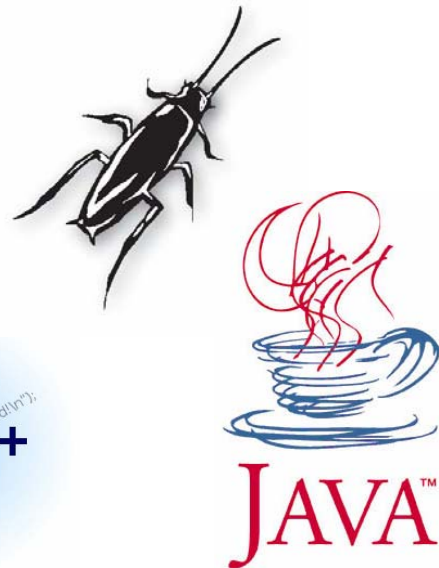




ÉCOLE POLYTECHNIQUE  
FÉDÉRALE DE LAUSANNE

# Software Verification Tools Overview

---



Clément Beffa  
Vincent Pazeller  
Olivier Gobet

# Overview of tools

---

- Many available tools
  - 37 non-commercial & downloadable
  - Huge disparity between them
    - Alpha version from a publication concept (ARMC)
    - Stable version used by ten of thousand people and sponsored by corporation (FindBugs)
  - Tools list available on the project page

# Languages repartition

---

- Popular languages (C, Java)
- Homemade for the purpose of proving (armc, abst/impl/spec, pale)





ÉCOLE POLYTECHNIQUE  
FÉDÉRALE DE LAUSANNE

# Java tools overview

---

# Jahob

---

- Only a subset of Java
- Prove dynamically allocated data structures and arrays
- require annotation (pre-condition, invariant and post-condition)
  - Not automated
- no GUI and require different packages like CVC and OCaml


# F-Rex

---

- composed of: Jreg and Jfree
- verify that program are memory safe
  - analyzing lifetime of objects
  - analyzing deallocation

```
.method public static
  main([Ljava/lang/String;)V
  .limit stack 1
  .limit locals 1
  create_ru 1 ;o3062:[:r2b |
  new_in_r 1 A
  astore_0
  push_r 0
  aload_0
  ; jreg added-> to attach
```

New object  
allocation in region  
;o3062:[:r2b



# Daikon

---

- Dynamic detector of likely invariants
- Automatically annotate program
  - For example for ESC/Java2
- Also supports C, C++ and Perl

# Purity Analysis Kit

---

- Check for purity of Java methods
  - method that does not mutate any object that existed before the method was invoked
  - do not interfere with other computations

```
void swapValRight(Value n)
NOT HEAP PURE
  "this": mutation on this.(right|value)
  "n": mutation on n.(value|right)
```



