Tyler Jackson

Project 2

CS-320-T1183

22EW1

Project 2

My unit testing approach was coordinated with my demonstration of white box testing. For all three assignments, I created JUnit tests to uncover errors in my code by testing various scenarios in this mobile application. Some examples from the requirements were to determine the length of the ID numbers. In the Task Class, the ID string could not be longer than 10 characters, shall not be null, and shall not be updatable. For my Task class, I created several variables to gather and hold the information. I included a check to assure that the field was not empty, or “null” and was no greater than 10 characters long (including spaces). These requirements were spread out through all three assignments, with various changes based on the requirements based on the length of the string, or even the date and time.

My experience working with the JUnit tests was a little bit of trial and error. Running the test was especially useful to determine what part of the code was failing. Really pinpointing and error on my end so it was able to be corrected pretty quickly. From reading and watching videos on JUnit testing, I have found several ways to write the code. This made it a little confusing at first, because writing JUnit test code isn’t just done in one specific way. But watching the different methods helped me understand how it works a bit more.

My TaskService code is as follows:

public class TaskService {

private ArrayList<Task> taskList = new ArrayList<Task>();

public boolean addTask(Task newTask) {

boolean addition = false;

for(Task task : taskList) {

if(task.equals(newTask)) {

addition = true;

}

}

if(!addition) {

taskList.add(newTask);

return true;

} else {

return false;

}

}

public boolean deleteTask(String id) {

boolean idStatus = false;

for(Task task : taskList) {

if (task.getID().equals(id)) {

idStatus = taskList.remove(task);

if(idStatus) {

return idStatus;

} }

}

return false;

}

public boolean updateTask(String id, String name, String desc) {

for(Task task : taskList) {

if(task.getID().equals(id)) {

task.setTaskName(name);

contact.setTaskDesc(desc);

return true;

}

}

return false;

}

}

This class specifically taught me the importance of staying disciplined in the quality of your code. The Eclipse IDE itself catches errors in your code for missed semicolons, or variables that don’t exist, or if you accidently made a class private instead of public. But what it doesn’t catch is the intended way that the code is supposed to work. The JUnit testing is just as important, if not more important because it discovers what isn’t working as intended, even if it seems like it is coded correctly.