Interactive Fault Localization Leveraging Simple User Feedback

Liang Gong[1], David Lo[2], Lingxiao Jiang[2], and Hongyu Zhang[1]

Funded by

清華大學 Tsinghua University [1] *Tsinghua University*

SMU SINGAPORE MANAGEMENT UNIVERSITY [2] *Singapore Management University*

Outline

Introduction & Framework

- **Motivation**
- Fault Localization

- **Interactive Fault Localization**
- Technical Motivation
- Detailed Approach

- **Experiments**
- Settings & Results

- **Conclusion & Future work**

Tsinghua University 清華大學

SMU SINGAPORE MANAGEMENT UNIVERSITY

Debugging

Problem

- Software errors cost the US economy 59.5 billion dollars (0.6% of 2002's GDP) [1]
- Testing and debugging activities are labor-intensive (30% to 90% of a Project) [2]

[1] National Institute of Standards and Technology (NIST). Software Errors Cost U.S. Economy $59.5 Billion Annually, June 28, 2002.

[2] B. Beizer. Software Testing Techniques. International Thomson Computer Press, Boston, 2nd edition, 1990.

Spectrum-based Fault Localization(abbr. SBFL)

- Automatically recommend a list of suspicious program elements for inspection.

- **Program Spectra** consists of coverage information and execution labels.

Approaches

Fault Localization

For a given statement S

The formula calculates the suspiciousness of S.

No. of failed traces covering S

Ochiai

$$\frac{a_{ef}}{\sqrt{(a_{ef}+a_{nf})(a_{ef}+a_{ep})}}$$

No. of failed traces *No. of traces covering S*

Intuition: *If* ***S*** *is covered* ***more*** *in* ***failed*** *traces and* ***less*** *in* ***passed*** *traces, it is more likely to contain faults.*

Research Goal:

- An interactive fault localization method leveraging user feedback
- Requires trivial or no additional effort