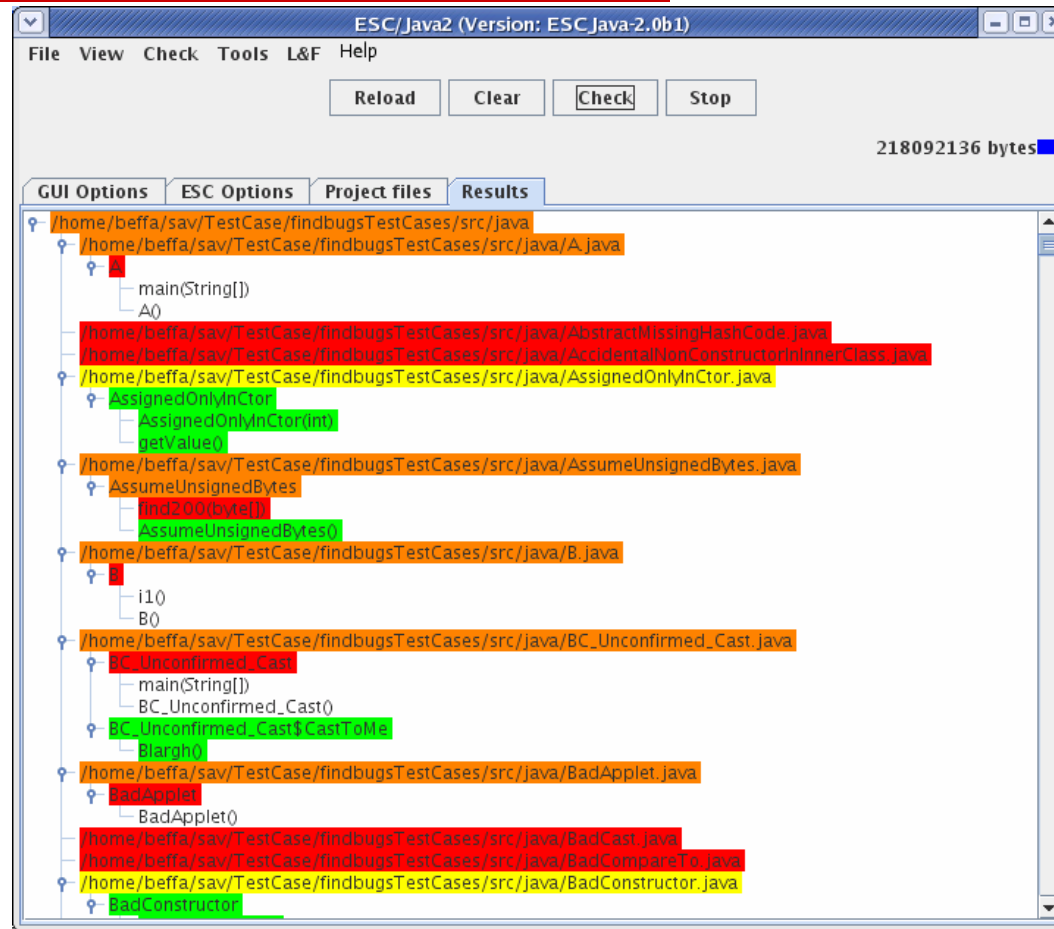
# ESC/Java2

---

- static analyzer of Java source code with formal annotations
  - parsing, type and static checking
- support only java 1.4
- reasons about each methods individually

```
class A{
  byte[] b;
  //@ ensures b != null && b.length = 20;
  public void n() { a = new byte[20]; }
  public void m() { n();
    b[0] = 2;
  }
}
```

