COSC345 Week 17

Static verification

4 August 2015

Richard A. O'Keefe

Aim of static verification

Find defects early

Work on incomplete components

Work on non-executable items

N.B. Testing and debugging cannot start until executable components are sufficiently complete, test data are available, test setup in place. That may be too late.

Value of static verification

estimated 50% to 90% faults findable

faults found early

no risk to system data

faults not masked by other faults

can find problems other than faults

sampling can be used (common faults are common)

but cannot replace testing

Automatic static verification

syntax checks

control flow (unreachable, some nonterminating)

data flow checks

interface checks

abstract execution

cross reference and spelling checks

layout rules (see ‘indent’, ‘astyle’)

Automatic verification for C

Some compilers do little, gcc and especially clang do more

enable *all* warnings to start with.

UNIX has lint, finds oddities as well

- does cross-file checks which even gcc doesn't
- warns about error results not examined

SPlint (né LClint) is free, UNIX/Windows, all lint does plus

- pointer tracking and name style enforcement
- and bounds checking (vital for security)

Not stated means not checked

A static checker cannot check requirements that it cannot be told about. ~/quasar/anthony.d/bp.c has

```
static void train(void)
    /*@globals
         nin,  in_units,  npat, in_pats,
         nhid, hid_units, ih_wts,
         nout, out_units, ho_wts, out_pats,
         momentum, criterion, learn_rate;@*/
    /*@modifies in_units, hid_units, out_units, ih_wts, ho_wts,
                kepoch, *stdin, *stdout, *stderr;@*/;
```

What's wrong with these loops?

```
1. for (i = 1; i <= N; i++) a[i] = 0;
2. for (i = 0; i <= N; i++) a[i] = 0;
3. for (i = 0; i <  N; j++) a[i] = 0;
4. for (i = 0; i < strlen(s); i++) s[i] = tolower(s[i]);
5. for (i = 0; i <  N; i++) a[i++] = 0;
6. for (i = 0; i <  N; i++) a[i] == 0;
7. p = a-1; for (i = 1; i <= N; i++) p[i] = 0;
```

All are *legal* C!

Answers

1. Will probably run off the top of a[]
2. Will almost certainly run off the top of a[]
3. Increments wrong variable (quite common)
4. $O(N^2)$ method; $O(N)$ is easy
5. Increments counter twice
6. Statement has no effect
7. Constructs illegal pointer (a-1)

FindBugs for Java

omitted null checks

pointless null checks

pointless new String(...)

exposing mutable private data

calling String.+ in a loop

cases where an encoding is advisable but missing

serialisable objects with non-serialisable parts

FindBugs for Java (II)

exception handlers that don't handle

dead assignments

(long)(a*b) where (long)a*b would be appropriate

(double)(x/y) instead of (double)x/y

unused fields (caused by shadowing, say)

clone() not calling super.clone()

implementing Cloneable but not mentioning clone()

FindBugs for Java (III)

implementing Comparator but not Serializable

implementing hashCode() or equals() without the other

switches without a default

writes to static fields (not good with threads)

exceptions that might not be caught

exceptions caught that can't be thrown

Formats using \n instead of %n

FindBugs for Java (IV)

unconditional wait()

wait() not in a loop

broken lazy initialisations (a concurrency bug)

inner classes that should probably be static

things that should be 'final' but aren't

...

and lots lots more

Example non-code verification

```
<!doctype exam system "exam.dtd">
<exam paper=cosc325 year=1998 semester=2 duration=3>
<title>Software Enginerring
<section choose=5>
<quest>
  <squest><p>What is prototyping?<mark>2
  <squest><p>What is the purpose of prototyping?<mark>4
  <squest><p>Where does prototyping fit in
  <name person>Boehm</name>'s <q>spiral</q> model of the
  software development process?<mark>4
```

What can we check?

Proper nesting of parts

Each part has mark or subparts but not both

Choose N implies all parts have equal marks

Question numbering consistent

Cross-references valid, consistently labelled

Quotes <q> balanced and alternate

None enforceable using WYSIWYG (both can check spelling)

All important

Manual static verification

high level semantics

style

common fault pattern recognition

don't keep a dog and bark yourself

Semi-automated

items go into structured data base

checker poses "pattern" questions

used in Y2K inspection

Program inspections

can be formal or informal

need your own checklist

is up-front expensive

but not as expensive as not doing it

requires practice

Very important indeed

People **must not** feel threatened by inspections. The policy **must** be that pay/tenure/status will **not** depend on inspection results, and this policy **must be adhered to**.

Inspection meetings **must not** be threatening. Consider making the code anonymous and having the author absent.

Inspections generate a list of actions; they **must** be followed up.

Rôles

Owner Claims item inspection-worthy; responsible for having defects corrected

Inspector Person who inspects item; need more than one

Reader Presents code at meeting

Scribe Takes minutes

Moderator Manages meeting; responsible for keeping it cool

Chief moderator Manages inspection process as whole

Resources

lint-paper.pdf — local copy of Lint paper

fi.pdf is the NASA Software Formal Inspections Guidebook; you were directed to that earlier.

fistd.pdf is the NASA Software Formal Inspections Standard; the guidebook expounds it.

its4-1.0.1.tar.gz "It's The Source, Stupid! Security Scanner"

Splint 3.1.2 at http://www.splint.org/

HTML Tidy and HTML Validator at http://www.w3.org

Resources II

http://findbugs.sourceforge.net/ is the site for FindBugs(TM), a user-extensible static checker for Java.

http://en.wikipedia.org/wiki/ESC/Java2 has links for the Extended Static Checker for Java.

Most Smalltalks now include the Refactoring Browser, which includes the SmallLint checker for Smalltalk.

AWK and Perl are good for checking data files. Don't overlook the possibility of using Lex (Flex) and Yacc (Bison) to write your own checkers (like the example last week).

Resources III

SPARK is a verifiable subset of Ada

There is a SPARK toolset for it.

It now comes with GNAT (part of gcc).

There's a book about it in the library.