


Unit Testing

Or, “How to Fill Out a T.P.S. Report”



T.P.S. REPORT
COVER SHEET

Prepared By: _____ Date: _____

Device/Program Type: _____

Product Code: _____ Customer: _____

Vendor: _____

Due Date: _____ Data Loss: _____

Test Date: _____ Target Run Date: _____

Program Run Time: _____ Reference Guide: _____

Program Language: _____ Number of Error Messages: _____

Comments: _____

C O N F I D E N T I A L

Goal of This Course

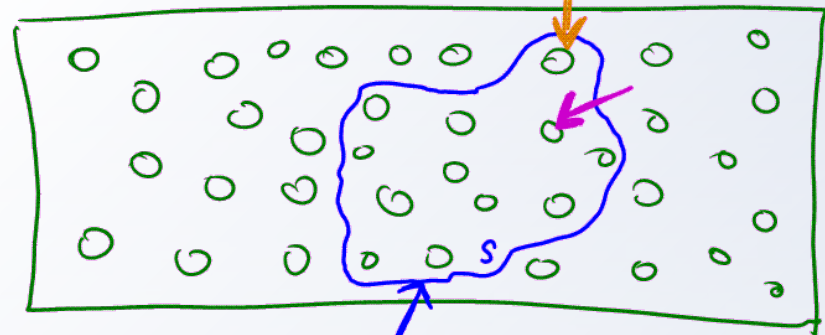
- The goal of this course is to help you become a better software engineer.
- Write Better Software, Better:
 - Easier Development Process
 - Higher Quality Result
- Testing your software and designing with testing in mind will catch bugs and result in a better design.

Let's Talk About Requirements

- We've talked about consciously designing software with desirable design properties: ease of change, reusability, security, etc.
- But don't forget: we still need the software to work!
- This lecture is about making sure the software you design works: meets the requirements.

For a given set of functional requirements, there are many possible ways to implement those requirements in software.

FULL SPACE OF ALL POSSIBLE SOFTWARE SOLUTIONS YOU COULD EVER WRITE



Requirements
 R

(subset S
includes all
software solutions
that meet R)

$$|S| > 1!$$

How To Ensure Your Program Meets Its Requirements?

- Stare at your code for a while and convince yourself it works
 - Easy to make mistakes
 - Easy to be lazy
 - Suffers from “tunnel vision”
 - Most software is “tested” this way
- Prove that it is correct
 - Difficult (very)
 - Not always possible
- Ship it and wait for your customers to complain
 - Not very nice

How To Ensure Your Program Meets Its Requirements:

In a word: **Testing.**

Testing is a way to increase your confidence that your program meets its requirements.

Making sure the program does what it's supposed to do

Making sure that you know what it's supposed to do!

Testing ensures that your software solution is among those that meet the requirements.

Know Thy Requirements

This Is Important:

In order to test whether code meets its requirements, you have to know what those requirements are.

Unit Testing: Meeting Requirements 1 Module At A Time

Unit testing tests each module in isolation to increase your confidence that it meets its requirements.

This in turn increases your confidence that the whole system will meet its requirements when the modules are put together.

For practical purposes:

Unit == Module == Class

Testing Against a Contract

What to test?

1. Before you can test a unit, you have to know what it is supposed to do.
Be very clear on what the unit and each of its methods are supposed to do.
2. Write this down precisely in the Javadoc. This is the unit's ***contract***.
3. Write test code to make sure the unit meets its contract.

Contracts: Stack Example

```
java.util.Stack<E>
```

```
/** Pushes an item onto the top of this stack. */
```

```
public E push(E item) ;
```

```
/** Removes the object at the top of this stack and  
    returns that object as the value of this function.  
    */
```

```
public E pop() ;
```

```
/** Looks at the object at the top of this stack  
    without removing it from the stack. */
```

```
public E peek() ;
```

Unit Testing

How to test a unit (a.k.a. module, a.k.a class)?

