Assignment 1 consists of three parts:

- Students need to make a C# console application that does what is asked in the assignment. I don't care how Students do it exactly as Students should be beyond that at this point. It simply needs to function correctly. It should not crash on incorrect inputs, etc.

- Students need to make a set of unit tests to test the methods created within their console application. Again there is no one answer to it, it depends on how Students set up their methods. But generally the methods are pretty easy in this assignment, so Students will need to create an object, run the method, then run an Assert to check the results, Assert.AreEqual() is good for probably all four asserts depending on how Students did it.

- Students need to set up a local repository using git and do at least three commits. Students just need to be as complicated as the lynda video shown in lesson 2. I tell them when to commit in the assignment. Of course Students can commit more, that is fine.

There is no one assignment to mark each against, as students are free to implement the program how Students want. When the assignment is due week 5, take some time in the class to have them present it. I have them all load the assignment, and then I walk around to each student and have them show me how it works quickly, and collect the hard copy of the assignment. I will make notes on the hard copy about what worked/what didn't work.

I have added a detailed rubric which breaks down how you can give marks for the assignment. I will go over that here in greater detail.

**Square Class**  
**1 Mark - Square class created as separate file.**

Students either do this or don't do this. If Students created the class inside their main Program.cs file, Students lose the mark.

**1 Mark - Square class side length attribute is private.**

If Students made the side length integer in the class public, Students lose the mark.

**2 Marks - Default and Non-Default constructor created and work properly.**

Students need to have both, one mark for each. Students need the default constructor even if Students don't actually use it in their program.cs. The default constructor should set the side length to 1.

**4 Marks - Four methods created, each function properly.**

One mark for each method that works as described. Students aren't very complicated methods.

**Console Application**  
**3 Marks - Square is created as described. Incorrect input is handled.**

The square object needs to be created using the non-default constructor, passing it the input from the user (1 Mark). It needs to handle correctly (not crash, ask the user to please try again) both non-numbers (1 Mark) and numbers zero or less (1 Mark).

**2 Marks - Menu Option 1 works as described.**

It should return and print to console the correct side length of the square object (2 marks).

**2 Marks - Menu Option 2 works as described. Incorrect input is handled.**

It should ask the user to enter a number and accept that number as input (1 mark). It needs to handle correctly (not crash, ask the user to please try again) both non-numbers and numbers zero or less (1 Mark for including both, since its the same code as before really).

**2 Marks - Menu Option 3 works as described.**

It should return and print to console the correct perimeter length of the square object (2 marks).

**2 Marks - Menu Option 4 works as described.**

It should return and print to console the correct area of the square object (2 marks). Students don't need to include units since it isn't mentioned. Just added "units squared" is fine if someone asks. If Students make up their own unit (ft., m, etc.) that is fine.

**1 Mark - Menu Option 5 exits the program.**

It should exit the program. This should be the only way out of the program (other than hitting the X to close the window). So it should always stay in the menu until you choose to quit (1 mark).

**Unit Tests**

**1 Mark - Unit test for GetSide() Method.**  
**1 Mark - Unit test for ChangeSide() Method.**  
**1 Mark - Unit test for GetPerimeter() Method.**  
**1 Mark - Unit test for GetArea() Method.**

One mark for completing each. All four need to have the correct Arrange-Act-Assert format. Their Act and Assert may be on the same line (for example, Assert.AreEqual(9, square.GetArea()); is both the Act [GetArea()] and Assert [the Assert.AreEqual).

Students need to include the [Test] flag before each Unit test method. I'll probably take off half a mark if Students forgot it but Students wrote the method (the method won't run in NUnit without the [Test] flag though.

**2 Marks - Unit test naming conventions.**

This is in the slides, but please emphasize it with your students. I will probably take off half a mark per mistake up to the 2 marks.

