# CS-320 Software Test Automation & QA

Dr. Albanie Bolton  
Dani Yosopov  
[Dani.yosopov@snhu.edu](mailto:Dani.yosopov@snhu.edu)  
Southern New Hampshire University

In all of the features that I built, I was focused on testing each attribute and each function in many ways to try and cover the most options for failures.  
I achieved it by running multiple scenarios of incorrect input, for example in the "ContactTest", "TaskTest", and "AppointmentTest" I tested if the program will send out an error message in many ways like if all the attributes are null or even if one of them is null.  
The same thing was done with attributes that were too long or too short.

I think that the overall coverage of my JUnit tests covers at least 90% of the possible options for failures, using the book "Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition)" I was able to plan the testing properly to ensure I was covering as much as possible.  
"Planning is determining what is going to be tested, and how this will be achieved" (Peter Morgan, 2015).

To ensure that my code was technically sound I had to test if proper memory allocation was made in the run time on top of that the I had to check if the proper attributes are inserted in their place this was done with all three features "ContactTest", "TaskTest", and "AppointmentTest".

@Test

void testContact() {

Contact contact = new Contact(contactID,firstName,lastName,phone,address);

assertNotNull(contact.getcontactID());

assertNotNull(contact.getfirstName());

assertNotNull(contact.getlastName());

assertNotNull(contact.getphone());

assertNotNull(contact.getaddress());

}

@Test

void testEquals() {

Contact contact = new Contact(contactID,firstName,lastName,phone,address);

assertEquals(contactID, contact.getcontactID());

assertEquals(firstName, contact.getfirstName());

assertEquals(lastName, contact.getlastName());

assertEquals(phone, contact.getphone());

assertEquals(address, contact.getaddress());

}

To ensure that my code was efficient I have to test the functions integrating with each other in the features service file for example I needed to add a function to search a piece of specific information in an array bested on a unique ID and then use it to either update the information or to remove it completely from the array.

public Task Search\_task(String uniqueTaskID) throws Exception {

int i = 0;

while (i < TaskList.size()) {

if (uniqueTaskID.equals(TaskList.get(i).gettaskID())) {

return TaskList.get(i);

}

i++;

}

throw new Exception("this contact ID does not exist!");

}

public void delete\_task(String uniqueTaskID) throws Exception {

TaskList.remove(Search\_task(uniqueTaskID));

}

public void updatename(String uniqueTaskID, String name) throws Exception {

Search\_task(uniqueTaskID).setname(name);

}

@Test

void test\_delete\_contact() throws Exception {

TaskService service = new TaskService();

service.NewTask(taskID, name, description);

service.delete\_task(taskID);

assertThrows(Exception.class, () -> service.Search\_task(taskID));

}

@Test

void test\_update\_name() throws Exception {

TaskService service = new TaskService();

service.NewTask(taskID, name, description);

service.updatename(taskID,update\_name);

assertEquals(update\_name, service.getTaskList().get(0).getname());

}

The software testing techniques that I used in all of the features were functional testing, unit testing, and integration testing. That was the best approach for me because in every class you have multiple attributes and methods and unit testing lets you the ability to check each one of them, sometimes I had to use integration testing because my code had methods that integrated with other methods. For example, I have a method that searches data by using the ID string and send it back to another method to delete the data or update it.

The software testing technique that you did not use for the features is system testing.  
In this case, my features did have an interface so I can't run them in a system which is why I can't perform the test.

This technique helps us to test all the components of the system by using all the other testing techniques.  
The purpose of system testing is to verify the whole system is tested and working properly, as well can to help verify the functional modules or user story.  
Using only this technique will make software development and delivery delayed because software system testing takes longer than planned, and sometimes even not to meet the customer’s requirements.

This technique helps us to test the functionality and the process of the system by sending different inputs and seeing the end result for each input.  
The purpose of functional testing is to verify that the functionality of the software is tested and working properly.  
The implication is that we can miss the logical errors in the application, on top of that functional testing can cause redundant testing.

This technique helps us to test all the code segments by testing the first segment of code and adding more segments if the tests passed.  
Using this technique will verify all the code segments that are needed to be integrated and if they working properly.  
This also will help detect the errors related to the interface.  
This technique can be very challenging if the code was devalued by two different teams or companies.

The unit testing technique requires separating written code into units to test each one and determine if it works as intended.  
The purpose of unit testing is to verify each component or individual code segment of the software is tested and working properly, this can help with detecting early bugs in code that may be more difficult to find later.  
The unit testing technique can take time to write test cases as well as a lot of time for maintenance.

While working on this project I was very careful with the appointment and the appointment service classes because of the time and date aspect.  
Unlike the other classes, I didn't know how to test it at first due to the fact that it is auto-generated.  
I had the look for some extra information on how the Date and Calendar libraries work when it comes to their attributes and methods.  
It was very important to me to know how to do setup the date in the proper way so I could be able to test it properly.

@SuppressWarnings("deprecation")  
@BeforeEach

void setUp() {

appointmentID = "7D1A8NI";

appointmentDate = new Date(2025, Calendar.APRIL, 8);

appointmentDescription = "good description";

past\_appointmentDate = new Date(0);

too\_Long\_appointmentID = "718DE78A227Q";

too\_Long\_appointmentDescription = "This will be the longest description ever made by me";

}

The way I tried to limit bias in my review of the code was by following the proper technics of code writing, which helped me with using the same code template that I came up with originally on week three just with little tweaks to fit the other object's requirements, this includes the tests files I have performed for each class.  
For example, for each test file, I performed a setup before each test.  
Another example is the contractors in the classes are in the same order just with the needed condition that the customer asked for.

The importance of being disciplined as a software engineer is to be able on providing high-quality software so the client or the customer won't have any issues while he uses the software.  
It is very important not to cut corners when it comes to writing or testing code because it will cost a lot of money to fix the poor code quality that may even not be fitting the customer's original requirements, and this will affect the reputation of the development for future jobs.   
The plan to avoid technical debt as a practitioner in the field is to   
Spread awareness of the importance of code quality the more developers are aware of It’s effects, it can reduce technical debt.

Reference

1. Morgan, P. (2015). Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition).