Name: Duy Nguyen

Class: 321

Final report

I believed that for the assignment 1, we learned about the first rule of Agan “Understand the System”. The reason was that we had to do a whole research about the dominion game. We learned about how to use each card in the game and also a little about how to play the game. Additionally, I believed that we also learned about the code by refactoring it as we had to read the whole code. I learned about some basic functions of the codes through this process. Through refactoring the code, I realized how bad the dominion code was written because it was really high coupling and was not cohesive at all. By running the Makefile, I was able to know what was given to me, so I didn’t need to do any extra steps. By adding errors into the code, I was able to be things about what possible errors that people used to make. That gave me a different perspective while looking at the code. The reason was that most people didn’t want to make any errors and broke their codes, however, I got the change to things of the ways people could unintentionally break their code through the process of making errors in the code.

For the second assignment, we learned about unit test. This showed us the “divide and conquer” technique. By assigning us to make the unit test for the functions and the card effects, I believed Dr. Groce wanted to show us how we could divide the dominion code into smaller sections. We also learned about gcov, which would let us see what portions of the dominion code that we covered after we ran the program. Based on the coverage report (Assignment 2/coverage2.txt), we have:

The coverage of shuffle function was 15.64%

The coverage of supplyCount function was 19.76%

The coverage of whoseTurn function was 20.27%

The coverage of numHandCard function was 16.15%

The coverage of Smithy card effect was 22.51%

The coverage of Council Room card effect was 23.71%

The coverage of Adventure card effect was 23.37%

The coverage of Great Hall card effect was 22.51%

Based on these values, we could see that the card effect test covered more than the function test. The reasons for the difference in the coverage was the card effect test used multiple function of the dominion program. Hence, I learned that if I wanted to increase the coverage of a program, I needed to write a test that implement multiple functions on that program. A way to do that was random testing, which we learn in the third assignment.

On the third assignment, we learned how to used random function in the unit test. I believed this allowed us to know about the coverage of the function. This was the result I got from the assignment:

The coverage of the 1st card test program went from 23.71% to 24.91% because of the random.

The coverage of the 2nd card test program went from 23.37% to 25.60% because of the random.

I believed this was when I made a mistake of a person that was new to debugging. The reason was that I had a look at the log coverage and saw all the commands in the card effect that I wanted to test were covered. Hence, I thought that I was done with the coverage because it covered 100% of the functions that I wanted to test. However, I didn't realize that the card effects accepted multiple different values. And, different values might provoke different functions that connect to the card effect function. Therefore, despite my code covered the effect card function 100%, it wasn't test all of the possible scenario that the card effects could accept. That mean I should make my code more random in every aspects of the functions, which leaded to the fourth project.

For the fourth project, I needed to play the whole game randomly. This was when I got a little confused with the second rule of Agan "Make it fail". The reason was that because it was random, I might won't be able to reproduce the error. Luckily, I was reminded of the random seed and I could create my code. I firstly test my code with a classmate code using the new testdominion function. Liked I had expected, I found a difference output from my classmate. That was because we had create different error which the first assignment. Next, I tried to test the same dominion code with different seed. I also got a different result from both run. That mean my testdominion code work properly.

Finally, for the final project, I had to find a bug from one of my classmate. This was when I had the change to use most of the things I learned in the class on my classmate. I mainly used divide and conquer technique to find the code in my classmate. The reason was that the assignment only asked us to find the bugs and didn't explicitly tell us to fit it. Hence, I only spent a sufficient time on reading the code and not tried to understand everything my classmate did. Next, I had a change to work with delta debugger. The program was supposed to run through my code and find a set of changes that would cause my code to fail. Then, the program supposed to minimize that set of change by finding interfere between the elements of the set. As the result, it would create a shorter version of the program that we wanted to find the bug. That shorter version was the minimal set of command that could reproduce the same bug as the original program. This would allow us to be able to read the program and make the program lighter.

These were some of my final thoughts about the dominion program. It was really unpredictable and that made it hard to debug it. The reason was that I sometimes didn't know if the dominion had a bug or my debugger itself had a bug. The program was highly couple, which made it hard to follow. Additionally, it lacked the comment for each function, hence, sometimes, I didn't know what that function supposed to do. There was some unnecessary files in dominion that made me a little confused about which program I could use and which program I couldn't use.