How Many of All Bugs Do We Find?
A Study of Static Bug Detectors

Andrew Habib, Michael Pradel
TU Darmstadt, Germany
software-lab.org
Static Bug Detection

Error Prone

Google

facebook

Infer

SpotBugs
Static Bug Detection

- General framework
- Scalable static analysis
- Set of checkers for specific bug patterns
How Many Bugs Do They Find?
How Many Bugs Do They Find?

Given a representative set of real-world bugs, how many of them do static bug detectors find?
How Many Bugs Do They Find?

Given a representative set of real-world bugs, how many of them do static bug detectors find?

This talk:
Empirical study with 594 real-world Java bugs and 3 popular static checkers
Real-World Bugs

- **594 bugs from 15 popular Java projects**
  - Extended version of Defects4J data set

- **Why this set?**
  - Gathered independently
  - Used in other bug-related studies *
  - Contains real fixes by developers

* Just et al., 2014 (mutation testing); Shamshiri et al., 2015 (test generation); Pearson et al., 2017 (fault localization); Martinez et al., 2017 (program repair)
Defects4J: Files Involved in Bug

Number of bugs

Number of buggy files

- 501 buggy files
- 64 buggy files
- 12 buggy files
- 10 buggy files
- 4 buggy files
- 1 buggy file
- 1 buggy file
- 1 buggy file
- 1 buggy file

Total number of bugs: 501
Defects4J: Size of Bug Fix

Diff size between buggy and fixed versions (LoC)

Number of bugs

1-4: 296
5-9: 128
10-14: 54
15-19: 29
20-24: 29
25-49: 44
50-74: 6
75-99: 6
100-199: 1
200-1999: 1
Previous Approach

How to determine which bugs are found?

[Thung et al., 2012]

- Get diff between buggy and fixed code
- Run tool on code with buggy lines
- If warning on buggy line: Bug found
- Result: 50% – 95% of all bugs found

Limitation:
- No check that warning points to bug
- One tool flags up to 57% of all lines
Previous Approach

How to determine which bugs are found?

[Thung et al., 2012]

- Get diff between buggy and fixed code
- Run tool on code with buggy lines
- If warning on buggy line: Bug found
- Result: 50% – 95% of all bugs found

**Limitation:**
- No check that warning points to bug
- One tool flags up to 57% of all lines
Methodology: Overview

<table>
<thead>
<tr>
<th>Bugs + fixes</th>
<th>Bug detectors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Methodology: Overview

Bugs + fixes

Bug detectors

Automated filtering of warnings
Methodology: Overview

Bugs + fixes

Bug detectors

Automated filtering of warnings

- Diff-based
- Fixed warnings-based
- Combined
Methodology: Overview

- Bugs + fixes
- Bug detectors

Automated filtering of warnings
- Diff-based
- Fixed warnings-based
- Combined

Candidates for detected bugs

Manual inspection of candidates

Detected bugs
Methodology: Diff-based

Bugs + fixes → Automated filtering of warnings → Candidates for detected bugs → Manual inspection of candidates → Detected bugs

Bug detectors
Methodology: Diff-based

1) Identify *lines changed* to fix bug
2) Intersect with *lines with warning*
Methodology: Diff-based

1) Identify **lines changed** to fix bug
2) Intersect with **lines with warning**

Buggy file:

固定文件：

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fixed file:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Methodology: Diff-based

1) Identify **lines changed** to fix bug
2) Intersect with **lines with warning**

Buggy file:

```
________
```

Fixed file:

```
________
```

Modified line
Methodology: Diff-based

1) Identify **lines changed** to fix bug
2) Intersect with **lines with warning**

Buggy file:

Fixed file:

Modified line

Removed line
Methodology: Diff-based

1) Identify **lines changed** to fix bug
2) Intersect with **lines with warning**

Buggy file:

- [ ]
- [ ]
- [ ]
- [ ]
- [ ]

Fixed file:

- [ ]
- [ ]
- [ ]
- [ ]
- [ ]

- [ ]
- [ ]
- [ ]
- [ ]

- [ ]
- [ ]
- [ ]
- [ ]

