Software Engineering: An Emerging Discipline

Rui Abreu

@rmaranhao
Submission deadline: 1 Feb. 2018

Publication: Sept./Oct. 2018

2018 will mark the 50th anniversary of the first NATO Software Engineering Conference, where the term “software engineering” was coined. IEEE Software will commemorate this occasion with a theme issue. By combining historical and insightful perspectives on various software engineering disciplines and trends, we hope to help guide the field’s further development.

Formats include, but aren’t limited to...
And the 3 most recurrent terms in the titles of #ICSE2018 submissions are... [drum roll please] ...Software, Analysis, and Code! Surprised?
Doing research in SE

• What do people expect?
  • Experiments with real, large software systems
  • Generality of the results
    • Or a strong threats to the validity section
  • Statistical analysis of the results
  • Reproducibility of the results

Or your papers won’t get accepted!
Rui Abreu aka Rui Maranhão

@rmaranhao
Jeff Bezos

• “[…] In the end, we are the choices we made.”
Associate Professor in Software Engineering, IST
Achievements

• I’ve published in top-tier SE/AI venues
  • ICSE, ASE, ICST, ISSTA, AAAI, IJCAI, TSE, AIJ, ...
  • IEEE Senior Member, since 2016
  • 98 Highly Influential Citations, cf. SemanticScholar.org

• 5 Best Paper Awards

• “A Survey on Software Fault Localization” was selected to be featured on ACM’s 21st Annual Best of Computing (notice received last week!)

• Entrepreneurial spirit
  • Crowbar (www.crowbar.io): raised $500k from VCs. Long story.
Testing and Debugging Software-Intensive Systems… Automatically!

Rui Abreu

@rmaranhao
Why do I find this topic exciting?
The birth of debugging: your guess?
Software Errors mentioned in Ada Byron’s notes on Charles Bababbage’s analytical engine
First actual bug and actual debugging: Admiral Grace Hopper’s associates working on Mark II Computer at Harvard University
Stallman’s GDB

**Input:** faulty program and 1 failed test case
A Survey on Software Fault Localization

Abstract:
Software fault localization, the act of identifying the locations of faults in a program, is widely recognized to be one of the most tedious, time consuming, and expensive - yet equally critical - activities in program debugging. Due to the increasing scale and complexity of software today, manually locating faults when failures occur is rapidly becoming infeasible, and consequently, there is a strong demand for techniques that can guide software developers to the locations of faults in a program with minimal human intervention. This demand in turn has fueled the proposal and development of a broad spectrum of fault localization techniques, each of which aims to streamline the fault localization process and make it more effective by attacking the problem in a unique way. In this article, we catalog and provide a comprehensive overview of such techniques and discuss key issues and concerns that are pertinent to software fault localization as a whole.

Published in: IEEE Transactions on Software Engineering (Volume: 42, Issue: 8, Aug. 1 2016)
Focus of what I work on

- Techniques that take into account spectra
  - aka abstraction of program traces
  - Spectrum-based Fault Localization (SFL)
    - Statistical vs. reasoning
- Lightweight, scalable
SFL: Principle (1)

Test suite:
- t1
- t2
- t3
- t4
- t5

Integrates well with testing
SFL: Principle (2)

Test suite

- t2
- t3
- t4
- t5

Status

- t1

Integrates well with testing
SFL: Principle (3)

Integrates well with testing

Test suite

Status

t1 ✓
t2 ✓

Not touched
Touched, pass
Touched, fail
SFL: Principle (4)

Integrates well with testing

Test suite

t4
t5

<table>
<thead>
<tr>
<th>Status</th>
<th>t1</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>t3</td>
<td>✗</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Not touched</th>
<th>Touched, pass</th>
<th>Touched, fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11 12</td>
<td>2 1 1 0 1 2 2 2 2 1 1 1</td>
<td>1 0 0 1 1 1 1 0 1 0 0 0</td>
</tr>
</tbody>
</table>
SFL: Principle (5)

Test suite

t5

Status

t1 ✓
t2 ✓
t3 ✗
t4 ✓

Integrates well with testing
SFL: Principle (6)

Test suite

Status

- t1 ✓
- t2 ✓
- t3 ✗
- t4 ✓
- t5 ✗

Not touched

Touched, pass

Touched, fail
SFL: Principle (7)

Components are **ranked** according to the likelihood of causing detected errors.

