Verification Using Static Analysis

Outline

- Today we will discuss static analysis and how it differs from dynamic analysis
- We will also look at the different types of static analysis including:
  - Control flow analysis, data use analysis, interface analysis, information flow analysis, and path analysis
- We will look at three specific static analyzers:
  - LINT, SpotBugs and CodeSurfer Path Inspector
- Finally, we will discuss a case study of the SCRUB tool at NASA JPL

Static vs. Dynamic Analysis

Dynamic Analysis

- involves execution the program and observing the outcomes.
  - e.g., Testing

Static Analysis

- involves examining a program without executing it.
  - e.g., Code Inspection

Automated Static Analyzers

What are Static Analyzers?

- Code inspection techniques that we have looked at involve examining the source code by hand.
- Static analyzers are tools that can be used during inspection to help identify problems in the code automatically.
- Static analyzers typically work best with language that lack strict type rules (such as C).
  - Languages such as Java have removed language features that cause errors (e.g., all variables must be initialized).

Automated Static Analyzers

Static Analyzer Process

- Static analyzers have 3 basic steps:
  - Scan source code
  - Perform an automated analysis of the code
  - Report any faults and/or anomalies

Different types of static analysis checks

Fault class	**Static analysis check**
Data faults	Variables used before initialisation Variables declared but never used Variables assigned twice but never used between assignments Possible array bound violations Undeclared variables
Control faults	Unreachable code Unconditional branches into loops
Input/output faults	Variables output twice with no intervening assignment
Interface faults	Parameter type mismatches Parameter number mismatches Non-usage of the results of functions Uncalled functions and procedures
Storage management faults	Unassigned pointers Pointer arithmetic

Source: Sommerville, *Software Engineering*, 7th Edition, Addison-Wesley 2004.

How do static analyzers find faults?

Types of Analysis

- Data use analysis: identify variable use such as variables used but not initialized, declared but not used, etc. (finds Data faults, Input/Output faults, Storage management faults)
- Control flow analysis: identify unreachable code, exit/entry points, loops. (finds Control faults)
- Interface analysis: checks consistency of declaration/use of procedures/routines. (finds Interface faults)
- Information flow analysis: identifies variables dependencies (e.g. input/output dependencies) but not faults.
- Path analysis: identifies all possible paths through the control flow graph.

A Static Analysis Tool

LINT

- One example of a static analyzer is LINT
  - LINT works on C code and is typically found on Linux/Unix systems.
  - We will now consider a small example using LINT.
  - More details:
    - http://www.unix.com/man-page/FreeBSD/1/lint
  - There is also a version of LINT for Java called Jlint
    - http://jlint.sourceforge.net/

A Static Analysis Tool

LINT Example

```
138% more lint_ex.c
#include <stdio.h>
printarray (Anarray)
 int Anarray;
{   printf(“%d”,Anarray);  }
main ()
{
 int Anarray[5]; int i; char c;
 printarray (Anarray, i, c);
 printarray (Anarray) ;
}
139% cc lint_ex.c
```

Code compiled with no errors

- Code compiles correctly but is it correct?

A Static Analysis Tool

LINT Example

```
138% more lint_ex.c
#include <stdio.h>
printarray (Anarray)
 int Anarray;
{   printf(“%d”,Anarray);   }
main ()
{
 int Anarray[5]; int i; char c;
 printarray (Anarray, i, c);
 printarray (Anarray) ;
}
139% cc lint_ex.c
140% lint lint_ex.c
lint_ex.c(10): warning: c may be used before set
lint_ex.c(10): warning: i may be used before set
printarray: variable # of args. lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) :: lint_ex.c(11)
printf returns value which is always ignored
```

LINT is run on compiled code

c, i not initialized before use

Inconsistent use of first arg

Function value is never used

Source: Sommerville, *Software Engineering*, 7th Edition, Addison-Wesley 2004.

Another Static Analysis Tool

SpotBugs (successor of FindBugs)

- An open source static analysis bug detection tool
  - Available at https://spotbugs.github.io/
- Analyzes Java bytecode
- Identifies bug patterns detected in the bytecode
  - A bug pattern is a code pattern that often results in a bug
  - Since SpotBugs detects bug patterns and not actual bugs it can produce false positives (e.g., bug patterns that are not really bugs).

