Software Testing

Unit Testing

- Ensure that every line of code does exactly what it is supposed to do.
- Change anything? You must do ***regression testing*** to make sure you didn't break anything.
- A "unit" = the smallest possible unit of code behavior that can be tested in isolation.
- Test the code for
  - Handling expected inputs properly
  - Handling unexpected inputs properly
  - Graceful error handling
  - Edge case: handling the extremes
    - 20 bytes in a 20-byte field
    - 21 bytes in a 20-byte field
    - Character data in a numeric field
    - Divide by zero

Software Testing

Why Automate Unit Testing?

- **Speeds up Unit Testing**
- **Enables Speedy, Reliable Regression Testing**
- **Ensures that Function meets Design**

Software Testing

How to Automate Unit Testing?

Use a “Testing Framework” to develop some testing software that will do this:

For a given action in your code:

- **Determine the expected value** (the value which should be produced if the software is working correctly)
- **Determine the actual value** (the value which the software is actually computing)
- **Compare the two**:
  - If they agree, the test passes
  - If they disagree, the test fails

Software Testing

Automated Unit Testing Frameworks:

- JUnit Tutorial: http://clarkware.com/articles/JUnitPrimer.html
- PyUnit: http://wiki.python.org/moin/PyUnit
- Python’s unittest https://docs.python.org/3/library/unittest.html
- PhpUnit: https://phpunit.de/
- Unit testing with C#: http://www.csunit.org/tutorials/tutorial7/
- Unit testing in Objective-C and Xcode:

http://developer.apple.com/mac/articles/tools/unittestingwithxcode3.html

- Unit testing for Ruby:

http://www.ruby-doc.org/stdlib/libdoc/test/unit/rdoc/classes/Test/Unit.html

Software Testing

Refactoring:

Code refactoring is the process of restructuring existing computer code without changing its external behavior.

Refactoring improves nonfunctional attributes of the software:

- Make it more Readable
- Make it Concise: Smallest possible size
- Reduces Complexity
- Improves Maintainability

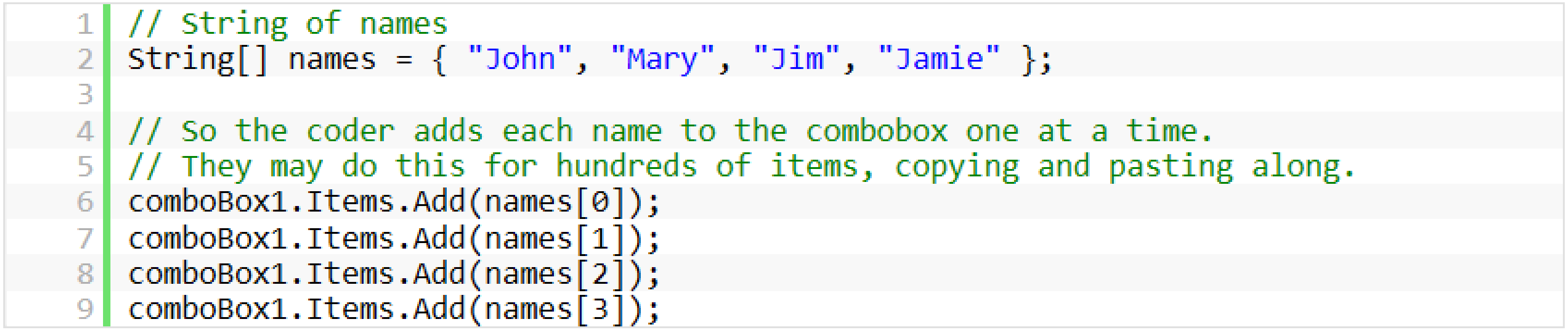
Software Testing

Refactoring:

