Residual Investigation: Predictive and Precise Bug Detection

Kaituo Li
U. Massachusetts, Amherst

Christoph Reichenbach
U. Massachusetts, Amherst

Christoph Csallner
U. Texas Arlington

Yannis Smaragdakis
U. Athens & U. Massachusetts, Amherst

Philosophy

- **You can solve all programming problems, if you change what the program does**
  - **results are not “wrong”, just “different”**
- Ok, not really what this paper is about ☺

Static Analysis vs. Testing for Bug Detection

Static Analysis	Testing
- False Positives • impossible paths/values • overgeneralization	+ No False Positives • realizable paths
+ Fewer False Negatives • covers more paths • covers more values	- False Negatives • most bugs missed • cannot generalize

Li, Reichenbach, Csallner, Smaragdakis, "Residual Investigation: Predictive and Precise Bug Detection"

Dynamic Analysis in the Middle?

Static Analysis	Dynamic Analysis	Testing
- False Positives • impossible paths/values • overgeneralization	?	+ No False Positives • realizable paths
+ Fewer False Negatives • covers more paths • covers more values	?	- False Negatives • most bugs missed • cannot generalize

Li, Reichenbach, Csallner, Smaragdakis, "Residual Investigation: Predictive and Precise Bug Detection"

Dynamic Analysis

- Often a synonym of testing
- **Good dynamic analyses should be more than testing**
  - predicting error (not just observing)
  - fewer false positives than static analysis
- E.g., *Eraser* for race detection
  - warns of inconsistent lock use: strong hint that race exists
- **Goal: "generalize with confidence"—predictive and precise (*PaP*) dynamic analysis**

Li, Reichenbach, Csallner, Smaragdakis, "Residual Investigation: Predictive and Precise Bug Detection"

Dynamic Analysis in the Middle?

Static Analysis	*PaP* Dynamic Analysis	Testing
- False Positives • impossible paths/values • overgeneralization	Few False Positives	+ No False Positives • realizable paths
+ Fewer False Negatives • covers more paths • covers more values	Fewer False Negatives than Testing	- False Negatives • most bugs missed • cannot generalize

Li, Reichenbach, Csallner, Smaragdakis, "Residual Investigation: Predictive and Precise Bug Detection"

“PaP Dynamic analysis sounds great! Get me a half dozen!”

- Problem: how to design predictive and precise dynamic analyses
- Few *PaP* dynamic analyses in literature
- No general recipe
- This paper: **informal recipe** for *PaP* dynamic analyses

This Work: Residual Investigation

- Recipe:
  1. take a **static analysis**
  2. examine its false positives: what is the common **objection** to the static analysis?
  3. design **dynamic test** to disprove objection

Important: residual investigation may be exercising completely different program paths/data than the bug it predicts

- This dynamic test is a "***residual investigation***" for the static analysis
  - "partner of static analysis at run-time"
  - cf. existing test suite
- Always same 3 parts in recipe:
  1) static analysis; 2) objection; 3) dynamic test

Example Residual Investigation

- 1) *Static analysis*: find program classes that override “equals” but not “hashCode”
  - common Java guideline violation
  - detected by FindBugs tool
- 2) *Objection*: “but I never use such objects in a hash table”
- 3) *Dynamic test*: execute program, see if such objects ever have “hashCode” called

Example In More Detail

- Overriding “equals” but not “hashCode” can be serious bug
  - lose object identity, two copies of same object in structure
- Testing is ineffective
  - very hard to reproduce bug
- Usual static warning is a false positive
  Many classes override “equals” but not “hashCode”
  - org.jboss.deployment.dependency.ContainerDependencyMetaData
  - org.jboss.management.mejb.SearchClientNotificationListener
  - org.apache.jasper.compiler.Mark
  - …

A PaP Dynamic Analysis:

- *predictive (warns of error although an existing test case runs fine)*
- *precise (high error confidence)*

Another Residual Investigation

- 1) *Static analysis*: return value of "read" call ignored
  - bug: "read" may not return the amount of data expected
- 2) *Objection*: "for this object, 'read' always returns the bytes I request"
    - org.eclipse.equinox.internal.p2.swt.tools.IconExe$LEDataInputStream
- 3) *Dynamic test*: execute program, see if "read" *ever* returns fewer bytes on *any* object of suspect type
  - *predictive*: not just on calls that ignore return value of "read"!