Another Static Analysis Tool

SpotBugs Bug Patterns

- A complete list of the bug patterns identified in SpotBugs is available at: https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html
- The bug patterns are classified into the following categories:
  - Bad practice
  - Correctness
  - Internationalization
  - Malicious code vulnerability
  - Multithreaded correctness
  - Performance
  - Security
  - Dodgy

Another Static Analysis Tool

Example Bug Patterns

- **Bad Practice Pattern:** "Method may fail to close stream (OS_OPEN_STREAM)"
  - Reports if an input/output stream is not closed
  - Possible outcome: file descriptor leak
- **Dodgy Pattern:** "Redundant nullcheck of value known to be non-null (RCN_REDUNDANT_NULLCHECK_OF_NONNULL_VALUE)"
  - Reports an unnecessary check that will most likely not lead to incorrect behavior

(Yet) Another Static Analysis Tool

CodeSurfer Path Inspector

- The CodeSurfer Path Inspector extension is a static analysis tool developed by GrammaTech (http://www.grammatech.com) that analyzes a C program with respect to a sequencing property.
  - **For example:**
    - A call to function X does not occur globally
    - Statement Y occurs after Statement Z
- Path Inspector will determine if a sequencing property is true or false.
  - If it is false the program will produce a counter-example (i.e. an execution path that shows the property cannot be true).

Case Study: Static Analysis at NASA's Jet Propulsion Lab

- Jet Propulsion Lab (JPL)
  - http://www.jpl.nasa.gov/index.cfm
- Case study details based on keynote seminar by Gerard Holzmann (formerly of NASA JPL) at OOPSLA titled – *"Scrub & Spin: Stealth Use of Formal Methods in Software Development"*

Jet Propulsion Laboratory

Case Study: Static Analysis at NASA’s Jet Propulsion Lab

Holzmann scrubbing away code defects

05/05/2009 01:15 PM ES

By Jeff Feinman

ı the NASA Jet Propulsion Laboratory, Gerard Holzmann likes to have things nice nd clean.

hat is why he created Scrub, a ode reviewing porgram that is sed for all module code reviews. crub collects warnings and error əports from NASA software nalyzers, and then presents the ata in a GUI to developers so ey can see all issues that need ɔ be addressed.

The tool collects all the nechanically produced error eports, but also peers and code eviewers can enter queries on the code as well by clicking on the line number," lolzmann said. "Human-generated input gets collected by the same tool in a uniform nterface. During a code review, the module developer is asked to respond to each eport and close them out. If there's a disagreement, then there's a second cycle of eview."

s Holzmann and his team gear up for the Mars Science Laboratory mission, which ill put a rover onto Mars to assess the planet's environment, it is code analyzers like crub that are making sure the software is working properly and that problems will be voided when it's time for the big launch.

★★★★★ Be the first to rate this post

hare this link: http://www.sdtimes.com/blog/1416

ags:

“The tool collects all the mechanically produced error reports, but also peers and code reviewers can enter queries on the code as well by clicking on the line number…Human-generated input gets collected by the same tool in a uniform interface. During a code review, the module developer is asked to respond to each report and close them out. If there's a disagreement, then there's a second cycle of review.”

