White-Box Testing

White-Box Testing - Review

- You know the code
  - Given knowledge of the internal workings, you thoroughly test what is happening on the inside
  - Close examination of procedural level of detail
  - Logical paths through code are tested
    » Conditionals
    » Loops
    » Branches
  - Status is examined in terms of expected values
  - Impossible to thoroughly exercise all paths
    » Exhaustive testing grows without bound
  - Can be practical if a limited number of “important” paths are evaluated
  - Can be practical to examine and test important data structures
**Build scaffolding for incomplete code**

Stubs and drivers are code that are (temporarily) written in order to unit test a program

- **Driver** is a software module used to **invoke a module under test and often, provide test inputs, controls, and monitor execution and report test results**
  ```java
  public void movePlayer(Player player, int diceValue) {
    player.setPosition(1);
  }
  ```

- **Stub** is a module that simulates components that aren't written yet, formally defined as a *computer program statement substituting for the body of a software module that is or will be defined elsewhere*
  ```java
  public void movePlayer(Player player, int diceValue) {
    player.setPosition(1);
  }
  ```

---

**Devising a prudent set of test cases**

- **Equivalence Class/Boundary Value Analysis**
  - Still applies!
- **Basis (Path) Set**
- **A metric for assessing how good your test suite is**
  - Method Coverage
  - Statement Coverage
  - Decision/Branch Coverage
  - Condition Coverage
- **Think diabolically**
Basis Set

- Compute Cyclomatic number - V(G)
  - This gives us an estimate of how many tests must be designed and executed to guarantee coverage
  - The number of independent paths that must be tested to ensure that all statements have been executed at least once and every condition will have been executed on its true and false side!!

- Edges – Nodes + 2

Flow Graph

Boolean phrases with sub-expressions are broken into pieces:
(if (x >= 5) AND (x <= 10))

Key

- Decision
- Statements
**Compound Predicate**

```
©L. Williams
```

```
100% Method Coverage

- All methods in all classes have been called
- Test case 1: Foo(0, 0, 0, 0, 0) = 0.0
- float foo (int a, b, c, d, e) {
  if (a == 0) {
    return 0.0;
  }
  int x = 0;
  if ((a==b) OR ((c==d) AND bug(a) )) {
    x =1;
  }
  e = 1/x;
  return e;
}
```

©L. Williams
100% Statement Coverage

- All lines in a method have been executed
- Test case 2: Foo(1, 1, 1, 1, 1) = 1.0
- float foo (int a, b, c, d, e) {
  if (a == 0) {
    return 0.0;
  }
  int x = 0;
  if ((a==b) OR ((c==d) AND bug(a) )) {
    x =1;
  }
  e = 1/x;
  return e;
}

100% Branch/Decision Coverage

- All predicates have been true and false
- Test case 3: Foo(1, 2, 1, 2, 1) ← division by zero!
- float foo (int a, b, c, d, e) {
  if (a == 0) {
    return 0.0;
  }
  int x = 0;
  if ((a==b) OR ((c==d) AND bug(a) )) {
    x =1;
  }
  e = 1/x;
  return e;
}
100% Condition Coverage

- All sub-expression predicates have been true and false
- Test case 4: Foo(1, 2, 1, 1, 1) ← a bug!
- float foo(int a, b, c, d, e) {
  if (a == 0) {
    return 0.0;
  }
  int x = 0;
  if ((a==b) OR ((c==d) AND bug(a)) {
    x = 1;
  }
  e = 1/x;
  return e;
}

<table>
<thead>
<tr>
<th>Predicate</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a==b)</td>
<td>Test Case 2</td>
<td>Test Case 3</td>
</tr>
<tr>
<td></td>
<td>foo(1, 1, x, x, 1)</td>
<td>foo(1, 2, 1, 1, 1)</td>
</tr>
<tr>
<td>(c==d)</td>
<td>return value 0</td>
<td>division by zero!</td>
</tr>
<tr>
<td>bug(a)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Coverage

Example free tools:
- djUnit
- eclEmma
Loops

- Write a test case such that you:
  - Don’t go through the loop at all
  - Go through the loop once
  - Go through the loop twice
  - Go through loop max times
  - Try to go through look max+1 times

Test-driven Development

Unit Test
(every few lines of code)

Acceptance Test
(every feature)

Features

Feature Delivery
**Mantra: Red-Green-Refactor**

![Diagram of the Mantra: Red-Green-Refactor](http://www.agileprogrammer.com/uploads/bradwils/red_2Dgreen_2Drefactor.png)

---

**IBM and Microsoft TDD**

<table>
<thead>
<tr>
<th>Metric Description</th>
<th>IBM: Drivers</th>
<th>Microsoft: Windows</th>
<th>Microsoft: MSN</th>
<th>Microsoft: VS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defects/KLOC of comparable team in organization but not using TDD</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>Actual defects/KLOC (using TDD)</td>
<td>0.61W</td>
<td>0.38X</td>
<td>0.16Y</td>
<td>0.09Z</td>
</tr>
<tr>
<td>Increase in time taken to code the feature because of TDD (%) [Management estimates]</td>
<td>15-20%</td>
<td>25-35%</td>
<td>15%</td>
<td>25-20%</td>
</tr>
</tbody>
</table>
### TDD vs. Test-last Unit Test Automation

<table>
<thead>
<tr>
<th>Metric Description</th>
<th>IBM: Drivers TDD</th>
<th>Microsoft: Windows TDD</th>
<th>Microsoft: MSN TDD</th>
<th>Microsoft: VS TDD</th>
<th>Microsoft: App Test-Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defects/KLOC of comparable team in organization but not using TDD</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>A</td>
</tr>
<tr>
<td>Actual defects/KLOC (using TDD)</td>
<td>0.61W</td>
<td>0.38X</td>
<td>0.16Y</td>
<td>0.09Z</td>
<td>.80A</td>
</tr>
<tr>
<td>Increase in time taken to code the feature because of TDD (%) [Management estimates]</td>
<td>15-20%</td>
<td>25-35%</td>
<td>15%</td>
<td>25-20%</td>
<td>30%</td>
</tr>
</tbody>
</table>

### Devising a prudent set of test cases

- **Equivalence Class/Boundary Value Analysis**
  - Still applies!
- **Basis Set**
- **A metric for assessing how good your test suite is.**
  - Method Coverage
  - Statement Coverage
  - Decision/Branch Coverage
  - Condition Coverage
- **Think diabolically**