Reflective Essay

Introduction

In this essay I will go over what have been the three main topics for me in the course TTM4115. Throughout the course we have covered many interesting topics, and I have learned from all of them, yet I believe some of them have impacted me more in different ways. The first topic is user requirements, often underrated; the second is state machines, the center of our development; and the last one is teamwork and course format, which shocked me from the beginning.

User requirements

Capturing and synthesizing requirements and use cases looks so simple and important that people tend to forget about it or overlook it. It might seem contradictory, how is it possible that if it's so important, it's overlooked? The fact is, requirements and use cases are foundational to every software development project (or any other project, really), and they grow and change throughout the process. You discuss them so frequently that you believe you know them perfectly and there is no need to revisit them. However, you might have understood them wrong, they might change, clients might change their mind, the implementation might reveal gaps, different understandings of the same use case come into place... This process works a bit as a broken phone. Every time you mention the requirements, small modifications occur in everyone's understanding. Through our implementation (E-scooter project) we could already see this happening, and we didn't even have a client who set them, we could make them ourselves.

User requirements are critical, yet they are often underestimated. Even in my own group, I felt like there were very few people referring to them or pushing for a stronger focus on them. Most just wanted to move on to the sequence diagrams and state machines. And this makes total sense, as in the course these were more emphasized in the course. Yet I found myself asking, "how are we going to implement a use case without defining the requirements and use case itself?". We were encouraged to define 3 use cases, almost symbolically, but it was hard for me to visualize the implementation without defining all of them.

I firmly believe that having good set of requirements in the form of use cases, with their proper DFDs and descriptions in table form is the best way of having a team aligned. While some may argue that focusing so much on them will only slow the process, we could see that through the development, especially with a large group where not everyone is present everyday, we noticed people would lose track of how the system should look and the current set of valid requirements. Everyone would start to get confused, trying to discern what is still valid among all the ideas discussed in meetings and later discarded during the

design and implementation phases. Add stakeholders to the equation, and everything is even more complex. So avoiding spending time and effort on them only delays the problem and will evolve into something more complex to approach.

To avoid this, it is crucial to follow the correct flow. After brainstorming or collecting the requirements from the client, each use case should be defined with a clear objective, including its preconditions (what must hold true before it begins), main success scenario, alternative flows, and postconditions (what is guaranteed after completion). All use cases should be documented following the same format in a Requirements Specification Document, which serves as the central reference for the team and must be continuously updated throughout the project. Additionally, Data Flow Diagrams (DFDs) and other supporting models should be used to visualize how data and responsibilities move across the system. This structured approach ensures traceability, reduces ambiguity, and helps all stakeholders maintain a shared and consistent understanding of the system's expected behavior.

State machines

The core of this course, for me, were state machines. As a computer science student, I learned about them a while ago, yet I had never worked in such an integrated and deep way with them. It was really interesting to see how they became the center of the whole development and implementation of our project from the very beginning. I lost track of how many changes and iterations we did on them. They seemed so simple at first, but every time we thought the state machine was done, someone realized that we needed more functionality. Did this imply a new state? A transition? An entry function? An entirely new state machine? In retrospective, once we finished all state machines and we looked at them, we wondered how could we have spent so much time figuring them out, when the resulting state machine was so simple and described the system so well. It was almost frustrating to see how much effort went into them when the resulting product was so small, and we still hadn't even started with the implementation. I will admit I was surprised when I realized that the system **was** the state machines. Once described them in STMPY, and defining a couple of more functions, the system worked.

So, how does this happen? State machines may seem simple at first, but their true complexity lies in identifying and defining the system's states and transitions correctly. State machines are a powerful tool for system design because they allow for an iterative and structured approach to modeling system behavior. They help by breaking down complex systems into manageable states and transitions, making it easier to track and adapt to changes as development progresses. But for it to work and not make you waste time, I believe the following procedure help: start with a simple model that captures the core states and transitions, and update it simultaneously to the requirements (never forget the requirements). Each time a new one is added, assess whether it introduces a new state, transition, or action. This ensures that the model evolves with the system. And then periodically, take a step back. Try to forget your current understanding of the requirements and state machine and revisit them. Is this still the best solution to the

current set of requirements? If so, great! Now everybody in the team has the same understanding, and starting or updating the implementation should be very easy and straightforward, as it directly reflects the state machines.

Course format

And lastly, I would like to reflect on how the course was structured. I am not sure if this is something to reflect on, as probably there are other courses with the same format, but I would like to do so either way. As an Erasmus student I had never encountered a course with this format. Lectures weren't lectures, they were time for you to do exercises that were not even graded. Time to spend with other students with the same level of knowledge as you. Initially, this is how I saw it, at least. I doubted that a structure like this could work. I expected people to slack off, to not come to the lectures, to use the time for other graded tasks. But the course content seemed useful and I wanted to give it a chance.

I was immensely surprised. Maybe I was lucky with my teammates, or maybe the course content is naturally fits this format. Or maybe it just works. Yes, lectures weren't lectures, they were time to tackle real problems and exercises that mirrored real-world challenges. Time to spend and learn from other students with different levels of knowledge and experiences that would promote a learning environment and were eager to share their knowledge. It allowed us to adapt and learn how to work together on small projects that would eventually lead to the team creating something bigger, a real solution (though partially implemented) to a real problem.

Throughout my study career I have sometimes questioned if things I am learning or studying are useful. It is easy for me to sometimes miss the point, buried between mathematical formulas or low-level programming details. Yet I was pleasantly surprised when this course (the one that seemed that wouldn't work) was actually one of the ones where I saw a lot of utility from what we were learning.

Conclusion

In summary, I believe user requirements and use cases were underrated and should be given more attention in the course and/or when designing. Mastering the state machines and implementing them more often into software and systems development is a good practice. And I also believe this was very enriching course. The teamwork, project and course content were very interesting.

References

Wiegers, K., & Beatty, J. (2013). *Software Requirements* (3rd Edition)

Sommerville, Ian (2011). *Software Engineering* (10th edition)

State Machines preparation materials from course TTM4115, NTNU.