**Brandon Hobbs**

**CS-320**

**July 26, 2022**

**Module 5 Journal Entry**

* **What were the software testing techniques that you employed for each of the milestones? Describe their characteristics using specific details.**  
    
  For the 3 modules in this course I have employed unit testing and static testing both of which are a form of whitebox testing. Static testing involves studying the code and comparing it to specifications to identify bugs. I often used this technique to find and fix the offending code the code after one of my JUnit tests failed.  
    
  As I was writing the class methods, I was attempting to design the logic to match the specification document. Then, during the unit test creation I re-read the specification and tried to find ways to prove that I made an error. A few times one of the Junit tests failed and I had to inspect the static code to figure out why. In one such case, I had used the wrong attribute within the logic branch – so of course the test failed.  
    
  I also figured out that merely making sure an exception is thrown when expected is not enough. Catching the correct and expected exception is also important. For example, in another case I was asserting that a constructor would fail due to bad input. An error **was** thrown but not the one I meant to be thrown. So, in Module 5’s assignment I started evaluating the exception type to ensure it was the exception I expected, and my test is not creating a false-positive.
* **What are the other software testing techniques that you did not use for the milestones? Describe their characteristics using specific details.**  
    
  Modules 3 and 4 have very little system testing and no integration tests. Module 5 has a few system tests. Each of the assignments had us create a base class and service class that drives and interacts with the base class. In theory, you can test the base class through the service class and treat this as a system. I did a bit of this in Module 5 but none in Modules 3 and 4.   
    
  Integration testing would test the entire application, bringing all the systems together for testing. In this case, the system and integration tests would be the same. I could, in theory, validate that the TaskService, AppointmentService, and any other classes all work together well at their integration layer – I have not.  
    
  I also did no automated testing – all my JUnit tests were manually run. In theory, I could have had a service that ran the JUnit tests after each build-event in Eclipse. This is more important for large applications.  
    
  There was also no security scanning performed. Libraries and components need to be scanned for security vulnerabilities, e.g., SPRING or even the Java components themselves. In this case, I am not extensively using any libraries, databases, or shared components so security testing could wait.
* **For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.**  
  Automated tests are important for large applications and should be built into a continuous integration pipeline – especially if continuous delivery is being practiced. Automated tests can be at the built stage, pre-deployment, or even post deployment. My projects tent to run Unit tests and System Tests at build and Integration Tests pre-deployment.  
    
  Unit testing is always useful and can help to ensure that small issues do not become huge problems. Unit tests can also help to harden the code against missed logic branches, e.g., what is the value is null, and missed specifications, e.g., maximum string length is supposed to be 50 characters.  
    
  System and Integration tests would become useful as the code base enlarges and there are more interconnected components the more important these flavors of testing become.  
    
  Security Testing is also always important, especially where data is concerned. Right now, there are no databases and no user input into the system so protecting against improperly formatted input, e.g., SQL injection or buffer overruns, is not needed.