# ESC/Java2 GUI

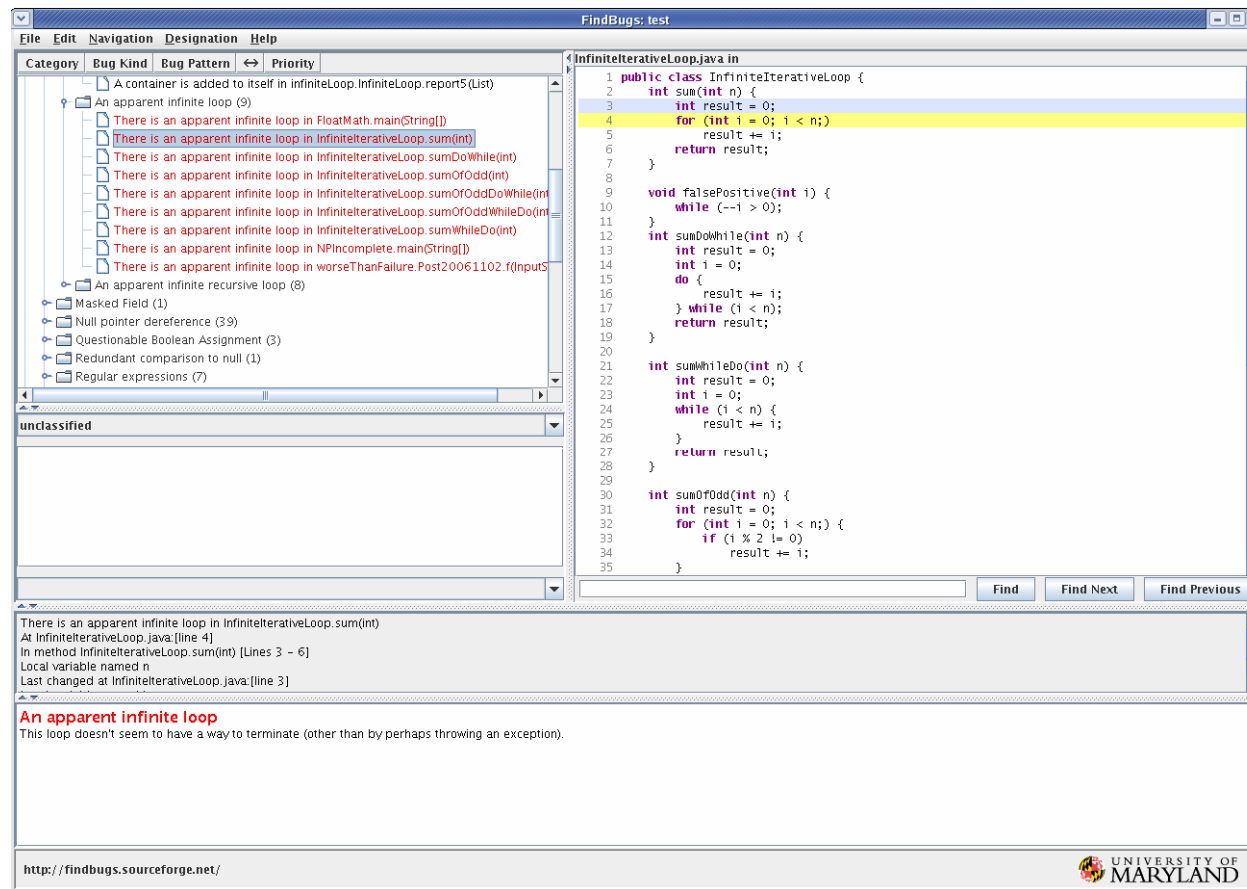


# FindBugs

---

- static analysis of Java bytecode
  - But adding source code helps reviewing warnings
- 3 categories of bugs
  - Correctness bug (probable bug)
  - Bad Practice (violate recommended coding practice)
  - Dodgy (confusing and prone to errors)

# FindBugs GUI



# JLint

---

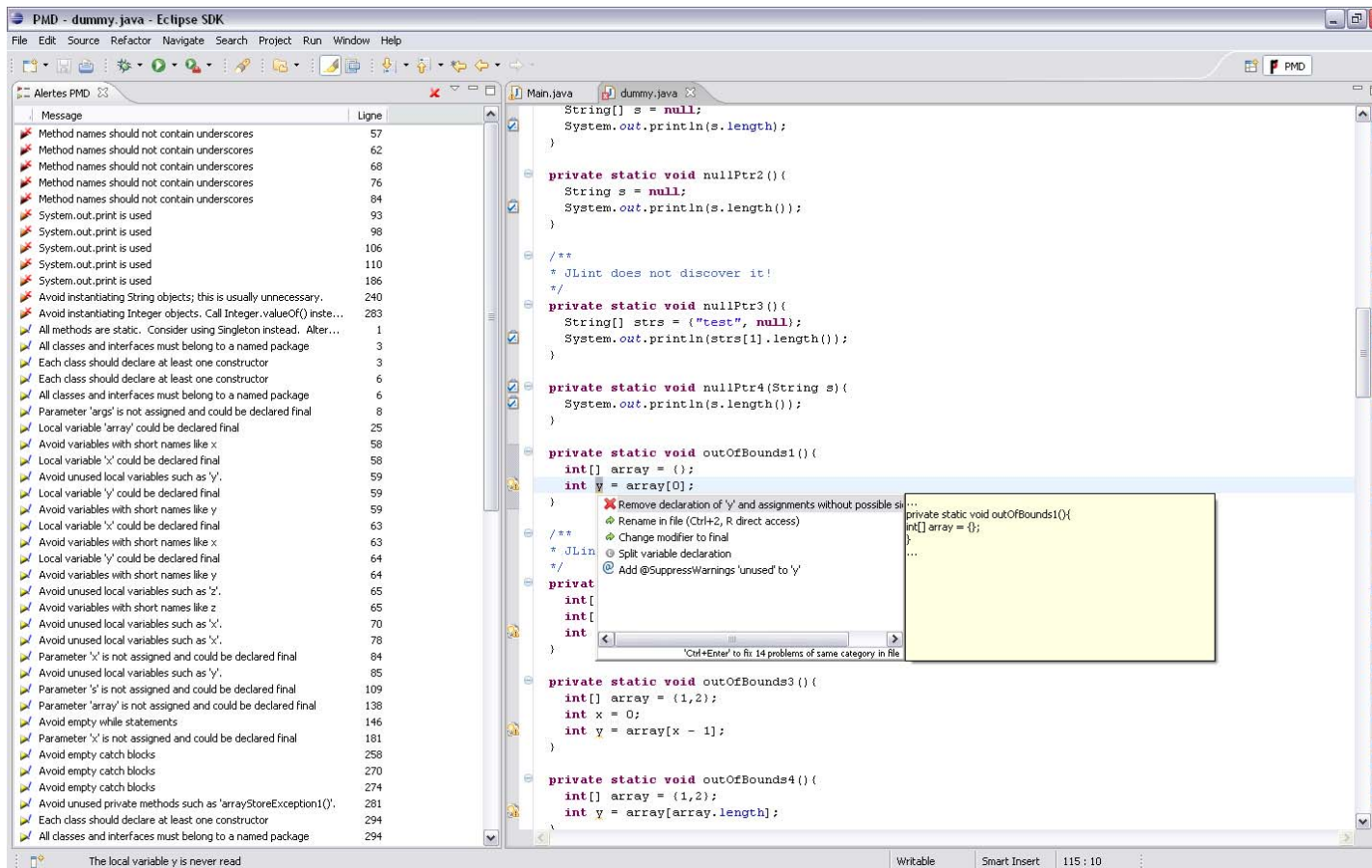
- Automated Java verification tool
  - With data flow analysis
- Intra-procedural only
- Many bugs and bad practices
  - nullpointer exception
  - arithmetic exception
  - array out of bounds
  - deadlocks
  - variable shadowing
  - zero operands
- No GUI

