# CISC/CMPE 327 Software Quality Assurance

Queen's University, 2019-fall

## Lecture #14 White Box Testing

# Guest Lecture

- Oct-11

***QA principles in practice: what to expect and how to get your offer landed.***

***by Derek Li***

# White Box Testing

# White Box Testing

# White Box Testing

- 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
  - White box static analysis

# White Box vs. Black Box

- Recall:
  - Systematic testing methods can be of two kinds
    - Black box and white box (or glass box)
  - Black box methods cannot see what the software code to test is (it may not exist yet), so they can only base their tests on the requirements or specifications
  - White box methods can see the software's code, so they can base their tests on the software's actual architecture or code itself

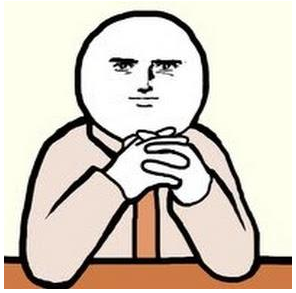
# White Box Testing

- Code Coverage
  - Design tests to cover (execute) every method, statement, or instruction of the program at least once
- Logic Path / Decision Point Coverage
  - Design tests to cover every path of execution in the program at least once
- Mutation Testing
  - Create different code “mutants” by mutating—randomly changing—the code in each version
  - Used to check sufficiency of test suites for detecting faults

# A Recurring Feature: **EVIL TIME**

(when we get to security,
it will be EVIL TIME all the time)

# EVIL TIME



```
/* Puny mortal!
   I scoff at your A1 black box testing! */
... createAccount (String number, String name) {
  evilCounter++;
  if (evilCounter == 327) {
     /* EVIL TIME */
      name = "  muahahahahaha ";
  }
  [non-evil code to write a line to the Transaction Summary]
}
```

# Code Coverage by Code Injection

- Injection is not itself a test method, but refers to modifying the source (or executable) code being tested to make tests more effective
  - Possible because this is white box testing
- Example:
  - Modify the program to log each statement's line number to a log file as it is executed, to check that every line is executed at least once by a test suite
  - Produces a file of executed line numbers, can check later that every line number is there

# Code Coverage by Code Injection

- Code Injection
  - Injection involves adding extra statements or instructions to execute that do not change what the original program does, but check or log additional information about execution of the program (such as which statements have been executed)
  - The original code is not changed, instead a separate copy with modifications is generated to run the tests on

# Applications of Code Injection

- Instrumentation Injection
  - Add code to "instrument" the actions of the program at every method, statement, or instruction during testing, to keep track of properties such as global invariants, resource usage or 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

# Applications of Code Injection

- 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
  - Example: before every C pointer dereference, add a check “!= NULL”
- Fault Injection
  - Involves adding code to simulate run-time faults to test fault handling

# Roles of White Box Testing

- Completeness for Black Box Methods
  - White box code coverage gives a measure of completeness for open-ended black box methods
  - 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

# Roles of White Box Testing

- Finds a Different Kind of Error
  - Black box testing finds errors of omission, something that is specified that we have failed to do
  - White box testing finds errors of commission, something that we have done, but incorrectly
- Automation
  - Because white box testing involves the program code itself, in a standard form, we can automate most of it

# Implementation of Code Injection

- Three Levels of Implementation
  - Although it is not a necessity, white box testing usually involves validation of code coverage using code injection
  - This can be implemented in three separate ways
    - At the source level
    - At the executable code level
    - At the execution sampling level

# Implementation of Code Injection

- Three Levels of Implementation
  - In the first two cases (source and executable levels), a copy of the program under test is altered to inject the additional source or executable code to log coverage as the program executes
  - In the third case (execution sampling level), 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*

| Source line number | Java Statement |
|---|---|
| 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;` |

# Source Level Implementation

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

- Log file method
  - Insert statements to print to a log file
  - Analyze log file

# Source Level Implementation

```
        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;
                }
        }
```