- [ ]
- [ ]
- [ ]
- [ ]

Methodology: Diff-based

1) Identify **lines changed** to fix bug
2) Intersect with **lines with warning**

Buggy file:

Fixed file:

---

Warnings by bug detector
Methodology: Diff-based

1) Identify lines changed to fix bug
2) Intersect with lines with warning

Buggy file:

Warnings by bug detector

Fixed file:

Candidate for detected bug
Example:

```java
public Dfp multiply(final int x) {
    return multiplyFast(x);
}
```

```
public Dfp multiply(final int x) {
    if (x >= 0 && x < RADIX) {
        return multiplyFast(x);
    } else {
        return multiply(newInstance(x));
    }
}
```

Bug fix

```java
public Dfp multiply(final int x) {
    if (x >= 0 && x < RADIX) {
        return multiplyFast(x);
    } else {
        return multiply(newInstance(x));
    }
}
```
public Dfp multiply(final int x) {
    return multiplyFast(x);
}

// Bug fix

public Dfp multiply(final int x) {
    if (x >= 0 && x < RADIX) {
        return multiplyFast(x);
    } else {
        return multiply(newInstance(x));
    }
}

Warning:
Missing @Override
Example:

```java
public Dfp multiply(final int x) {
    return multiplyFast(x);
}
```

```
public Dfp multiply(final int x) {
    if (x >= 0 && x < RADIX) {
        return multiplyFast(x);
    } else {
        return multiply(newInstance(x));
    }
}
```

Warning: Missing @Override

Candidate for detected bug
Method.: Fixed Warnings-based

Bugs + fixes → Bug detectors

Automated filtering of warnings

Fixed warnings-based

Candidates for detected bugs

Manual inspection of candidates

 Detected bugs
Method.: Fixed Warnings-based

1) Compare *warnings before and after fix*
2) Warning that *disappears* was for bug
Method.: Fixed Warnings-based

1) Compare **warnings before and after fix**
2) Warning that **disappears** was for bug

<table>
<thead>
<tr>
<th>Buggy file:</th>
</tr>
</thead>
<tbody>
<tr>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fixed file:</th>
</tr>
</thead>
<tbody>
<tr>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
</tr>
</tbody>
</table>
Method.: Fixed Warnings-based

1) Compare *warnings before and after fix*
2) Warning that *disappears* was for bug

Buggy file:

```
_________
_________
_________
_________
```

Fixed file:

```
_________
_________
_________
_________
```
Method.: Fixed Warnings-based

1) Compare **warnings before and after fix**
2) Warning that **disappears** was for bug

Buggy file: __________

__________

__________

__________

__________

Fixed file: __________

__________

__________

__________

__________

**Candidate for detected bug**

Warnings by bug detector
Example

```java
public Week(Date time, TimeZone zone) {
    this(time,
         RegularTimePeriod.DEFAULT_TIME_ZONE,
         Locale.getDefault());
}
```

**Bug fix**

```java
public Week(Date time, TimeZone zone) {
    this(time,
         zone,
         Locale.getDefault());
}
```
Example

class

class

public Week(Date time, TimeZone zone) {
    this(time,
        RegularTimePeriod.DEFAULT_TIME_ZONE,
        Locale.getDefault());
}

public Week(Date time, TimeZone zone) {
    this(time,
        zone,
        Locale.getDefault());
}

Warning:
Chaining constructor ignores argument

Candidate for detected bug
Methodology: Combined

- Bugs + fixes
- Bug detectors

Automated filtering of warnings

- Diff-based
- Fixed warnings-based
- = Combined

Candidates for detected bugs

Manual inspection of candidates

Detected bugs
Results
### Warnings to Inspect

#### All warnings

<table>
<thead>
<tr>
<th>Tool</th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Prone</td>
<td>0</td>
<td>148</td>
<td>7.58</td>
<td>4,402</td>
</tr>
<tr>
<td>Infer</td>
<td>0</td>
<td>36</td>
<td>0.33</td>
<td>198</td>
</tr>
<tr>
<td>SpotBugs</td>
<td>0</td>
<td>47</td>
<td>1.1</td>
<td>647</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td>5,247</td>
</tr>
</tbody>
</table>