# PMD

---

- bug depending on programming style
  - empty try/catch/finally
  - dead code unused local variables
  - overcomplicated expression
    - unnecessary "if", "for" that could be a "while"
- based on rulesets (Java or XPath expression)
  - Possibility to choose which one to use
- Some warning examples
  - method names should not contain underscores
  - System.out.print is used
  - Avoid instantiating String, Integer objects

# PMD Eclipse plugin



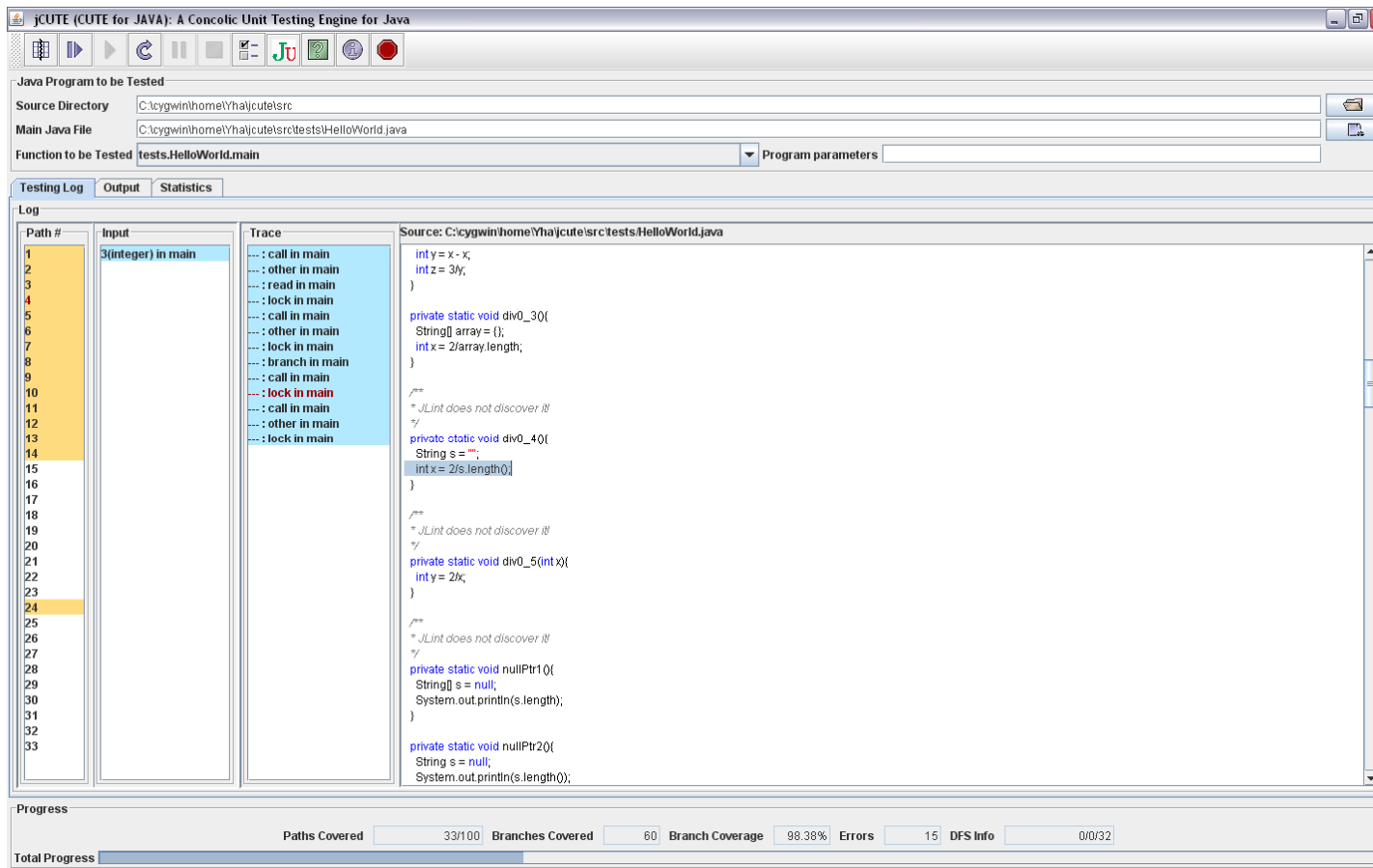
# jCUTE (Concolic Unit Testing Engine)

