COMP5111 – Fundamentals of Software Testing and Analysis

Pattern Based Static Analysis

Shing-Chi Cheung

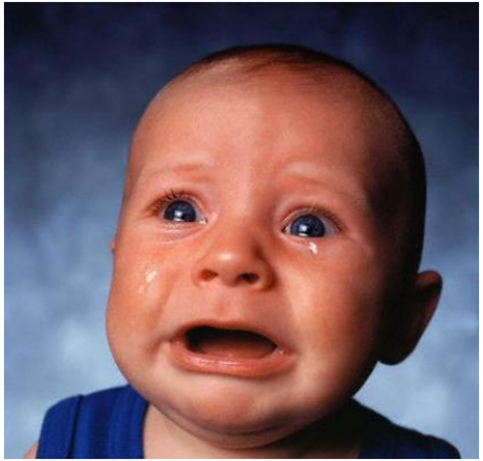
Computer Science & Engineering

HKUST

Adapted from Bill Pugh's talk on Effective use of SpotBugs for Large Software Development Efforts, Emerging Technologies for the Enterprise, April 2012.

FIRST ATTEMPT

My assignment is due tomorrow. Can someone find the bugs for me, ppplease …

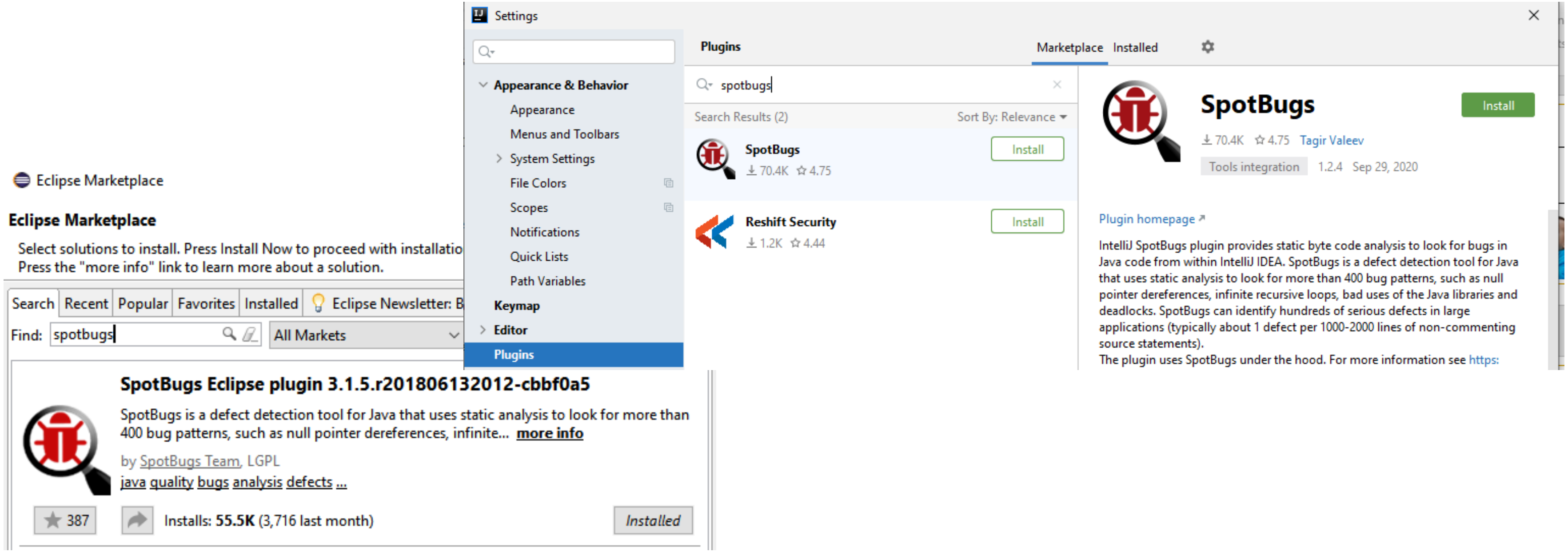


BUT I don't have time to

- write tests
- prepare input data files
- explain what my code is supposed to do

Can I have someone to find my mistakes before I have finished coding …

SpotBugs



Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installatio
Press the "more info" link to learn more about a solution.

Find: spotbugs

SpotBugs Eclipse plugin 3.1.5.r201806132012-cbbf0a5

SpotBugs is a defect detection tool for Java that uses static analysis to look for more than 400 bug patterns, such as null pointer dereferences, infinite... **more info**

by SpotBugs Team, LGPL

java quality bugs analysis defects ...

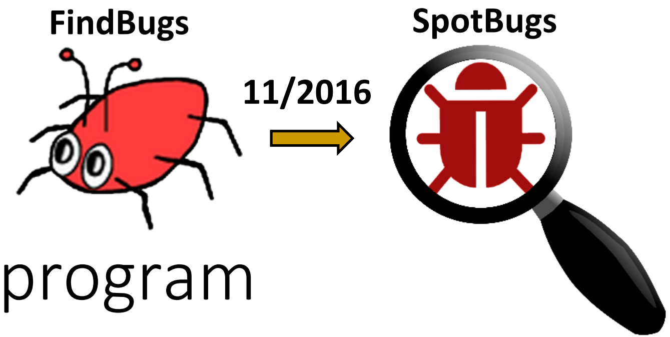
Installs: **55.5K** (3,716 last month)

spotbugs

IntelliJ SpotBugs plugin provides static byte code analysis to look for bugs in Java code from within IntelliJ IDEA. SpotBugs is a defect detection tool for Java that uses static analysis to look for more than 400 bug patterns, such as null pointer dereferences, infinite recursive loops, bad uses of the Java libraries and deadlocks. SpotBugs can identify hundreds of serious defects in large applications (typically about 1 defect per 1000-2000 lines of non-commenting source statements).
The plugin uses SpotBugs under the hood. For more information see https:

One of the popular Eclipse and IntelliJ Plugins!!

