D – Testing and Debugging

# LOG Review

These are some very brief points regarding the concepts and skills listed in LOGs for this topic.

**General Programming Concepts and Terms**

* **Explain the role of testing as it applies to software development** 
  + Testing is a form of “quality assurance” in which steps are taken to verify that a program is behaving or working as intended.
  + Testing has traditionally been treated as a “follow-up” activity that takes place *after* a program is written. Currently, other approaches, such as TDD (Test-Driven Development) place the “testing” activity at the start of the programming efforts, thereby integrating tests as a driving element in software development.
* **Define and compare the terms "compile-time error" and "run-time error"** 
  + A “compile-time error” is an error where a program does not “compile”. In other words, compile-time errors are due to code that does not follow the rules of the programming language.
  + A “run-time error” is an error where a running program does not produce the desired result. In other words, run-time errors are due to logical errors in a computer program. Sometimes these errors are “fatal” (meaning that they result in “abnormal program termination” – the program “crashes”). Other times the errors are subtle; the subtle errors can be the most difficult to detect, particularly when relying on “manual” verification, such as found with ad-hoc test drivers.
* **Define the term "test driver"** 
  + A “test driver” is a driver (which is a program with a main method) whose sole purpose is to “test” the components of a program. Test drivers provide a simple “ad-hoc” form of testing. Test drivers are basically considered “throw-away” programs, as their usefulness does not extend beyond the tests.
* **Create simple ad-hoc test drivers to test for run-time errors** 
  + *Demonstrate your ability to meet this learning outcome guide by completing the related assignments.*
* **Define the term "TDD"** 
  + TDD refers to “Test Driven Development”. Test Driven Development is an approach to programming that starts with creating “unit tests” that reflect the business and design requirements of a program, and then writing the classes and methods that will pass the unit tests. The idea here is to help drive the programmer toward code that “stays on focus” of the needs of and solutions for a given problem.
* **Compare and contrast Test Drivers and Unit Tests**
  + Test Drivers are small programs with a main() that tests classes by creating objects, calling methods, and output results to the console.
  + Unit Tests are small “units” (which are simply methods) that calls methods on objects and classes and “tests” if they work as intended by running “asserts”. Unit Tests display the results of the test in Red/Green/Yellow results, where Red () represents run-time errors, Yellow () represents ignored tests, and Green () represents success.
* **Define the terms "false positive" and "false negative"** 
  + A “false positive” is a test result that shows success for a test which in reality should be a failure.
  + A “false negative” is a test result that shows failure for a test which in reality should be a success.
  + Both false positives and false negatives are a result of an error in the unit test (or test driver).
* **List three downfalls of using Test Drivers** 
  + Can grow in size and become bulky & difficult to use (poor scalability)
  + Requires considerable effort to generate and distinguish tests
  + Difficult to verify tests due to heavy manual intervention & verification
* **Identify four benefits of using Unit Tests** 
  + Provides simple tests to verify specific parts of a system
  + Easy to verify tests (minimal manual effort); errors are clearly identified and described
    - A clear graphical response it provided, leading to the popular development mantra "Keep it Green"
  + Increases productivity (by focusing development efforts)
  + Captures design requirements in code
  + Increases programmer confidence when modifying and expanding code
  + Easily isolate smaller parts (“units”) of the program for verification; these tests can run independently of each other (even if the other tests have compiler issues with the classes being tested).
* **Add unit testing libraries and unit testing packages to existing programs** 
  + *Demonstrate your ability to meet this learning outcome guide by completing the related assignments.*
* **Use unit tests to validate the requirements of a class’ design and behaviour**
  + *Demonstrate your ability to meet this learning outcome guide by completing the related assignments.*
* **Diagnose and correct software problems by using unit tests** 
  + *Demonstrate your ability to meet this learning outcome guide by completing the related assignments.*