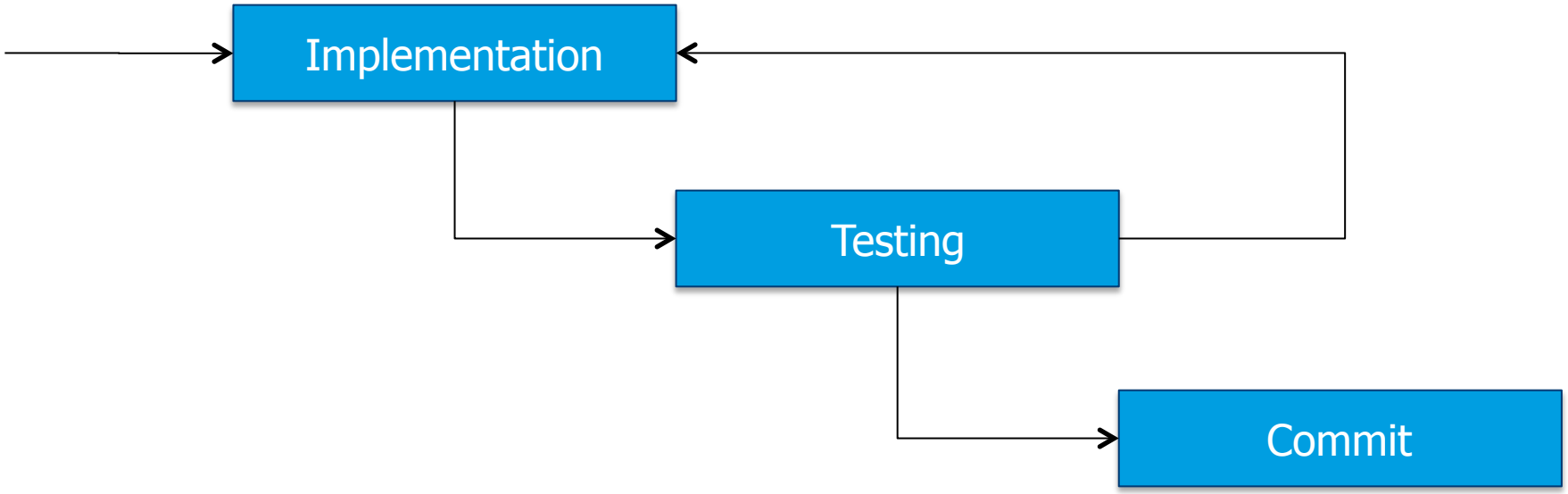
Exploiting Count Spectra for Bayesian Fault Localization

9/13/10

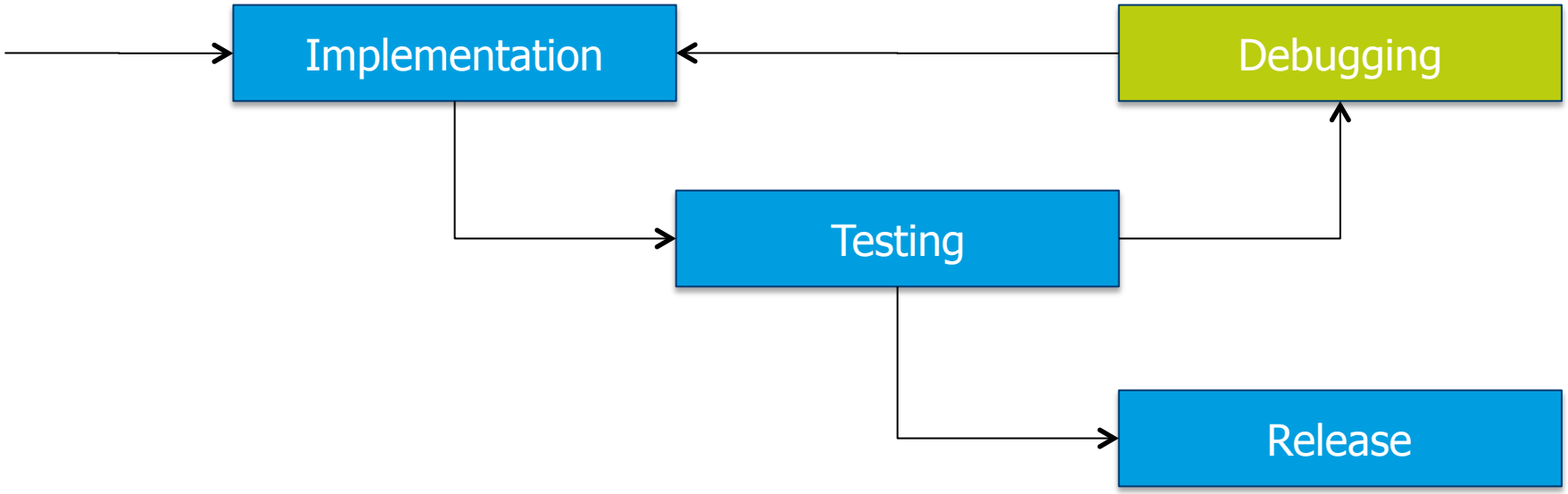
Rui Abreu
Alberto González-Sánchez
Arjan van Gemund