Our Method

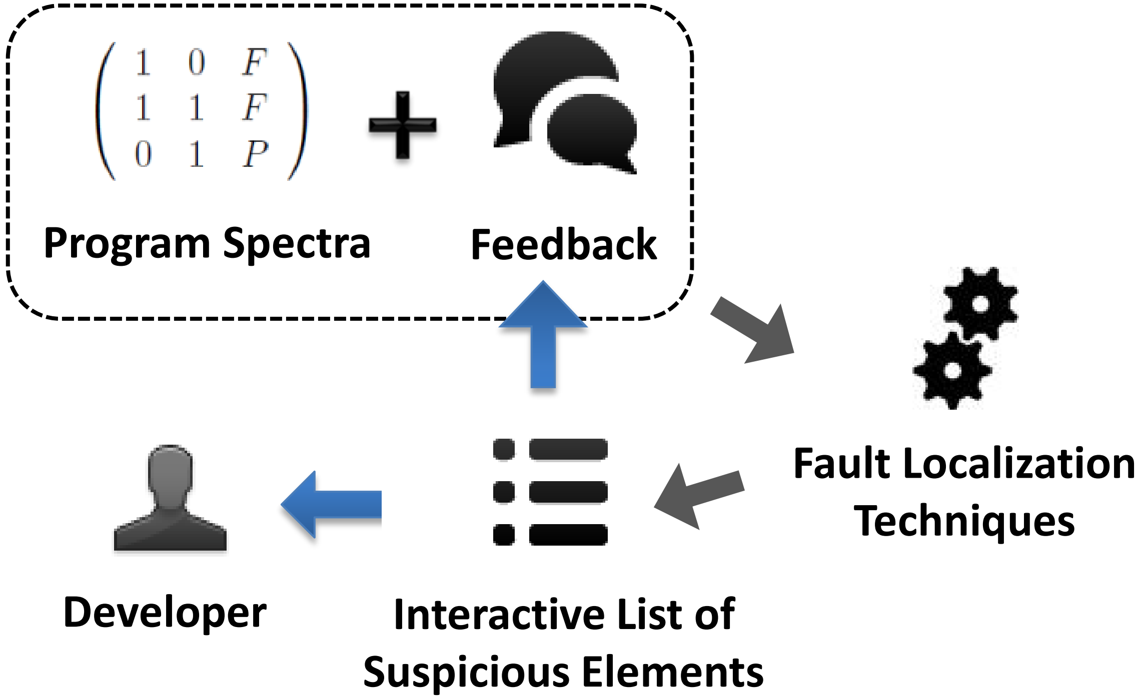
Introduction

FaulT LocAlization Leveraging User FeedbacK (TALK)

- Interactively and iteratively **updates** model according to feedback
- Leveraging only **simple** feedback
- **one-size-fits-all** approach

Process

Motivation



Interactive Fault Localization

Feedback

Opportunities

How to provide feedback which requires trivial additional effort?

No.	Susp.	Program Element
S_{12}	*0.756*	`other += 1;}`
S_{11}	*0.707*	`else if(isprint(c))`
S_{8}	*0.671*	`let += 1;`
S_{9}	*0.667*	`else if('0'<=c && '9'>c)`
S_{5}	*0.603*	`if('A'<=c && 'Z'>=c)`

Conventional Inspection List (*static*)
Proposed Inspection List (*interactive*)

When developer examine the inspection list, they must judge if those statements are clean or faulty.

Tsinghua University
SMU
SINGAPORE MANAGEMENT UNIVERSITY

Feedback

How to utilize ?

With provided feedback, how to **improve** fault localization accuracy?

Once a false positive(symptom) has been found

Find the likely root cause for that symptom

Adjust the suspiciousness of root cause and re-rank

No.	Susp.	Program Element	Buggy?
S_{12}	0.756	`other += 1;}`	✔ ✘
S_{11}	0.707	`else if(isprint(c))`	✔ ✘
S_8	0.671	`let += 1;`	✔ ✘
S_9	0.667	`else if('0'<=c && '9'>c)`	✔ ✘
S_5	0.603	`if('A'<=c && 'Z'>=c)`	✔ ✘

likely root cause

How to find the likely root cause of a symptom?

Investigating **co-occurrences** of program elements in **failed** executions to identify **root cause candidate**

	...	s_1	s_2	s_3	s_4	s_5	s_6	...	p/f
t_1	...	○	●	●	○	○	○	...	*fail*
t_2	...	●	●	●	○	○	○	...	*fail*
t_3	...	○	●	●	○	○	○	...	*pass*

False positive (Symptom)

Intuition: ***If s_3 has been identified as false positive, then s_2 is more likely to be the root cause than s_1.***
As s_2 co-appears more with s_3 in failed traces.

Root Cause

How to find ?

Tsinghua University
SMU SINGAPORE MANAGEMENT UNIVERSITY

Identifying a Root Cause from Its Symptom.

- Evaluate the **co-appearance score** of statements (root cause candidate)

$$\mathcal{P}_s(s_c) = \sum_{t \in T_{fail}(s) \wedge s_c \in t} \frac{|\mathcal{D}|}{|\{s' | s' \in t \wedge s' \notin \mathcal{I}\}|}$$