A PaP Dynamic Analysis:

- *predictive (warns of error although the existing test case runs fine)*
- *precise (high error confidence)*

Yet Another Residual Investigation

- 1) *Static analysis*: find possible races in a program
  - static race detection is a problem with well-known false positives
- 2) *Objection*: "sure, this variable is not consistently protected, but it's thread-local!"
- 3) *Dynamic test*: execute program, see if variable is ever accessed by a second thread
  - *predictive*: not watching for race at all

Stephen Freund came up with this in under a minute

Li, Reichenbach, Csallner, Smaragdakis, "Residual Investigation: Predictive and Precise Bug Detection"

Our Paper

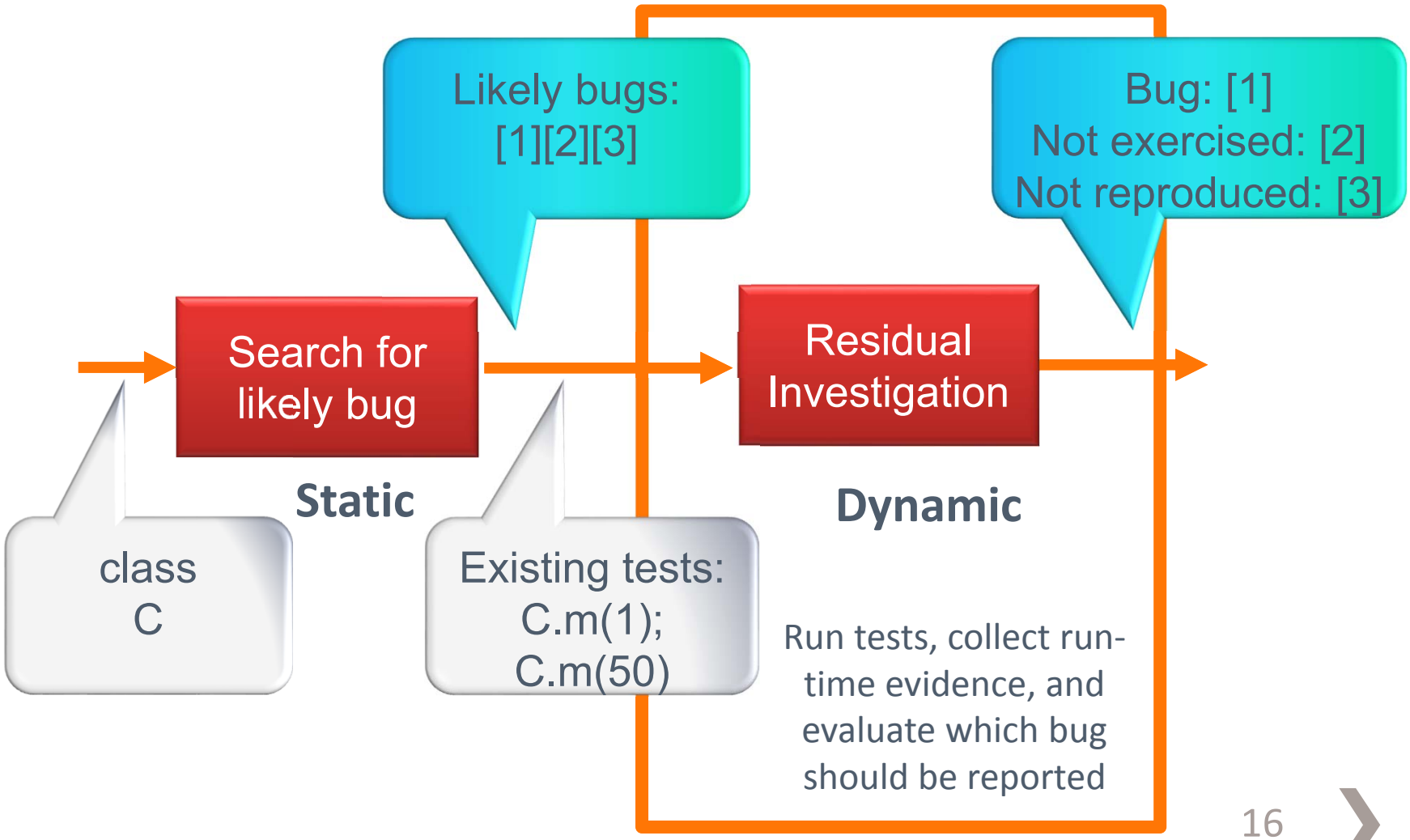
- Recipe for Residual Investigation: design a dynamic analysis to accompany a static one
  - confirm reports, or downgrade them
- Applied recipe repeatedly to show feasibility
  - on 7 static analyses from FindBugs
- Implemented dynamic analyses using bytecode rewriting and AspectJ
- Result: *RFBI* tool (*Residual FindBugs Investigator*)
- Evaluation on several large projects

