Part 1: Introduction

Bookmark this page

Advanced Testing

Part 1: Introduction

As you write more lines of code for a programming project, or as the number of programmers working on a project increases, your code becomes more complex. Some projects become so complex that programmers have a difficult time anticipating how even the simplest change made on any given line of code will effect the rest of the program. This lesson will cover three possible, and complementary, answers to the challenge of managing complexity in your software:

- Testing: defining the expected behavior of your code, and expressing that behavior in a software script, consisting of automated tests, that can quickly tell you whether your code changes have broken existing code, or whether your changes have fixed code that wasn't working.
- Coverage: measuring what percentage of your code is invoked by your automated tests. If only 10% of your code is invoked, then your tests will not detect if anything is broken in the other 90%.
- Linting: a "report card" on how well your code conforms to certain standards of style and syntax. The more closely your code conforms to standards of style, the more likely that another programmer will be able to read, understand, and maintain it. Linters also often analyze your code for complexity: less complex code is easier to read, to understand, and to maintain than code that is more complex.

We'll also address the following general concepts of software engineering:

- Writing testable code: not all styles of code writing lend themselves to automated testing. Fortunately, many strategies for making code more testable also make it easier to maintain. One such method is called dependency injection.
- Unit tests vs. integration tests: testing each piece of your software individually vs testing it as a whole. If you only test your software as a whole, then when you find unexpected/broken behavior it may become difficult to know exactly which piece of your software is responsible for the behavior. "Unit testing" is the practice of testing each piece individually so that you can know exactly which piece to attribute a failure to. That said, you still want to test your software as an integrated whole to make sure that the individual pieces are also working together correctly.