Software Testing Overview

> Software testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. - Hetzel88

During the software development process, developers often perform actions to determine whether a program achieves its desired functionality. In a practical sense, this often involves running a program multiple times under a variety of scenarios to cover all possible combinations of user interactions and inputs.

Logging and debugging are helpful parts of the software development and testing processes, but ultimately the purpose of a software **test** is to explicitly define a program's intended functionality and verify the program is producing said functionality.

Automated Testing

Rather than repeatedly performing manual actions, and rather than trying to remember all possible user experience scenarios, software developers write test code to accompany a program's source code. These tests often perform small components of program functionality by directly referencing parts of the accompanying source code in controlled scenarios. By testing individual components of a program's source code, developers help guarantee the program as a whole performs as desired.

Most modern programming languages include built-in features and third-party packages which help developers test their code.

Testing Philosophies and Benefits

Different people test software for different reasons. Some software development teams track metrics related to "test coverage" because it matters to them what percentage of an application's functions are described by tests. Other developers only test public-facing components of an application's source code. Some developers don't bother to write tests at all.

The professor suggests you use tests to save yourself time, to improve the quality and maintainability of your programs, and to communicate your program's desired functionality to other developers. If you find yourself performing a manual action multiple times, consider automating that action by writing a corresponding test case. If you expect a specific function to operate in a certain way, write down your expectations in a new test. If you find yourself writing comments to describe some part of the source code which isn't clearly understood, make its function plain by describing it in a test.

Well-tested applications in general tend to be of a higher quality than untested applications, at least because there is intentional effort spent in pursuit of such a goal, but perhaps also because tests serve their purpose.

Written expectations and descriptions of desired functionality mitigate risks of "brain drain" over time as development teams experience attrition. In this way, well-tested applications tend to be easier to maintain over time.

Test-driven Development (TDD)

Sometimes software developers write tests after completing the development process. In other cases, developers write tests before and during the development process. This latter approach is called **Test-driven Development (TDD)**. When following TDD, tests become a bridge between requirements of desired functionality (i.e. "the app should do xyz") and explicit checks for that functionality.

Your professor prefers to practice TDD and a related approach of **Document-driven Development** whereby the software development process follows the testing process which in-turn follows from the process of writing the software's documentation (most likely in a `README.md` file).

The test-driven development process is closely related with the refactoring process. When performing the two processes in conjunction with each other, as a best practice developers follow the approach of "Red Light, Green Light, Refactor". This refers to a process of first writing a failing test, then writing functional code to make the test pass, then refactoring the functional code and re-running tests to make sure they are still passing.

Continuous Integration

Once tests have been automated, they can be configured to run on a separate server anytime a new feature is proposed. This process is called **Continuous Integration (CI)**, and is one of the techniques for preventing bugs and defects from reaching users. A codebase (i.e. Git repository) can be configured to prevent new versions of the software from being "released" or "merged" until / unless accompanying tests have passed.

> In recent years CI has become a best practice for software development and is guided by a set of key principles. Among them are revision control, build automation and automated testing. - Code Ship website

> Continuous integration is a DevOps software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. Continuous integration most often refers to the build or integration stage of the software release process and entails both an automation component (e.g. a CI or build service) and a cultural component (e.g. learning to integrate frequently). The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates. - Amazon AWS website

> Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly. This article is a quick overview of Continuous Integration summarizing the technique and its current usage. - Martin Fowler

> CI emerged as a best practice because software developers often work in isolation, and then they need to integrate their changes with the rest of the team's code base. Waiting days or weeks to integrate code creates many merge conflicts, hard to fix bugs, diverging code strategies, and duplicated efforts. CI requires the development team's code be merged to a shared version control branch continuously to avoid these problems.

> CI keeps the master branch clean. Teams can leverage modern version control systems such as Git to create short-lived feature branches to isolate their work. A developer submits a "pull request" when the feature is complete and, on approval of the pull request, the changes get merged into the master branch. Then the developer can delete the previous feature branch. Development teams repeat the process for additional work. The team can establish branch policies to ensure the master branch meets desired quality criteria. - Visual Studio