What is SpotBugs?



- A static analyzer that detects potential bugs without executing a program
- Mandated by many open source projects or organizations (Google, Goldman-Sachs, …) as a presubmit check in code review
- Able to detect functional and performance issues
- Have been popularly used by practitioners to detect many types of common, hard-to-find bugs

Creators of FindBugs [OOPSLA 2004]

- Bill Pugh & David Hovemeyer

- Professors at the University of Maryland
- Succeeded by SpotBugs under GPL license in 11/2016

Tackles hard programming issues …

- Aliasing
- Infeasible paths
- Inappropriate call targets
- Providing feedback to developers under what conditions an error can happen
- Operates directly on Java bytecode

Full List of Issues Supported:

- Bad practice (BAD_PRACTICE)
- Correctness (CORRECTNESS)
- Experimental (EXPERIMENTAL)
- Internationalization (I18N)
- Malicious code vulnerability (MALICIOUS_CODE)
- Multithreaded correctness (MT_CORRECTNESS)
- Performance (PERFORMANCE)
- Security (SECURITY)
- Dodgy code (STYLE)

Insight

- There are common coding mistakes across projects
- The bug patterns of these mistakes can be mined from the patches in open source projects
- Detect bugs by the existence of such patterns in target code
- No program execution is needed
- No specification is needed
- No tests are needed

How it works?

- Use "bug patterns" to detect potential bugs
- Example patterns

NullPointerException

```
if (infile == null)
   infile.close()
```

When programmers are tired or on Friday afternoon, they can confuse || with && and != with ==

```
Address address = client.getAddress();
if ((address != null) || (address.getPostCode() != null))
   ...
```

What’s wrong with it?

- Eclipse 3.7 (org.eclipse.update.internal.ui.views.FeaturesStateAction)

```
public void run() {
  try {
    if ((adapters == null) && (adapters.length == 0))
      return;
    IStatus status = OperationsManager.getValidator().validatePlatformConfigValid();
    if (status != null)
      throw new CoreException(status);
   ...
}
```

What’s wrong with it?

```
public void setData(String keyName, String valName, HashMap hashMap) {
  if (hashMap != null)
    this.hashMap = hashMap;
  else
    this.hashMap = new HashMap(true);

  if (hashMap.size() > 0)   {
     …
  }
}
```

Sounds like missing some updates from hashMap to this.hashMap in program refactoring.

More example patterns

Uninitializedfield

```
public class ShoppingCart {
  private List items;
  public addItem(Item item) {
    items.add(item);
  }
}
```

Dead code

```
x = null;
… does not assign x …
if (x != null) {
  // non-trivial dead code
  x.aMethod();
}
```

More example patterns

Bad covariant definition of equals

public boolean equals(Foo obj) { …}

Equal objects have different hashcodes

A class redefines equals() but not hashcode(), or vice versa

Inconsistent synchronization

There seems to be a high (1/3 or more) proportion of unlocked access to a mutable field:
2(RU + WU) > (RL +2WL)

…

What does it print?

```
public class ShortSet {
  public static void main(String args[]) {
    Set<Short> s = new HashSort<Short>();
    for (short i = 0; i < 100; i++) {
      s.add(i);  // add a new item
      s.remove(i – 1);  // remove the previous item
    }
    System.out.println(s.size());
  }
}
```

(a) 1
(b) 100
(c) Throws exception
(d) None of the above

What does it print?

(a)1

(b)100

(c)Throws exception

(d)None of the above

What does it print?

```
public class ShortSet {
  public static void main(String args[]) {
    Set<Short> s = new HashSort<Short>();
    for (short i = 0; i < 100; i++) {
      s.add(i);  // add a new item
      s.remove(i – 1);  // int-valued expression
    }
    System.out.println(s.size());
  }
}
```

What does it print?

```
public interface Set<E> extends Collection<E>{
  public abstract boolean add(E e);
  public abstract boolean remove(Object o);
  …
}
```

- Method takes a nearly untyped parameter Object
- Fails to detect mismatched type at both compile time and run time
- Simply does nothing
- Common to classes in Collection Framework!
- So implemented for backwards compatibility

How to fix it?

```
public class ShortSet {
  public static void main(String args[]) {
    Set<Short> s = new HashSort<Short>();
    for (short i = 0; i < 100; i++) {
      s.add(i);  // add a new item
      s.remove((short) (i – 1));  // short-valued expression
    }
    System.out.println(s.size());
  }
}
```

What SpotBugs can do?

