Software Testing, Quality Assurance and Maintenance Winter 2015

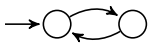
Lecture 11 — January 28, 2015

Patrick Lam *version 1*

Assertions are a key ingredient in writing tests, so I thought I would explicitly describe them.

Definition 1 *An* assertion *contains an expression that is supposed to evaluate to* `true`.

For instance, if we have a linked list, then the doubly-linked list property states that `prev` is the inverse of `next`, i.e. for nodes `n`, we have `n.next.prev == n`. After inserting a node, one might expect this property to be true of that node (possibly with caveats, for instance about the end of the list.)



We use assertions in unit tests to say what's supposed to be true.

Preconditions and postconditions. More generally, we can express what is supposed to be true upon entry & exit from a method.

We saw this code in Linux:

```
1 /* LOCKING: caller. */
2 void ata_dev_select(...) { ...}
```

This expresses an assertion (although not written as a program statement) that the lock is held upon entry.

Assume/Guarantee Reasoning. Why would you use preconditions and postconditions? Reasoning about programs is difficult, and preconditions and postconditions simplifies reasoning.

When reasoning about the callee, you get to assume that the precondition holds upon entry.

When reasoning about the caller, you get to guarantee the precondition holds before the call.

The reverse holds about the postcondition.

aComment

We talked about the aComment approach for locking-related annotations. In particular, it:

- extracts locking-related annotations from code;
- extracts locking-related annotations from comments; and
- propagates annotations to callers.

Tools

To provide some context, consider the OS X Mavericks goto fail bug:

```
1   if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
2       goto fail;
3   if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
4       goto fail;
5       goto fail; /* MISTAKE! THIS LINE SHOULD NOT BE HERE */
6   if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
7       goto fail;
8       err = sslRawVerify(...);
9
10  fail:
11      return err;
```

The bug:

`opensource.apple.com/source/Security/Security-55471/libsecurity_ssl/lib/sslKeyExchange.c`

No bug:

`www.opensource.apple.com/source/Security/Security-55179.13/libsecurity_ssl/lib/sslKeyExchange.c`

A writeup:

`nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch/`

Detecting goto fail. In retrospect, a number of tools could've found this bug:

- compiler `-Wunreachable-code` option
- PC-Lint:warning 539: Did not expect positive indentation
- PVS-Studio:V640: Logic does not match formatting

The problem is that these tools also report many other issues, and live issues can be buried among those other issues.

The Landscape of Testing and Static Analysis Tools

Here's a survey of your options:

- manual testing;
- running a JUnit test suite, manually generated;
- running automatically-generated tests;
- running static analysis tools.

We'll examine several points on this continuum today. More on this later (Lecture 23), thanks to guest lecturer. Some examples:

- Coverity: a static analysis tool used by 900+ companies, including BlackBerry, Mozilla, etc.
- Microsoft requires Windows device drivers to pass their Static Driver Verifier for certification.

Tools for Java that you can download

FindBugs. An open-source static bytecode analyzer for Java out of the University of Maryland.

`findbugs.sourceforge.net`

It finds bug patterns:

- off-by-one;
- null pointer dereference;
- ignored `read()` return value;
- ignored return value (immutable classes);
- uninitialized read in constructor;
- and more...

FindBugs gives some false positives. Here are some techniques to help avoid them:

`patricklam.ca/papers/14.msr.saa.pdf`

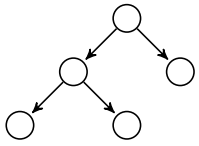
Java Path Finder (JPF), NASA. The key idea: Implement a Java Virtual Machine, but explore many thread interleavings, looking for concurrency bugs.

"JPF is an explicit state software model checker for Java™ bytecode."

JPF can also search for deadlocks and unhandled exceptions (`NullPointerException`, `AssertionError`); race conditions; missing heap bounds checks; and more.

`javapathfinder.sourceforge.net`

Korat (University of Illinois). Key Idea: Generate Java objects from a representation invariant specification written as a Java method.

For instance, here's a binary tree.  Binary Tree!

One characteristic of a binary tree:

- left & right pointers don't refer to same node.

We can express that characteristic in Java as follows:

```
1  boolean repOk() {
2    if (root == null) return size == 0;   // empty tree has size 0
3    Set visited = new HashSet(); visited.add(root);
4    List workList = new LinkedList(); workList.add(root);
5    while (!workList.isEmpty()) {
6      Node current = (Node)workList.removeFirst();
7      if (current.left != null) {
8        if (!visited.add(current.left)) return false; // acyclicity
9        workList.add(current.left);
10     }
11     if (current.right != null) {
12       if (!visited.add(current.right)) return false; // acyclicity
13       workList.add(current.right);
14     }
15   }
16   if (visited.size() != size) return false;  // consistency of size
17   return true;
18 }
```

Korat then generates all distinct ("non-isomorphic") trees, up to a given size (say 3). It uses these trees as inputs for testing the `add()` method of the tree (or for any other methods.)

`korat.sourceforge.net/index.html`

Randoop (MIT). Key Idea: "Writing tests is a difficult and time-consuming activity, and yet it is a crucial part of good software engineering. Randoop automatically generates unit tests for Java classes."

Randoop generates random sequence of method calls, looking for object contract violations.

To use it, simply point it at a program & let it run.

Randoop discards bad method sequences (e.g. illegal argument exceptions). It remembers method sequences that create complex objects, and sequences that result in object contract violations.

`code.google.com/p/randoop/`

Here is an example generated by Randoop:

```
1  public static void test1() {
2     LinkedList list = new LinkedList();
3     Object o1 = new Object();
4     list.addFirst(o1);
5
6     TreeSet t1 = new TreeSet(list);
7     Set s1 = Collections.synchronizedSet(t1);
8
9     // violated in the Java standard library!
10    Assert.assertTrue(s1.equals(s1));
11  }
```