Usage Overview

Important Usage Note

- Residual Investigation ***does not compete*** with static analysis, it ***complements*** it
- Static analysis is a prerequisite
- Static analysis reports are always available
- Residual investigation only prioritizes them
- Three outcomes:
  - high alert / *bug*: suspicious, based on dynamic analysis
  - medium alert / *not exercised*: dynamic analysis failed to confirm, due to lack of exercising
  - low alert / *not reproduced*: dynamic analysis failed to confirm, but not due to lack of exercising

Usage Overview



Li, Reichenbach, Csallner, Smaragdakis, "Residual Investigation: Predictive and Precise Bug Detection"

Implementation

The RFBI Tool

Example Implementations (1)

- Residual Investigation for “class overrides ‘equals’ but not ‘hashCode’”
- *Dynamic test*: execute program, see if such objects ever have “hashCode” called
- Implementation: add our own “hashCode”
  - using ASM (bytecode transform lib):

```
class org.apache.tomcat.util.buf { …
   @Override
   public int hashCode() {
        registerHashCodeObservedOn(
                                this.getClass());
        return super.hashCode();
   }
}
```


Li, Reichenbach, Csallner, Smaragdakis, "Residual Investigation: Predictive and Precise Bug Detection"

Example Implementations (2)

- Residual Investigation for “return value of ‘read’ not checked”
- *Dynamic test*: execute program, see if “read” *ever* returns fewer bytes on *any* object of suspect type
- Implementation:
  - AspectJ Advice to instrument read calls and register them per-type

```
after(byte[] b, int off, int len)
returning(int value):readcalljoinpoint(b,off,len)
{
  if(value == len)
   registerReadEqual(thisJoinPointStaticPart);
  else if(value < len)
   registerReadFewer(thisJoinPointStaticPart);
}
```


Li, Reichenbach, Csallner, Smaragdakis, "Residual Investigation: Predictive and Precise Bug Detection"

Residual Investigation Catalog

Analyses in RFBI

Other 5 Analyses

Bug Pattern	Run-time evidence that reinforce static warnings	Implementation Tool
Clone Method Does Not Call super.clone()	A subclass's clone can be shown dynamically to never reach super.clone()	Source generation + AspectJ
Dropped Exception	Any method in the call graph of the try block ever throws the dropped exception anywhere	First pass: ASM Second pass: AspectJ
Equals Method May Not Be Symmetric	Two equals methods ever disagree	AspectJ
Non-Short-Circuit Boolean Operator	Actual side-effects on the right-hand side of a non-short-circuiting boolean operator	ASM+AspectJ
Bad Covariant Definition of Equals	Object.equals(Object) is called on suspect class	ASM run-time/ JDK class build-time instrumentation

Li, Reichenbach, Csallner, Smaragdakis, "Residual Investigation: Predictive and Precise Bug Detection"

Evaluation

Sample of Results

Evaluating Residual Investigation

- 7 large open source systems
  - JBoss
  - BCEL
  - NetBeans
  - Tomcat
  - JRuby
  - Apache Commons Collection
  - Groovy

Evaluating Residual Investigation

- Test suites run take anywhere from 23sec to 3 hours
  - 4-core 2.4GHz Intel i5 with 6 GB RAM
- Runtime slowdown
  - 2-3 factor
  - except for Dropped Exception, which goes up to 6
    - execute test suites twice
    - watch a large number of calls

Evaluating Residual Investigation

- FindBugs reports 436 bugs
- For 393, the test suite does not exercise conditions relevant to the bug at all
  - few true bugs, based on our sampling and inspection
- RFBI does very well in the other 43
  - Summary: ≥ 77% precision, ≥ 96% recall

Dynamic Reports	Bug	Non-bug	undetermined
31 reinforced	24	6	1
12 rejected	0	11	1
43 total	24	17	2

Li, Reichenbach, Csallner, Smaragdakis, "Residual Investigation: Predictive and Precise Bug Detection"

Threats to Validity

- Choice of subject applications
- Choice of FindBugs patterns
- Choice of static analysis system

Conclusions

(See paper for related work, technical insights and more)

Conclusion

- Residual Investigation =
  way to produce predictive and precise (PaP) dynamic analyses
  - fewer false positives than static analysis
  - more bugs caught than testing
- Using a standard recipe on a static analysis pattern
- Applied to 7 FindBugs analyses, evaluated on large systems

Questions?