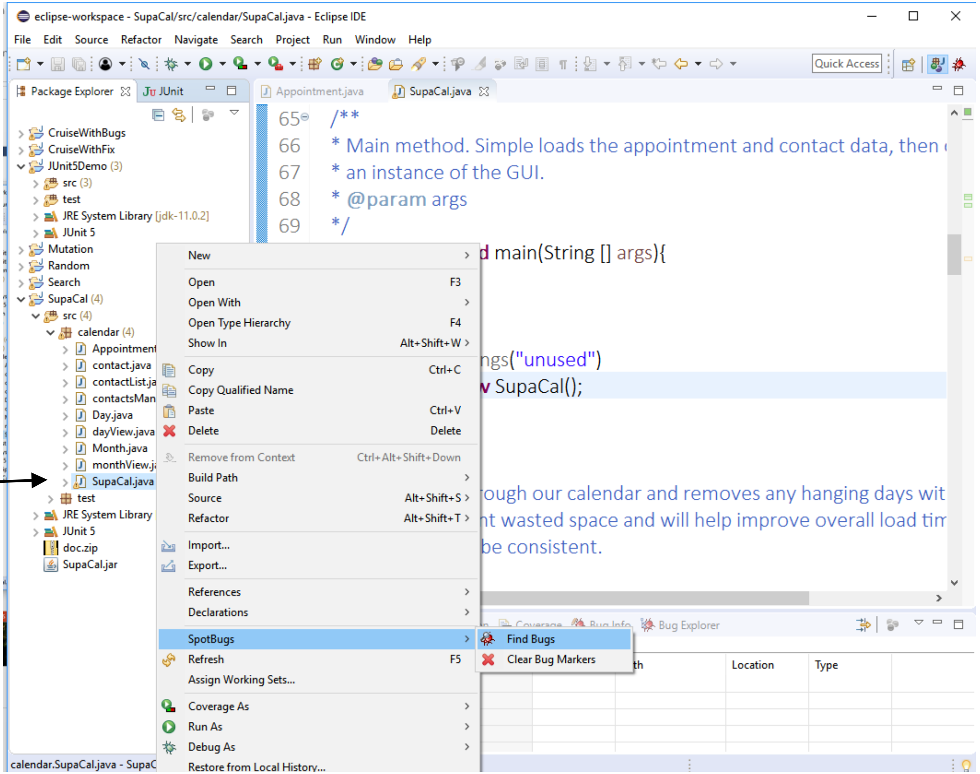
- It comes with 200+ rules divided into different categories:
  - Correctness
    - e.g., infinite recursive loop, reads a field that is never written
  - Bad practice
    - e.g., code that drops exceptions or fails to close a file
  - Performance
  - Multithreaded correctness
  - Dodgy
    - e.g., unused local variables or unchecked casts

Using SpotBugs – SpotBugs Attributes

- Each bug warning has five attributes
  - Confidence (low, medium, high)
  - Rank (1 – 20)
    - Scariest (1-4) / Scary (5-9) / troubling (10-14) / of concern (15-20)
  - Type (e.g., HE_EQUALS_USE_HASHCODE)
  - Pattern (e.g., HE)
  - Category (e.g., Bad Practice)

Invoking SpotBugs

Select the project or a particular Java file

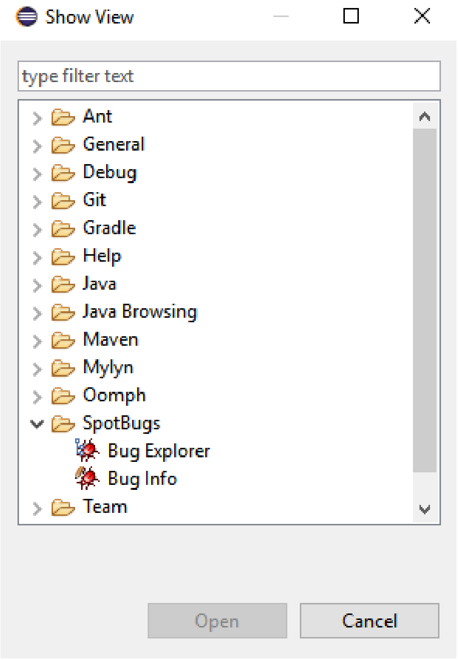


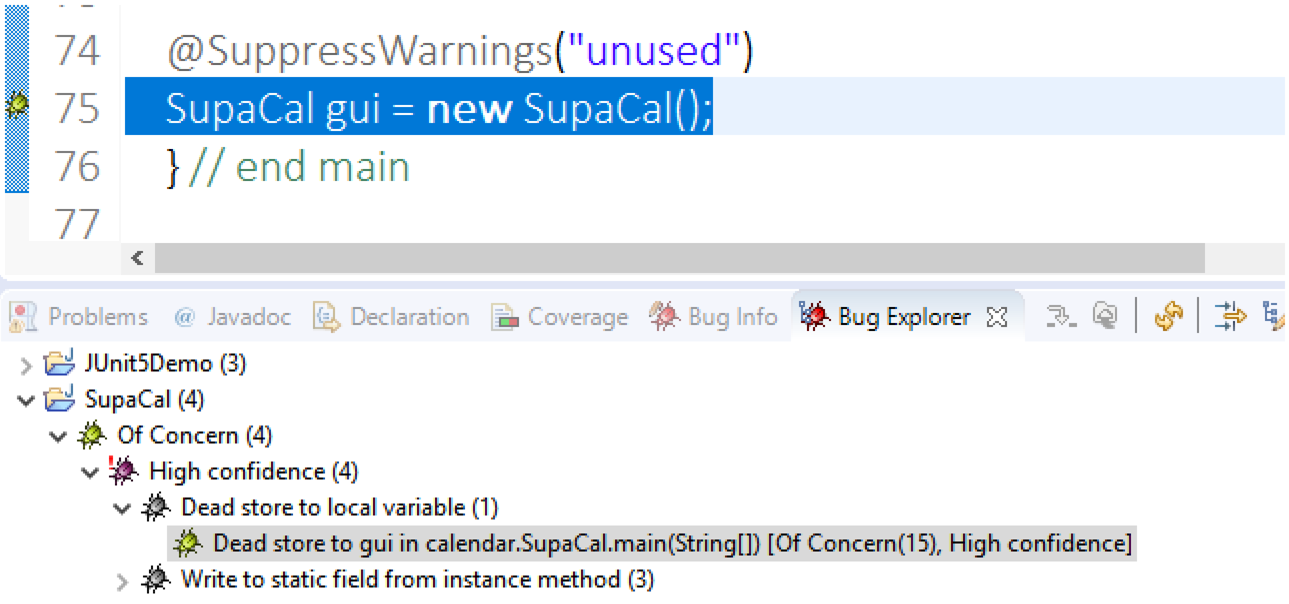
```
/**
 * Main method. Simple loads the appointment and contact data, then
 * an instance of the GUI.
 * @param args
 */
d main(String [] args){

ngs("unused")
v SupaCal();

ough our calendar and removes any hanging days wit
nt wasted space and will help improve overall load tim
be consistent.
```

