Week 3 Objectives

By the end of this week, you will be able to:

- **Remember**: Identify and accurately define core software quality vocabulary you'll use all week (e.g., quality attributes/non-functional requirements, defect vs. failure, verification vs. validation, QA vs. QC, maintainability, reliability, usability, performance, security). Recall at least a few widely used quality attributes and be able to list concrete indicators of each one (what it "looks like" in code or in user behavior).
- **Understand**: Explain—in your own words—what "software quality" means beyond "it works," including how quality attributes shape design decisions and tradeoffs (for example, readability vs. performance, speed of delivery vs. test coverage). Summarize the key ideas from the week's materials and connect them to at least one real or familiar software example (a class project, an app you use, or a system you've worked with) to demonstrate genuine conceptual understanding.
- **Apply/Analyze/Evaluate/Create**: Apply software quality practices while building a medium-complexity Java program by decomposing the program into clear modules/classes, validating input, handling errors with exceptions, and writing clean, readable code (naming, structure, and comments where needed). Analyze your own design as you work—identify likely failure points, code smells, or maintainability risks—and revise accordingly. Evaluate your program by testing key paths (including edge cases) and ensuring the behavior matches requirements. Create your Discussion Board post on Software Quality by making a clear claim about what quality means, supporting it with specific examples from your Java program (what you did to improve quality), and responding substantively to classmates by comparing approaches or asking a thoughtful follow-up question.