---

- ❑ Explore all distinct execution paths of a program
- ❑ Automatic testing by running path
- ❑ Catch dataraces and deadlocks
- ❑ How it found errors
  - detects with java runtime exception
  - detects infinite recursion by StackOverflow
  - detects infinite loop by OutOfMemory
  - "while(true){}" problem



# jCUTE GUI



The screenshot shows the jCUTE GUI interface. At the top, the title bar reads "jCUTE (CUTE for JAVA): A Concolic Unit Testing Engine for Java". Below the title bar is a toolbar with various icons. The main area is divided into several sections:

- Java Program to be Tested:**
  - Source Directory: C:\cygwin\home\Yha\jcute\src
  - Main Java File: C:\cygwin\home\Yha\jcute\src\tests\HelloWorld.java
  - Function to be Tested: tests.HelloWorld.main
  - Program parameters: (empty)
- Testing Log:**
  - Log: A table with columns Path #, Input, Trace, and Source. Path # 1-33 are listed. Input is "3(integer) in main". Trace shows call stack entries like ": call in main", ": other in main", ": read in main", ": lock in main", ": call in main", ": branch in main", ": lock in main", ": call in main", ": other in main", ": lock in main". Source shows code snippets for div0\_30, div0\_40, div0\_5, and nullPtr1/2.
- Progress:**
  - Paths Covered: 33/100
  - Branches Covered: 60
  - Branch Coverage: 98.38%
  - Errors: 15
  - DFS Info: 0/0/32

# Experiment results

Type	DIV 0					Null ptr				Bounds					Inf L				Inf R		
Bug	1	2	3	4	5	1	2	3	4	1	2	3	4	5	1	2	3	4	1	2	3
JLint	x	x	x				x		x	x		x	x	x				x			
jCUTE	x	x	x	x	x	x	x	x	x	x	x	x	x	x	-	x	x	x	x	x	x
ESC / Java 2	x	x	x	x	x	x	x	x	x	x		x	x								
FindBugs						x	x			x		x	x					x			
PMD																					

Type	Type Ov		Cast	String Cmp		Stream Close		Store	Deadlock		Score
Bug	1	2	1	1	2	1	2	1	1	2	
JLint											10 / 31
jCUTE			x							x	23 / 31
Esc / Java 2	-	-	x	-	-	-	-	x			14 / 31
FindBugs						x	x				8 / 31
PMD											0 / 31

# Meta tools

---

- Main challenges
  - Avoid duplicated error messages
  - Use underlying tools efficiently

# Duplicated errors

---

- Standardize error system
  - Giving pairs (err\_no, line\_range)
  - By modifying tools (if opensource)
  - By writing wrappers otherwise
- This enable some new features
  - Duplicated message detection
  - Ordering bugs by severity

# Bottom-up approach

---

- Developers debug in a bottom-up fashion
- => Use same layered approach for meta-tools
  - First check for syntax
  - Check intra-procedural problems
  - Check inter-procedural problems
  - Finally global bugs
- Mask inappropriate messages

# Thoughts about the miniproject

---

- Many bugs are not found
  - However many warning around could help to find them
- Writing bug cases is tedious
- Bug finding tools are not bug free
  - Not so easy to get them working

# Future work directions

---

- Build the theoretical meta-tool
  - Testing it on real project
- Analyzing new java bug tools
  - Chord, ..
- Analyzing C/C++ bug tools



ÉCOLE POLYTECHNIQUE  
FÉDÉRALE DE LAUSANNE

# Questions?

---