Using SpotBugs – Bug Explorer



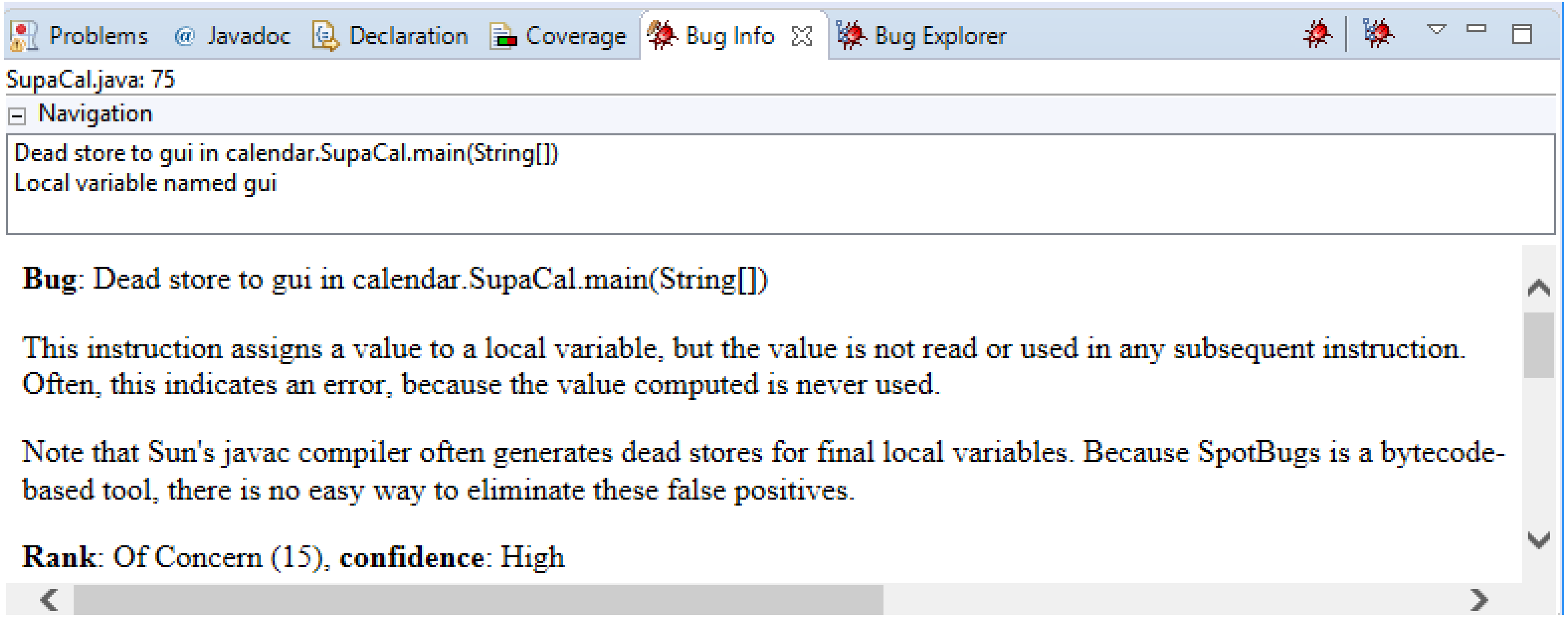


```
    @SuppressWarnings("unused")
    SupaCal gui = new SupaCal();
} // end main
```

```
> JUnit5Demo (3)
v SupaCal (4)
  v Of Concern (4)
    v High confidence (4)
      v Dead store to local variable (1)
          Dead store to gui in calendar.SupaCal.main(String[]) [Of Concern(15), High confidence]
      > Write to static field from instance method (3)
```

Windows → Show View → Other … → SpotBugs

Using SpotBugs – Bug Info



SupaCal.java: 75

Navigation

Dead store to gui in calendar.SupaCal.main(String[])
Local variable named gui

Bug: Dead store to gui in calendar.SupaCal.main(String[])

