people with more technical knowledge. The solution to this problem is not by doing more training or hiring more SP-consultant, that would be a waste of money, but instead they should have looked at the end-users and done some user-insight in the form of user interviews and the like to gain an understanding of the end-users knowledge and needs. This way both the customer and supplier can actually make and receive a system that works for the end-users. As a software engineer it is partly your job to make sure the customer is aware and knowledgeable of what they are receiving from their contract and as part of the development process it is important to iterate on the contract and test the systems on the end-users to make sure you are making a system that works for them.

As the developer/supplier of these systems you have an ethical duty to inform the customer of the development process and the customer has a ethical duty to make sure the contract is factual or changed if the situation changes. It is the shared responsibility of both parties to make sure the system is being developed probably because while the supplier actually develops the system the customer has to keep the supplier updated on the situation and assist the supplier so the system can be tested and be as good as it can be.