

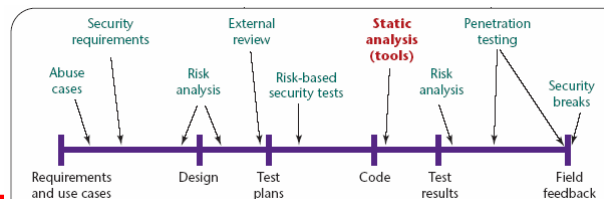
## Static Analysis Intro

- **Static analysis** is the process of evaluating a system or component based on its form, structure, content, or documentation [IEEE]
- Does not involve the execution of the program
- Software inspections are a form of static analysis
- “even well tested code written by experts contains a surprising number of obvious bugs” [Hovermeyer/Pugh]
- “Java has many language features and APIs which are prone to misuse.” [Hovermeyer/Pugh]
- Static analysis tools “can serve an important role in raising the awareness of developers about subtle correctness issues. . . . prevent future bugs” [Hovermeyer/Pugh]

NC STATE UNIVERSITY

## Static Analysis Tools

- Search through code to detect **bug patterns** (error prone coding practices that arise from the use of erroneous design patterns, misunderstanding of language semantics, or simple and common mistakes.
- Increasingly being used to identify security vulnerabilities
- “can peer into more of a program’s dark corners with less fuss than dynamic analysis”
- “the operator of a good static analysis tool can successfully apply that tool without being aware of the finer points of security bugs”



[Hovermeyer/Pugh, Chess/McGraw]

## Problems with static analysis tools

- **False positive:** the tool reports bugs the program doesn't contain
  - A static analysis tool will brag about having only 50% false positives.
  - Need to manually review and decide whether to fix or ignore. Some tools allow you to create filters of the types of bugs you don't want to see.
- **False negative:** the tool contains bugs the tool doesn't report
  - May increase as static analysis tool develop works to reduce false positives
- May also detect “**harmless bugs**” which need human judgment to sort out

## A Comparison of Bug Finding Tools for Java

- **Bug pattern detection**
  - PMD
  - FindBugs [also dataflow]
  - JLint [also dataflow]
- **Theorem proving [involves annotation]**
  - ESC/Java 2
- **Model checking [involves annotation]**
  - Bandera
- **Result:** little overlap in the warning generated by the tools & no correlation between the warning count and the tool → always be a need for multiple separate tools

Bug Category	Example	ESC/Java	FindBugs	JLint	PMD
General	Null dereference	√*	√*	√*	√
Concurrency	Possible deadlock	√*	√	√*	√
Exceptions	Possible unexpected exception	√*			
Array	Length may be less than zero	√		√*	
Mathematics	Division by zero	√*		√	
Conditional, loop	Unreachable code due to constant guard		√		√*
String	Checking equality using == or !=		√	√*	√
Object overriding	Equal objects must have equal hashcodes		√*	√*	√*
I/O stream	Stream not closed on all paths		√*		
Unused or duplicate statement	Unused local variable		√		√*
Design	Should be a static inner class		√*		
Unnecessary statement	Unnecessary return statement				√*

√ - tool checks for bugs in this category \* - tool checks for this specific example

Figure 3. The Types of Bugs Each Tool Finds

Snapshot Tool	Name	NCSS (Lines)	Class Files	Time (min:sec.csec)			Warning Count				
				ESC/Java	FindBugs	JLint	PMD	ESC/Java	FindBugs	JLint	PMD
	Azureus 2.0.7	35,549	1053	211:09.00	01:26.14	00:06.87	19:39.00	5474	360	1584	1371
	Art of Illusion 1.7	55,249	676	361:56.00	02:55.02	00:06.32	20:03.00	12813	481	1637	1992
	Tomcat 5.0.19	34,425	290	90:25.00	01:03.62	00:08.71	14:28.00	1241	245	3247	1236
	JBoss 3.2.3	8,354	274	84:01.00	00:17.56	00:03.12	09:11.00	1539	79	317	153
	Megamek 0.29	37,255	270	23:39.00	02:27.21	00:06.25	11:12.00	6402	223	4353	536

Figure 4. Running Time and Warnings Generated by Each Tool

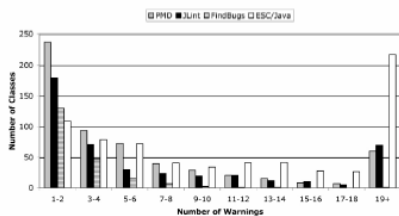


Figure 5. Histogram for number of warnings

	ESC/Java	FindBugs	JLint	PMD
Concurrency Warnings	126	122	8883	0
Null Dereferencing	9120	18	449	0
Null Assignment	0	0	0	594
Index out of Bounds	1810	0	264	0
Prefer Zero Length Array	0	36	0	0

Figure 6. Warning Counts for the Categories Discussed in Section 4.1

## FindBugs

- 45 bug pattern detectors
  - Single-threaded correctness issue
  - Thread/synchronization correctness issue
  - Performance issue
  - Security and vulnerability to malicious untrusted code

Code	Description
Eq	Bad Covariant Definition of Equals
HE	Equal Objects Must Have Equal Hashcodes
IS2	Inconsistent Synchronization
MS	Static Field Modifiable By Untrusted Code
NP	Null Pointer Dereference
OS	Open Stream
RR	Read Return Should Be Checked
RV	Return Value Should Be Checked
UR	Uninitialized Read In Constructor
UW	Unconditional Wait
Wa	Wait Not In Loop

Figure 1: Summary of selected bug patterns

## References

- Chess, Brian and McGraw, G. *Static Analysis for Security*, IEEE Security & Privacy, Nov/Dec 2004.
- Hovermeyer, David and Pugh, William, *Finding Bugs is Easy*, OOPSLA 2004
- Rutar, N., Almazan, C., and Foster, J., *A Comparison of Bug Finding Tools for Java*, ISSRE 2004.