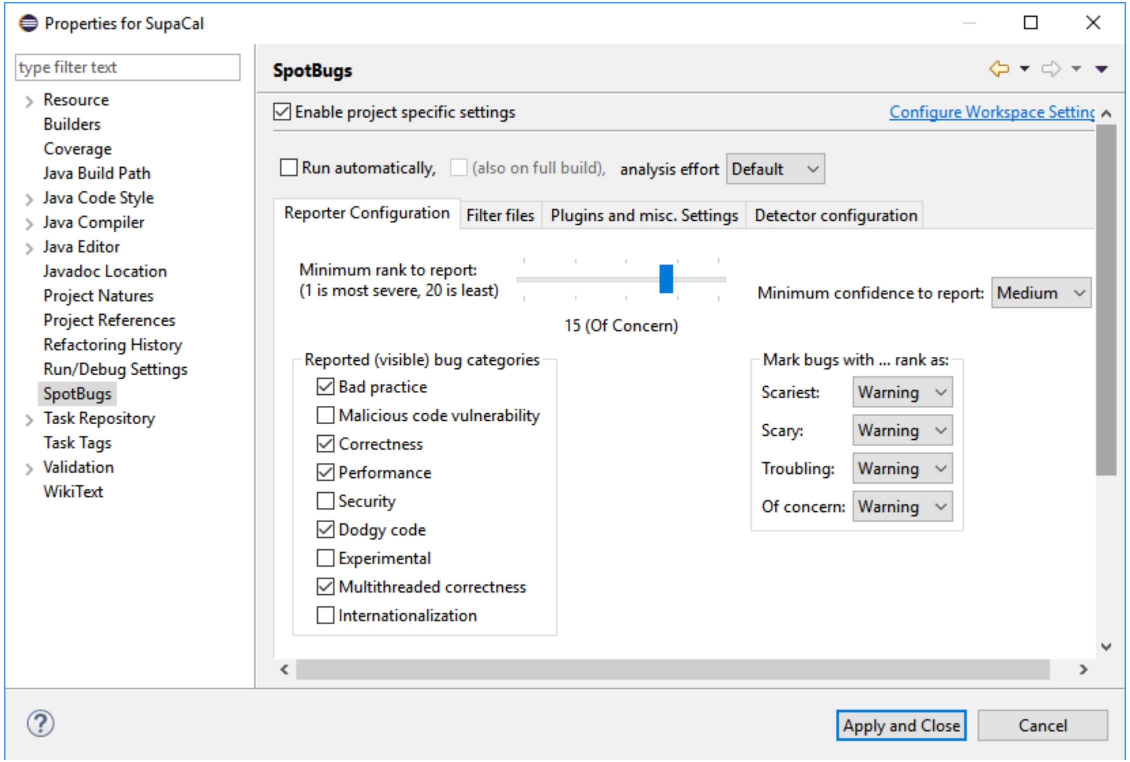
This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used.

Note that Sun's javac compiler often generates dead stores for final local variables. Because SpotBugs is a bytecode-based tool, there is no easy way to eliminate these false positives.

Rank: Of Concern (15), **confidence**: High

Using SpotBugs - Filtering Bugs

Select the Properties of a target project -> SpotBugs



Properties for SupaCal

SpotBugs

☑ Enable project specific settings

Configure Workspace Settinc

□ Run automatically, □ (also on full build), analysis effort Default

Minimum rank to report:
(1 is most severe, 20 is least)

15 (Of Concern)

Minimum confidence to report: Medium

Reported (visible) bug categories

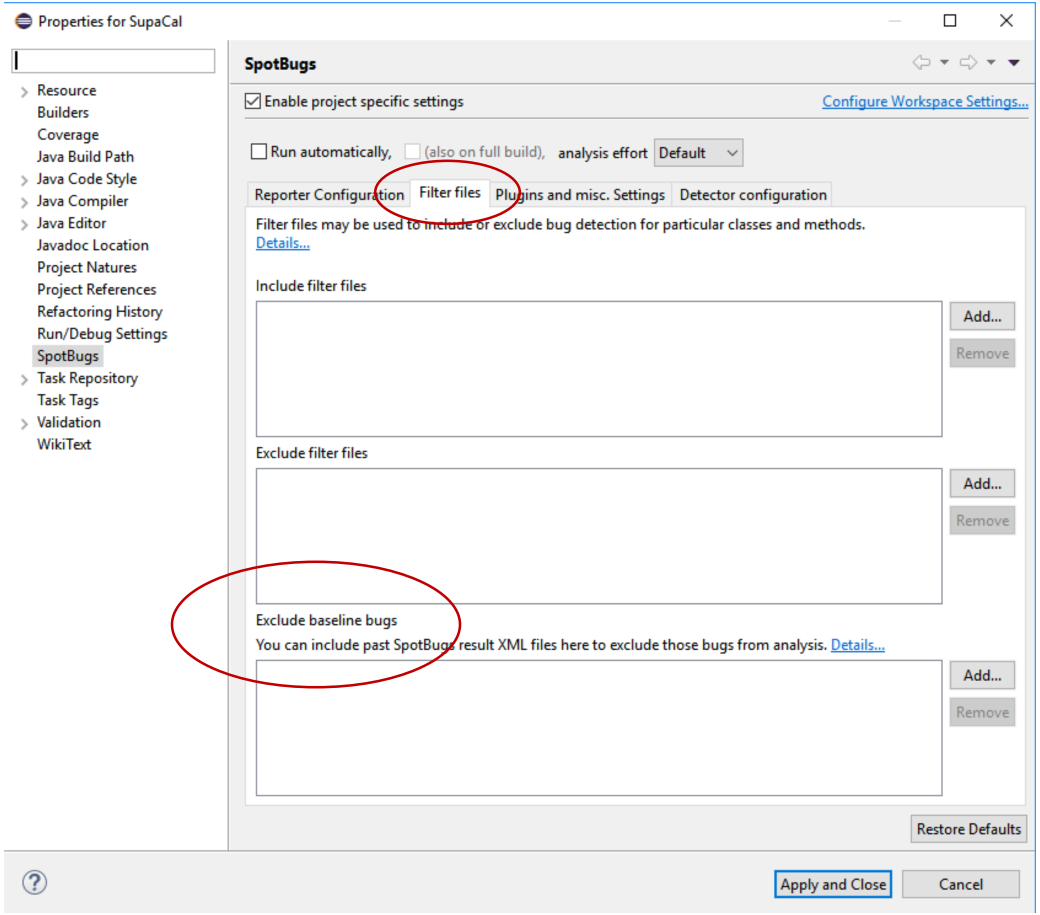
☑ Bad practice
□ Malicious code vulnerability
☑ Correctness
☑ Performance
□ Security
☑ Dodgy code
□ Experimental
☑ Multithreaded correctness
□ Internationalization

Mark bugs with ... rank as:

Scariest:	Warning
Scary:	Warning
Troubling:	Warning
Of concern:	Warning

Baseline bugs

- Show only new bug warnings
- Exclude all warnings arising from the previous release
- Commonly used by practitioners



Properties for SupaCal

SpotBugs

☑ Enable project specific settings — Configure Workspace Settings...