Co-appearance from traces covering less statements weights higher

- Select candidate with most co-appearance score as the **root cause**

$$s_r \leftarrow \arg\max_{s_c \in T} \{\mathcal{P}_s(s_c)\}$$

Rule: R1

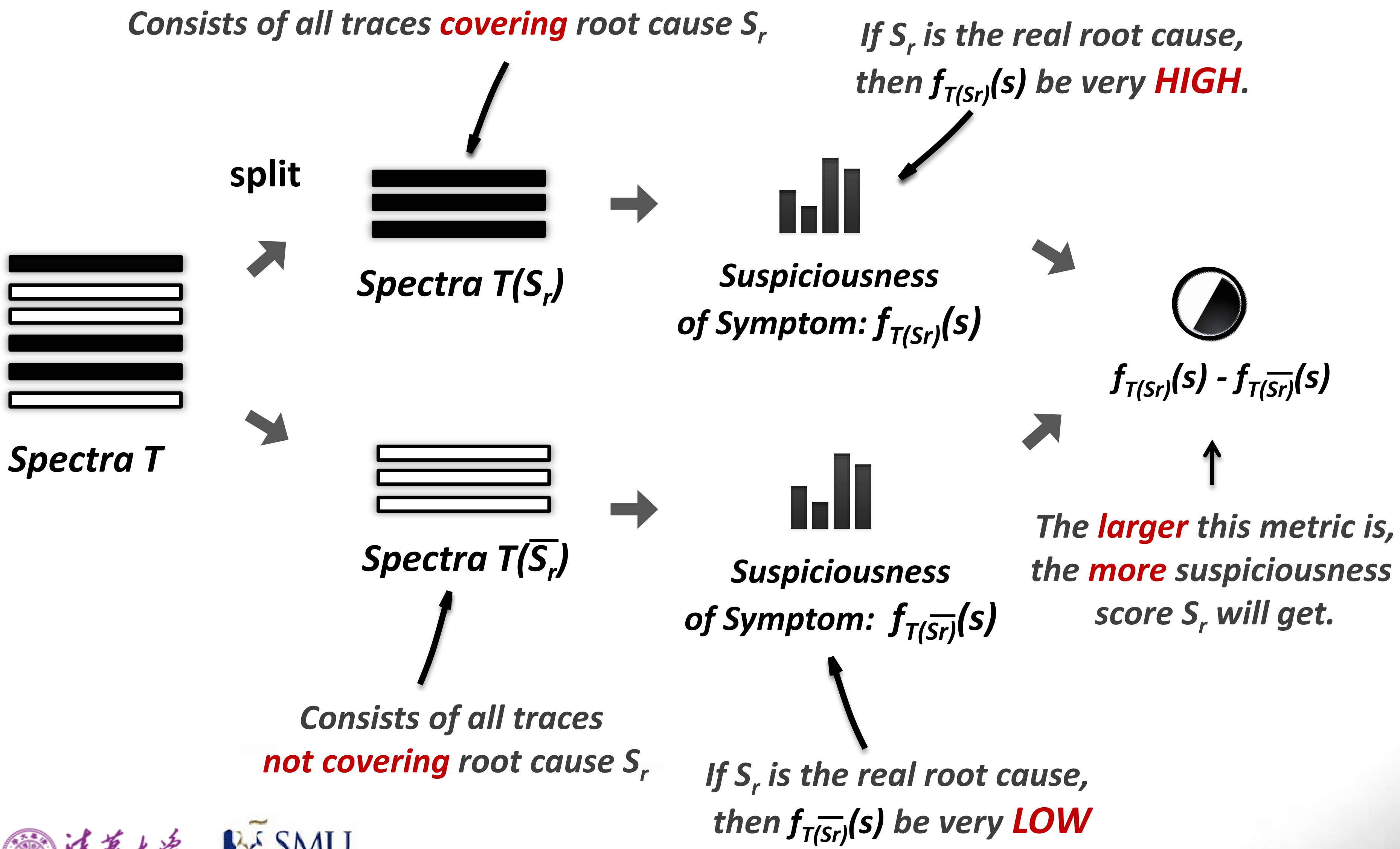
Detailed Approach

Intuition: ***Statements co-appeared more with symptom in failed traces covering less statements are more likely to be chosen as the root cause.***

Tsinghua University

SMU SINGAPORE MANAGEMENT UNIVERSITY

With the spectra *T* and symptom *S*, and how to **adjust** the suspiciousness score of the root cause S_r ?