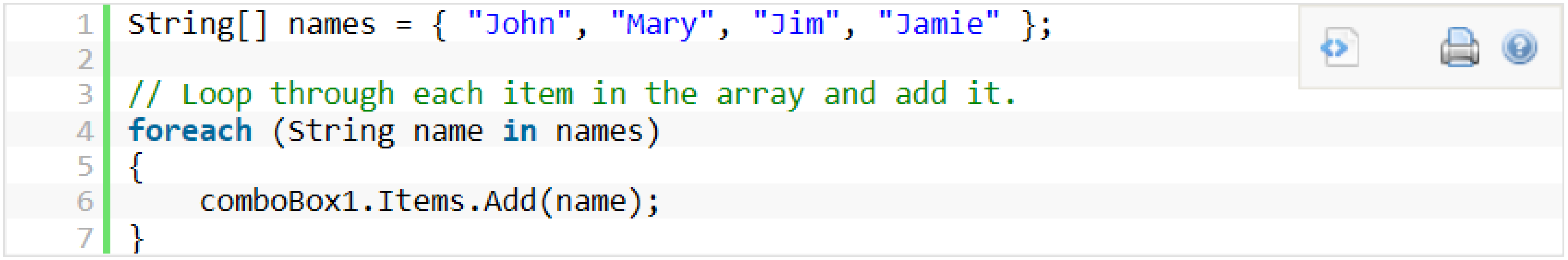
You must ensure that you have solid, repeatable, automated unit tests to ensure that refactoring does NOT impact code functionality AT ALL.

Refactoring Tips

Tip 1 – Look for multiple lines virtually doing the same thing



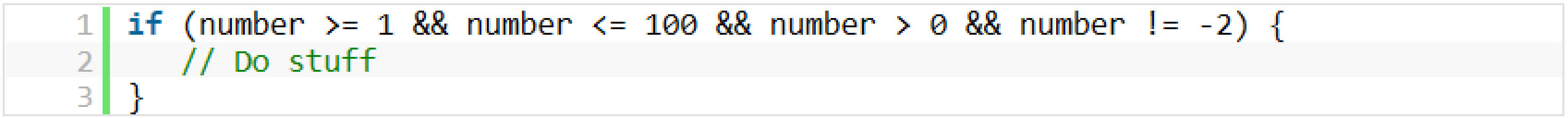
```
// String of names
String[] names = { "John", "Mary", "Jim", "Jamie" };

// So the coder adds each name to the combobox one at a time.
// They may do this for hundreds of items, copying and pasting along.
comboBox1.Items.Add(names[0]);
comboBox1.Items.Add(names[1]);
comboBox1.Items.Add(names[2]);
comboBox1.Items.Add(names[3]);
```



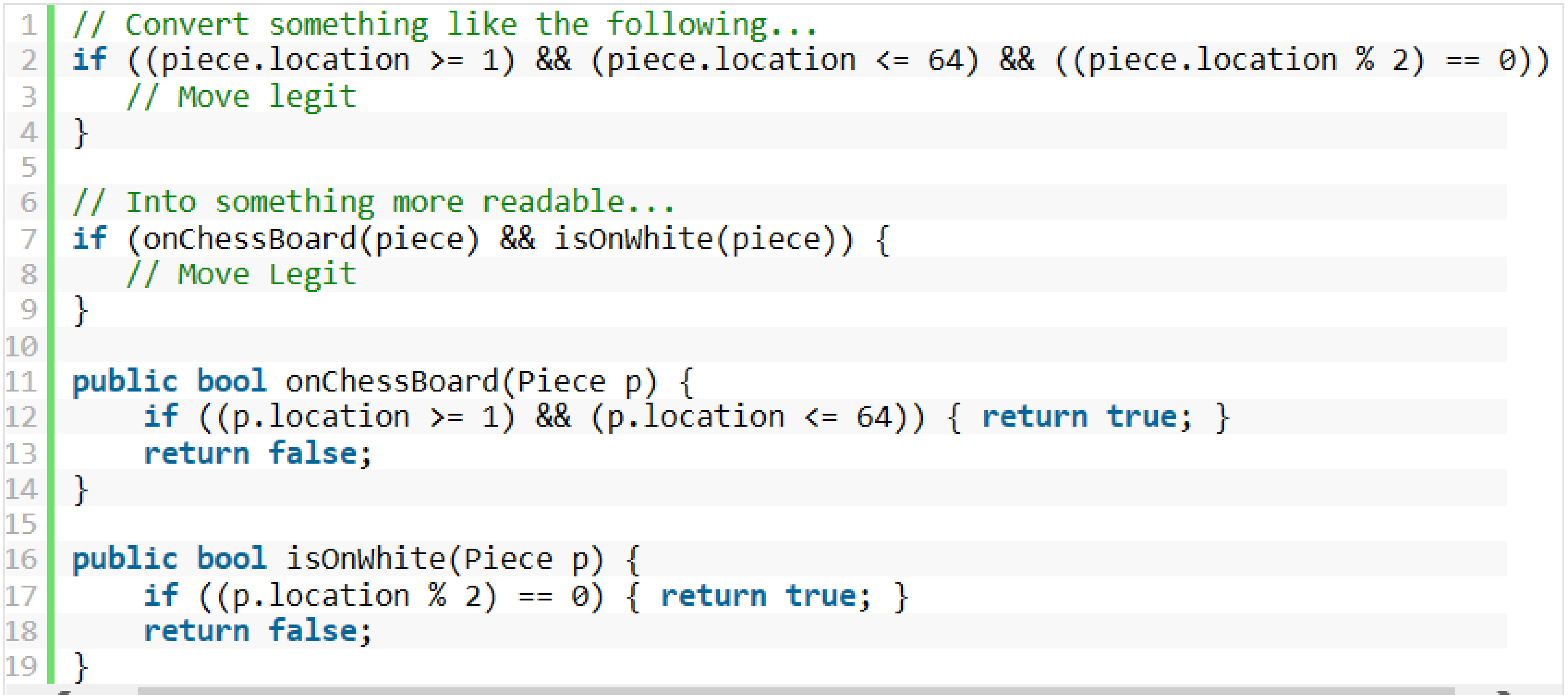
```
String[] names = { "John", "Mary", "Jim", "Jamie" };

// Loop through each item in the array and add it.
foreach (String name in names)
{
    comboBox1.Items.Add(name);
}
```

