As I was doing the projects threw out the modules. I noticed that the majority of the steps did not change. The steps and the requirement for testing with JUnit were consistently the same procedures. Testing of every variable created and every getter / setter. Needing to test for the most common possible mistakes a user would more than likely do. Along with some that we might not have thought of would happen.

With JUnit, I will admit I was a bit lost the first time I was going to do my testing. I’ll be honest, I didn’t realize I had to create the entirety of the code for the test. I had to go through the resources again to realize the steps that needed to be done. After my first assignment with JUnit all the remainder assignments were a breeze. Of course, knowing exactly what case scenarios I needed to input for the fails that might occur.

Constantly testing for any syntax errors was my first step to assuring that my code was efficient. After the obvious bugs that can occur because of syntax errors. I proceeded to check the code for loop, variable or any other random mistake as such that may also cause a bug. Once that was completed. I followed the same steps when it came to the JUnit testing.

When it comes to software and the development of any code. I have been doing it the same way I was shown when I first started coding in my Python class. Do all your code in small chunks. This was if there are any bugs or errors, you’ll be able to catch them early. Of course, the first thing we need to make correct is our syntax. After that making sure that the process we are creating is not going to result in an infinite loop or the application crashing. Taking all this into consideration, you follow the same process when it comes to the JUnit testing. In the small section, but this time as we are writing the code we make sure that the process that we are testing for results in a failure. If the process of writing a name allows 20 characters per say. You want to test for more than 20 characters.

I know there is more than one way to do a software test. Honestly, I did not investigate other processes too much. I found one process, and I stuck to that one process. I sort of stick to the concentrate on one till you are very good at that one. Then and only then I will move on to learning a new process or steps.

For each technique you discussed, explain the practical uses and implications for different software development projects and situations. I am not sure of exactly what this question is asking, I’ll explain it the best I can. As for checking for syntax, checking for syntax is one of the fastest and easiest ways to find bugs. There are typos and we tend to type fast and the more we type the faster we get. This at times will cause error, specially when our hands start getting tired. This can cause typos and syntax errors. One of the most common mistakes. After that checking for mistakes in the way you do you steps in the process. Know and make sure the steps are correct and in the right order. Will reduce the chances of crashing the application, infinite loop, or a stack overflow.

I think we all know the phrase “being a code monkey”. Computer Science and Software Engineering is about problem solving. We can all type code and yes as we are developing our processes in our code, as we are being “code monkeys” we are doing a small part of problem solving. Software testing brings our skills more toward being a problem solving, more away from the “code monkey” and closer to the software engineer. This is how I felt when we started doing software testing. It felt as if that was the missing part of being a software engineer. I had a lot of fun breaking my head attempting to figure out the testing.

When it comes to bias, this is going to be off the subject of testing and away far from software. First, I will answer the best way I can understand the question. So, to the question of bias. I worked in retail for 10 years. You see all sorts of people from so many demographics, cultures, etc., etc. We also work with all sorts of people from so many backgrounds, demographics, cultures, etc., etc. You learn to take everything, even something like code or testing any process or procedure, you don’t tend to like or dislike. I would not judge my or someone else’s code. As a developer or tested. You do the process that you need to do take care of the corrections and pass no judgement.

I have undiagnosed ADHD; the signs are there. Just haven’t been able to get to a doctor to diagnose me. Possible why I must rant before getting to my answer. Yet, as a person with all the signs of ADHD for a person with ADHD discipline is very important. If I don’t have my agenda for the next day and the coming week or so. I am either going to lounge around and work on like 5 different things back-to-back and never complete one. I must have my field journal, my journal, my scratch journal, my google calendar (agenda), and obsidian. This keeps me disciplined and committed to the process of what needs to be done. So, committing to quality with the discipline the process of testing and being a professional software engineer. You might as well toss that the door. Software engineering is long hours on a chair behind the screen and sometimes staring at the same line for hours reading the same code over and over trying to solve a problem. Without discipline you might as well be a “code monkey” after all.

As bad as my ADHD situation might be. I still enjoy being behind these monitors staring at a code trying to solve a problem. As has as it might sound, having so many journals and agendas and apps. Yet, they help each one has its purpose. MY agenda keeps track of the times I will dedicate to working on xyzzy project or ABC class. Obsidian keeps my projects and learning journal. My journal is a bullet journal is thought I’d like to do they might get done if they move to the agenda or obsidian. It is more of a brain dump. This discipline helps, yes discipline is important to be a successful software engineer.