Tsinghua University
SMU SINGAPORE MANAGEMENT UNIVERSITY

Rule: R1

Detailed Approach

With the symptom, how to **adjust** the suspiciousness score of the root cause?

- Calculate the suspiciousness difference of symptom in spectra **covering** and **not covering** root cause.

$$\mathcal{W}_{s_r \to s} = f_{T(s_r)}(s) - f_{T(\overline{s_r})}(s)$$

- Contribute the suspiciousness difference of symptom to the suspiciousness of its root cause

$$Susp_{s_r} = f_T(s_r) + \mathcal{W}_{s_r \to s} \cdot f_T(s)$$

Intuition: ***If the suspiciousness of symptom becomes larger when root cause is covered, the root cause is more suspicious.***

Focusing on a Single Failed Execution Profile

	...	s_1	s_2	s_3	s_4	s_5	s_6	...	p/f
t_1	...	○	●	●	○	○	○	...	fail
t_2	...	●	●	●	○	●	●	...	fail
t_3	...	○	●	●	○	○	○	...	pass

Rule 2

Introduction

In this case, focusing on statements covered by t_1 will quickly identify at least one bug. As only two statements have to be examined.

- Find out the failed profile t_{min} covering the least number of unexamined elements.
- For each program element s_i that is covered in t_{min}

$$Susp'_{s_i} = K_{s_i} \cdot Susp_{s_i}$$

A constant making sure that statements covered by t_{min} are examined first.

Tsinghua University

SMU SINGAPORE MANAGEMENT UNIVERSITY

Initial Process: Conventional Fault Localization

Processing False Positive with Rule 1

If a bug has been confirmed, record it for Rule 2

Apply Rule 2

Processing Feedback

Procedure *TALK*
Input:
  T - Program spectra
Output:
  $\mathcal{L}$ Ranked list of suspicious program elements
Method:
  Build a fault localization model f_T using T
  Obtain a ranked list $\mathcal{L}_T$ according to f_T
  $\mathcal{L} \leftarrow \mathcal{L}_T$; ***show_result***$(\mathcal{L})$
  while ***wait_feedback()*** **do**
    $\mathcal{F} \leftarrow$ ***obtain_feedback()***
    for each $\langle s, l_s \rangle \in \mathcal{F}$ **do**
      $\mathcal{L}_T \leftarrow \mathcal{L}_T \setminus s$; $\mathcal{I} \leftarrow \mathcal{I} \bigcup \{s\}$
      if l_s =*clean* **then**
        //Identify the root cause
        $\forall s_c \in T$, calculate $\mathcal{P}_s(s_c)$ by Equation 1
        Select s_r by Equation 2
        Update $Susp_{s_r}$ in $\mathcal{L}_T$ according to Equation 3
      else if l_s =*faulty* **then**
        $\mathcal{E} \leftarrow \mathcal{E} \bigcup \{s\}$
      end if
      $\mathcal{L} \leftarrow \mathcal{L}_T$
      //Focus on one failed profile
      $t_{min} \leftarrow$***least_fail_profile***$(\mathcal{E})$
      Update $Susp_{s_r}$ in $\mathcal{L}$ according to Equation 5
      show_result$(\mathcal{L})$
    end for
  end while
  return $\mathcal{L}$

Experiment

Dataset & Evaluation Metric

Benchmarks for Fault Localization

Program	Description	LOC	Tests	Faults
tcas	Aircraft Control	173	1609	41
schedule2	Priority Scheduler	374	2710	8
schedule	Priority Scheduler	412	2651	8
replace	Pattern Matcher	564	5543	31
tot_info	Info Measure	565	1052	22
print_tokens2	Lexical Analyzer	570	4055	10
print_tokens	Lexical Analyzer	726	4070	7
space	ADL Compiler	9564	1343	30
flex	Lexical Parser	10124	567	43
sed	Text Processor	9289	371	22
grep	Text Processor	9089	809	17
gzip	Data Compressor	5159	217	15

1 **Siemens Suite** 2 **UNIX Programs**

Evaluation Metric for Fault Localization:

$$cost = \frac{|\{j \mid f_{T_S}(d_j) \geq f_{T_S}(d_*)\}|}{|\mathcal{D}|}$$

Experiment

Research Questions

Research Questions Investigated:

- Is user feedback helpful for improving fault localization accuracy?
- What is the relative effectiveness of the two rules to improve fault localization?

