Team GulDukat
11/15/2016
CP03 Test Report

Test Report

Our team did not have enough use cases for every team member to complete one from start to finish. Ethan was tasked with completing Use Case 4, Modify Accounts. Zach and Rachel were tasked with completing Use Case 3, Generate Report. Zach did the majority of the coding of Use Case 3, while Rachel did the majority of the documentation and planning. Dan was tasked with putting the finishing touches on Use Case 2, Run Scenario, which was implemented last iteration.

Zach created unit tests for the Report Controller class. He had two unit tests checking whether or not the Report would be receiving the correct value of total damage, regardless of whether or not the damage was from a melee attack or a spell attack. These unit tests passed without an issue. The equivalence partitions for input to the Report Controller and GUI would negative numbers, positive numbers, zero, and extremely large positive numbers that may not be able to be contained by an integer. To test these equivalence partitions using boundary value analysis, the numbers used to test would be negative one, zero, positive one, fractional amounts such as negative one half, positive one half, and a large amount such as two million. The internal algorithms of our system do not allow erroneous numbers to be propagated to the GUIs, so a negative number or an outrageously large number would be changed by our system to a number that is within the realm of reason. Positive whole numbers and decimals will be represented appropriately in text boxes on the Report GUI. The cyclomatic complexity of generating a report is four, because there are only a few loops through which data can go. To test these basis paths, you would need to generate a new report, an old report, select a nonexistent report, which generates an error, and skip the loop that fills the list box on the Report GUI. The last situation would require changing the internal code in order to test that path.

Team GulDukat
11/15/2016
CP03 Test Report

Ethan created two unit tests for the Modify Accounts GUI. One test was used to determine whether or not the username field was empty. If so, send the user an error message. The other test was used to determine whether or not the password field was empty. Again, the user would see an error message if the textbox was left empty when submitted. Ethan had these unit tests commented out, but they appear to have worked otherwise. The equivalence partitions for input to the Modify Accounts GUI would be a string including null, punctuation, capital letters, and an extremely long string. To test these equivalence partitions using boundary value analysis, the numbers used to test would be a blank string, a period, GulDukat, and a string with 2 million characters. The internal code of the Modify Accounts GUI would not update a field if it was blank, which takes care of that erroneous input. Punctuation, capital letters, and longer strings are accepted within the means of the system. User error is validated by making the user type the new username and password twice. They also cannot select to be a user and an admin at the same time, because they must only be one or the other. The cyclomatic complexity of the Modify Accounts GUI is nine. To test these basis paths, you would need to log in once as a user and another time as an admin. Then, you must try to update using the types of erroneous data mentioned above, such as leaving text boxes blank, not matching paired textboxes, not selecting an account type, or selecting two account types.