**Unit test:**

Advantage

* Time is saved in not having to test methods that have been removed as the design evolved over time. What is left is what really has to get tested.
* By adding tests, this allows an opportunity to review all the aspects in the app and determine what other optimizations one could add now that a working prototype is ready.

Disadvantage:

* Large time investment to get the tests written, new functionality may be delayed for some time to generate all the tests.
* Bugs may have been introduced that the tests will discover that may cause this to be longer than initially planned.

**GUI testing:**

Advantage

* You don't have to be as familiar with the specific implementation, or even how coding works to write automated UI tests. Many tools allow you to just click record, perform some actions, and save a script.
* You are also going to find a lot more user-impacting bugs up front, because you are working directly with the application at the user's perspective.

Disadvantage

* GUI testing can require a lot of programming and is time consuming whether manual or automatic.
* GUI testing also has to be done for each individual device environment.

**Coverage measurement**

Coverage measurement helps in checking that how thoroughly the testing has been done.

By applying coverage measurement one can

* Identify the coverage items;
* Calculate the percentage of coverage items that were tested by a set of tests;’
* Report the coverage items that have not been tested yet;

Coverage measurement shows

* Which lines of code are executed.
* How much of your code is covered by your testes
* It helps to achieve reliable quality through identifying untested areas of the application.

But

The coverage measurement measures only the coverage of the items that we can identify. Because our tests are achieved 100% coverage, does not mean that our software is 100% tested!

**Exercise 1:**

**How does Visual Studio Test Explorer and Code Coverage work (the general technique used to collect measurement data)?**

* To determine what proportion of your project's code is actually being tested by coded tests such as unit tests, you can use the code coverage feature of Visual Studio.
* Code coverage is an option when you run test methods using Test Explorer. The results table shows the percentage of the code that was run in each assembly, class, and method. In addition, the source editor shows you which code has been tested.
* When you build the test project, the tests appear in Test Explorer. If Test Explorer is not visible, choose Test on the Visual Studio menu, choose Windows, and then choose Test Explorer.

**How can you interpret the results displayed in Visual Studio Test Explorer and EclEmma?**

* As you run, write, and rerun your tests, Test Explorer displays the results in default groups of Failed Tests, Passed Tests, Skipped Tests and Not Run Tests.

**Were all statements executed by your BlackBox tests? If not, why not? What were the missing**

**test cases?**

* No, new line and closing statements were not executed by the blackbox tests.

**Is it even possible to reach all statements in the given triangle.cpp with BlackBox testing?**

**If not, why?**

* No, the same reason as the blackbox is not able to execute the opening and closing statements of each methods

**Exercise 2**

**What is the difference between line coverage, condition coverage, decision coverage,**

**condition/decision coverage?**

**Line coverage**: It is a testing in which each statement in the program are executed

**Condition coverage**: It is a testing in which each one of the Boolean expression have been evaluated to both TRUE and FALSE.

**Decision coverage**: is a testing method, which aims to ensure that each one of the possible branch from each decision point is executed at least once and thereby ensuring that all reachable code is executed.

**Condition/decision coverage**: a test that requires both decision and condition coverage are satisfied

**Explain the main benefits of TDD**

* Maintainable, Flexible, Easily Extensible

#### **Unparalleled Test Coverage & Streamlined Codebase: it facilitates easy maintenance and helps alleviate scope creep**

**A short discussion about the contrast between the results of the black box testing in**

**Exercise 1 and the white box testing in Exercise 2.. In Exercise 4, is there a fair reason why the initial test cases only reached 85% coverage? Is it necessary to reach 100% coverage? What is your opinion regarding how to treat requirements of “full coverage” in realistic settings?**

* Blackbox testing is testing without knowing the internal implementation of the code where as white box testing is testing where the internal implementation of the code is known to the tester.
* It is not necessary to reach 100% coverage, since measuring test coverage does not tell anything about the quality of the tests and the code being tested

**Finally, your discussion must include a comparison between testing in C++ and Java. From your own experience; explain the differences and similarities when thinking about unit testing and coverage measurements. Which was easy/difficult to use? Why?**

Testing C++

* Testing in C++, we use Visual Studio, and we can create unit tests for unmanaged or naive codes using native test project.
* Unit tests can give a quick way to check for logic errors in the methods of classes in the code.
* A unit test can be created one time and run every time that source code is changed to make sure that no bugs are introduced.
* Coverage measurements also support C++ naive code

Testing Java

* Testing Java uses eclipse under the tool EclEmma
* Launches from within the workbench like JUnit test runs can directly be analyzed for code coverage.
* Coverage results are immediately summarized and highlighted in the Java source code editors.
* EclEmma does not require modifying your projects or performing any other setup.

For me both are directly related. The more effective unit test made, the high percentage of code coverage will be. If we have a good unit test then code coverage percentage will also be hig.

Sources:

* msdn.microsoft.com/
* http://www.eclemma.org/