Refactoring Tips & Examples

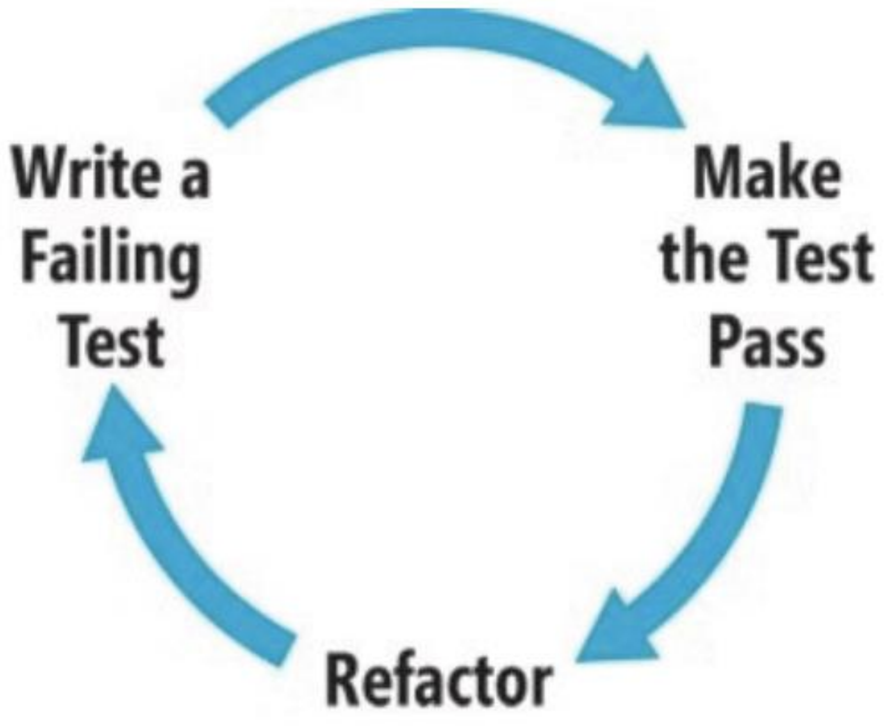
Tip 2 – Cut down complex conditionals



```
if (number >= 1 && number <= 100 && number > 0 && number != -2) {
    // Do stuff
}
```



```
// Convert something like the following...
if ((piece.location >= 1) && (piece.location <= 64) && ((piece.location % 2) == 0))
    // Move legit
}

// Into something more readable...
if (onChessBoard(piece) && isOnWhite(piece)) {
    // Move Legit
}

public bool onChessBoard(Piece p) {
    if ((p.location >= 1) && (p.location <= 64)) { return true; }
    return false;
}

public bool isOnWhite(Piece p) {
    if ((p.location % 2) == 0) { return true; }
    return false;
}
```



TDD

Software Testing

- Traditionally (i.e. “waterfall”) large systems are designed and coded up front, then tested by QA teams when coding is done.

