For my software engineering artifact, I selected an old project I did to learn C#, a sudoku program that allows for working on a sudoku puzzle as well as checking a sudoku solution for validity. I had written this for my own learning, but also because I frequently enjoy doing these puzzles and wanted something that I could use for my own entertainment. In looking at this artifact, I found that my initial development several years ago was incomplete, and I found a lot of areas for improvement. The goal of my code review and improvements here was to find the areas where work could be done and then work on those that would be simpler fixes, while keeping the more complicated work documented for future improvements. Overall, this was the most major undertaking of any of my projects in this course, and that I perhaps underestimated what I was dealing with since I didn’t fully recall the state this project was in. This was a learning experience in estimating and preparation that I will take with me in my career.

There were two primary areas that improvements were needed on this system: the GUI front end system and the sudoku solving/checking algorithms. I found that the GUI mostly needed tweaks to make everything lineup and be viewed better and more consistent, but the sudoku algorithm side was a lot more complicated. First, the algorithm that I had written was recursive, which always adds complexity to understanding and completion. Second, the algorithm didn’t quite work right in the first place, so instead of just optimizing the system, it was an exercise in troubleshooting why it wasn’t working correctly. It was interesting to note how far I’ve come in my algorithms since 2014 when this was originally written, and also reminded me that it’s important to document how much of a project I’ve finished if I’m going to host it somewhere (it was on my personal Github repo) and work on it later. Being able to quickly find pain points and identify what to fix is important, and documentation is a great way to make this simple.

When I did my changes, I was also working on syntax and style issues, which were prevalent throughout this code. Similar to the algorithms project that I worked on for this course, the original used a set of coding style standards that I now find to be wasteful, such as putting all curly braces on individual lines, keeping debug comments everywhere to toggle them on and off, making one line functions entirely all in the same line as the declarative statement, blocking if statements into one line and trying to make things as compact as possible. I find now that making things compact often sacrifices readability, so making things a bit more consistent and in longer for is better overall. This also emphasizes the importance of making code as modular and simple as possible to prevent code bloat of having massive files. I found that when I cleaned up my code, I also found unnecessary areas and places where things could be further modularized.

Overall this project has been a demonstration in how difficult it can sometimes be to clean up old (or legacy) code. While I made quite a few changes, it wasn’t really anywhere near close to as much as I wanted to by the time the project needed to be submitted. With plenty to still work on, I’ll find this to be something I keep improving for months in the future. I think the comments I made and organization that I’ve done on this has steered in in the right direction where I can keep building on this after my degree. It’s good to have personal projects to work on, particularly as employers continue to ask for them as part of the interviewing process.