Status
- t1  ✓
- t2  ✓
- t3  ✗
- t4  ✓
- t5  ✗
```java
class Triangle {
    static int type(int a, int b, int c) {
        int type = SCALENE;
        if ( (a == b) && (b == c) )
            type = EQUILATERAL;
        else if ( (a*a) == ((b*b) + (c*c)) )
            type = RIGHT;
        else if ( (a == b) || (b == a) ) /* FAULT */
            type = ISOSCELES;
        return type; }

    static double area(int a, int b, int c) {
        double s = (a+b+c)/2.0;
        return Math.sqrt(s*(s-a)*(s-b)*(s-c)); }
}
```

<table>
<thead>
<tr>
<th></th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
<th>Suspiciousness</th>
</tr>
</thead>
<tbody>
<tr>
<td>int type = SCALENE;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.09998</td>
</tr>
<tr>
<td>if ( (a == b) &amp;&amp; (b == c) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.09998</td>
</tr>
<tr>
<td>type = EQUILATERAL;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.10001</td>
</tr>
<tr>
<td>else if ( (a<em>a) == ((b</em>b) + (c*c)) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.09999</td>
</tr>
<tr>
<td>type = RIGHT;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.10001</td>
</tr>
<tr>
<td>else if ( (a == b)</td>
<td></td>
<td>(b == a) ) /* FAULT */</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>type = ISOSCELES;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.10001</td>
</tr>
<tr>
<td>return type;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.09998</td>
</tr>
<tr>
<td>static double area(int a, int b, int c) {</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.10000</td>
</tr>
<tr>
<td>double s = (a+b+c)/2.0;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.10000</td>
</tr>
<tr>
<td>return Math.sqrt(s*(s-a)<em>(s-b)</em>(s-c)); } ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.10000</td>
</tr>
</tbody>
</table>
```
Suspiciousness score

• Each component (row) is **ranked** according to their **similarity** to the **error vector**

• Many similarity coefficients exist.

\[ \text{Ochiai}(a, b) = \cos(\theta) \]

• **Ochiai** similarity is equivalent to the cosine of the angle between two vectors in a n-dimensional space

<table>
<thead>
<tr>
<th>Rank Position</th>
<th>Suspicious Statement</th>
<th>Line number</th>
<th>Suspiciousness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1º</td>
<td>type = EQUILATERAL;</td>
<td>3</td>
<td>0.10001</td>
</tr>
<tr>
<td>2º</td>
<td>type = RIGHT;</td>
<td>5</td>
<td>0.10001</td>
</tr>
<tr>
<td>3º</td>
<td>type = ISOSCELES;</td>
<td>7</td>
<td>0.10001</td>
</tr>
<tr>
<td>4º</td>
<td>else if ( (a == b)</td>
<td></td>
<td>(b == a) ) /* FAULT */</td>
</tr>
<tr>
<td>5º</td>
<td>double s = (a+b+c)/2.0;</td>
<td>9</td>
<td>0.10000</td>
</tr>
<tr>
<td>6º</td>
<td>return Math.sqrt(s*(s-a)<em>(s-b)</em>(s-c));</td>
<td>10</td>
<td>0.10000</td>
</tr>
<tr>
<td>7º</td>
<td>else if ( (a<em>a) == ((b</em>b) + (c*c)) )</td>
<td>4</td>
<td>0.09999</td>
</tr>
<tr>
<td>8º</td>
<td>int type = SCALENE;</td>
<td>1</td>
<td>0.09998</td>
</tr>
<tr>
<td>9º</td>
<td>if ( (a == b) &amp;&amp; (b == c) )</td>
<td>2</td>
<td>0.09998</td>
</tr>
<tr>
<td>10º</td>
<td>return type; }</td>
<td>8</td>
<td>0.09998</td>
</tr>
</tbody>
</table>

\[ C_d = 4 \]

R. Abreu, P. Zoeteweij, and A. J. van Gemund, “Spectrum-Based Multiple Fault Localization”, ASE ’09
Spectrum-based reasoning

1. Generate sets of components that *explain* observed erroneous behavior
   - Equivalent to compute minimal hitting set (Staccato/MHS2**)
   - Given failed executions

2. Rank candidates according to their probability of being the true fault explanation ➤ Baye’s rule
   - Given both passed and failed executions

---

R. Abreu, P. Zoeteweij, and A. J. van Gemund, “Spectrum-Based Multiple Fault Localization”, ASE ’09
**https://github.com/npcardoso/MHS2  (citable via https://zenodo.org/record/10037) ➤ contribute to the project; send pull requests; email us!
Diagnostic Performance

![Graph showing diagnostic performance with percentage of faulty versions on the y-axis and effort (% of program to be examined to find the fault) on the x-axis. There are two lines: one for the worst technique (grey) and one for the ideal technique (red). The worst technique remains at 100% until the effort reaches 80%, while the ideal technique shows a sharp increase from 0% to 100% at the same effort level.]
How good are we?

• Best Performing techniques still require to inspect 10% of the code…

