Assignment 1 BDSA

frepe, adrka

September 16th 2022

Exercise 1

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

Nouns: Red, Verbs: Blue
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.

In the table below we’ve tried to assign each noun to a class in our system. Some nouns have been merged with others (e.g. *system* has been merged with *version control system*) as they represent the same thing. The verbs are shown attached to the nouns they refer to. This isn’t perfect but it gives a good early overview of the system.

Nouns	**Verbs**
version control system	records (changes) recalls (versions of files) revert (files) compare (changes)
change	stores (versions of files, time, users)
file	contains (source code, configuration data, diagrams, binaries)
version	stores (file)
state	stores (changes)
user (who)	modifies (files) causes (problems & issues)
time	
problem/issue	stores (time)
source code	
configuration data	
diagrams	
binaries	

Exercise 2

The Coronapas App we've primarily classified as *interactive transaction-based applications.*

We've done this, as it's essentially just access to a central data-store, with information about every individuals current health.
The app wouldn't work without access to the data-store, as it serves no further purpose then fetching and authenticating health-data
Furthermore this information is updated by transactions, by a different part of the system, every time a test result has been submitted.

For git we've instead chosen to classify it as a *standalone application*, as you run git locally on your machine and create local branches.

You could make a case for also calling this an interactive transaction-based application, as you typically would also store your code in a remote repository. But seeing as git could in theory run entirely locally, we thought it to more closely resemble a standalone application.

Exercise 3

The two kinds of software products as described by Sommerville:

"*1. Generic products These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them. Examples of this type of product include apps for mobile devices, software for PCs such as databases, word processors, drawing packages, and project management tools. This kind of software also includes "vertical" applications designed for a specific market such as library information systems, accounting systems, or systems for maintaining dental records.*

2. Customized (or bespoke) software These are systems that are commissioned by and developed for a particular customer. A software contractor designs and implements the software especially for that customer. Examples of this type of software include control systems for electronic devices, systems written to support a particular business process, and air traffic control systems."
(Software Engineering P. 20-21)

Both the Coronapas App and Git are generic products, as they are not customized to each specific user, but rather available as a single product for everyone on the internet. The customers of the Coronapas App are every person who lives in Denmark, and the app is the same for everyone and is therefore a generic product. The customers of Git are probably software developers and such, and once again the product is pretty much the same for everyone and is therefore a generic product as well.

Exercise 4

While the *dependability, security, maintainability* and *efficiency* are important for almost every piece of software, there is some variance in how much each factor is weighted for different projects.

From the example projects, you can show some interesting differences.
The Coronapas app is a great example of an application requiring a great amount of focus on security.
This is because it handles a high amount of personal data, and having it leak, or shown to the wrong users would be a huge breach of GDPR law.
Furthermore the Coronapas App was used to grant access to a heap of social-activities and businesses, and having a security flaw that allowed users to display spoofed data would be detrimental to the app (This part also ties neatly into why it should be dependable)

In direct contrast git, being a standalone app, won't have much connection with the outside world, and therefore doesn't have to prioritize security as highly.
Even when having your code in a remote repository, git shouldn't be responsible for the security of your code. That responsibility should be left to the server that runs the remote repository instead.

While Coronapas has a high focus on security, and Git a correspondingly low focus, the opposite can be said when it comes to maintainability
The Coronapas App was build as a solution to a relatively short term problem, under a tight time constraint. This meant that developers could take shortcuts, to fasten development time in exchange for worse maintainability. It's interesting to note however, that the purpose of the app could be generalised to be used for more than this specific case, at which point a lack of maintainability would be more critical than it is now.
Git on the other hand was build as a long term version control system designed primarily for development of the linux kernel. Additionally git is open-source, this means that everyone has the option to read the code, and contribute. Writing unmaintainable code would go directly against the spirit of open source as fewer people would be able to comprehend and add to the code.

There are few cases where the dependability of a piece of software is more important than the insulin pump control system.
In the case of a system failure, you need to have fail-safe measures in check to ensure that the user doesn't receive a potentially fatal dose of insulin.
With the health of the users at stake, the system should be as dependable as can be, and should be tested for every single potentially breaking scenario.

Exercise 5

Exercise 6

Despite the theme of the cases being similar and the suggested solution being alike, the two problems stem from wildly different sources.
In the "Sundhedsplatformen" case, the problem is caused by the integration of code changes, causing users to receive incorrect data.
These types of problems would usually be caught by automatic tests, but weren't in this case. This would mean the obvious solution is a change in the test process, be it unit tests or integration tests.
In the article they suggest hiring consultants to aid the hospital staff work with the platform. It seems indicative of poor design, if a system meant to be used by a demographic primarily consisting of people not in tech-jobs, requires "teachers" to help you work with the application.
While we do think there are certain cases where having somebody *guide* you in a given piece of software could be beneficial, it wouldn't fix the issue at hand, and certainly wouldn't have circumvented it, in a way like a better test process would.

This solution was also proposed in the other article regarding the "Oracle Cerner"-system. The system was shipped with a feature called "unknown queue" which would place patient journals with incomplete data in said queue, to be handled later. This wasn't completely understood by the end user, who

instead thought the patient journal where filed correctly, and moved on, causing some patients to wait indefinitely for examinations and test.
The problem as described in the article, seems like a clear case of specifications of the project being unclear, and further communication between developers and product owners failing. It would however, also have been clearer design if the application had clearly indicated that information was missing from the journal, causing it to be placing in the unknown queue.

As previously mentioned, the article suggested that the solution was giving training in the software to the staff of the hospital. While it makes more sense in this case than in the "Sundhedsplatformen" case, as it is a feature misunderstood by the end-user, it still could have been entirely avoided by writing clearer product specifications and clearer communication between project manager and product owner (perhaps also consulting a test-person in the target demographic)