Jerry Perez

December 10th, 2020

CS4233 – Design Document

**Escape**

Alpha:

Alpha was an interesting project for me, and one that I to be completely honest, did not look forward to. Why? Because I feel like I understood the concepts conceptually but never really enough to implement them in a way that was meaningful to me. I knew that this was a big chunk of the grade, and I was overwhelmed by the Developers Guide. It had so much information and I had such a hard time just understanding where to get started. Which of the SOLID principles can I use? Should I use a different design pattern? There were so many questions that were going through my head even before starting this assignment. However, once I picked up, I wasn’t able to get off. I won’t ruin the surprise because it will be coming towards the end… but one thing that this class taught me was to just get started. Where? Testing.

So, what did I do? Overwhelmed… I started writing tests that got anticipated values from the egc, and eventually it got to the point where I was creating classes that implemented interfaces in order to get various classes to communicate with each other. This was expressed in Module 5, where we ought to program to an interface and not to an implementation. I cannot tell you how beneficial this was for me. It started with SquareCoordinate and Coordinate, and eventually led to being extremely beneficial in Beta as well.

Another principle that I tried to keep near and dear to me was to favor object composition instead of inheritance, which would refer to creation as opposed to recycling. Without knowing what CodeMR was at that moment, sticking to this principle seemed to give me fairly good A picture containing application

Description automatically generatedresults when it came down to cohesion, which is exactly what this principle is meant to obtain.

My intention was to follow this principle closely since we learned about inheritance breaking encapsulation and creating code bloat, so, I wanted to make sure that everything that was to be implemented had unique methods. At that point in time there weren’t many classes that I would group together into one “grouping” so to say, and so the methods among those classes are expected to be fairly cohesive.

One principle that I don’t feel like I followed all too well despite wanting to, was the Open/Closed Principle which had that classes would be open for extension but closed for modification. I will explain why in the next section.

Beta:

Beta was a fun one, and I cannot tell you how many hours of sleep that I lost from it haha. However, if it is one iteration that taught me the most, I would say that it was this one. This iteration is where we had to implement another coordinate type, a different board, add descriptors to pieces, and develop algorithms for almost everything ☹ . It was a long battle that I eventually got through, but it wasn’t without vigor for sure.

I said that I didn’t adhere to the Open/Closed principle because of Beta, which is where this principle would have come in handy. This principle says that classes should be open for extension but closed for modification, and the only thing I felt like I did in Beta was modify classes in order to give the user exactly what they need. However, even throughout Beta, I programmed to an interface, whether that be through the board interface that I created or the coordinate interface that was needed for the coordinates.

Another principle that I believe I did a great job of using was the Interface Segregation Principle, which states that no client should depend upon methods that it does not use. I’d like to believe, until proven otherwise, that my code does a tremendous job of using only the methods that are needed, and none more. In fact, programming to an interface helped me achieve just that, and Gamma was nothing different. I continued to code the necessary classes using the methods that were in my interface, and the methods inside of my interface were only methods that would be absolutely necessary to make my implementation meaningful.

Lastly, before I roll over into Gamma, I’d also like to mention that I kept the dependency inversion principle in mind throughout as well, focusing on keeping classes coupled and trying my hardest not to involve classes that weren’t needed. Running a CodeMR on Beta shows just that!

Graphical user interface, application

Description automatically generated

One talking point that I have been anticipating is this huge spike in complexity, which stems from the Coordinate classes and the Board classes that handle the bulk of the functionality for the program. However, I didn’t know about this up until Gamma!

Gamma:

Gamma was interesting because it was the first portion of the project where I worked knowing what I needed to improve on (complexity mainly). Not functionality wise, but code wise. Beta had not been graded yet and it was my first time trying CodeMR, and of course, the only project to run it on was on Beta. Attached above is the graph for the initial CodeMR run on my Beta code, which shows an incredible spike in complexity. Gamma increases that complexity with the added logic inside of the board classes that constantly run through checks for end of game, keeps track of player piece counts, their scores, and what attributes they possess. In hindsight, a solution that I was able to think of that I would have approached completely differently knowing what I know now. No, I don’t consider this as “bad”, but I also do not see this as “great”. I believe that this could have been avoided and still could, if the logic portions of my board classes were extracted to a service class that handled it instead of the board.

It’s a little frustrating because I felt really good as to where my code was, and I honestly thought it wasn’t complex at all! It just goes to show that re-factoring is ridiculously important, but it is not going to save the complexity of a program. One thing that I carried in my head was “re-factoring”, “re-factoring”, “re-factoring”, but it didn’t save me from being too complex, or at least I don’t think it did.

Looking back at it, I also almost regret not making service classes at first and making them work at that point in time. I was a little too concerned on getting the project done with the way would be the fastest so that I would be able to revisit it and fix it later. However, I will use this as a segway into the next portion of this paper… I have learned to not wait, and just do it. Sometimes it takes starting over, and sometimes it takes deleting a class, removing a method, adding an interface, a package, whatever it may be, in order to truly obtain the design that works best. I, however, really do believe that knowing what I know now, it will be different in the real world.

All in all, this class has been a wild ride, and one that I could not have been more grateful for. This was honestly the first class I have come across at WPI that scared me, and I almost set myself up for failure with my mindset. However, after weathering the storm, if it’s one thing that I can say this class has done better than ANY class at WPI, its teach. I have learned more in this class than I have in the past few years, and I am a pretty good student. This class has not only showed me tools that are useful for coding, but principles that will last a lifetime and coding practices that will never be forgotten. I am truly shocked at how much I have taken away from this class and I am extremely proud of the work I have been able to produce with being enrolled in this class. Professor Pollice’s ability to clearly communicate topics, requirements, and teach skills that I will never forget is something that I am truly grateful to have had.