• 100 LOC ➤ 10LOC

• 10,000 LOC ➤ 1,000LOC

• 1,000,000 LOC ➤ 10,000LOC
<table>
<thead>
<tr>
<th>Case</th>
<th>To Inspect</th>
<th>Out of / Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Problem</td>
<td>2 logical threads</td>
<td>315</td>
</tr>
<tr>
<td>Teletext Lock-Up</td>
<td>2 blocks</td>
<td>60K</td>
</tr>
<tr>
<td>NVM corrupt</td>
<td>96 blocks, 10 files</td>
<td>150K, 1.8K</td>
</tr>
<tr>
<td>Scrolling Bug</td>
<td>5 blocks</td>
<td>150K</td>
</tr>
<tr>
<td>Invisible Pages</td>
<td>12 blocks</td>
<td>150K</td>
</tr>
<tr>
<td>Tuner Problem</td>
<td>2 files</td>
<td>1.8K</td>
</tr>
<tr>
<td>Zapping Crash</td>
<td>1 run (15 mins)</td>
<td>1 day (develop)</td>
</tr>
<tr>
<td>Wrong Audio</td>
<td>1 run (15 mins)</td>
<td>½ day (expert)</td>
</tr>
</tbody>
</table>
Humm....

- Are we properly quantifying diagnostic accuracy?
- Comparing techniques based on the rankings
- Assuming perfect bug understanding
- Are we showing providing an ecosystem offering this techniques?
Crowbar

http://www.crowbar.io
Visualizations
User Study: Setup

- 40 participants
- Intention: GZoltar vs. IDE’s features
- Program: Xtream
  - 17,389 LOC
  - 306 classes and 22 packages
  - 1418 unit test cases
  - Injected 1 logical fault

User Study: Results

**RQ1**: Do the proposed visualizations efficiently aid the user to quickly find a fault?
RQ2: Is Crowbar a usable toolset?
Importance of Testing

“A confounding factor for the usefulness of SFL is the dependency on the quality of the existing test suite”
matrices), whereas high values mean that test cases tend to
exercise small parts of the program (sparse work [major factors to determine uncertainty in the ranking. Previous
test cases. The variety and number of test cases are two
reasons in terms of multiple faults. To illustrate this approach,
information about the framework and underlying technique
applied. In the Triangle example we have 10 components
cannot distinguish which components are most probably at
faulty can explain the fault better. The minimum value (i.e.,
the candidates in the example with probability 0.5 of being
approximately ideal) for

\[
\tilde{\rho}^T_{\{s\}} \begin{bmatrix} \theta \end{bmatrix} = \begin{bmatrix} T + \{t_7, t_8\} \\
T + \{t_7, t_8, t_9\} \end{bmatrix}
\]

Test case outcome (pass = ✔, fail = ✗)

<table>
<thead>
<tr>
<th>(\tilde{\rho})</th>
<th>0.400</th>
<th>0.457</th>
<th>0.475</th>
<th>0.500</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mathcal{H})</td>
<td>3.322</td>
<td>2.651</td>
<td>2.445</td>
<td>2.437</td>
</tr>
<tr>
<td>(C_d)</td>
<td>4.000</td>
<td>2.500</td>
<td>1.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

\(C_d = 0.0\)
Out of the oven
• Available as an Eclipse plug-in
• A Visual Studio plugin will be released soon
• Also available as a library
• Instrumentation and diagnosis
• Testing features are yet to be deployed
• Only test suite minimization available

http://www.gzoltar.com
Let’s use it

- Open Eclipse
- Install Crowbar
  - Help ➤ Install New Software
    - http://crowbar.io/plugin/eclipse
  - Window ➤ Other… ➤ Crowbar Views ➤ Diagnostic Reports
- Import (as maven project) buggy yodaTime
  - http://crowbar.io/plugin/tarot/buggy_jodatime.zip
- Find the bug!
Opportunities and Challenges

• Integration with software repository mining
• Use fitness function in test suite prioritization
• Generation: How to solve the oracle problem?
  • Human in the loop
  • Leverage program invariants
• Explore idiosyncrasies of mobile devices
Questions?

Importance of Testing

“A confounding factor for the usefulness of SFL is the dependency on the quality of the existing test suite.”

Let’s use it!

- Available as an Eclipse plug-in
- A Visual Studio plugin will be released soon
- Also available as a library
- Instrumentation and diagnosis
- Testing features are yet to be deployed
- Only test suite minimization available

- Open Eclipse
- Install Crowbar
- Help ➤ Install New Software
- http://crowbar.sourceforge.net/
- Window ➤ Other... ➤ Crowbar Views ➤ Diagnostic Reports
- Import (as Maven project) buggy yodaTime
- http://crowbar.sourceforge.net/buzzy_yodatime.zip
- Find the bug!

SFL: Principle (1)

Test suite
11 12 13 15
11 12 13 15

- Not touched
- Touched, pass
- Touched, fail

Hummm...

- Are we properly quantifying diagnostic accuracy?
- Comparing techniques based on the rankings
- Assuming perfect bug understanding
- Are we showing providing an ecosystem offering this techniques?