Krisna Irawan

CS 362

testreport.pdf

Testing Dominion:

This is my first time working on a code designed solely for testing purposes. I do learn a lot about new stuff in this class. Before I took this class, I have barely had any ideas about mutant testing, random testing, or even mutant testing. My first real testing experience start when I was working on the assignment 2. That was my first time writing a unit test for a code. In this assignment I created the unit test for Council Room, Smithy, Garden, Adventurer card function and initializeGame, numHandCards, buyCard, supplyCount dominion functionality. During this assignment, I found a couple bug in my code, including the bug that I purposely put on my code. I found that my council room and adventurer card functionality has a bug in it. It was such a new and rewarding experience to be able to write a unit test and see it in action. My code coverage was 38.71% which is pretty good for a first timer.

On my third assignment for this class, I create a random testing algorithm for Adventurer, Smithy, and Council room functionality. In this assignment I found an interesting findings. I found that there is a bug on the unit test that I created for Smithy. This is really interesting since on the previous test I stated that there is a bug on my Smithy card algorithm, but in reality, the real bug was found on the unit test by itself. I am really glad that I created these random testing and found a bug on my previous testing. This reminds me of one of Agan's principle of checking the plug. Sometimes, the real bug is can be found in our test, and it can caused a lot of problem. Thus, we should never fully trust our test. In this test, I am able to get 100% line coverage for the all the functionality that I have been testing. It is interesting to see the result of a random test cases since we put a lot of random input in our test code. This can result in a lot of interesting cases that we might never think of. Thus, it makes me feel confidence enough about my random test code reliability.

I think assignment 4 was the most challenging assignment for this class. The amount of testing that I have to write is longer than usual and the code coverage that I have to reach is substantially larger than before (around 60% minimum). Moreover, I also have to compare my dominion code with my classmates' dominion code and find if by running the same test, we can achieve a different test result. In our class circumstances, I found that using this testing method is not as useful as using random or unit testing. One of the biggest reason why I don't think this test method is useful for finding bugs in our code is the fact that nobody in our class have the right implementation of dominion. Comparing two buggy code will not tell us, which code has the real bug, or even which code is better than others. However, this method will be really useful if we have a bug free code to compare with. In this assignment, I achieve 65.97% of code coverage which is substantially larger than two of my previous testing method.

Reliability of my Classmates' Dominion:

Jiangzh:

I found that when I run my unit tests and random tests on Jiangzh's dominion code, it result in lower code coverage. The result shows that we have around 3% to 8% in code coverage differences. One of the biggest code coverage differences that I got when running my test code in Jiangzh's dominion code is 8.71%. This happens when I run my unit tests. In details, I got 30% code coverage when running my unit tests, 23.93% code coverage when running my random tests, and 57.14% code coverage when running my testdominion. This differences might be caused by our differences in refactoring the card functionality for assignment 1. Despite our differences in test coverages, there are no real differences in our test result. This can be caused because we barely change our code from the default code that our professor gave to us, which makes our code similar in some way. I also find a bug in Jiangzh's council room implementation when running my test. In his version of Council Room implementation, he only draw 2 cards instead of 4 cards. I found this bug by running my unit test for the Council Room implementation. It makes me feel happy that my test code can be used to find a real bug in my classmate code, which increase the reliability of my test code.

Other than the Council Room bug that I found in Jiangzh's dominion code, I couldn't say more about which dominion codes are better. We are expected to introduce new bug for the assignment 1 and barely make a change in our code to improve the reliability of our dominion code, which result in pretty similar test results. It is hard to see which dominion code is better, especially when you are comparing two buggy dominion codes. I haven't seen the bug free dominion code and have no idea what it will looks like. Based from our similar test result, I can only conclude that our dominion code are similar in reliability.

Washburd:

I also get a lower code coverage result when running my test code in Washburd's dominion code. Although, the differences are not as big as when I run my test code in Jiangzh's dominion code, it still result in a lower code coverage. I got 29.48% code coverage when I run my unit test, 23.45% code coverage when I run the random test, and 54.71% code coverage when I run the testdominion. Again, the only differences that I found is the refactoring that we created for assignment 1. He refactor Council Room, Smithy, Village, Outpost and Mine card instead of Council room, Village, Smithy, Great Hall, and Embargo card functionality like what I did in my dominion.c. I don't think Washburd make any changes to improve his dominion code logic. Thus, making our code still similar in some ways. There are no real differences in test result when I run my test in his dominion code. However, I do found two bugs in his card functionality. The first bug that I found is located in his Adventurer card. My test shows that he has one extra card at the end of his Adventurer card logic. I think this is because he forgot to discard the Adventurer card after he use it. I think discarding the Adventurer card at the end of the function might solve this bug. Another bug that I found in his bug is located in his Smithy

card. He add 2 cards instead of 3 cards in his Smithy card logic. I think we can fix this bug by changing the for loop stopping condition from i<2, to i<3.

Other than these two bugs that I found, there are no real differences in our dominion test code result. I cannot said that my dominion code is better because my dominion code also has a bug in council room and adventurer card functionality. It is hard to decide which one is a better dominion code since all card functionality is as important as the other card functionality. Thus, I think our dominion code is the same in terms of reliability.

Conclusion:

This class is a challenging but rewarding class. I do learn a lot from this class since this is my first time learning about making a test code. I believe that I have significantly improved my knowledge about testing from this class. Although, it is still a long way for me to become a real software tester, I believe with more practice, I can build on from the knowledge that I learn from this class and become a real software tester. I enjoy the learning experiences that I get from this class and I think this knowledge will be helpful for my computer science career in the future. Although, writing a test sometimes can be challenging, it was a fun experiences for me. I also learn that Agan's principles are the golden rule of creating a test code. Without this principles it is really easy to see people getting frustrated in creating a test code. Another things that I learn from this class is the fact that every test method have their own advantages and disadvantages. It is our job as a software tester to choose which methods are the most appropriate for the situations. This experiences of learning how to create a bunch of test method will absolutely increase my testing tool arsenal and hopefully, as I learn from creating these test methods, I can use this knowledge to make a better decision in selecting the best test method. Overall, I really learn a lot from this class and I believe that this class has help me to become a better software tester in the future.