(Simplified) Software Lifecyle




WHERE IS THE FAULT?

(Simplified) Software Lifecyle



Preliminaries

- Diagnostic system DS = <SD, COMPS, OBS>
  - SD, system description (propositional theory)
  - COMPS = $\{c_1,\ldots,c_M\}$
  - OBS, set of observable variables
- Each component c_j has a health variable h_j
  - $\forall j : h_j \Rightarrow inp\_ok_j \Rightarrow out\_ok_j$
- A failed test is an observation *obs*
- Diagnostic candidates d_k that explain the observations
  - Defining the states of the variables h_j
- Diagnostic ranking $D = \{d_1, \ldots, d_k, \ldots, d_K\}$

System and Observation Model

$$h_1 \Rightarrow (x_1 \Rightarrow y_1)$$
$$h_2 \Rightarrow (x_2 \Rightarrow y_2)$$
$$h_3 \Rightarrow (x_3 \Rightarrow y_3)$$
$$h_4 \Rightarrow (x_4 \Rightarrow y_4)$$
$$h_5 \Rightarrow (x_5 \Rightarrow y_5)$$
$$x_5 = y_4$$
$$x_4 = y_3$$
$$x_3 = y_2$$
$$x_2 = y_1$$
$$x_1 = \text{true}$$
$$y_4 = \neg e_n$$

$$h_1 \wedge h_2 \wedge h_3 \wedge h_4 \wedge h_5 \Rightarrow false$$

$$\neg h_1 \vee \neg h_2 \vee \neg h_3 \vee \neg h_4 \vee \neg h_5$$

(conflict)

{1} {2} {3} {4} {5}

Spectrum-based Reasoning

$$N \text{ spectra} \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1M} \\ a_{21} & a_{22} & \dots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NM} \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix}$$

M components

errors

Figure 1: Input for SFL

Spectrum-based Reasoning

c_1	c_2	c_3	e
1	0	0	1 (F)
0	1	0	1 (F)
1	0	1	1 (F)
0	1	1	1 (F)
1	1	0	0 (P)

(conflicts)

$\neg h_1$

$\neg h_2$

$\neg h_1 \vee \neg h_3$

$\neg h_2 \vee \neg h_3$

$\neg h_1 \wedge \neg h_2$

diagnosis candidates
(*minimal hitting set*)

1 diagnosis candidate $\{c_1, c_2\}$ => correct diagnosis

Candidate Ranking

- Probabilities updated according to Bayes' rule

$$\Pr(d_k|obs) = \frac{\Pr(obs|d_k)}{\Pr(obs)} \cdot \Pr(d_k)$$

- where

$$\Pr(obs|d_k) = \begin{cases} 0 & \text{if } d_k \text{ and } obs \text{ are inconsistent} \\ 1 & \text{if } d_k \text{ logically follows from } obs \\ \varepsilon & \text{if neither holds} \end{cases}$$

Candidate Ranking

- One of the best ε for software fault localization (ASE'09)

$$\varepsilon = \begin{cases} g(d_k)^\eta & \text{if run passed} \\ 1 - g(d_k)^\eta & \text{if run failed} \end{cases}$$

- Where

$$g(d_k) = \frac{\sum_{n=1..N} [(\bigvee_{j \in d_k} a_{nj} \neq 0) \wedge e_n = 0]}{\sum_{n=1..N} [\bigvee_{j \in d_k} a_{nj} \neq 0]}$$

Example

```
/* block 0: main */
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;
      }
    }
  }
}
```

Example (good)

	block						
T	**0**	**1**	**2**	**3**	**4**	**5**	**e**
1	1	1	1	1	0	1	0
2	1	1	1	1	1	1	1

#	d_k	$Pr(d_k)$
1	{4}	0.44
2	{1}	0.11
2	{2}	0.11
2	{3}	0.11
2	{0}	0.11
2	{5}	0.11

Example (bad)

	block						
T	**0**	**1**	**2**	**3**	**4**	**5**	**e**
1	1	1	1	1	1	1	0
2	1	1	1	1	1	1	1

#	d_k	$Pr(d_k)$
1	{0}	0.16
1	{1}	0.16
1	{2}	0.16
1	{3}	0.16
1	{0}	0.16
1	{4}	0.16

Count Spectra as Tie-breaker

	block						
T	**0**	**1**	**2**	**3**	**4**	**5**	**e**
1	1	1	4	6	3	6	0
2	1	1	4	6	5	6	1

?

Zoltar-C

	block						
T	**0**	**1**	**2**	**3**	**4**	**5**	**e**
1	1	1	4	6	3	6	0
2	1	1	4	6	5	6	1

$$\varepsilon = \begin{cases} g(d_k)^t & \text{if } e_i = 0 \\ 1 - g(d_k)^t & \text{if } e_i = 1 \end{cases}$$

$$t = \sum_{j \in d_k} a_{ij}.$$

#	d_k	$Pr(d_k)$
1	{0}	0.27
2	{1}	0.21
2	{2}	0.21
2	{3}	0.19
2	{0}	0.07
2	{4}	0.07

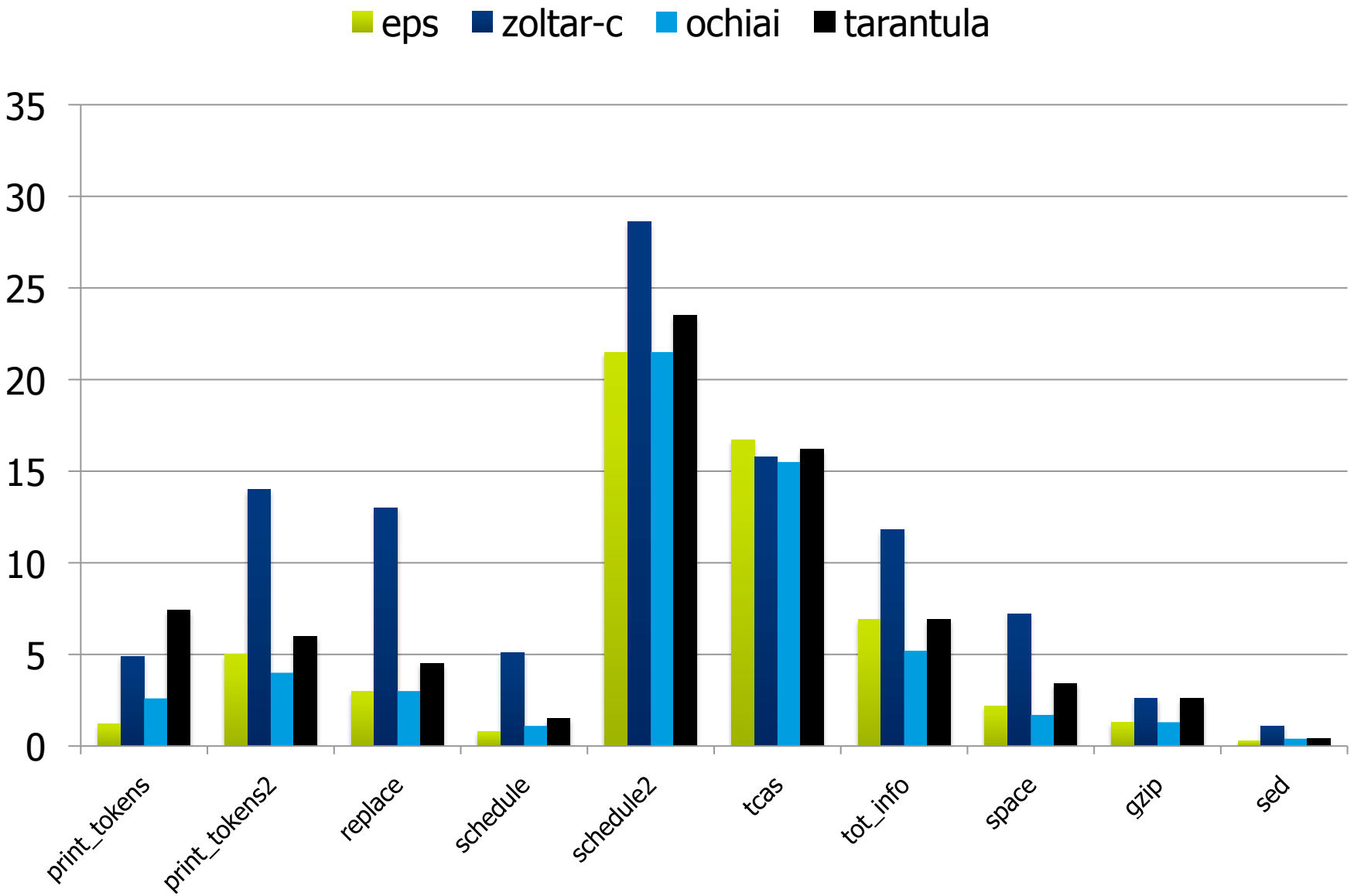
Evaluation

- Real-world software programs
  - Software infrastructure repository
  - O(10) faulty versions
  - O(100) LOC, O(1000) test cases
  - 1, 2, 5 faults
- Programs have been modified to accommodate multiple faults

Evaluation Metric - W

- Excess work incurred in finding the faulty components
- Independent of number of faults

- As an example, suppose
  - A M=10-component program, where c_1 and c_2 are faulty
  - D = <{1,3}, {3,5}, {2,4}>
  - Inspecting order: 1,3,4,2
  - ***W* = 2/(10-2)**


eps
zoltar-c
ochiai
tarantula
35
30
25
20
15
10
5
0
print_tokens
print_tokens2
replace
schedule
schedule2
tcas
tot_info
space
gzip
sed

Fail ☹

- Unrealistic assumption:
  - Bit-spectra work because observations are independent
  - Each "cover" in the count is NOT independent
- $g(d_k)$ is calculated on a bit fashion
- Pass/fail of test cases sample is highly biased in siemens
- tcas works because most of the times the count is 0-1

Conclusions

- Current spectrum-based approaches only exploit component involvement, which leads to ties
- Zoltar-C: tie-breaking by exploiting count spectra
- Zoltar-C does not improve the diagnostic process
  - Assumes (unrealistic) or-model
  - Highly biased sample of pass/fail test cases

Future Work

- Theoretically study the performance of count spectra
- Study the performance impact of the (more/less) number of runs
- Predict number of faults in system

Co-located with ICST

Deadline December 20th

http://paginas.fe.up.pt/~tebug2011/

Mulțumesc!