#### Per bug

This table provides a summary of warnings per bug for different tools.
## Warnings to Inspect

<table>
<thead>
<tr>
<th>Tool</th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
<th>Total</th>
<th>Candidates only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Prone</td>
<td>0</td>
<td>148</td>
<td>7.58</td>
<td>4,402</td>
<td>53</td>
</tr>
<tr>
<td>Infer</td>
<td>0</td>
<td>36</td>
<td>0.33</td>
<td>198</td>
<td>32</td>
</tr>
<tr>
<td>SpotBugs</td>
<td>0</td>
<td>47</td>
<td>1.1</td>
<td>647</td>
<td>68</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td>5,247</td>
<td>153</td>
</tr>
</tbody>
</table>
### Warnings to Inspect

#### All warnings

<table>
<thead>
<tr>
<th>Tool</th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
<th>Total</th>
<th>Candidates only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Prone</td>
<td>0</td>
<td>148</td>
<td>7.58</td>
<td>4,402</td>
<td>53</td>
</tr>
<tr>
<td>Infer</td>
<td>0</td>
<td>36</td>
<td>0.33</td>
<td>198</td>
<td>32</td>
</tr>
<tr>
<td>SpotBugs</td>
<td>0</td>
<td>47</td>
<td>1.1</td>
<td>647</td>
<td>68</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td>5,247</td>
<td>153</td>
</tr>
</tbody>
</table>

97% of all warnings are removed by the automated filtering step
Manual Inspection

Distinguish coincidental matches from actually detected bugs

Candidate = (bug, warning)
public Dfp multiply(final int x) {
    return multiplyFast(x);
}

public Dfp multiply(final int x) {
    if (x >= 0 && x < RADIX) {
        return multiplyFast(x);
    } else {
        return multiply(newInstance(x));
    }
}

Candidate for detected bug

Warning: Missing @Override

Bug fix
public Dfp multiply(final int x) {
    return multiplyFast(x);
}

public Dfp multiply(final int x) {
    if (x >= 0 && x < RADIX) {
        return multiplyFast(x);
    } else {
        return multiply(newInstance(x));
    }
}

Warning: Missing @Override

Mismatch
public Week(Date time, TimeZone zone) {
    this(time,
        RegularTimePeriod.DEFAULT_TIME_ZONE,
        Locale.getDefault());
}

<table>
<thead>
<tr>
<th>Bug fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warning: Chaining constructor ignores argument</td>
</tr>
</tbody>
</table>

Candidate for detected bug
**Manual Inspection: Example (2)**

```java
public Week(Date time, TimeZone zone) {
    this(time,
        RegularTimePeriod.DEFAULT_TIME_ZONE,
        Locale.getDefault());
}
```

**Warning:** Chaining constructor ignores argument

**Bug fix**

```java
public Week(Date time, TimeZone zone) {
    this(time,
        zone,
        Locale.getDefault());
}
```

**Full match**
Most Bugs are Missed

Three tools together:
Detect 27 of 594 bugs (less than 5%)
Why are Most Bugs Missed?

Manual inspection of random sample of 20 missed bugs:
Why are Most Bugs Missed?

Manual inspection of random sample of 20 missed bugs:
14 are domain-specific

- Unrelated to any of the supported bug patterns
- Application-specific algorithms
- Forgot to handle special case
- Difficult to decide whether behavior is intended
Manual inspection of random sample of 20 missed bugs:

6 are near misses

- Root cause is targeted by bug detector, but current implementation misses the bug
- Detector targets similar, but not the same, problem
Conclusion

- **Novel methodology** to measure how many of a set of bugs are detected
- Popular static **bug detectors miss most bugs**
- Main reason: **Domain-specific bugs** vs. generic bug patterns
- **Huge potential** for future work on bug detection
Implications for Future Work

Huge potential for:

- Bug detectors that catch domain-specific bugs
- More sophisticated yet precise static analyses
- Generalizations of existing bug checkers
- Bug finding techniques other than static analysis, e.g., test generation