On the Accuracy of Spectrum-based Fault Localization

Rui Abreu
Peter Zoeteweij & Arjan van Gemund

‘All truths are easy to understand once they are discovered; the point is to discover them’

Galileo Galilei
Testing & Debugging

• Spectrum-based Fault Localization (SFL)
  – Automatic debugging technique
  – Based on execution profiles and error detection info
  – Improves the efficiency of the debugging stage
    • more bugs solved leads to more reliable systems

• BUT, TAIC PART is a testing conference
  • Testing (is there an error?) and SFL (what causes it?) are closely related
TRADER project

• Improve the user-perceived reliability of high-volume consumer electronics devices
• Partners:
  – Several Universities in NL, , and Philips {Research, TASS}
• Test case: TV platform from NXP
• Preliminary successful experiments triggered a ‘knowledge transfer’
Overview

• Background
  – Terminology
  – SFL

• Experiments
  – Benchmark and Metric
  – What is the effect of various external factors on the diagnostic accuracy?
    • Impact of the error detector
    • Impact of the number of runs

• Conclusions
Terminology

- **fault**: the cause of an error in the system
  - *(bug: array index un-initialized)*
- **error**: system state that may cause a failure
  - *(index out of bounds)*
- **failure**: delivered service ≠ correct service
  - *(segmentation fault)*

TU Delft

Embedded Systems Institute
Program Spectra

• Execution profiles
  – indicate, or count which parts of a software system are used in a particular test case

• Many different forms exist:
  – Spectra of program locations
  – Spectra of branches / paths
  – Spectra of data dependencies
  – Spectra of method call sub-sequences
Block hit spectra

\[ x_1 \quad x_2 \quad \ldots \quad x_i \quad \ldots \quad x_n \]

1: block \( i \) executed
0: block \( i \) not executed

Block:
- C statement (compound stmt)
- cases of a switch statement
SFL

1. Spectra for $m$ test cases

$m$ cases

$n$ blocks

<table>
<thead>
<tr>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>…</th>
<th>$x_{1n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>…</td>
<td>$x_{2n}$</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>$x_{m1}$</td>
<td>$x_{m2}$</td>
<td>…</td>
<td>$x_{mn}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$e_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_2$</td>
</tr>
<tr>
<td>…</td>
</tr>
<tr>
<td>$e_m$</td>
</tr>
</tbody>
</table>
## SFL

### 1. Spectra for $m$ test cases

<table>
<thead>
<tr>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>…</th>
<th>$x_{1n}$</th>
<th>$e_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>…</td>
<td>$x_{2n}$</td>
<td>$e_2$</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>$x_{m1}$</td>
<td>$x_{m2}$</td>
<td>…</td>
<td>$x_{mn}$</td>
<td>$e_m$</td>
</tr>
</tbody>
</table>

Row $i$: the blocks that are executed in case $i$
SFL

1. Spectra for $m$ test cases

<table>
<thead>
<tr>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>...</th>
<th>$x_{1n}$</th>
<th>$e_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>...</td>
<td>$x_{2n}$</td>
<td>$e_2$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$x_{m1}$</td>
<td>$x_{m2}$</td>
<td>...</td>
<td>$x_{mn}$</td>
<td>$e_m$</td>
</tr>
</tbody>
</table>

Column $j$: the test cases in which block $j$ was executed
## SFL

1. Spectra for \( m \) test cases
2. Error detection per test case

<table>
<thead>
<tr>
<th></th>
<th>( x_{11} )</th>
<th>( x_{12} )</th>
<th>( \ldots )</th>
<th>( x_{1n} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_{21} )</td>
<td>( x_{22} )</td>
<td>( \ldots )</td>
<td>( x_{2n} )</td>
<td></td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td></td>
</tr>
<tr>
<td>( x_{m1} )</td>
<td>( x_{m2} )</td>
<td>( \ldots )</td>
<td>( x_{mn} )</td>
<td></td>
</tr>
</tbody>
</table>

|   | \( e_1 \) | \( e_2 \) | \( \ldots \) | \( e_m \) |

\( e_i = 1 \) : error in the \( i \)-th test
\( e_i = 0 \) : no error in the \( i \)-th test
**SFL**

Compare every column vector with the error vector

<table>
<thead>
<tr>
<th>block $j$</th>
<th>error vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{11}$ $x_{12}$ ... $x_{1n}$</td>
<td>$e_1$</td>
</tr>
<tr>
<td>$x_{21}$ $x_{22}$ ... $x_{2n}$</td>
<td>$e_2$</td>
</tr>
<tr>
<td>... ... ... ...</td>
<td>...</td>
</tr>
<tr>
<td>$x_{m1}$ $x_{m2}$ ... $x_{mn}$</td>
<td>$e_m$</td>
</tr>
</tbody>
</table>

Similarity $s_j$
SFL

Jaccard similarity coefficient

\[ s_j = \frac{a_{11}}{a_{11} + a_{10} + a_{01}} \]
SFL

Jaccard similarity coefficient

$$s_j = \frac{a_{11}}{a_{11} + a_{10} + a_{01}}$$
SFL

Jaccard similarity coefficient

\[ s_j = \frac{2}{2 + a_{10} + a_{01}} \]

block \( j \)  
\[
\begin{array}{c}
1 \\
0 \\
1 \\
0 \\
1 \\
\end{array}
\]

error vector  
\[
\begin{array}{c}
0 \\
1 \\
1 \\
0 \\
1 \\
\end{array}
\]
SFL

Jaccard similarity coefficient

\[ s_j = \frac{2}{2 + 1 + a_{01}} \]

block \( j \)

error vector

\[
\begin{array}{c}
1 \\
0 \\
1 \\
0 \\
1 \\
\end{array}
\quad
\begin{array}{c}
0 \\
1 \\
1 \\
0 \\
1 \\
\end{array}
\]
SFL

Jaccard similarity coefficient

\[
s_j = \frac{2}{2 + 1 + 1}
\]
### SFL

For every block: similarity with the error “block”

<table>
<thead>
<tr>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>…</th>
<th>$x_{1n}$</th>
<th>$e_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>…</td>
<td>$x_{2n}$</td>
<td>$e_2$</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>$x_{m1}$</td>
<td>$x_{m2}$</td>
<td>…</td>
<td>$x_{mn}$</td>
<td>$e_m$</td>
</tr>
</tbody>
</table>

$s_1$ $s_2$ … $s_n$

Output: ranking of blocks in order of likelihood to be at fault
SFL: Example

```c
void RationalSort(int n, int *num, int *den)
{
    /* block 1 */
    int i, j, temp;

    for (i = n - 1; i >= 0; i--)
    {
        /* block 2 */
        for (j = 0; j < i; j++)
        {
            /* block 3 */
            if (RationalGT(num[j], den[j],
                            num[j + 1], den[j + 1]))
            {
                /* block 4 */
                temp = num[j];
                num[j] = num[j + 1];
                num[j + 1] = temp;
            }
        }
    }
}
```

<table>
<thead>
<tr>
<th>input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1 = {}$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$I_2 = {\frac{1}{2}}$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$I_3 = {\frac{1}{2}, \frac{1}{3}}$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$I_4 = {\frac{1}{2}, \frac{1}{3}, \frac{0}{1}}$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$I_5 = {\frac{3}{2}, \frac{1}{3}, \frac{1}{2}}$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$I_6 = {\frac{1}{2}, \frac{1}{3}, \frac{1}{2}, \frac{1}{1}}$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$s_J$ | .17 | .20 | .25 | .33 | .25 |
Overview

• Background
  – Terminology
  – SFL

• Experiments
  – Benchmark Metric
  – What is the effect of various external factors on the diagnostic accuracy?
    • Impact of the error detector
    • Impact of the number of runs

• Conclusions
Benchmark and Metric

• Siemens Benchmark set
  – 7 programs, 20 – 124 blocks
  – 7 – 32 faulty versions per program: 132 faults in total
    • Correct version available → Pass/fail error vector
  – Up to 1000 – 5000 test cases per program:
    full code coverage

• Evaluation Metric
  – Diagnostic quality \( (q_d) \)
    • \( q_d = 1 - \frac{\text{block position}}{\#\text{blocks}} \)
    • Measures % of code that NEED NOT be inspected
Experiments

• Previous study [PRDC06] showed that Ochiai outperforms the other coefficients
  • $q_d = 84\%$ (vs. 79\% of Jaccard vs. 77\% of Tarantula)

• E1: What is the impact of error detection quality?
  – Faulty activations do not necessarily lead to failures!

• E2: What is the impact of the number of runs?
  – Thousands of test cases might not be available!
E1: Error Detection Impact

• Error Detection Quality
  – \( q_e = \frac{a_{11}}{a_{11} + a_{10}} \)
  – Approximate measure: not all fault activations lead to failures
    • e.g., \( if( x > 3 ) \) instead \( if( x \geq 3 ) \)

• passed / failed runs that activate the fault were randomly discarded

• SFL is run for the above (sub-)set of test cases
E1: Error Detection Impact

- Small fraction of fault activations detected is enough
- Ochiai performance gain is structural, and strongest for low $q_e$
E2: #Runs Impact

- #Runs (Test cases available)
  - Number of passed runs ($N_p$)
  - Number of failed runs ($N_f$)

- $N_p$ and $N_f$ randomly selected

- SFL is run for the above (sub-)set of test cases
E2: #Runs Impact

- For the Siemens set, two trends were observed
  - Adding failed runs does not harm $q_d$
  - Passed runs, however, have unpredictable effect
E2: #Runs Impact

- On average,
  - 6 failed tests are enough
  - It stabilizes around 20 passed tests
Remarks on the Coefficients

- Ochiai structurally outperforms the other coefficients → Why??
- Tarantula
  \[ s_j = \frac{a_{11}}{a_{11} + a_{01}} \]
  \[ \frac{a_{11}}{a_{11} + a_{01}} + \frac{a_{10}}{a_{10} + a_{00}} = \frac{1}{1 + c \frac{a_{10}}{a_{11}}} \]
- Jaccard
  \[ s_j = \frac{a_{11}}{a_{11} + a_{01} + a_{10}} \]
- Ochiai
  \[ s_j = \frac{a_{11}}{a_{11} + a_{01} + a_{10} + \frac{a_{10}a_{01}}{a_{11}}} \]
Overview

• Background
  – Terminology
  – SFL

• Experiments
  – Benchmark Metric
  – What is the effect of various external factors on the diagnostic accuracy?
    • Impact of the error detector
    • Impact of the number of runs

• Conclusions
Conclusions

• SFL improves the efficiency of debugging, and can be easily integrated with testing
  – Low memory / CPU overhead
  – Little infrastructure needed, no models required
• The diagnosis results are useful, even with low quality error detection
• Adding failed runs is always safe, whereas passed runs may have a negative impact on the accuracy
• Ochiai structurally outperforms other coefficients
Future Work

• Passed tests may deteriorate the ranking, so which passed tests to use?
  – Strategies for selecting passed tests

• How to exploit knowledge about a system?
  – Known data / control dependencies, hierarchies, …

• SFL is robust to error detection quality
  – Techniques for automatic error detection
  – Writing a paper on the use of generic program invariants as error detectors
    • Automatic error detection and fault localization technique
Questions

• www.esi.nl/trader