

Testing

An Exhaustively Research Presentation

```
if(itNotWork)
{
    cout << "Oh Crap" << endl;
}
```

What do we mean by 'Testing'?

- Unit Testing
 - Small class or single routine.
- Component Testing
 - Large class or subsystem.
- Integration Testing
 - 2 or more packages, or subsystems.
- Regression Testing
 - Unit Testing, Component Testing, or Integration Testing... again. Just to be sure.
- System Testing
 - Complete program.

White Box vs Black Box

- Tester does not know how the code works.
- As a programmer, you are expected to take part in White Box testing.
- Tester does not know how code works.
- Testing done by non-programmers/community college graduates.

Why Developers Are Not Good Testers

- Testing breaks programs.
- Testing cannot prove a program is bug free.
- Testing does not improve programs; it shows developers what needs to be improved.
- Testing requires developers to be humble enough to think maybe, just maybe, there could be errors in their code.

Write Tests First!

- Studies have shown that projects can reduce the costs of defects by writing test cases before writing the code.
- Testing first requires the same amount of work as testing last.
- Forces developers to understand design and requirements before code is written.

I AM GLAD
WE TESTED
FIRST



WHY IS MY
HAIR BLUE?

Limitations of Developer Tests

- Developers tend to only do 'Clean' tests.
 - A Clean Test checks if the code works with expected data.
 - A Dirty Test attempts to break the code with unexpected data.
 - Most developers have only 1 Dirty Test for 5 Clean Tests. The ratio should be 5 Dirty to 1 Clean.
- Developers are optimistic.
 - Of course my tests are comprehensive!

How To Test

It is impossible to test every possible case.
That's why we have different testing techniques!

- Structured Basis Testing
 - Can we get every line of code to execute successfully?
- Data-Flow Testing
 - How is our data being used within our program?
- Equivalence Partitioning
 - How can we reduce the number of tests?
- Error Guessing
 - “Let's see if 'ab3\$4/=-13 nbe!%234\$2' breaks the calculator!”
- Boundary Analysis
 - Testing booleans the smart way!

Structured Testing

We start with 1 path and add 1 path for every `if`, `while`, `repeat`, `for`, `and`, `or`. Also add 1 path for each case in a case statement, plus an additional path if the case statement has no default.

LET'S PRACTICE

HOW MANY TESTS DO WE NEED?

```
...
DisplayAdapter da = new DisplayAdapter();
for(da.availableFrames() < 12)
{
    if(da.isFree() && da.bufferInitialized())
    {
        da.fillFrame(imageArray);
        da.copyBuffer();
    }

    int framesStored = da.availableFrames - 1;
}
if(!da.bufferEmpty())
{
    DisplayBuffer db = da.getBuffer();
}else
    DisplayBuffer = new DisplayBuffer();
...
```



5!

We have 5 paths through the code!

Ah Ah Ah Ah!

Data Flow Testing

We can describe the state of our data to check for potential problems.

- Defined
- Used
- Killed (aka deallocated)

Describing the behavior of routines helps us check for problems too!

- Entered
- Exited

We now combine these terms to check for problems.