All Unit tests MUST be NameOfTheMethodTesting\_WhatTheInputIs\_WhatTheExpectedOutputIs

I realize this leads to really long method names, and that is fine, that is the point. When you have 1000 unit tests all running at once, and one fails, you need to know just from the name what was tested and what was supposed to happen. **For example:**A good unit test name for testing the GetPerimeter() method could be:

GetPerimeterOfSquareSizedThreeShouldReturnTwelve()

Anything less specific than that is wrong.

Other naming conventions Students must follow:

All the tests must be in their own project file, and the project must be named WhateverTheirProjectIsNamed.Tests

It must be .Tests, with the dot, and plural.

The .cs file that contains the tests must be WhateverTheirClassIsNamedTest

Students will mix this up, particularly in the first assignment. I have included an example project screenshot in the assignment to show what it should look like. Again I'll do a half mark off for each up to the 2 Marks.

**1 Mark - Screenshot of completed unit tests run successfully.**

One mark for using the NUnit 2.6.4 GUI and running their tests. If Students show some failing, I'll still give them the mark here for running the GUI as Students have probably already lost the marks for that earlier.

**Git**  
**1 Mark - Screenshot showing git repository log and required commits.**

Students should have at least 3 commits, each labelled with a message corresponding to when I told them to commit in the assignment. If Students commit more, it is fine. 1 Mark for this. 0 if Students don't do it.

**Programming standards deductions.**

These are deductions. If Students have no problems Students don't lose any marks. Students cannot gain marks from them.

You will need to check their C# code and mark up on their hard copy any programming standards Students have missed. Students should be familiar with this, and the standards are all listed on the IT coversheet that Students include in the assignment. I just mark which P# Students have got wrong beside the error on their hard copy. The IT coversheet says 10% each, but I am using -0.25 each as it is easier to count them up when its in quarters rather than 0.3. So -0.25 each, up to a maximum of -6. So even if Students have more than 20 errors, Students only lose 6 marks.

Common ones to look for:

P1 - Variable names not meaningful, should tell you what it is, generally be a noun ("int a" is no good).

P3 - Case conventions. Variables MUST be camalCase, method names MUST be PascalCase.

P12 - Line length, their lines of code should not wrap when printed on the hard copy. I always tell them to look for the black arrows on the right side of the assignment before submitting. If Students have a long line, Students need to break it up by pushing enter at some correct point in the line of code.

P13 - No giant blocks of white space

P23 - Unwise coding practice, anything Students shouldn't do (use goto, whatever else is not a good idea).

Those are the most common, but any on the list is fair game. Be consistent with your marking over the course of the assignments. The goal of this is to get them making professional looking code.

**Failure to present deduction.**

This is here to get them to show up and present and hand in the code. If Students email you and have a reasonable reason why Students can't make it, I will waive the deduction and download their assignment from eConestoga and mark it myself. Otherwise if Students don't present Students lose 6 marks. It takes way too long to mark programming assignments if Students don't present so this is here to minimize that.

**Hard copy not handed in or not all documents listed handed in.**

Students must hand in a hard copy and it must include all the documents asked for:

a.    Assignment Cover sheet properly filled (found on eConestoga)  
b.    Assignment Rubric left blank (found on eConestoga)  
c.    Copy of Program source code (The Program.cs in example Figure 1 screenshot)  
d.    Copy of Square class source code (The Square.cs in example Figure 1 screenshot)  
e.    Copy of Unit Test class source code (The SquareTest.cs in example Figure 1 screenshot)  
f.    Print out of Doc showing the screenshot with results of your unit tests being run, and a screenshot/output of your git repository log.

This deduction is here to get them to do that, otherwise Students get lazy. You can remove part marks if Students only forget one page or whatever. If Students can't make it for a reasonable excuse, I will get them to drop the hard copy off at my office. You guys can get them to drop it off next class, or just print it off from eConestoga.

Finally, Students must submit their documents, and the entire zipped Solution folder to eConestoga. Tell them to zip the entire solution folder, not just the source code. Since I already have seen their assignment run, and have a hard copy, I just use this for a timestamp of when Students submitted. Or if the student could not make it to class, I download and run it.