James Galbreath

MOD 5 Journal Guidelines

2/11/2024

In the development of my software project, I employed several software testing techniques across the milestones in Modules Three, Four, and Five. These techniques varied in their characteristics and application, each serving a specific purpose in ensuring the quality and functionality of the software.

For Milestone Four, I primarily focused on unit testing. This technique involves testing individual units or components of the software in isolation to ensure they function correctly. I used JUnit, a popular framework for Java, to automate these tests. By writing test cases for each method or function within the codebase, I could verify their behavior under different scenarios. Unit testing helped me catch bugs early in the development process and provided a safety net for refactoring code.

In Milestone Two, I expanded my testing approach to include dynamic testing. This technique involves executing the software and observing its behavior in real-time to validate its functionality. I performed various types of dynamic testing, such as functional testing to ensure the software met specified requirements and regression testing to detect any unintended changes introduced during updates or modifications. Dynamic testing allowed me to assess the overall performance and usability of the software from an end-user perspective.

For Milestone Two, I also incorporated static testing into my testing strategy. Unlike dynamic testing, static testing does not involve executing the software. Instead, it focuses on reviewing the code and documentation to identify errors, defects, or potential improvements. I utilized code reviews, static analysis tools, and documentation inspections to conduct static testing. This technique helped me identify issues related to code readability, maintainability, and adherence to coding standards.

While I utilized unit testing, dynamic testing, and static testing for the milestones, there are other software testing techniques that I did not employ. One such technique is integration testing, which involves testing the interactions between different modules or components of the software. Integration testing ensures that the integrated modules function correctly together as a whole system. Another technique is acceptance testing, which validates whether the software meets the expectations and requirements of stakeholders. Acceptance testing involves end-users or stakeholders testing the software in a real-world environment to determine its suitability for deployment.

Each software testing technique has practical uses and implications for different software development projects and situations. Unit testing is essential for verifying the correctness of individual components and facilitating code maintenance. Dynamic testing helps assess the software's behavior under various conditions, uncovering bugs and ensuring reliability. Static testing aids in identifying issues early in the development process, improving code quality and reducing technical debt. Integration testing ensures the seamless integration of different modules, while acceptance testing validates the software's compliance with user expectations and business requirements. Depending on the project's requirements and constraints, a combination of these testing techniques may be employed to achieve comprehensive test coverage and ensure the software's quality and reliability.