7-2 Project Two Submission

Lynn McCargar

CS-320-T2644 Software Test Automation& QA 21EW2

Prof. Angelo Luo, M.S

Southern New Hampshire University

The project requirements were based on the theoretical Grand Strands Systems software engineering company. The whole project is meant to simulate real world projects software developers and testers will encounter. No databases were used, all the data housing was in-memory data for storing information. No user interfaces (UI) were used in this simulation. Classes had requirements, as well as, services for manipulating data. These parameters were consistent with each class and services in this real world project simulation.

The first requirement was with the contact services. The project one requirements for the contact class and service entailed,

|  |  |
| --- | --- |
| Class Requirements | Service Requirements |
| The contact object shall have a required unique contact ID String that cannot be longer than 10 characters.  The contact ID shall not be null and shall not be updatable.  The contact object shall have a required firstName String field that cannot be longer than 10 characters.  The contact object shall have a required lastName String field that cannot be longer than 10 characters.  The contact object shall have a required phone String field that must be exactly 10 digits.  The contact object shall have a required address field that must be no longer than 30 characters.  All these field shall not be null. | The contact service shall be able to add contacts with unique ID.  The contact service shall be able to delete contacts per contactId.  The contact service shall be able to update contact fields per contactId. The following fields are updatable:   1. firstName 2. lastName 3. PhoneNumber 4. PhoneNumber |

The JUnit 5 tests were based on the above sets of requirements from the contact services. Starting with the annotation @Test, which denotes the testing method we used to test the requirements. We used a Boolean condition called assertTrue to ensure that the results are what we are looking for based on the above requirements. In order to test whether input outside the requirements would throw exceptions, we used assertFalse to help us understand if the input used improper naming conventions. In the JUnit 5 services tests we added a @Before method to setup the services tests. The contact JUnit tested creating, succusses, and failures of the contactID, first name, last name. Also tested whether there too many or not enough alpha numeric characters, etc. We were able to get a higher JUnit test coverage by following the industry standards JUnit 5 layout for the testers.

The next area tested was with the task services. As mentioned in the beginning of this paper, the task services had similar structure as contact services, so the above is true for the task services too. List below are the requirements for this part for the test analysis.

|  |  |
| --- | --- |
| Class Requirements | Service Requirements |
| The task object shall have a required unique task ID String that cannot be longer than 10 characters.  The task ID shall not be null and shall not be updatable.  The task object shall have a required name String field that cannot be longer than 20 characters.  The task object shall have a required description String field that cannot be longer than 50 characters.  All fields shall be null. | The task service shall be able to add tasks with a unique ID.  The task service shall be able to delete tasks per taskId.  The task service shall be able to update task fields per taskId. The following fields are updatable:   1. name 2. description |

The task test analysis had fewer requirements but nonetheless the same testing methods as with the contact services procedures. These JUnit 5 tested creating, successes, and failures of the taskID, name, and descriptions.

The appointment services used in-memory for the data structures, so no databases where required. There was no user interface (UI) for this part if the requirements for the appointment services.

|  |  |
| --- | --- |
| Class Requirements | Service Requirements |
| The appointment object shall have a required unique appointment ID String that cannot be longer than 10 characters.  The appointment ID shall not be null and shall not be updatable.  The appointment object shall have a required appointment Date field.  The appointmentDate field cannot be in the past.  The appointment object shall have a required description String field that cannot be longer than 50 characters.  All of the field shall be null. | The appointment service shall be able to add appointments with a unique appointmentId.  The appointment service shall be able to delete appointments per appointmentId. |

The appointment services JUnit 5 tests analysis was similar to prior two tests with an exception of testing the date fields. In order to ensure that dates before the current date could not be used within the code. So we setup a functions to aide with the JUnit 5 test analysis,

@BeforeEach

**void** setUp() {

id = "0987654321";

description = " b c d e f g h i j k l m n o p q r s t u v w x y z";

date = **new** ~~Date~~(3210, Calendar.***JANUARY***, 1);

tooLongId = "9998888777666655544333222111";

tooLongDescription =

"a b c d e f g h i j k l m n o p q r s t u v w x y z";

pastDate = **new** Date(0);

}

The JUnit test analysis we used testing methods such as, @Before and @Test methods during these analysis work. But there are many other JUnit testing methods as our disposal such as, @RepeatedTest which is a test template to repeat tests in the event we need to repeat tests during our analysis. Another being @TestClassOrcder which is used to configure test class execution order for another method called, @Nested test classes. These are just a few examples of the capability of JUnit 5 testing strategy.

These projects in this course if I may use an analogy, are like solving a puzzle. Each piece needs to be placed in proper order to ensure the puzzle is constructed methodically. Meaning as part of the software development process, quality assurance and testing ensure that the pieces are working as expected. This is a learned aspect that must be just as important as the coding is, because poor coding practices can lead to harm with institutions, businesses, and even us humans. An example of this is with the Boeing software bug. This is a good example of why quality assurance and testing is apart of the software development lifecycle. People lost lives due to a software bug that should have been detected before the release of the airliners software.

I am a computer science student with the goal of developing systems to solve problems with computing. This is not an easy program to complete, I would go as far to say that not too many people can make it through a computer science program. And with that said, there is great pride in such an accomplishment in a life time. There will be some arrogance and a sense being able to do it all ourselves. But this is not realistic in the real world. We just need to look at our past to understand why this is not realistic. To demonstrate my point I will use empirical evidence and an anecdote from my past experience with a company named Best Buy, the multi-billion dollar retailer here in the U.S. I was a young Windows infrastructure engineer working for Best Buy and hired to help with the bestbuy.com ecommerce development. At that time Andersen Consulting was the big name in accounting and IT consulting. We used Andersen’s Methodology One which was a waterfall hybrid methodology. This was during the time where technologist drove how a solution were built, not the business units who employed us. The company invested millions towards the effort to move the company into the ecommerce revenue strategy. Much of the testing was done by the technologists who developed the system(s), e.g., developers tested their own code, systems engineers tested their own setups, etc. After about 15 months of building the commerce.bestbuy.com and putting all the pieces in place, the company had a big promotions of the unveiling of the ecommerce site. Newspapers’, television stations, and all attending this red carpet event. The site was launched and millions of users hit the site only to find an error message stating the site could not be reached.

In conclusion, there is an old adage that comes to mind by Theodore Roosevelt: “Nothing worth having comes easy.” And if we take a moment to absorb the meaning of these words we can start to understand the depth of the meaning. Most of us work hard in our relationships, our careers, our families to provide best compliment of ourselves. Because if we do not we will lose what is worth having in life. If Theodore Roosevelt did not hold these values to heart, he would not be a U.S. President. Nor would we have national parks we have today to enjoy. These are the disciples that are needed to be successful a software engineers. All the pieces are an intrigue part of our profession.

References:

Bechtold, S. Et Al (Apr 12, 2021). *JUnit 5 User Guide*. Retrieved from, <https://junit.org/junit5/docs/current/user-guide/#overview>  
Hambling, Brian Morgan, Peter Samaroo, Angelina Thompson, Geoff Williams, Peter. (2015). Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition). BCS The Chartered Institute for IT. Retrieved from

<https://app.knovel.com/hotlink/toc/id:kpSTAIST01/software-testing-an-istqb/software-testing-an-istqb>

Smith, N. (Apr 19, 2019). A software bug caused Boeing’s new plane to crash twice. Retrieved from, https://techtrends.com/2019/04/19/a-software-bug-caused-boeings-new-plane-to-crash-twice/