Shanghai Jiao Tong University
Software Theory and Practice Group

XFindBugs: eXtended FindBugs for AspectJ

Haihao Shen, Sai Zhang, Jianjun Zhao, Jianhong Fang, Shiyuan Yao

Software Theory and Practice Group (STAP)
Shanghai Jiao Tong University, China

A code snippet in AspectJ

```
public class A {
    public String s = "Initialize s";
1   public static void  main (String args[]){
2       new A();
    }
}
public aspect B {
3   pointcut beforeInitialize(A a):execution(A.new())&&this(a);
4   before(A a):beforeInitialize(a){
5       if(!a.s.equals("some value")) {...}
    }
}
```

No !!

a is null !!

// Bug Pattern AFBI

Bugs are common in AspectJ programs

- Why ?
  - Most AspectJ beginners are used to writing code with ***Java programming specification***, which may not consist with AspectJ one.
  - AspectJ is a new paradigm, and ***AspectJ compiler*** is not so robust as Java compiler.

- **XFindBugs: eXtended FindBugs in AspectJ Programs**

Outline

- Background
- Bug patterns in AspectJ
- Implementation issues on XFindBugs
- Empirical evaluation
- Related work
- Conclusion

Outline

- **Background**
- Bug patterns in AspectJ
- Implementation issues on XFindBugs
- Empirical evaluation
- Related work
- Conclusion

Background

- **FindBugs**
  - **FindBugs** is one of the most widely-used bug finding tools in Java community.
  - **FindBugs** analyzed all **89** publicly available builds of JDK and generated over **370** warnings.
  - There are totally **1127** warnings reported by **FindBugs** in Google's Java code base.

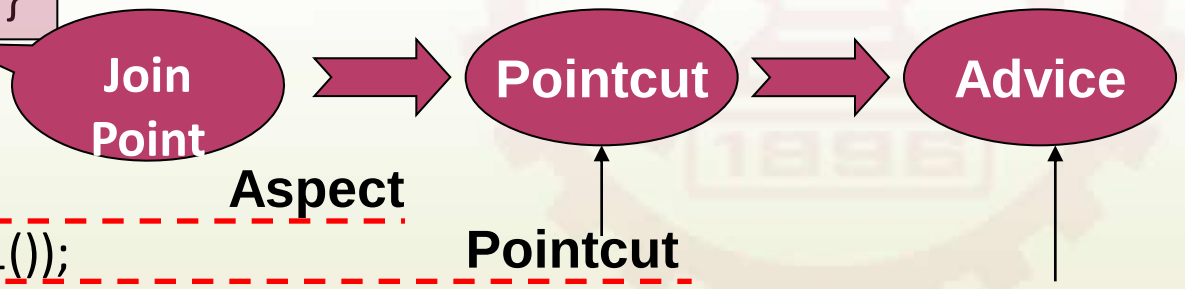
An AspectJ program

```
1. public class C {
2.    private int f1 = 0;
3.    public void m (){ m1 ();}
4.    public void m1(){}
5. }
6. public aspect A {
7.    pointcut pc():call(* m1());
8.    before():pc(){
9.      System.out.println("…");}
10. }
11. public aspect B {
12.   private float C.f2 = 0.0f;
13.   public int C.m2() {return 0;}
14. }
```

Class

Fields: f1, f2

Methods: m, m1, m2



Intertype-field

Intertype-method

Introduction

Error-prone features in AspectJ programs

- **Pointcut**
  - Join point model in a lexical-level
  - Using wildcards in pointcut designators e.g., call (* *.*)
- **Advice**
  - Inconsistent advice invocation sequence
  - *Proceed* in around advice
- **Introduction (Intertype-declaration)**
  - Altering the original class hierarchy dramatically
  - Introducing a same field many a time

Outline

- Background
- **Bug patterns in AspectJ**
- Implementation issues on XFindBugs
- Empirical evaluation
- Related work
- Conclusion

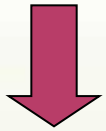
Bug patterns in AspectJ *

Pattern ID	Short Description	Category	Priority
AFBI	Access Field Before Object Initialization	Advice	Medium
MOAR	Mismatching Of After Returning	Advice	Medium
SA	Singleton Aspect	Advice	High
TROP	The Return Of Proceed	Advice	Medium
TNP	The Negated Pointcut	Pointcut	Low
UID	Unchecked Intertype Declarations	Introduction	Medium
**** The other 11 bug patterns could be found in our technical report.***			

Table 1

TROP: The Return Of Proceed

```
public interface I {
1   public Integer i = new Integer(3);  // ➔ public static final
}
public aspect B {
    Integer around(int val): call(Integer.new(int))&& args(val) {
2       Object result = proceed(val);  // ➔ assign twice
            return (Integer)result;
    }
}
public class A implements I {
    public static void main(String[] args) {...}
}
```



Outline

- Background
- Bug patterns in AspectJ
- **Implementation issues on XFindBugs**
- Empirical evaluation
- Related work
- Conclusion

XFindBugs Implementation

- Build on top of the FindBugs analysis framework
- Add corresponding bug detector for each bug pattern
- Search and compare the signature of bug pattern

- XFindBugs can support AspectJ compiler version 1.5 now.

Outline

- Background
- Bug patterns in AspectJ
- Implementation issues on XFindBugs
- **Empirical evaluation**
- Related work
- Conclusion

Empirical evaluation

- Research questions
- Subject programs
- Experimental procedures
- Experimental results
- Experimental conclusions

Research questions

- Do the bug patterns defined in this paper exist in real-world AspectJ applications?
- Can the tool XFindBugs find real potential defects?
- Can XFindBugs scale to large applications, or is there a real necessity for the usage of our tool?

Subject programs

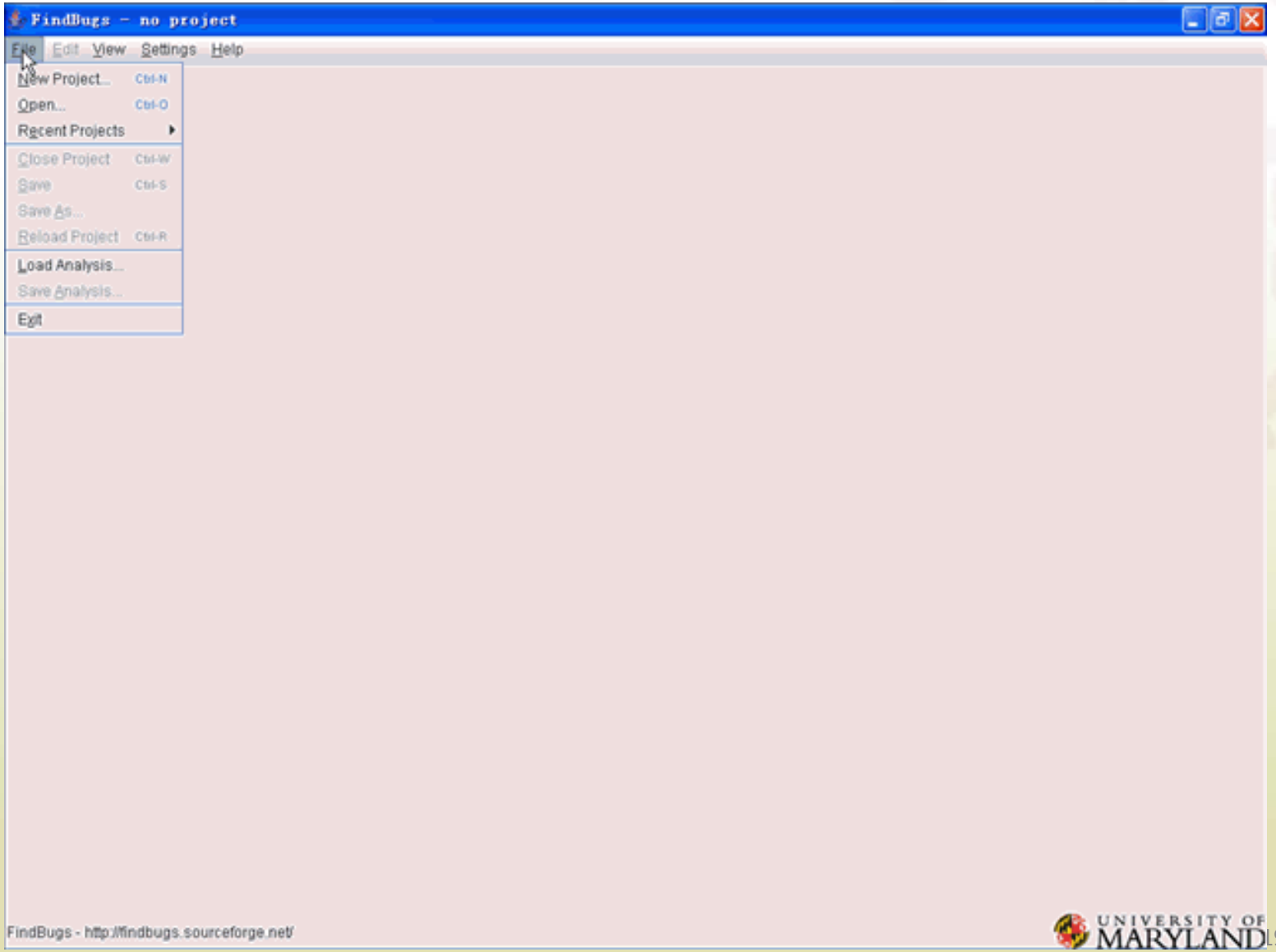
Name	LOC	#Advice	#Pointcut	#Introduction
AJHotdraw	38846	48	33	54
AJHSQLDB	123661	30	38	0
GlassBox	39220	132	183	44
ajc Benchmarks	4656	44	30	27
Abc Benchmarks	89596	54	54	87
Design Patterns	10821	15	24	43