□ Run automatically, □ (also on full build), analysis effort: Default

Filter files may be used to include or exclude bug detection for particular classes and methods. Details...

Include filter files

Exclude filter files

Exclude baseline bugs

You can include past SpotBugs result XML files here to exclude those bugs from analysis. Details...

SpotBugs communal cloud

- SpotBugs project is hosting a free server to record and share information about bugs
- It records when a bug was first seen, and any evaluation of the issue by developers
  - e.g., “On Jan 11th, Sam marked this as a “Should Fix” issue and said “…”
- Useful for open source and other non-confidential source code

Lessons Learnt with SpotBugs

- May not need to fix all warnings
  - Many warnings do not necessarily cause problems in program execution
  - They are often just inconsistencies
- Engineering effort is limited and zero sum
- Aim at getting the best return of our time in using SpotBugs

A Case Study of FindBugs

- According to a Google report in May 2009,
  - 4,000 issues were to review based on the bug patterns most relevant to Google
  - Out of 4,000 reviews conducted:
    - 600+ fixes
    - 1,800+ bugs were filed
    - 1,500+ issues removed

Lessons Learnt with SpotBugs

- Static analysis, at best, might catch 5-10% of software quality problems
- Need to understand which types of bugs matter to our project
  - If we have a public static final field pointing to an array, anyone can change the content of the array
  - It is likely a big concern if our program run with other untrusted code in the same VM
  - Otherwise, it is likely a minor concern

Lessons Learnt with SpotBugs

- Only turn on those rules of our interest
- Analogous to compiler warnings
  - Try to fix the ones we care
  - Disable the ones we don’t care
- Less than 2% of changes introduce new serious issues and detected by SpotBugs
- Rank the reported issues from 1 to 12
- Incomplete/bad bug fixes ranges from 5-30%

Lessons Learnt with SpotBugs

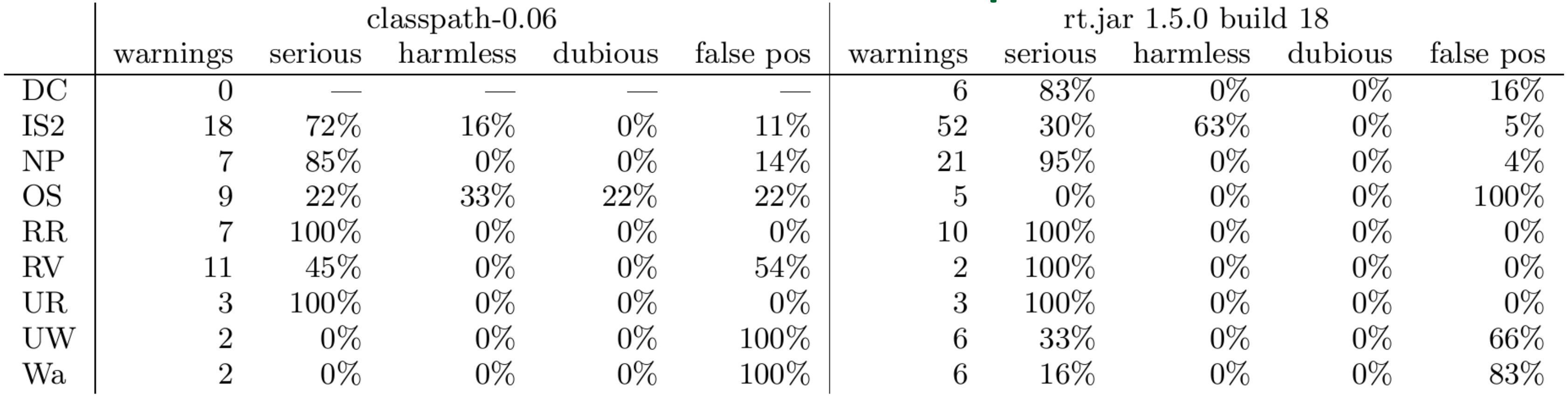
- High false positives
  - Only a small portion of warnings are real; most of them are false alarms
  - Allow users to enable/disable the rules individually
  - Focus on the new ones induced by the last commit

Facts of real projects (based on FindBugs)

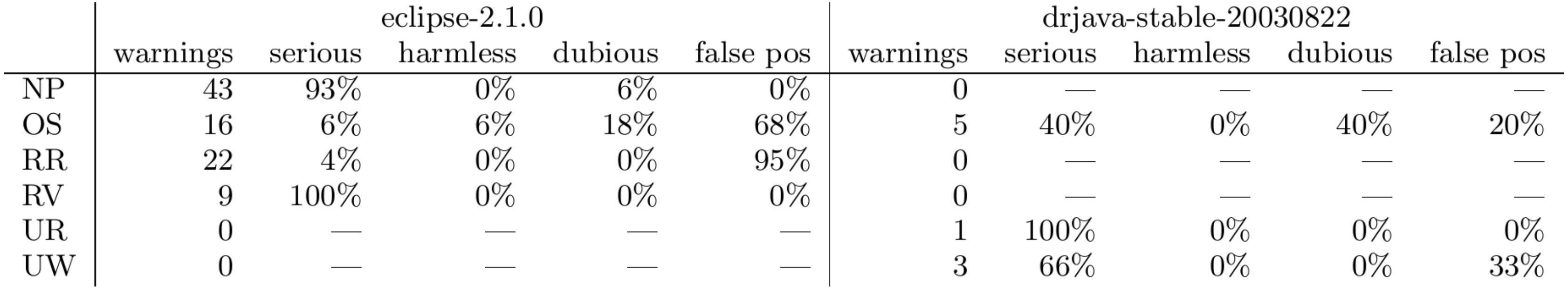
- Eclipse is known for its high reliability
- Defect density of Eclipse 3.0
  - Scariest defects: 30 per million LOC
  - Scary: 160 per million LOC
  - Troubling: 480 per million LOC
  - Of concern: 6,000 per million LOC