- Gerard Holzmann, SD Times

SD Times SOFTWARE DEVELOPMENT

Case Study: Static Analysis at NASA's Jet Propulsion Lab

SCRUB

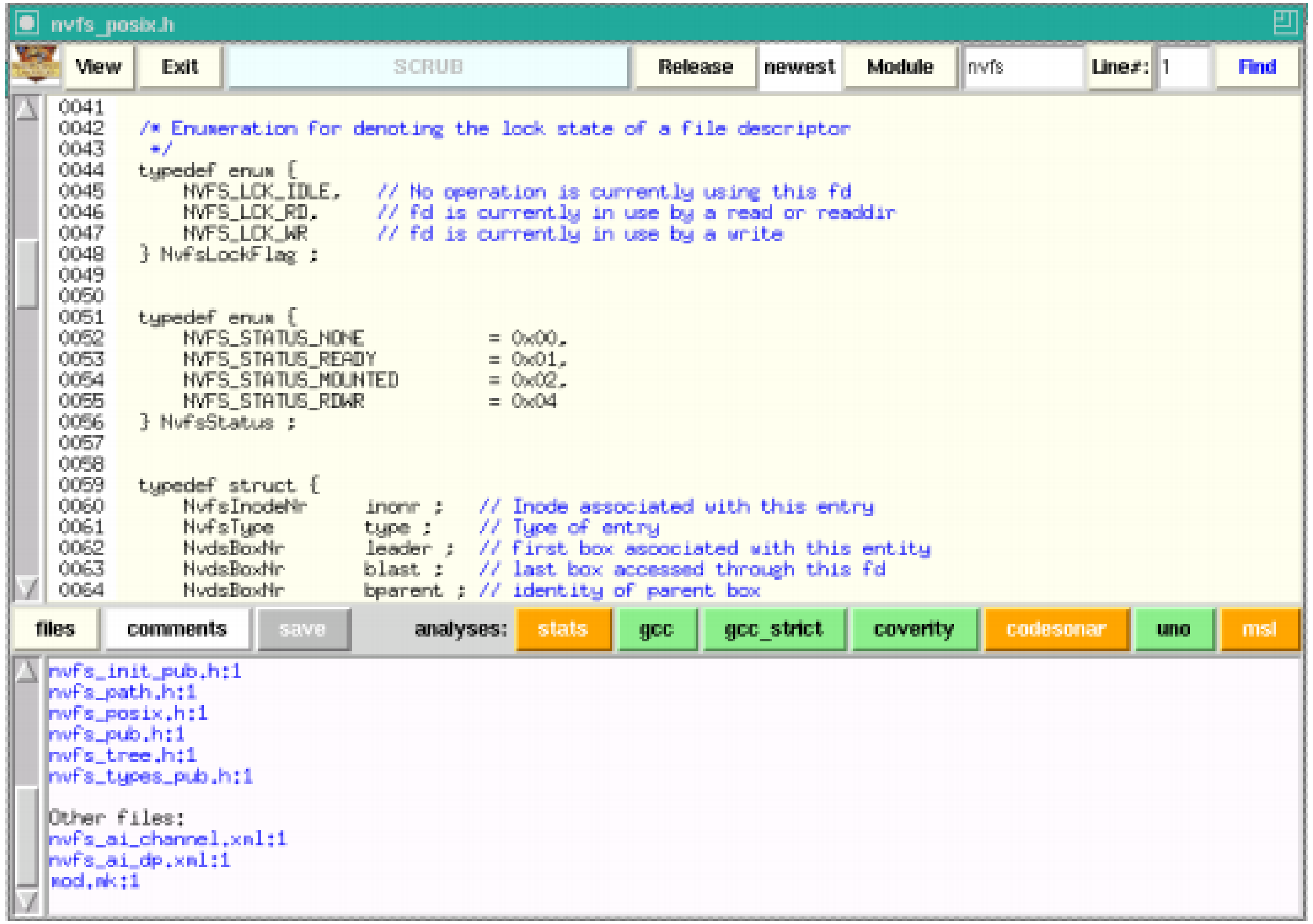
=

Source Code Review User Browser

NOTE: In addition to static analysis tools, SCRUB also uses formal analysis tools (next class). Examples of tools used by SCRUB are Codesonar, Coverity, gcc, uno etc.

Case Study: Static Analysis at NASA’s Jet Propulsion Lab

SCRUB Interface



```

/* Enumeration for denoting the lock state of a file descriptor
 */
typedef enum {
    NVFS_LCK_IDLE,    // No operation is currently using this fd
    NVFS_LCK_RD,      // fd is currently in use by a read or readdir
    NVFS_LCK_WR       // fd is currently in use by a write
} NvfsLockFlag ;


typedef enum {
    NVFS_STATUS_NONE              = 0x00,
    NVFS_STATUS_READY             = 0x01,
    NVFS_STATUS_MOUNTED           = 0x02,
    NVFS_STATUS_RDWR              = 0x04
} NvfsStatus ;


typedef struct {
    NvfsInodeNr     inonr ;   // Inode associated with this entry
    NvfsType        type ;    // Type of entry
    NvdsBoxNr       leader ;  // first box associated with this entity
    NvdsBoxNr       blast ;   // last box accessed through this fd
    NvdsBoxNr       bparent ; // identity of parent box
```

```
nvfs_init_pub.h:1
nvfs_path.h:1
nvfs_posix.h:1
nvfs_pub.h:1
nvfs_tree.h:1
nvfs_types_pub.h:1

Other files:
nvfs_ai_channel.xml:1
nvfs_ai_dp.xml:1
mod.mk:1
```

Source: http://spinroot.com/gerard/pdf/ScrubPaper_rev.pdf

Case Study: Static Analysis at NASA's Jet Propulsion Lab

- *"Scrub & Spin: Stealth Use of Formal Methods in Software Development"* keynote available on ACM Digital Library: http://dl.acm.org.uproxy.library.dc-uoit.ca/citation.cfm?id=1639950.1705499&coll=DL&dl=ACM&CFID=70536916&CFTOKEN=76245316