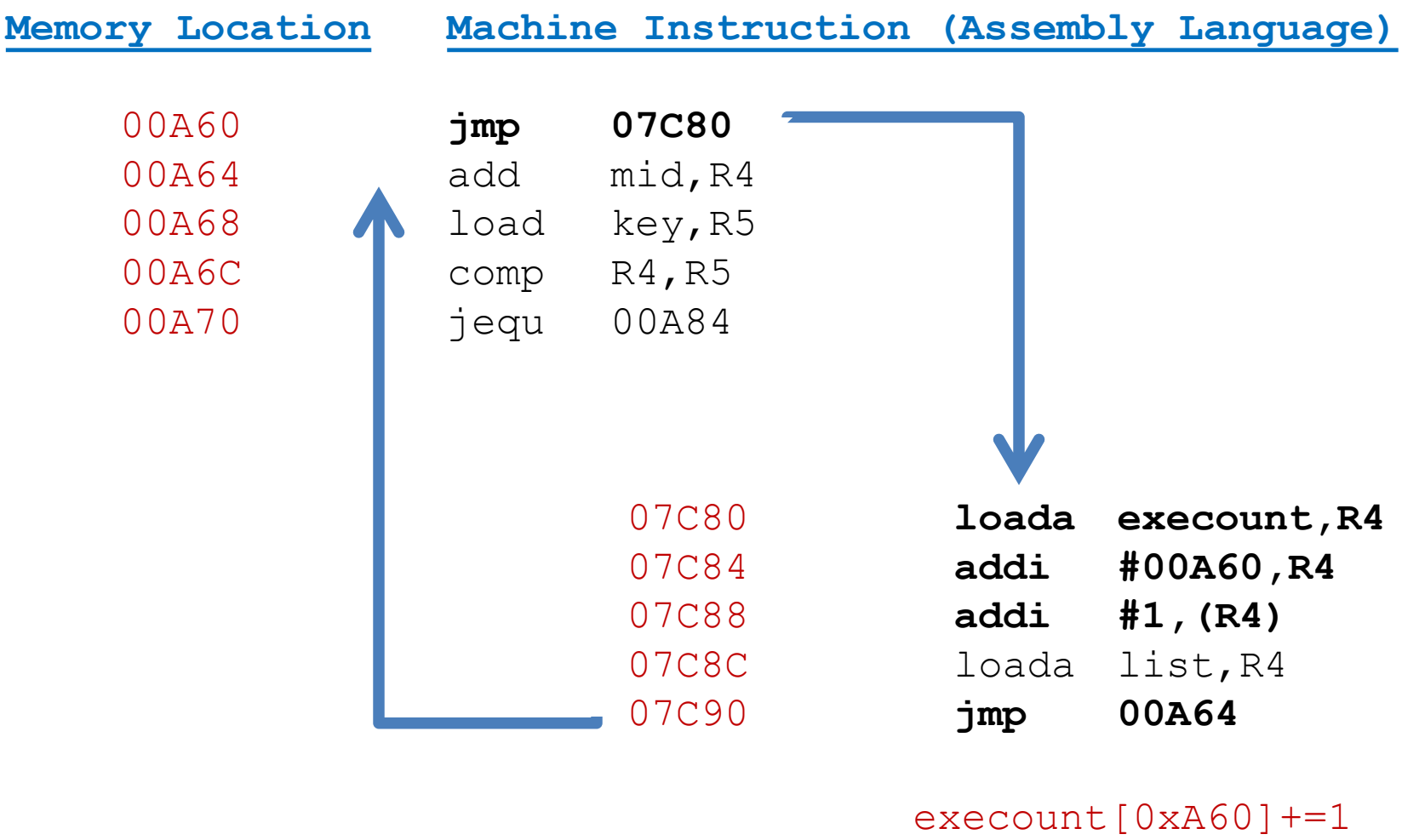
- Coverage array method
  - Insert statements to increment global array elements
  - Analyze array

# Executable Code Level Implementation

- Implementing Code Injection by Executable Code Modification
  - Create a copy of the executable program code with instructions inserted to log coverage
  - In order not to change addresses, modify code to execute new instructions out of line
  - Example: Unix prof and gprof

| Memory Location | Machine Instruction (Assembly Language) |
|---|---|
| 00A60 | loada list,R4 |
| 00A64 | add mid,R4 |
| 00A68 | load key,R5 |
| 00A6C | comp R4,R5 |
| 00A70 | jequ 00A84 |

# Executable Code Level Implementation



# Sampling Level Implementation

- 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  timer: loada  execount,R4
07C84         add    (SP),R4
07C88         addi   #1,(R4)
07C8C         rti
```

execount[where_at]+=1

# White Box Tools

- Testing Tools
  - Implementing these strategies by hand would be tedious
  - White box coverage testing is almost always supported by tools to implement the necessary code injections
  - Some test analysis and selection of test cases for white box testing can be done automatically by modern tools

# Static Analysis

- Have Code: Why Not Prove?
  - The source analysis to automatically generate tests is a complex and sophisticated flow analysis of the program
  - A very similar analysis can actually prove many of the cases, automatically finding problems or eliminating test cases before they are ever run

# Static Analysis

- Example: Euclid compiler
  - Euclid was a precursor to the Turing language
  - The Euclid compiler was designed to prove that subscripts and pointers were never out of range, that pre- and post- assertions were always true, and so on
  - The compiler inserted code to check these "legality" conditions at run time only in cases where it could not prove them at compile time ("statically")
  - In practice, the compiler was able to prove almost all legality conditions, reducing the overhead of run time checking to less than 10% of run time of the program

# (Static typing is static analysis)

- Removing subscript bounds checking also possible through more powerful **type systems**
- Example: Dependent ML eliminated up to 79% of bounds checks [Xi and Pfenning 1998]

# Summary

- White Box Testing
  - White box testing includes code coverage, logic path, and mutation testing
  - White box methods often involve code injection to instrument execution using source modification, executable code modification, or run time sampling
  - Static analysis can reduce white box testing effort and cost using automatic proofs
- Next time
  - Code coverage methods