Test Driven Development “TDD”

- TDD is a developer process of writing unit-tests first, then writing code to pass the tests.
- Tests are executable requirements/specifications
- End-result is a complete system (working code) with corresponding, automated unit tests
  - Assist with regression testing
  - Enable refactoring
- NO CODE is ever written without a test being created first

Software Testing

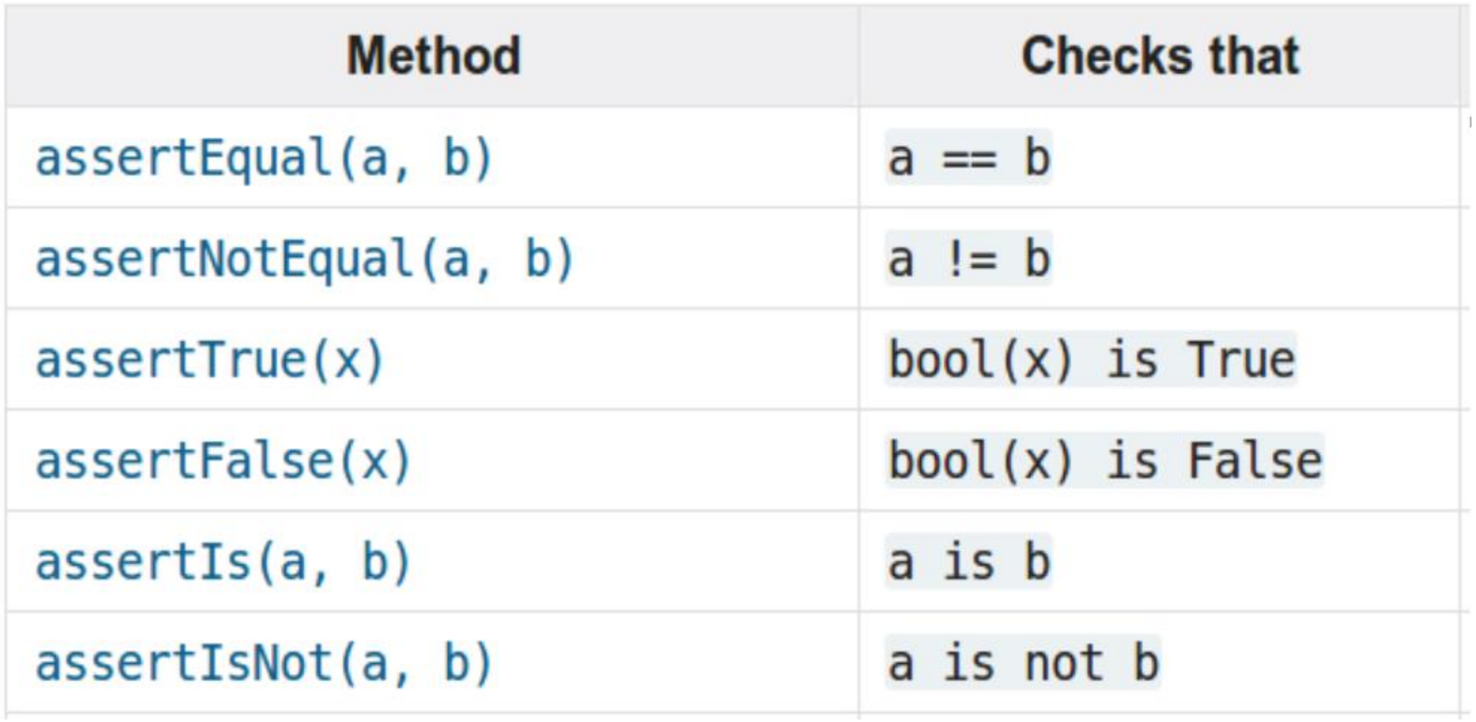
TDD

- *First* write the test, *then* do the design/implementation
- Part of agile approaches like XP (Extreme Programming)
- Supported by testing framework tools (like Junit, PyUnit)
- TDD Is more than a mere testing technique; it incorporates much of the detail design work
- Useful for many code behaviors, but not really for GUI or Database functionality
- It is not a replacement for UA, System, Performance testing

Software Testing

TDD

- Assertions: a method that allows verification of ACTUAL results versus EXPECTED results



Method	Checks that
`assertEqual(a, b)`	`a == b`
`assertNotEqual(a, b)`	`a != b`
`assertTrue(x)`	`bool(x) is True`
`assertFalse(x)`	`bool(x) is False`
`assertIs(a, b)`	`a is b`
`assertIsNot(a, b)`	`a is not b`