Evaluation with selected patterns

Inconsistent
Sync
Null Pointer



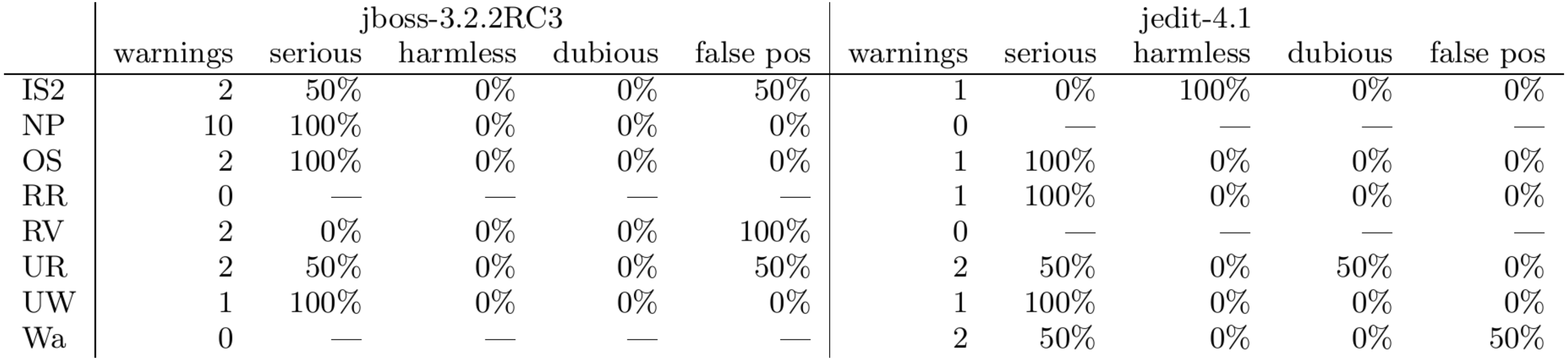
	classpath-0.06					rt.jar 1.5.0 build 18				
	warnings	serious	harmless	dubious	false pos	warnings	serious	harmless	dubious	false pos
DC	0	—	—	—	—	6	83%	0%	0%	16%
IS2	18	72%	16%	0%	11%	52	30%	63%	0%	5%
NP	7	85%	0%	0%	14%	21	95%	0%	0%	4%
OS	9	22%	33%	22%	22%	5	0%	0%	0%	100%
RR	7	100%	0%	0%	0%	10	100%	0%	0%	0%
RV	11	45%	0%	0%	54%	2	100%	0%	0%	0%
UR	3	100%	0%	0%	0%	3	100%	0%	0%	0%
UW	2	0%	0%	0%	100%	6	33%	0%	0%	66%
Wa	2	0%	0%	0%	100%	6	16%	0%	0%	83%



	eclipse-2.1.0					drjava-stable-20030822				
	warnings	serious	harmless	dubious	false pos	warnings	serious	harmless	dubious	false pos
NP	43	93%	0%	6%	0%	0	—	—	—	—
OS	16	6%	6%	18%	68%	5	40%	0%	40%	20%
RR	22	4%	0%	0%	95%	0	—	—	—	—
RV	9	100%	0%	0%	0%	0	—	—	—	—
UR	0	—	—	—	—	1	100%	0%	0%	0%
UW	0	—	—	—	—	3	66%	0%	0%	33%

http://SpotBugs.sourceforge.net/bugDescriptions.html

Evaluation with selected patterns



	jboss-3.2.2RC3					jedit-4.1				
	warnings	serious	harmless	dubious	false pos	warnings	serious	harmless	dubious	false pos
IS2	2	50%	0%	0%	50%	1	0%	100%	0%	0%
NP	10	100%	0%	0%	0%	0	—	—	—	—
OS	2	100%	0%	0%	0%	1	100%	0%	0%	0%
RR	0	—	—	—	—	1	100%	0%	0%	0%
RV	2	0%	0%	0%	100%	0	—	—	—	—
UR	2	50%	0%	0%	50%	2	50%	0%	50%	0%
UW	1	100%	0%	0%	0%	1	100%	0%	0%	0%
Wa	0	—	—	—	—	2	50%	0%	0%	50%

Comparison of similar tools

Product	License	Type	Languages	Features
LAPSE+	Open Source GNU GPL	Eclipse Plugin	Java	Variable Traceback, security vulnerabilities detection (e.g., injection & cross-site scripting)
SpotBugs	Open Source GNU GPL	Eclipse Plugin	Java	General purpose bug detection, nice interface, security vulnerabilities detection
Orizon	Open Source GNU GPL	Standalone Text-based	Java, PHP, C, JSP	Report-based scheme, no fancy UI, security vulnerabilities detection
FindNPE	Free for non-commercial use	Eclipse Plugin	Java	Detect potential null pointer dereferences, pretty precise
PMD	Open Source BSD, EPL	Eclipse Plugin	Java, Javascript, XML, XSL	Generic code quality tool, nice user interface, extensible by specifying additional rules*

