**Summary**

In the project you had me do we took three features the setting up of contacts, tasks, and appointments. In Appointments you wanted the date of the appointment as well as a description of why the appointment was made. In the contacts section the requirements were to have an ID number for the customer to be identified with. As well as their first and last name, and the address of the contact person. Lastly in the tasks section the requirements were made to have a task Id, the name of the task to be done, as well as a description of the assigned task. There was also the question of length of the information provided. The length of the Id’s were to be ten digits that were not supposed to be changed. The descriptions were supposed to be less than fifty characters total. The names were limited to ten characters, and the address was limited to thirty. The rest of the variables were able to be changed as needed.

The three features would also have a code for each part to be used for adding, deleting, and changing the information given. This includes the First and last name address and description of the task or assignment.

Next there would be the Junit tests that were run. There were six different codes to run, that contained a total of eighteen tests total. I tried to cover the requirements in my tests to cover the requirements that were made. In a good test all tests should pass, but there were times that a test should not pass and that’s where it does what you do not want it to. This is where I ran into an issue in my thinking on how to structure a test in order to set it up to where you want it to pass and give all green.

How do you know that you have an effective base coverage percentage? From the List of requirements that were given you think first can the test make sure that this works the way it is supposed to. Update when it is supposed to, or delete the contact when the user wants to, Then you can start to create tests based on trying to make the program do things that it is not supposed to do. And you make the description of the task longer than fifty characters long or seeing if you can change the ID number even though you are not supposed to be able to. These tests should fail. This is based on the tester ability to see what the code is supposed to do, as well as things its not supposed to.

As far as the efficiency of the code, that is set with the setter and getter methods. This was also done by combining if possible, the if statement requirements.

Example, (number == **null** || number.isBlank() || number.length() != 10)

This could have been written as three different if statements and added roughly ten to twenty lines. Doing this once may not be adding a lot but if you do this over every time the code could be extremely long.

**Testing Practices**

There were several techniques added to test the program. I used equivalent partitioning. This is a type of test that is a pass or fail test. If the Id number is ten not less or more then it was a valid test. If either the number is less or more it would be invalid. A boundary test was also conducted to make sure the address, names and descriptions do not go past the boundary or the test would fail. This would make sure that there would be a boundary and clearly show where that boundary is located. Also I set down to a checklist to make sure that all the requirements were to be added and checked off as they were added. This could help in the static testing to set up a program before its run.

Some tests were not used like a decision table, since there was no real need to test for cause and effect in the code. I also did not check to make that an ID would bring up both the contact information and tasks, as that was not a requirement.

**Mindset**

As a coder , you feel your role is to create. However as a software tester I felt that the

goal was to Push the envelope. Since nothing would blow up, I did not deal with any real

caution. But when dealing with sophisticated technology, I can see the real need for caution. If

things can blow up or hurt some one in the process, there should be testing caution. In this case I

would suggest that you implement testing in stages. In the case of malfunction equipment you do

not have to have the same type of caution, but you must be cautious in the fact that you should

for see several problems that could come up and then test to make sure that they do not happen.

When it comes to bias in this code I do not see the bias other than the fact that I can only

test what I think to test. This can be fixed by talking to fellow testers and ask for other ways that

I might be able to test the program. When creating a program that is designed to poll people for

something. I can see a lot of bias can happen here. In fact doing online polls I often wonder if

others see the same thing I see as I vote. Things in this case can be skewed in order to show favor

to the side you want to win. There is also paid bias for putting your webpage toward the top of

the list.

Computers are a tool that we currently use in our everyday life. And the saying is true we

are only as good as the tools that we have. So with this in mind I believe that it is important to be

disciplined in our approach to testing. We need to make sure that our computer software is able

to function as it is supposed to and not beyond the scope of what is necessary. For this reason

cutting corners is a no-no. If we did that then most of the technology tools we use today would

not exist. I would not want my camera on my laptop to send out pictures to Facebook at random

times. Some things should not be seen. Nor would I want some websites to pop up just because

they can. I hate when ads do that, I don’t want webpages to be that way. It would make the

internet tool terrible. So we should test the software and be happy when it does what we

requested of it. This has been a wonderful class of which I learned a lot.

Citations

software testing help June 25 2022, popular software testing Techniques with examples. <https://www.softwaretestinghelp.com/software-testing> techniques2/#Equivalence\_Partitioning