Subject: Final Reflection-Secure Software Development
University: Essex University
Place & date: Remote; 03-09-2022
Name: Nicolas Haas
Study: Computer Science
ePortfolio: https://nicha1986.github.io/ePortfolio/
Supervisor: Dr. Cathryn Peoples
Word Count: 1082

Final Reflection-Secure Software Development

Secure Software Development (SSD) seems to be a simple term and most newcomers to IT possibly think that it is about security in the development of an application. It is true, that security is an essential factor in SSD but I had to learn that it is also here only complementary. In this sense, a developer should get to the next level and more think like a Software Engineer who thinks about system (code) complexity/simplicity, testing and security.

Starting with code/system complexity, I can suggest to read (Ferrer et al., 2013) which gives a short introduction to different measures of code complexity. Personally, I can fairly say that I now (since the last few weeks) often use the McCabe's Cyclomatic Complexity measure as a reference to test the simplicity/complexity of my code. This measure for example helps yourself to entirely rethink your system/code when you think too complex because in this measure complexity increases when your system is too strongly connected (= too many edges). Why can this be? Firstly, it can be that the system simply is strongly connected. Secondly, your code simply starts from a weak point of view. As an example of the latter, think about the different types of triangles and their properties. When you want to include all the properties, the system becomes huge and some properties are shared (180° rule of inner angle) and some are different (equilateral vs. rectangular). A simpler solution could be to rethink your approach entirely and think about the Pythagorean theorem. If there is a triangle that is not rectangular then you break it into two pieces which are rectangular. This at the end leads to the use of only one rule, the Pythagorean theorem. So, the McCabe's measure now often helps me personally to choose my approach because, even when the system is difficult to fragment, it gives me at least the hint that I should probably first put more effort in analysing my system (Perhaps a

Data Engineer can help to see patterns). Nonetheless, I also had to learn that the McCabe's measure has its limitations because it uses logic as reference for evaluating your code complexity. For example, the simple lines of code are not incorporated. In other words, writing a 1000-line code compared to a 100-line code makes no difference (but it does, in reading and writing code).

And what about testing complexity? Generally, when your code is complex, it will also be difficult to test it sufficiently (Pillai, 2017, Chapter 3) (Watson et al., 1996). But disregarding code complexity for a moment, I generally learned that even if you are an experienced programmer, it can always be helpful to give your code to a third party for testing (= black-box testing). Sometimes, it is already sufficient to use third party libraries for testing but it will probably not be enough when your system (not necessarily your code) gets more complex. Hence, it is generally a good idea to keep always in mind that "high cohesion and low coupling" are important for fast error tracking. Because cohesion refers to the responsibility of objects (class, functions, etc.) it is important not overdoing it within an object. So, one would assume to give for example a certain class only one responsibility/task (to achieve high cohesion) and use connections (edges) between modules only when it is really needed. What I learned here is that in case of a code-error, this should then lead to the fact that only some portions of the modules are breaking down while others (which are not related to the module where the error initiates) are still alive. Why can this be helpful? In case of an error message, seeing that at least some modules are still working already gives me personally the hint that these modules are ok and not the reason of the breakdown (Pillai, 2017, Chapter 2). Unfortunately, the testing capabilities of a Software Specialist do not end in thinking about the functionality of a software, it also includes performance, reusability, readability, etc.

Getting now the bridge to security, one has to mention that there is always some kind of trade-off between performance (testing) and security. For example, security prefers the use of local rather than global variables but performance does not (Remember, a local variable always has to be first created and at the end of its use it has to be destroyed what leads to a lot of temporary memory usage). However, security is essential and should probably have priority in most choices that a Software Engineer has to make. Why? Because if your application is under attack, it is very likely that your database is it too (except the database is stored decentralized). This makes you (or your company) vulnerable against legal requirements (Data Protection rules) but also against the trust of your users. What can you do while developing an API? A general advice that I learned, take a look at the web page of OWASP which presents everyone with a list of current threats in cyberspace and think about solutions. After that, it is also important to choose a strategy of protection. There is ongoing research and depending on my personal concerns (for example authorization), I sometimes have to focus more on one particular strategy (in case of authorization security perhaps “Least Privilege” rather than “Defense in Depth”).

To conclude SSD in terms of the three mentioned main points (complexity, testing and security), complexity let you think about the structure of your code, testing about the correctness, performance, reusability and readability of your code and security cares about the safety of your software and the safety of your database/data flow. However, because there are trade-offs between these different parts, I learned that two general tips are helpful. Firstly, do your research. Secondly, do it not on your own. Why? Because developing a reliable API is a huge task. You have to think about a lot and there is a lot to do. Hence, different people have different strengths. Don’t be shy to use these strengths, even it leads to increasing cost and time. I can

personally say that I also want to attend more team work seminars in the future which are focused on web development. Team work helps you to keep going on a constantly moving path and better skills in web development can help you in several directions (for example, aside from learning specific frameworks, it also helps you to become a better and faster programmer).

Reference List:

Pillai, A. B (2017). Software Architecture with Python. Birmingham, UK. Packt Publishing Ltd.

Ferrer, J., Francisco, C. & Enrique, A. (2012) Estimating Software Testing Complexity. Information and Software Technology.

Watson, A. H. & McCabe, T. J. (1996) Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric. US Department of Commerce Technology Administration National Institute of Standards and Technology. Available from: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication500-235.pdf [Accessed 9 September 2021].