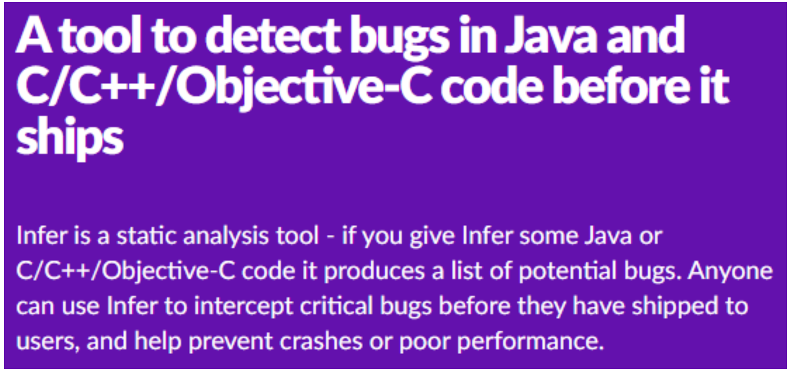
Adapted from Lewis Sykalski's presentation on Security WebApps – A Survey of Vulnerabilities & Static Analysis Tools

Comparison of similar tools

Product	License	Type	Languages	Features
FxCop	Open Source MS-PL	Visual Studio Plugin	.NET	Security-specific static analysis, UI built into Visual Studio
RIPS	Open Source GPL	Standalone	PHP	Professional user-interface, security-specific analysis
FlawFinder	Open Source GPL	Standalone Text-based	C++	Security-specific and dangerous function analysis
PreFast	Open Source MS-PL	Visual Studio Plugin	C++	General static analysis, UI built into Visual Studio
BrakeMan	Open Source MIT	Standalone Text-based	Ruby	Security-specific analysis
CheckThread	Open Source MIT	Eclipse Plugin	Java	Report thread confinement violations and race conditions

Adapted from Lewis Sykalski's presentation on Security WebApps – A Survey of Vulnerabilities & Static Analysis Tools

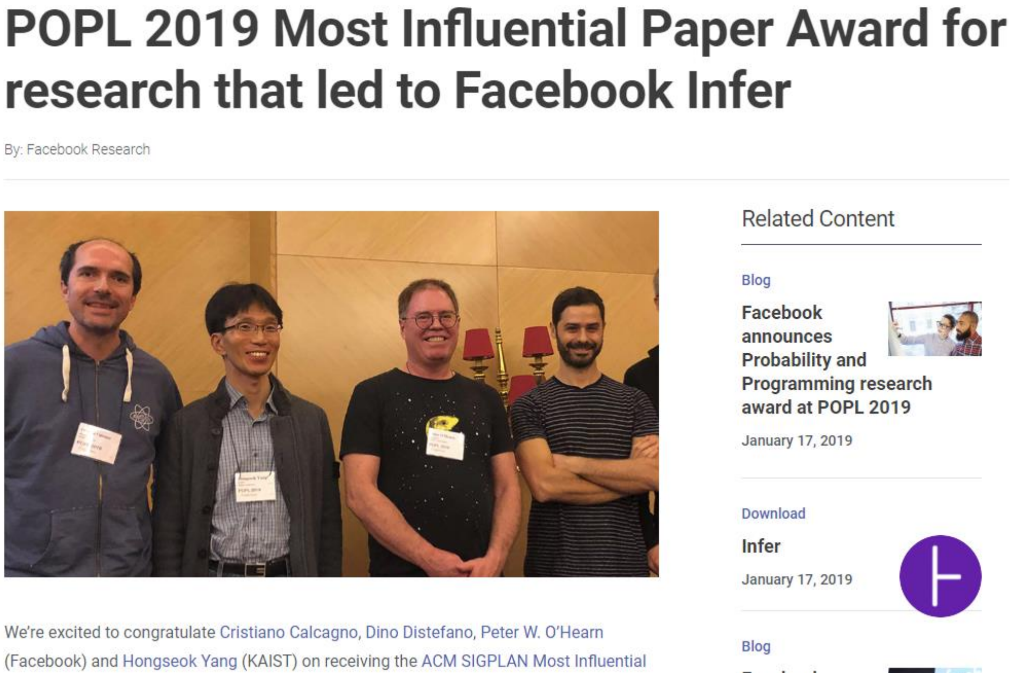
INFER – Pattern-Based Analyzer by Facebook



A tool to detect bugs in Java and C/C++/Objective-C code before it ships

Infer is a static analysis tool - if you give Infer some Java or C/C++/Objective-C code it produces a list of potential bugs. Anyone can use Infer to intercept critical bugs before they have shipped to users, and help prevent crashes or poor performance.

https://fbinfer.com/



POPL 2019 Most Influential Paper Award for research that led to Facebook Infer

By: Facebook Research

We're excited to congratulate Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn (Facebook) and Hongseok Yang (KAIST) on receiving the ACM SIGPLAN Most Influential

Related Content
Blog
Facebook announces Probability and Programming research award at POPL 2019
January 17, 2019
Download
Infer
January 17, 2019
Blog

After-class Exercise

- Install SpotBugs on your Eclipse and apply it to a couple of Java programs.

- SpotBugs Manual
  - https://spotbugs.readthedocs.io/en/latest/

- PMD (https://pmd.github.io/)

Further readings

- Effective Use of FindBugs in Large Software Development Efforts (video)
  - https://www.infoq.com/presentations/Effective-Use-of-FindBugs
- SpotBugs References
  - Manual: https://spotbugs.readthedocs.io/en/stable/
  - Github site: https://spotbugs.github.io/