**Reflection**

It was easy to get a clear picture of how the program’s modules should look like, as well as what kind of functions they should have. In addition, deciding whether elements needed to be static, thus only manipulated by functions in their own module was relatively straightforward. The same can be said about const values’ need to be protected from any kind of alternations e.g. the “current\_user” in the library’s interface in this case. Overall, developing each module independently before combining them together was the easiest part. This way I had a clear view of what was running properly or had to be reviewed.

Having no previous contact with the concept, developing tests using unity was a little difficult because even though I knew I had to make more detailed and advanced test functions, I often found myself unable to figure out how to create a particular test, or what was it that needed testing.

Finally, the most difficult part for me was having to deal with memory allocation for pointers, storing and loading data to and from text files and creating the final library interface. Even though I managed to make the functions inside each module to execute as intended when it came down to connecting all the elements together in the last parts of my code, complicated and inexplicable errors began to arise and in spite of trying for hours, plus searching on the internet for possible explanations, I could not figure out why they occurred let alone how to fix them as most of them concerned segmentation faults due to memory leaks I could not locate.

To conclude, even though my modules were satisfying before the implementation of the library’s interface, when it all came down to assembling them into one program, the strong dependencies of functions from different files to one another along with memory leaks due to bad memory manipulation, despite the enormous amount of effort, lead to a problematic final outcome.