

# Testing Methods: White Box Testing I

---

## Outline

- Today we begin to look at **white box** testing
- We'll look at:
  - white box vs black box
  - role and **kinds** of white box testing
  - implementation - source, executable and sampling
  - code **coverage** testing
    - **statement** coverage
    - **basic block** coverage

# White Box vs. Black Box

---

## Recall: Kinds of Tests

- We divide tests into:

Black Box  
Testing  
Methods

White Box  
Testing  
Methods

- **Black box** methods – cannot see the software code (it may not exist yet!) – can only base their tests on the **requirements** or **specifications**
- **White box** (aka glass box) methods – can see the software's code – can base their tests on the software's actual **architecture** or **code**

# Kinds of White Box Testing

---

## Code Coverage

- Code coverage methods design tests to **cover** (execute) every method, statement or instruction of the program at least once

## Logic Path/Decision Point Coverage

- Logic path methods design tests to cover every **path of execution** in the program at least once

## Data & Data Flow Coverage

- Data coverage methods explicitly try to cover the **data** aspects of the program code, rather than the control aspects

## Fault-Based Testing (e.g. Mutation Testing)

- Mutation testing involves creating many slightly different versions of the code by **mutating** (changing operations) in each version
- Used to check **sufficiency** of test suites in detecting faults

# Role of White Box Testing

---

## Completeness for Black Box Methods

- White box code coverage gives a measure of **completeness** for open-ended black box methods
  - **Example:** Black box **shotgun** testing becomes a systematic method if we use code coverage (all statements executed at least once in the set of tests) as the completion criterion

# Role of White Box Testing

---

## Finds a Different Kind of Errors

- Black box testing finds errors of **omission** - that is, something that is specified that we have failed to do
- White box testing finds errors of **commission** - that is, something that we have done, but incorrectly

## Automation

- Because white box testing involves the program code itself, which has a standard form, we can automate most of it

# White Box Testing & Code Injection

---

## Code Injection

- Injection is not itself a test method, but refers to **modifications** of the source or executable code being tested in order to make tests more effective (possible because white box)
  - **Example:** Modify the program to log each statement's line number to a log file as it is executed, in order to check that every line is executed at least once by a test suite (*Produces a file of executed line numbers – check every line there*)
- Injection involves adding **extra** statements or instructions to execute that do not change what the original program does but checks or logs additional information about **execution** of the program (such as which statements have been executed)
- Original code is not changed, instead a **separate copy** with modifications is generated to run the tests on

# Applications of Code Injection

---

## Instrumentation Injection

- Involves adding code to instrument the actions of the program at every method, statement or instruction during testing, to keep track of properties such as execution coverage

## Performance Instrumentation

- Involves adding code to log the actual time or space used by each method or statement of the program during execution

## Assertion Injection

- Involves adding strict run-time assertion code to every method, statement or instruction in the program during testing, to help localize the cause of failures

## Fault Injection

- Involves adding code to simulate run-time faults, to test fault handling

# Implementation of White Box Code Injection

---

## Three Levels of Implementation

- Although it is not a necessity, white box testing usually involves validation of coverage using **code injection**
- This code injection can be implemented in three separate ways:
  1. At the **source** level
  2. At the **executable code** level
  3. At the **execution sampling** level



# Implementation of White Box Testing

---

## Three Levels of Implementation

1. At the **source** level
2. At the **executable code** level

**1 & 2:** A **copy** of the program under test is altered to inject the additional source or executable code to log coverage as the program executes.

3. At the **execution sampling** level

**3:** The **original** program under test is run but with regular timer interrupts - at each interrupt, the current state and execution location at interrupt time can be **sampled** and logged before continuing execution.

# Source Level Implementation

---

## Implementing Code Injection by Source Modification

- Create a copy of the program with new statements inserted to log coverage

### Example: JTest

```
13  final int mid = (lo + hi) / 2;  
14  if (list[mid] == key)  
15      result = mid;  
16  else if (list[mid] > key)  
17      hi = mid - 1;  
18  else  
19      lo = mid + 1;
```

```
13      log.println (13);  
14      final int mid = (lo + hi) / 2;  
15      log.println (14);  
16      if (list[mid] == key)  
17      {  
18          log.println (15);  
19          result = mid;  
16      }  
17      else  
18      {  
19          log.println (16);  
16          if (list[mid] > key)  
17          {  
18              log.println (17);  
19              hi = mid - 1;  
16          }  
17          else  
18          {  
19              log.println (18);  
16              log.println (19);  
17              lo = mid + 1;  
16          }  
17      }
```

```
    execount[13] += 1;
13    final int mid = (lo + hi) / 2;
    execount[14] += 1;
14    if (list[mid] == key)
    {
        execount[15] += 1;
15        result = mid;
    }
    else
    {
        execount[16] += 1;
16        if (list[mid] > key)
        {
            execount[17] += 1;
17            hi = mid - 1;
        }
18        else
        {
            execount[18] += 1;
            execount[19] += 1;
19            lo = mid + 1;
        }
    }
}
```