Software Testing

TDD Step-By-Step

1. Write a single test
2. Run that test, system fails *(because the actual code is not written yet)*
3. Write a “stub” of the code function
4. Run that test, system fails *(because the stub doesn’t do anything)*
5. Fill in the stub with code to make the test pass
6. Run the test again, and verify that the code runs properly
7. Refactor the code as needed to improve its design
8. Run the test again to ensure clean regression test

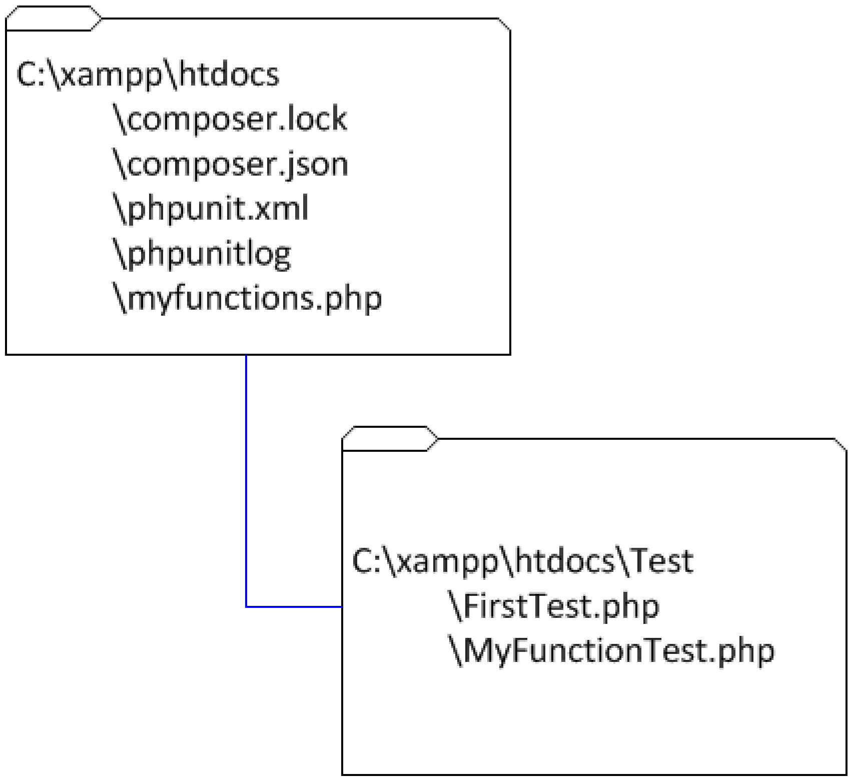
Software Testing

Let’s Look at an Example:

- Using “phpunit” in my XAMPP environment on my PC
  (Apache, MySQL, Php)
- I used Composer to Install phpunit from https://phpunit.de/
- It runs from within my c:\xampp\htdocs folder
  (from where apache executes programs for the localhost web server)
- I tell phpunit what is my expected result, and it compares that to the actual result of executing my code
- Uses an “assertion” to measure the outcome
  - `assertEquals(expected result, actual result);`

Software Testing

Here's my setup:


```
C:\xampp\htdocs
        \composer.lock
        \composer.json
        \phpunit.xml
        \phpunitlog
        \myfunctions.php

C:\xampp\htdocs\Test
        \FirstTest.php
        \MyFunctionTest.php
```

Software Testing

C:\xampp\htdocs

- \composer.lock – composer config info
- \composer.json – composer metadata
- \phpunit.xml – phpunit config info
- \phpunitlog – we write the output of the phpunit tests here
- \myfunctions.php – a php program defining functions used in testing

C:\xampp\htdocs\Test

- \FirstTest.php – simple boolean true/false
- \MyFunctionTest.php – simple math A + B

Software Testing

FirstTest.php

```
<?php # FirstTest.php

class FirstTest extends \Phpunit_Framework_TestCase
        {
                public function testUselessness()
                        {
                                $this->assertTrue(true);
                        }
        }

?>
```

Software Testing

MyFunctionTest.php

```
<?php
class MyProceduralTest extends \Phpunit_Framework_TestCase {
/*
 * Testing the addition function
*/
          function my_addition($arg1, $arg2){
        return $arg1 + $arg2;
    }
public function testAddition(){
         include('my_functions.php');
         $result = my_addition(4,1);
         $this->assertEquals(6, $result);
  }
}
?>
```

Software Testing

1. Edit the code to enter my expected result

2. Invoke phpunit, with option to write results to log file

For this week’s lab, you will do basically the same thing, but using a different framework – “Python3UnitTest” -- and running in your lab VM environment.