A class is a bundle of capabilities, specified in its contract and embodied in its public methods. We test the contract of a class by testing those methods.

We test a method by calling it with sample inputs and checking that it produces the correct result.

Important: we both call the method and check its results *automatically*.

Automated Testing

Testing by hand is tedious, slow, error-prone and not fun.

Would you want to do something matching that description?

Automated testing to the rescue: write code to test your code.

Unit testing is automated testing: you write code to set up the unit (class), call its methods with test inputs, and compare the results to the known correct answer.

Once tests are written, they are easy to run, so you are much more likely to run them.

Black Box Testing

- You can write unit tests just by looking at the contract for the unit.
- You don't ever have to see the source code for a method to write a test for it.
- You don't have to be the author of a method to author a test for it.
- ...*if* you have a precise specification of the method's contract.
- This is called “black box” testing – you treat the method as a black box whose inner workings are unknown. You only know what it is supposed to do from its “instruction manual” -- its contract.

Black Boxes Hide Information

- You don't have to know anything about the implementation of a module to test it
- This should sound familiar: it's yet another benefit of information hiding
- You can change the implementation of a module without having to touch the tests
- You can change the tests without disturbing the module (or even having access to it)

Unit Testing is Black Box Testing

- Stick in input, crank the handle, out comes result.
- Then compare the result with the known correct result.
- How do you know the correct result? It's in the contract.

A Framework For Unit Testing In Java: JUnit

- You could go write a bunch of boilerplate code for defining, enumerating, running, logging, and analyzing unit tests yourself.
- Or you could download a ready-to-use, well-tested (!), industry-standard testing framework.
- **JUnit** is such a framework for Java code.
- JUnit is the Java member of a loosely-related family of unit testing frameworks called **xUnit**. Chances are, there is an xUnit framework for your favorite value of x.

Advantages of JUnit

- You don't have to reinvent the wheel
- It's Standard: everybody writes their tests the same way, so everybody knows:
 - Where to find tests
 - How to run tests
 - How to interpret results
 - How to add/modify tests
- Testing is easily automated
- Integrates with your favorite IDE

Show Me The Code

- Now would be a good time for a live demo in Eclipse, don't you think?

Design for Test

- Think about a method's contract *while you are designing the method*. Think about how you would test it. Write testable code.
- Which is easier to test?

```
/** Prints s to standard out in reverse. */  
public void reverse(String s) ;
```

```
/** Returns the reverse of s. */  
public String reverse(String s) ;
```

Testing Tips

- Test Early
 - Bugs are easier and cheaper to fix when there is less code. Testing up front is less work than debugging later. “Code a little, test a little.”
- Test Often
 - Run your tests every time you compile and execute
- Test Automatically
 - You are much more likely to test if it's easy
- Find Bugs Once
 - Then code up a unit test to catch 'em in the future
- Work the corner cases

You can find these tips and many more in *The Pragmatic Programmer* and *Pragmatic Unit Testing*

by Andrew Hunt and Dave Thomas

When to Write Tests

- You can write a test as soon as you have a contract
- You can write tests before any code exists
 - Often, this helps you clarify exactly what to put in the module contract
- You can write tests while you write code
 - You can co-design a module and its tests together, in an iterative fashion
- You can write tests immediately after writing the code
 - Instant gratification: you have something on which to run the tests
- *Do not* wait until the whole program is done
 - You will either run out of time or inclination to test
 - Testing is a *design aid* – it will not only catch bugs but help you design better modular, information-hiding code

How To Test Testing Code

Test code is code too: it can have bugs.
How to test the testing code?

Break (a copy of!) the production code and see if the tests catch it.

Take Home Message

Testing can help you catch bugs and produce higher-quality software.

Thinking about testing while designing your software results in a better design and an easier development process.

In order to make sure code does what it's supposed to, you first have to know what it's supposed to do (requirements).

Automated unit testing is the way to go – write code to test your code.

JUnit is a standardized framework that helps you test Java code.