

Lab 8

It is important that you do not begin this lab by looking at the code!

In the previous lab we used JUnit to test code for a queue. During the lectures other forms of testing have been suggested for system level testing. It is these which will be the focus of this lab.

Application specification

Our application prints a prompt which asks the user for a name, and adds this name to a list of displayed names. Names should not be displayed if they include characters which are not letters, or if they are shorter than 3 letters and longer than 10. Should a user input an invalid name then the message “Invalid name!” should be shown. Typing “quit” should end the application’s execution.

Assignment Project Exam Help

Black box testing

Still do not look at the code!

Test the application by running it on a suitable choice of user inputs. With equivalence classes in mind choose values which allow you to check that the application conforms to the specifications. Take care to note which inputs cause the application to behave as expected, and which do not.

Add WeChat powcoder

Code coverage, making sure the whole system is tested

Consider the input data which you used in the previous section. Do you have any redundant data with respect to testing? To see if this is the case we can use the built-in code coverage tools in Eclipse. Click on “Run > Coverage As > Java Application” and input what you think is a set of inputs that fully covers the specifications. When you terminate the program lines of code which have been executed will be highlighted in green, and those which have not will be highlighted in red. What is your statement coverage and branch coverage? Rerun the code coverage tool with all your input data from the previous section. Does your statement coverage and branch coverage improve?

The main purpose of considering code coverage when testing isn’t to check that your set of tests is minimal, but is instead to make sure that your test cases check for errors in all parts of your code. Our code has sections that are not often called in normal use. Having now taken a white box approach, can you comprehend the code to ascertain which test data might improve coverage? Do not fix errors at this stage.

Mutation testing

Notice that there are some JUnit tests included in the project directories to assess the correctness of the Validate class. The rest of this lab is spent checking their usefulness. Run these tests and fix the errors that cause them (don't hesitate to ask for help if necessary). Notice that not all errors in the code are covered by the tests. Write a couple more tests which will catch these and repair the code.

Now that the code works as intended choose a mutation to make to the validate class. Rerun the tests. Consider whether or not your mutated line of code gets called in the execution of your tests, and if your tests are sufficient to kill the mutant. Try and come up with mutations that will be killed, and mutations which will not. You can just note what changes you make and where, you do not need to save a copy of each mutant.

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder