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

TU Delft

Delft University of Technology

Embedded Systems INSTITUTE

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

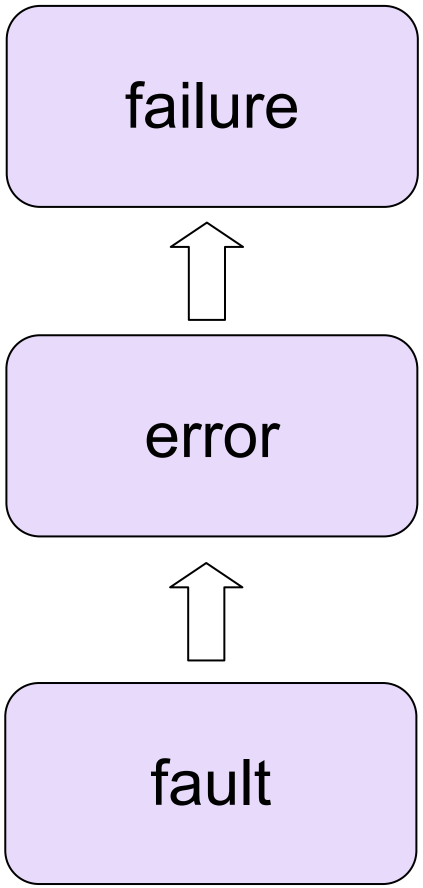
TRADER project

- 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



delivered service ≠ correct service

(segmentation fault)

system state that may cause a failure

(index out of bounds)

the cause of an error in the system

(bug: array index un-initialized)

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	x_2	…	x_i	…	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

n blocks

m cases

x_{11}	x_{12}	…	x_{1n}		e_1
x_{21}	x_{22}	…	x_{2n}		e_2
…	…	…	…		…
x_{m1}	x_{m2}	…	x_{mn}		e_m

SFL

1. Spectra for *m* test cases

x_{11}	x_{12}	…	x_{1n}		e_1
x_{21}	x_{22}	…	x_{2n}		e_2
…	…	…	…		…
x_{m1}	x_{m2}	…	x_{mn}		e_m

Row *i*: the blocks that are executed in case *i*

SFL

1. Spectra for *m* test cases

x_{11}	x_{12}	…	x_{1n}		e_1
x_{21}	x_{22}	…	x_{2n}		e_2
…	…	…	…		…
x_{m1}	x_{m2}	…	x_{mn}		e_m

Column *j* : the test cases in which block *j* was executed

SFL

1. Spectra for *m* test cases
2. **Error detection per test case**

x_{11}	x_{12}	…	x_{1n}
x_{21}	x_{22}	…	x_{2n}
…	…	…	…
x_{m1}	x_{m2}	…	x_{mn}

e_1
e_2
…
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

block j

x_{11}	x_{12}	…	x_{1n}
x_{21}	x_{22}	…	x_{2n}
…	…	…	…
x_{m1}	x_{m2}	…	x_{mn}

error vector

e_1
e_2
…
e_m

similarity s_j

SFL

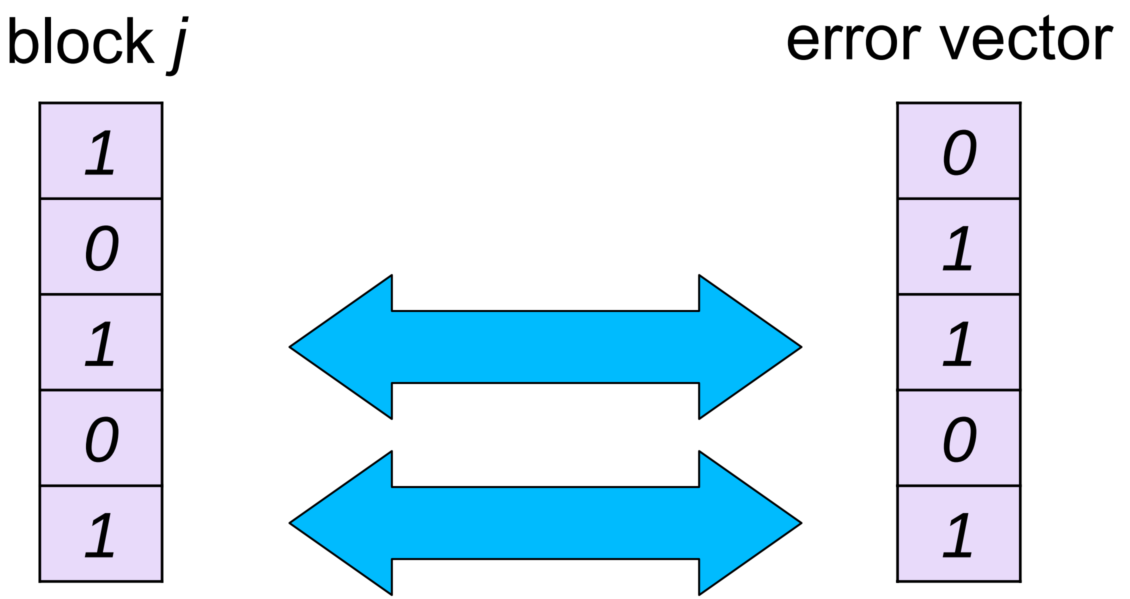
Jaccard similarity coefficient

block j	error vector
1	*0*
0	*1*
1	*1*
0	*0*
1	*1*

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

SFL

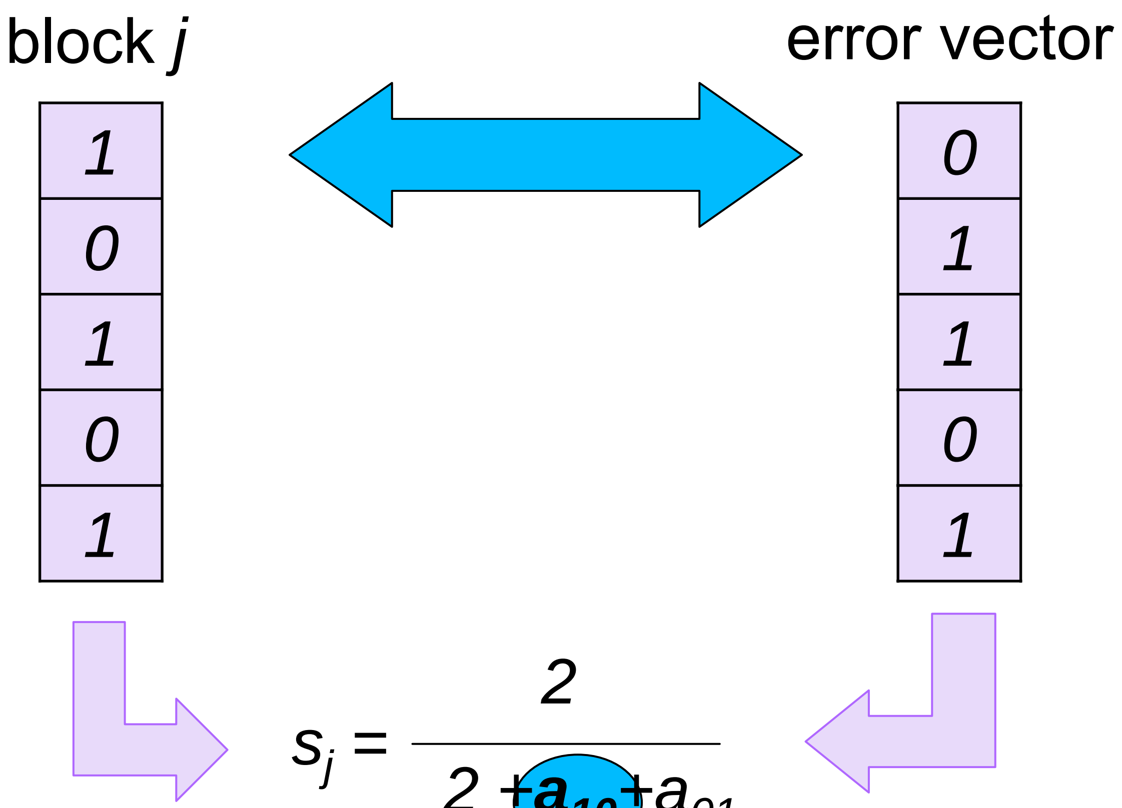
Jaccard similarity coefficient



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

SFL

Jaccard similarity coefficient



$$s_j = \frac{2}{2 + \mathbf{a_{10}} + a_{01}}$$

SFL

Jaccard similarity coefficient

block j	error vector
1	0
0	1
1	1
0	0
1	1

$$s_j = \frac{2}{2 + 1 + \mathbf{a_{01}}}$$

SFL

Jaccard similarity coefficient

block *j*	error vector
1	*0*
0	*1*
1	*1*
0	*0*
1	*1*

$$s_j = \frac{2}{2 + 1 + 1}$$

SFL

For every block: similarity with the error “block”

n blocks | error vector

	x_{11}	x_{12}	…	x_{1n}		e_1
	x_{21}	x_{22}	…	x_{2n}		e_2
m cases	…	…	…	…		…
	x_{m1}	x_{m2}	…	x_{mn}		e_m
	s_1	s_2	…	s_n		

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

SFL: Example

```
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;
            }
        }
    }
}
```

input	block 1	2	3	4	5	error
$I_1 = \langle\,\rangle$	1	0	0	0	0	0
$I_2 = \langle \frac{1}{4} \rangle$	1	1	0	0	0	0
$I_3 = \langle \frac{2}{1}, \frac{1}{1} \rangle$	1	1	1	1	1	0
$I_4 = \langle \frac{4}{1}, \frac{2}{2}, \frac{0}{1} \rangle$	1	1	1	1	1	0
$I_5 = \langle \frac{3}{1}, \frac{2}{2}, \frac{4}{3}, \frac{1}{4} \rangle$	1	1	1	1	1	1
$I_6 = \langle \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{1}{1} \rangle$	1	1	1	0	1	0
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 – (block position / #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

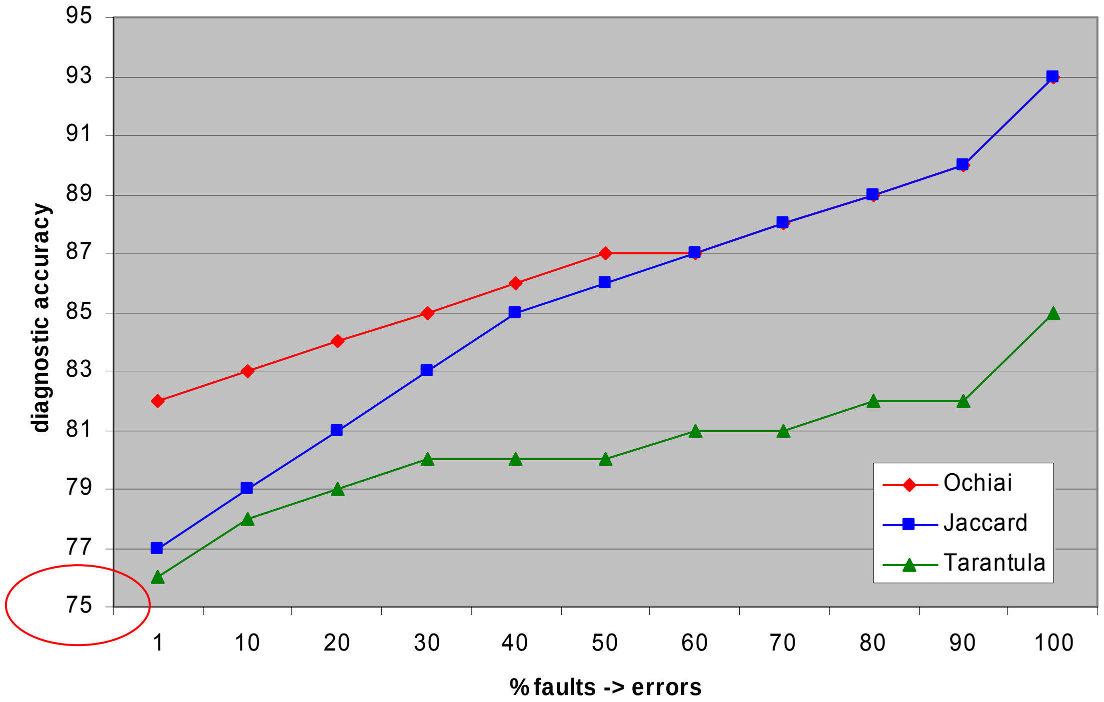
- Error Detection Quality
  - $q_e = a_{11} / (a_{11} + a_{10})$
  - Approximate measure: not all fault activations lead to failures
    - *e.g., if(x > 3)* instead *if(x >= 3)*

- 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

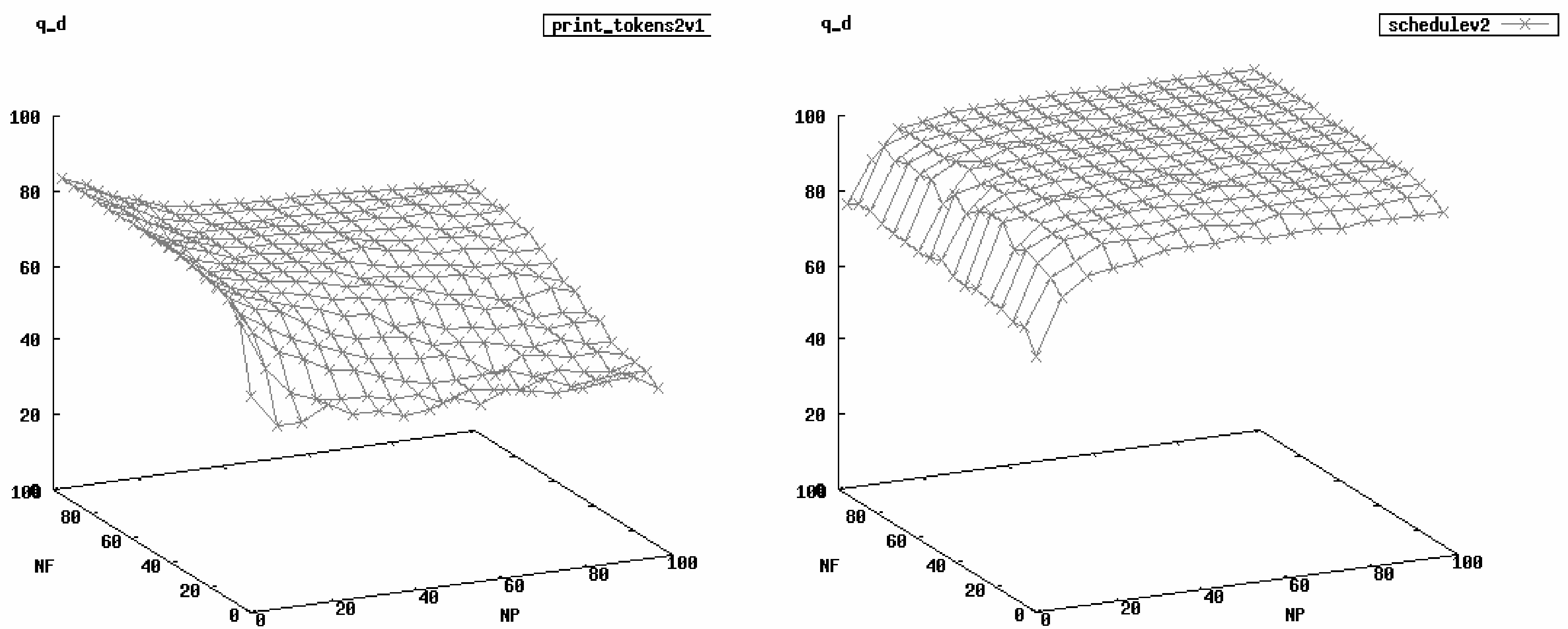
- #Runs (Test cases available)
  - Number of passed runs (N_p)
  - Number of failed runs (N_f)

- N_p and N_f randomly selected

- SFL is run for the above (sub-)set of test cases

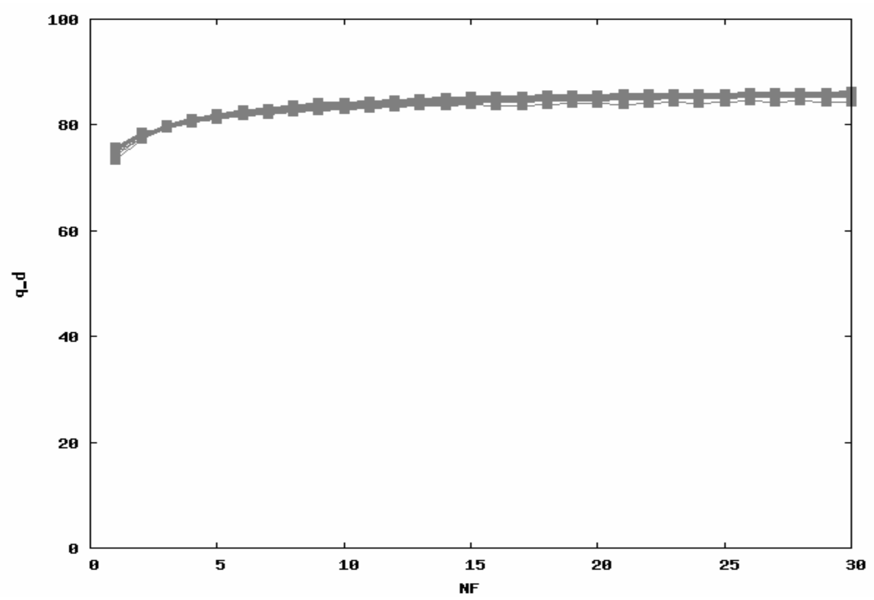
E2: #Runs Impact

- 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

$$s_j = \frac{\dfrac{a_{11}}{a_{11} + a_{01}}}{\dfrac{a_{11}}{a_{11} + a_{01}} + \dfrac{a_{10}}{a_{10} + a_{00}}} = 1 \Big/ 1 + c\,\frac{a_{10}}{a_{11}}$$

- Jaccard

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

- Ochiai

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

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
- [PRDC06] R. Abreu, P. Zoeteweij and A.J.C. van Gemund. *An Evaluation of Similarity coefficients for Software Fault Localization.* PRDC'06