Evaluating a Lightweight Defect Localization Tool

Valentin Dallmeier Christian Lindig Andreas Zeller
Saarland University – Germany
JUnit Tests in Eclipse

Passing Run (1+)

Failing Run (1)
Ample Plugin

AspectJ Bug #30168

Suspect Classes

Bug fixed here

2,929 classes

Faults correlate with differences in traces between a correct and a faulty run.
Tracing Objects

myObj  anInStream  anOutStream  aLogger

InStream.read  Logger.getName
OutStream.write  Logger.log
OutStream.flush

InStream.read  Logger.getName  OutStream.write  Logger.log  OutStream.flush
Call-Sequence Sets

Call-Sequence Set – sequences of length k
Benefits: simple, compact, set semantics
Aggregating Traces

class

object

object

Sequence Set

Sequence Set

Trace
Comparing Program Runs

class-by-class

passing run

failing run

MyClass

MyClass

common sequence (weight 0)

new sequence (weight 1)

missing sequence (weight 1)

average sequence weight for ranking classes
Search Length

- search length: classes in front of faulty class in ranking
- smaller is better
- evaluated for programs with one known bug

search length: 9
Evaluation Subjects

NanoXML - Java XML Parser (Do et al.)

- 4 Versions, 16–23 classes, 4.3–7.6 kLOC
- 33 known bugs, 214 test cases
- 386 rankings, each for:
  - 1 bug, 1 failing run, 1+ passing runs

AspectJ - Java Compiler (v1.1.1)

- 979 classes, 112 kLOC
- 5 rankings for real bugs from bug db
# Results

<table>
<thead>
<tr>
<th>Subject</th>
<th>Rand</th>
<th>Guess</th>
<th>search length</th>
<th>window size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>NanoXML</td>
<td>4.78</td>
<td></td>
<td>2.53</td>
<td>2.31</td>
</tr>
<tr>
<td>AspectJ</td>
<td>209</td>
<td></td>
<td>32.4</td>
<td>31.8</td>
</tr>
</tbody>
</table>

Sample beats random guessing (no surprise)
Sequences perform better than coverage
Inspecting the 3 top-ranked classes, a programmer finds over 50% of all bugs in NanoXML.
Conclusions

- Ample works (NanoXML) and scales (AspectJ)
- Sequence sets facilitate aggregation and comparison of runs
- Ample is first approach to leverage objects
- Search length is measure for performance
- Sequences outperform coverage analysis

1 Failing, 3 Passing Runs

passing runs

1/3

1 2/3

failing run

1/3

0 2/3

1 1/3

2/3
Runtime Overhead

- Measured for SPEC JVM 98 Benchmarks
- Memory: factor 1.1 – 22.7 (typical: $\leq 2$)
- Time: factor 1.2 – $\geq 100$ (varies widely)
- Comparable to coverage analysis (JCoverage)
- Found low overhead for AspectJ