Table 2

Experimental procedures

- We extract the existing bug reports from AspectJ Bugzilla and run XFindBugs on the reported buggy code.
- We also run XFindBugs on the subject programs listed in Table 2.

Demo for finding bugs in AJHotdraw



Experimental results

- **Defects from AspectJ Bugzilla**
  - *Bugzilla number:* ***195794, 148644, 165810, 145391, 218023,*** and ***72834***
  - *XFindBugs confirmed* ***7*** *bug instances.*
- **Defects in Subject Programs**
  - *XFindBugs reported* ***257*** *bug instances in all.*
  - *Among them, there are* ***1***, ***10***, *and* ***147*** *instances in* ***GlassBox***, ***AJHotdraw***, *and* ***AJHSQLDB***, *respectively.*

Some typical bug instances (1)

#org.jhotdraw.ccconcerns.commands.UndoableCommand.aj

```
84: void around(DrawingView drawingView):
      callCommandFigureSelectionChanged(drawingView) {

85:       AbstractCommand command =
            (AbstractCommand)thisJoinPoint.getTarget();
86:       command.hasSelectionChanged = true;
87:       proceed(drawingView);
88: }
```

static method

return null

nullpointer exception

Misuse Of GetTarget in AJHotdraw !!

Some typical bug instances (2)

#glassbox.monitor.resource.JdbcMonitor.aj

```
182: before(Statement statement, String sql) :
         topLevelDynamicSqlExec(statement, sql) {
183:     if (sql == null) {
184:         sql = “I won’t be changed.";
185:     }
186:     ...
187: }
```

Warning !!

The Scope Of Advice in GlassBox !!

Experimental conclusions

- Overall false positive ratio is **8.0%**.
- XFindBugs scales well to over **300KLOC**.
- XFindBugs not only confirms the reported bugs, but also reports **257** previously unknown defects.

Related work

- **Bug patterns**
  - E. Allen **[Bug patterns in Java]**
  - E. Farchi *et al.* **[PODC' 03]**
  - W. Pugh *et al.* **[OOPSLA' 04][PODC' 04] [PASTE' 07]**
- **Bug finding techniques**
  - *Partial verification* **R. Jhala *et al.* [POPL' 02]**
  - *Dynamic slicing* **R. Gupta *et al.* [ICSE' 03]**
  - *Formal proof* **B. Cook *et al.* [PLDI' 06]**

Conclusion

- XFindBugs supported a catalog of 17 bug patterns.
- XFindBugs can scale well and report a lot of warnings in real-world software systems.
- **Future work**
  - Identify more bug patterns in AspectJ
  - Refine our bug detectors in XFindBugs