COM4506/6506: Testing and Verification in Safety Critical Systems

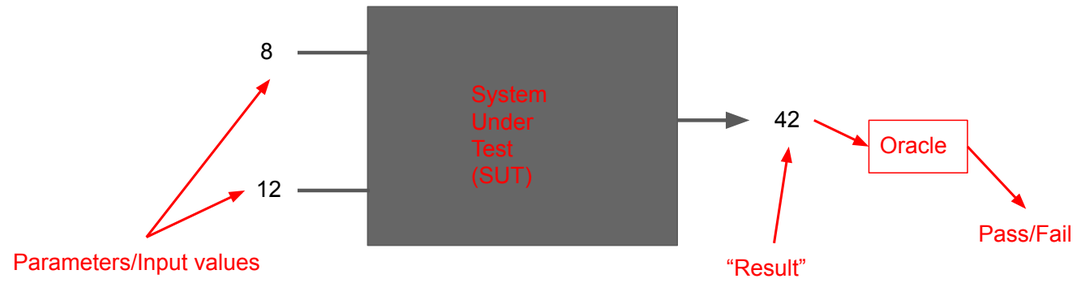
Dr Ramsay Taylor

Contents

- “Unit” testing
- Test case strategies
- Test limitations

Testing Objectives

Testing Objectives



Testing Objectives

“Testing shows the presence, not the absence of bugs.”

Edsger Dijkstra, 1969

Our tests should be designed to *provoke* failures.

Test Case Selection

```c
float current(float voltage, float resistance) {
    float result;

    /* I = V/R */
    result = voltage / resistance;

    return(result);
}
```

- “Whitebox” testing:
  - Specific cases from the spec
  - Code Coverage*
  - Expected error conditions
  - “Obvious” problems?
  - Static Analysis identified potential problems.

Test Case Selection

- “Blackbox” testing:
  - Specific cases from the spec
  - ~~Code Coverage*~~ Specification coverage?
  - Expected error conditions
  - ~~“Obvious” problems?~~ Usual problems with “this sort of thing”
  - ~~Static Analysis identified potential problems.~~ Search Based Testing

Metamorphic Testing

We don’t know the right answer...

```
sin(  3.000000) =   0.141120
sin(  3.141590) =   0.000003
sin(  6.283100) =  -0.000085
sin(  4.000000) =  -0.756802
sin( -0.858407) =  -0.756802
sin( -1.500000) =  -0.997495
```

Metamorphic Testing

```
sin(  3.000000) =   0.141120
sin(  3.141590) =   0.000003
sin(  6.283100) =  -0.000085
sin(  4.000000) =  -0.756802
sin( -0.858407) =  -0.756802
sin( -1.500000) =  -0.997495
```

We don't know the right answer…

...but we know something about the function:

However, we do know that $sin(\pi - x) = sin(x)$

In Neil's slides from last year… (also in the Wikipedia page on Metamorphic Testing!)

Metamorphic Testing

```
sin(  3.000000) =   0.141120
sin(  3.141590) =   0.000003
sin(  6.283100) =  -0.000085
sin(  4.000000) =  -0.756802
sin( -0.858407) =  -0.756802
sin( -1.500000) =  -0.997495
```

We can make tests that compare multiple results from the function:

```
    x = 4;
    sx = sin(x);
    printf("sin(%10f) = %10f\n",x,sx);
    x = PI-x;
    sx = sin(x);
    printf("sin(%10f) = %10f\n",x,sx);
```

Metamorphic Testing

This is far from perfect! There might be other functions that have the same property:

```
➜  COM4506 java SinTest
Does sin(4) == sin(PI-4)? true
```

Metamorphic Testing

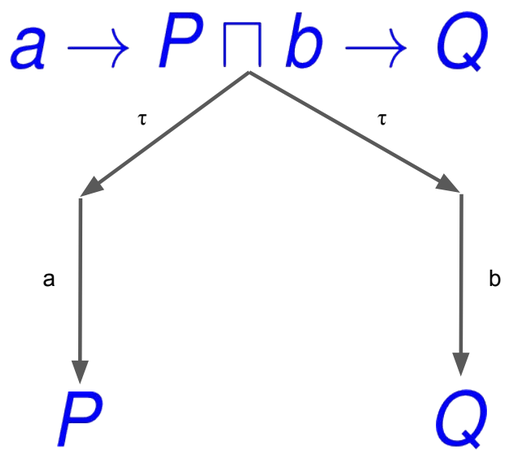
This is far from perfect! There might be other functions that have the same property:

```
➜  COM4506 java SinTest
Does sin(4) == sin(PI-4)? true
```

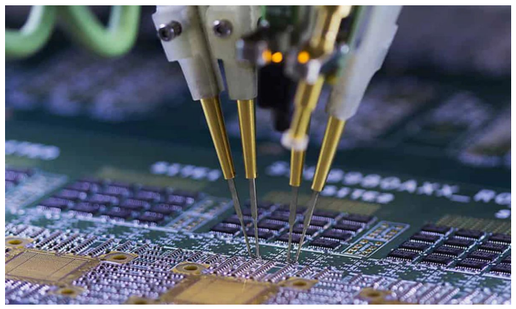
```java
public class SinTest {
    public static double sin(double x) {
        return(0.0);
    }

    public static void main(String[] args) {
        System.out.println("Does sin(4) == sin(PI-4)? " + (sin(4) == sin(Math.PI - 4)));
    }
}
```

Test Case Limitations

- Non-Determinism
  - Random Number Generators
  - Multi-threaded or multi-processor systems
  - Internal state reset
  - External “state”



Test Case Limitations



- Parameter Control
  - Hardware Interactions
  - Stub code

Test Case Limitations

- Timing
  - Expected timeout
  - Unexpected timeout?
  - Time-dependent behaviour

Summary

- Choosing tests for our SUT is important…
- We want to *cause* faults to occur if we can.
- Whitebox testing lets us use the details of the code to direct our testing
- Blackbox testing doesn’t - but there are still sensible strategies
- Metamorphic testing uses properties of the SUT as well as explicit results.
- … but there are limits to what any of this can do.