a small sample of

Problem States

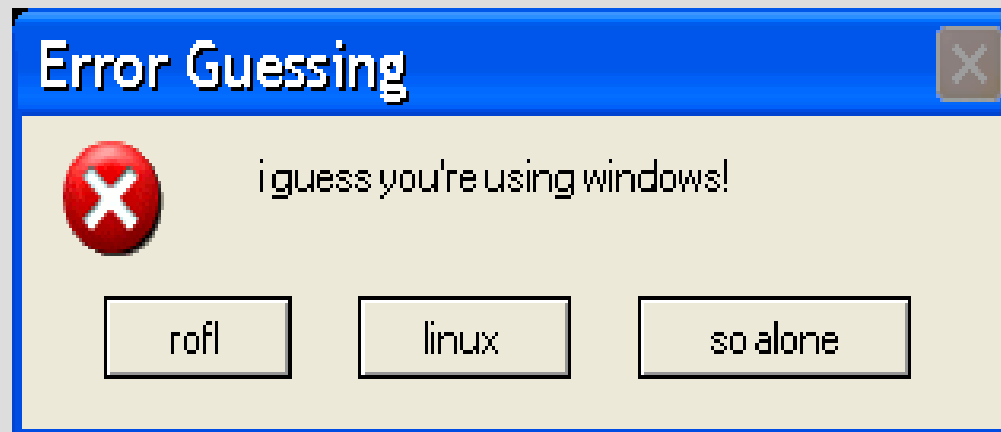
- Defined-Exited
 - We defined a local variable and didn't use it.
- Defined-Killed
 - We defined a variable and killed it before use!
- Killed-Killed
 - We got rid of our variable... twice?
- Killed-Used
 - Dead variables tell no values.
- Used-Defined
 - We used data before it was defined.
- Entered-Killed / Entered-Used
 - If we aren't using local variables, this is bad.

Equivalence Partitioning

“Equivalence Partitioning” is a fancy name for only testing the part of the program you're testing.

Error Guessing

- Test what you think will cause problems.
- Check basic conditions.
 - 0 values.
 - 1 value.
 - Infinite values.



Boundary Analysis

- When checking booleans, we can minimize our test values.

```
If (oldLady.numCats() => 0 && oldLady.numCats() < 15)  
    oldLady.setCrazy(false);
```

- We only need to check 3 values:
 - -1 for low outside our bounds.
 - 5 for inside our bounds.
 - 16 for high outside our bounds.
- A comprehensive test would check 0 and 15 as well.

Creating Test Data

Bad vs Good

- Too little data.
- Too much.
- The wrong kind.
- The wrong size.
- Uninitialized data.



- Expected values.
- Minimum expected configuration
- Maximum expected configuration.
- Old data.

Note!

You should double check values by hand. Pick easy test numbers so that if your hand calculation does not match your system, you can be sure the system is wrong and not you!

The Truth About Errors

- Most errors are pretty small and caused by programmers, not designers.
- 36% of errors are clerical errors.
- 16% of errors are caused by programmers who don't understand the design of the software.
- 85% of errors can be fixed in less than a few hours.

See Code Complete 520, 521, and 522 for more statistics on the nature of errors in programs.

How to Avoid Errors

- Check your work.
- Plan how to test your code while designing it.
 - You are designing it before coding it, right?
- Keep your test cases available to test again on new versions of the software.
- Use a testing framework likeJUnit.

So How Should We Test?

- Test Scaffolding
 - 'Mock Object' Scaffold is a class connects to the class being tested.
 - It allows you to keep track of when and where in the code errors occur, with what values, and can print this information out or store it to a log.
 - 'Fake Routines'
 - Using the class we want to test, feed in data interactively or from a set list of potential data.



So How Should We Test?

- Test-Data Generators
 - Using random data uses data you wouldn't think to test.
 - Test-Data Generators allow testers to focus on realistic tests while the generators focus on massive amounts of unlikely cases.
- Code Coverage Monitor
 - Tells you how many lines of code you've run.
 - Testing without Code Coverage generally only tests 50-60% of code.
 - A good way to measure how thoroughly you've tested your program.

So How Should We Test

- Symbolic Debuggers
 - Allows you to run the code line by line, step by step, day by day.
 - Feature of most IDEs.
- System Pertubers
 - Makes your test machine unstable.
 - Does you program function when memory is:
 - Full?
 - Constantly changing?
 - Failing?
 - Out of bounds?
- Error Databases
 - Helpful for diagnosing, recording, and tracking errors.

The Keys to Testcess!

- Plan to test!
- Test it again!
- Write test cases before you code!
- Be smart about the tests you run!
- Let automation run bigger, less likely test cases!
- I made the word Testcess up.
- Keep records of the tests you do.
- Test it again!
- Test it again!
- Seriously.

This Was the #2 Google Image Result for 'The End'

