The past three milestones, classes were created for the new application made by Grand Strand Systems software company. Tests were created to ensure those classes were working properly. JUnit4 was used as the software of choice to test those classes.

Screenshot from 2021-12-10 19-10-00

On Module Three Milestone, I created the Contact and ContactService classes. The Contact class has five attributes: contactID, firstName, lastName, phone and address. It has setter and getter methods for firstName, lastName, phone and address. The contactID has only a getter. I created a ContactTest class to test setFirstName, setLastName, setPhone and setAddress. The idea is to test the setter and getter by comparing the string that is setted to the string that is getted. If it is correct both methods are valid. As shown above, the test coverage for Contact test is 100% for the class and 81% for the methods.

Screenshot from 2021-12-10 18-04-35

On Module Four Milestone, I created the Task and TaskService classes. The Task class has three attributes: taskID, name and description. The Task class has setters and getters for the name and description. It also, has the getter for the taskID. I tested the Task class by using JUnit4 and creating a TaskServiceTest class. The Task Test class tests the methods: setName, setDescription and getTaskID. The idea is that the getter method is used to test if the string used to set equals. This tests both the getter and setter methods. As shown above, the task class was tested along with coverage. It shows that the class coverage is 100% and the method coverage is 85%.

Screenshot from 2021-12-10 18-02-38

The TaskService class has one attribute: the task list. The TaskService class has 3 methods: addTask method, deleteTask method and editTask method. The TaskServiceTest class tests all 3 methods. As shown above, the class coverage is 100% and the method coverage is 100%.

Screenshot from 2021-12-10 18-47-05

On Module Five Milestone, I created the Appointment and AppointmentService classes. The Appointment class has three attributes: ID, date and description. It has setter and getters for date and description. The ID only has a getter. I created an AppointmentTest class to test the Appointment class. It tests the setDate, setDescription and getID methods. The idea is to test the setDate method to test both the setDate and getDate methods. Also, the setDescription test was used to test both the setDescription and getDescription methods. As shown above, the class coverage is 100% and the method coverage is 85%.

Screenshot from 2021-12-10 18-52-12

The AppointmentService methods are: addAppointment and deleteAppointment. The attribute is the list of appointments. I created a AppointmentService test class that tests the addAppointment and deleteAppointment methods. As shown above, the class test coverage is 100% and the method test coverage is 100%.

The software testing techniques used was unit testing. JUnit4 was used to unit test the code. Its a good way to test each individual method and attribute of a class. This is to ensure the class is sound and does not have any problems. Integration testing was not performed on the software. It would be beneficial to include integration testing to the application. Many different services and objects were created as apart of the application. Integration testing would ensure those services and classes would integrate with each other fluidly (Hamilton, 2021). Integration testing would focus more on the flow between the modules rather than testing each individual unit (Hamilton, 2021). For example, in an email client you are directed to a login module. You input your credentials and you get directed to your email. This ensure the transfer between each module works without any problem.

Unit testing is great for testing individual modules to ensure they work correctly. Integration testing ensures each module work with each other properly. In the example of the milestones that I worked on, unit testing was great to ensure the services and classes I created are working and that the individual requirements are being met. However, I think the next step should be integration testing where I combine all those services and classes together and ensure each of them are working properly between each other. Ensuring that data and direction are being transferred between each other properly and accurately. I think using both would be beneficial for the application.

The mindset I had was being cautious and thorough with creating the tests. I tested each part of the classes. The attributes and the methods of the class were tested. I also ensured that the classes followed the software requirements. In an example, the Contact class should have a contactID with a limit of ten characters. The ContactTest class ensures that the contactID can not have more than ten characters.

The ways to limit bias was to use the test coverage and to run the actual unit test. I actually found some flaws in my code when running the unit test. Simply running the unit test ensures that there aren’t any bugs in the code.

I mentioned in the 7-1 discussion, that it is very important to keep quality of code, and to always test code. A real life example of code not working properly was the Therac-25 machine. The Therac-25 machine was a radiation therapy machine for cancer treatment (Fabio, 2015). It overdosed 6 patients because there was a bug in the code (Fabio, 2015). If a thorough test was conducted on the machine, the bug could have been found and patched.

As mentioned before, testing is very important to be conducted when developing software. Tests should always be made to find bugs in system to prevent any problems in the system. There are several testing techniques that can be used. I used the unit testing technique to ensure the software is correct.

Hamilton, T. (2021, October 8). *Integration Testing: What is, Types, Top Down & Bottom Up Example*. Guru99. Retrieved December 3, 2021, from <https://www.guru99.com/integration-testing.html>

Fabio, A. (2015, October 26). *Killed By A Machine: The Therac-25*. Hackaday. https://hackaday.com/2015/10/26/killed-by-a-machine-the-therac-25/