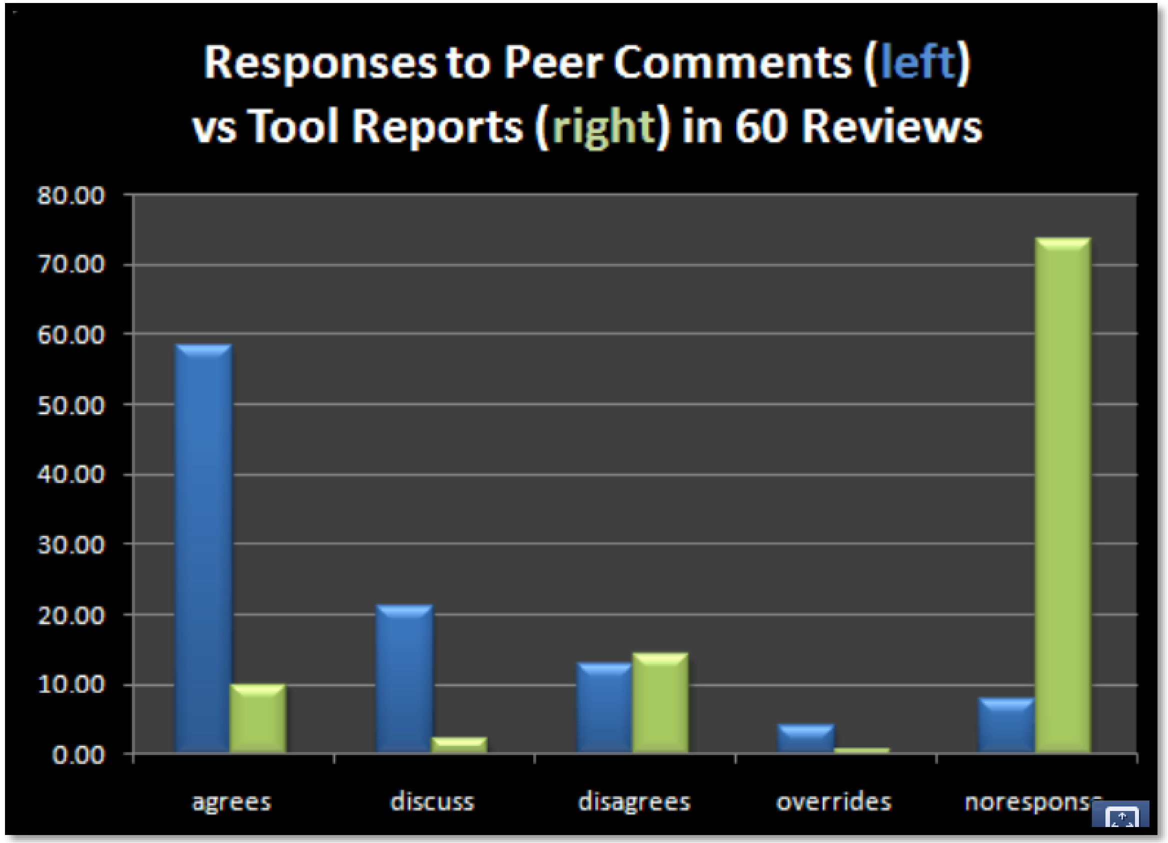
Case Study: Static Analysis at NASA's Jet Propulsion Lab

For class watch 29:28 to 42:04

- *"Scrub & Spin: Stealth Use of Formal Methods in Software Development"* keynote available on ACM Digital Library: http://dl.acm.org.uproxy.library.dc-uoit.ca/citation.cfm?id=1639950.1705499&coll=DL&dl=ACM&CFID=70536916&CFTOKEN=76245316

Case Study: Static Analysis at NASA's Jet Propulsion Lab

- Use of SCRUB on 227,041 loc in the first year (60 different code reviews)



Source: http://spinroot.com/gerard/pdf/ScrubPaper_rev.pdf

Verification Using Static Analysis

Static Analysis

- Automatic static analyzers are complementary to both testing and code inspection (discussed next)
  - Analyzer use different types of analysis to find and report possible faults

Readings

- G.J. Holzmann. Scrub: a tool for code reviews. Innovations in Systems and Software Engineering, 2010, Vol. 6, Nr. 4, pp. 311-318. http://spinroot.com/gerard/pdf/ScrubPaper_rev.pdf

References

- This lecture was partially based on Section 22.3 in Sommerville' s *Software Engineering* book