# Executable Code Level Implementation

---

## Implementing Code Injection by Executable Code Modification

- Create a **copy** of the program code with instructions inserted to log coverage
- In order not to change addresses, modify code to execute new instructions **out of line**

# Executable Code Level Implementation

---

**Example:** Unix prof and gprof

<u>Mem. Loc.</u>	<u>Machine Instruction</u>	
00A60	loada	list,R4
00A64	add	mid,R4
00A68	load	key,R5
00A6C	comp	R4,R5
00A70	jequ	00A84

# Executable Code Level Implementation

**Example:** Unix prof and gprof

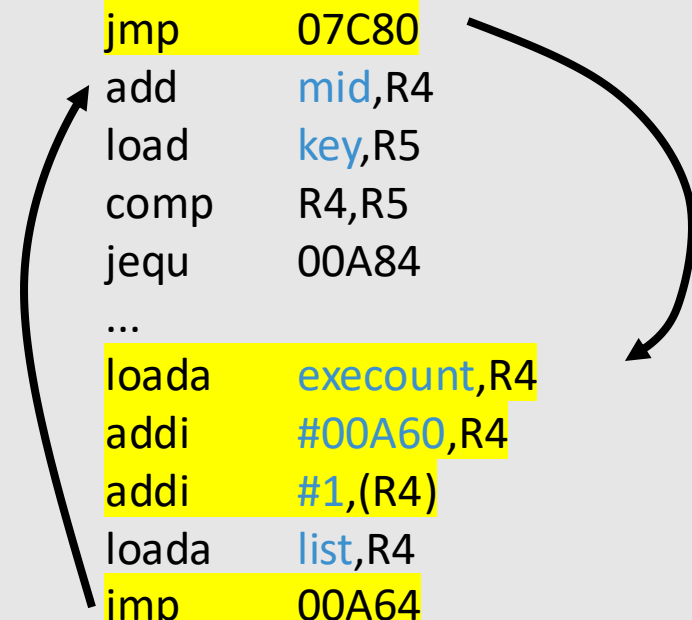
<u>Mem. Loc.</u>	<u>Machine Instruction</u>
00A60	jmp 07C80
00A64	add mid,R4
00A68	load key,R5
00A6C	comp R4,R5
00A70	jequ 00A84
...	...
07C80	loada execount,R4
07C84	addi #00A60,R4
07C88	addi #1,(R4)
07C8C	loada list,R4
07C90	jmp 00A64

# Executable Code Level Implementation

**Example:** Unix prof and gprof

Mem. Loc.      Machine Instruction

00A60	jmp	07C80
00A64	add	mid,R4
00A68	load	key,R5
00A6C	comp	R4,R5
00A70	jequ	00A84
...	...	
07C80	loada	execount,R4
07C84	addi	#00A60,R4
07C88	addi	#1,(R4)
07C8C	loada	list,R4
07C90	jmp	00A64





# Sampling Level Implementation

## Implementing Code Injection by Execution Sampling

- Do not change the executable code at all
- Use a **timer** or other frequent regular interrupt to randomly **sample** where we are executing
- Interrupt **return address** tells us where we are executing when each interrupt happens
- After a large number of samples, results become **statistically valid**

07C80	<b>timer:</b>	loada	<b>execount</b> ,R4
07C84		add	(SP),R4
07C88		addi	<b>#1</b> , (R4)
07C8C		rti	

# White Box Tools

---

## Testing Tools

- Obviously implementing these strategies by hand programming would be **tedious** and time consuming
- White box coverage testing is almost always supported by **tools** to implement the necessary code injections
- Often test analysis and selection of test cases for white box testing can also be done **automatically** by modern tools

# Code Coverage Testing

---

## Code Coverage Methods

- Two kinds: statement analysis (flow independent), decision analysis (flow dependent)
- Statement analysis methods
  - statement coverage
  - basic block coverage
- Decision analysis methods
  - decision coverage
  - condition coverage
  - loop coverage
  - path coverage

# Code Coverage Testing

---

## Code Coverage Methods

- Two kinds: statement analysis (flow independent), decision analysis (flow dependent)
- Statement analysis methods
  - statement coverage
  - basic block coverage
- Decision analysis methods
  - decision coverage
  - condition coverage
  - loop coverage
  - path coverage

