Portfolio I

Theodor Baur

15/12/2021

—

CM1301: Principles, Tools, and Techniques for Secure Software Engineering

Section 1: Reflection on Class Material

3 weeks ago I decided to read "Requirements Engineering: A Roadmap" (Nuseibeh and Easterbrook, 2000) which gave me insight into certain details about requirements, including difficulties, different processes and qualities that requirements should have. The paper is mainly divided into a few key sections including: Eliciting requirements to form initial requirements by data collection techniques as well as best trying to identify the fundemental goals of the software being written; modelling to give requirements for different purposes, for example buisiness goals, or security goals; the process of communicating said requirements; and finally some fundemental problems with writing requirements. In particular, the section on eliciting requirements has been relevant as it goes over some approaches to finding initial requirements and states that often the most important goals is to find out what problem must be specifically solved.
My old approach to coding would be to essentially tackle the problem as a whole, with no elicited requirements, however this would mean that if the code did not perform properly I would often have to spend time changing the essential structure of the code as I was not solving the right problem. However for my website which requires a python script to either display dates verbosely or numberically I decided to clearly define requirements that would solve my given problem and noticed 2 benefits, firstly time was saved as when troubleshooting, I was confident that the essential structure was fine and just need bug fixing whereas if I had not defined requirements I would need to do both, and secondly that time was saved as in the process of writing requirements as I had already abstracted the problem into clear goals meaning that I had a clearer roadmap to making the code. For example, instead of just starting the code with the idea that I needed a verbose function I noted down that the year number should always remain constant, the day number should have the last digit read, then assorted into a list of suffixes. Therefore I have learned the importance and benefits of writing good requirements.

Section 2: Independent Research

Declarative programming is a different style of programming to traditional imperative programming. Instead of using step by step logical commands to describe how a program should work, but not what outcome is wanted like in imperative programming, declarative programming describes what outcome is wanted but not necessarily how the program should work.
A good example of this difference would be the process of trying to double each number in a list. For imperative programming each step could be outlined: create a new list; create a for loop for each item in list; set index to 0 and make sure it stops after the index reaches the number of items in the list; inside the loop double the numbers and push it into an new list; then finally print the new list. For declerative programming all that is required is a return statement with a map function inside creating a new list with each map value multipled by 2. The difference between these two programs is that nowhere in the imperative is the outcome described, and in the declarative program the process is not described in as much detail, but the outcome is more apparent. (Nxstack, 2017)
"Practical Advantages of Declarative Programming" (Lloyd, 1994) outlines some of the advantages of declarative programming that I share the same perspective on, they mostly stem from the fact that declarative programming is more readable. Firstly that declarative programming has made programming more accesible to ordinary people. This is because ordinary people generally are not immediately interested in low level details which are involved in more imperative programs but they are more interested in logic, which is clearer to read in declarative programs. Secondly that declarative programming has increased programmer productivity. This is because often there is less of the programmer's time spent, and maintaining and upgrading code can be done in less time as well if done in a higher level declarative way. Thirdly it enables the logic and the control (how a logical component is executed) to be separated, which makes a program easier to transform. It is worth noting that imperative programs can be made more efficient due to their lower level, so whilst there are upsides to declarative programs, they also often sacrifice some level of efficency.

Appendix

- Nuseibeh, B., Easterbrook, S. 2000, *Requirements Engineering: A Roadmap*, Available at: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.4288&rep=rep1&type=pdf [Accessed: 15 December 2021].
- Lloyd, J. 1994, *Practical Advantages of Declarative Programming*, Availiable at: https://www.programmazionelogica.it/wp-content/uploads/2015/12/GP1994-I-000-031.pdf [Accessed: 15 December 2021].
- Nxstack, S. 2017, *Lecture 17 - Imperative vs Declarative Programming*, Available at: https://www.youtube.com/watch?v=Mt7KnvnNGfk [Accessed: 15 December 2021]., (4.06-6.26)