# Statement Coverage

---

## Statement Coverage Method

- Causes every statement in the program to be executed at least once, giving us confidence that every statement is at least *capable* of executing correctly
- System: Make a test case for each statement in the program, independent of the others
  - Test must simply cause the statement to be run, ignoring its actions and sub-statements (but still must check that result of test is correct)
- Completion criterion: A test case for every statement
  - Can be checked by *instrumentation injection* to track statement execution coverage

# Example: Statement Coverage

```
// calculate numbers less than x  
// which are divisible by y  
1  int x, y;  
2  x = c.readInt ();  
3  y = c.readInt ();  
4  if (y == 0)  
5      c.println("y is zero");  
6  else if (x == 0)  
7      c.println("x is zero");  
8  else  
9      {  
10         for (int i = 1; i <= x; i++)  
11             {  
12                 if (i % y == 0)  
13                     c.println(i);  
14             }  
15     }
```

# Example: Statement Coverage

---

## Statement Coverage Tests

- We blindly make one test for each statement, analyzing which **inputs** are needed to cause the statement to be executed
- Create test case for each unique set of inputs

Statement x input y input

1     ?     ?

...

# Example: Statement Coverage

## Statement Coverage Tests

Statement x input y input Test x y

1	0	0	T1	0	0
2	0	0			
3	0	0			
4	0	0			
5	0	0			
6	0	1	T2	0	1
7	0	1			
8	1	1	T3	1	1
9	1	1			
10	1	1			
11	1	1			



# Basic Block Coverage

---

## Basic Block Analysis Method

- Causes every **basic block** (indivisible sequence of statements) to be executed at least once - (usually) generates fewer tests
- System: Identify basic blocks by sequence analysis, design test case for each basic block
  - Sequence of statements in a row, ignoring sub-statements, such that if first is executed then following are all executed
- Completion criterion: A test case for every basic block
  - Can be checked by **instrumentation injection** to track statement execution coverage

# Example: Basic Block Analysis

```
// calculate numbers less than x  
// which are divisible by y
```

```
int x, y;  
x = c.readInt ();  
y = c.readInt ();  
if (y == 0)  
    c.println("y is zero");  
else  
    if (x == 0)  
        c.println("x is zero");  
    else  
    {  
        for (int i = 1; i <= x; i++)  
        {  
            if (i % y == 0)  
                c.println(i);  
        }  
    }  
}
```

# Example: Basic Block Analysis

*// calculate numbers less than x  
// which are divisible by y*

```
int x, y;  
x = c.readInt ();  
y = c.readInt ();  
if (y == 0)  
    c.println("y is zero");  
else  
    if (x == 0)  
        c.println("x is zero");  
    else  
    {  
        for (int i = 1; i <= x; i++)  
        {  
            if (i % y == 0)  
                c.println(i);  
        }  
    }  
}
```

The diagram illustrates basic block boundaries in the provided code. Red arrows point to specific lines, and red numbers 1 through 7 are placed near the arrows to identify the blocks:

- 1: Points to the first `int` declaration (`int x, y;`).
- 2: Points to the first `println` statement (`c.println("y is zero");`).
- 3: Points to the `if (x == 0)` condition.
- 4: Points to the `c.println("x is zero");` statement.
- 5: Points to the `for` loop header (`for (int i = 1; i <= x; i++)`).
- 6: Points to the `if (i % y == 0)` condition.
- 7: Points to the `c.println(i);` statement.

Vertical red lines extend from these points to the corresponding closing braces, defining the basic blocks. For example, block 1 spans from the first `int` declaration to the final closing brace of the entire function.

# Example: Basic Block Analysis

---

## Basic Block Coverage Tests

- We make one test for each block, analyzing which inputs are needed to cause the block to be entered
- Create test case for each unique set of inputs

<u>Block</u>	<u>x input</u>	<u>y input</u>
1	?	?
...		

# Example: Basic Block Analysis

## Basic Block Coverage Tests

<u>Block</u>	<u>x input</u> <u>y input</u>		<u>Test</u>	<u>x</u>	<u>y</u>
1	0	0	T1	0	0
2	0	0			
3	0	1	T2	0	1
4	0	1			
5	1	1	T3	1	1
6	1	1			
7	1	1			

# Testing Methods: White Box Testing I

---

## Summary

- White box testing includes: [code coverage](#), [logic path/decision point coverage](#), [data & data flow coverage](#), [fault-based](#) testing (e.g. mutation testing)
- White box methods often involve [code injection](#) to instrument execution using source modification, executable code modification or run time sampling
- Today we started to look at one class of code coverage methods: Statement analysis methods ([statement](#), [basic block](#) coverage)

## Next Time

- More code coverage methods and data coverage methods