Tsinghua University

SMU SINGAPORE MANAGEMENT UNIVERSITY

Comparison

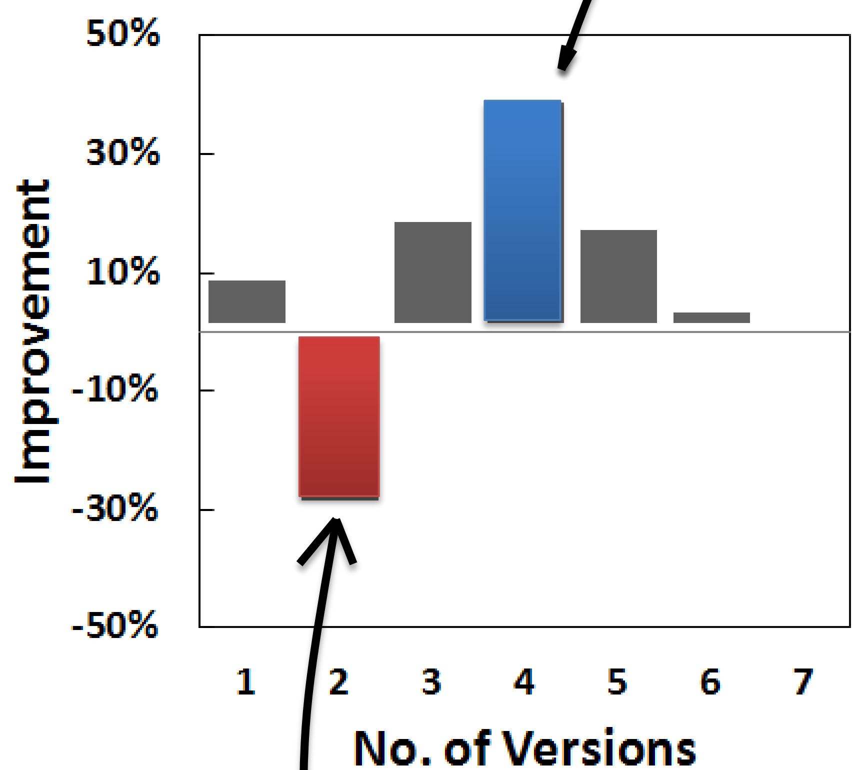
Introduction

Conventional Fault Localization Technique f

Interactive Fault Localization Technique f+

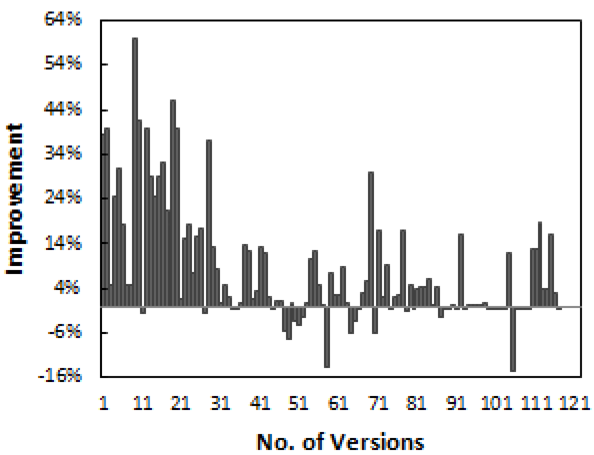
f+ vs f

☺ *f+ requires 40% less debugging cost than f on faulty version 4*

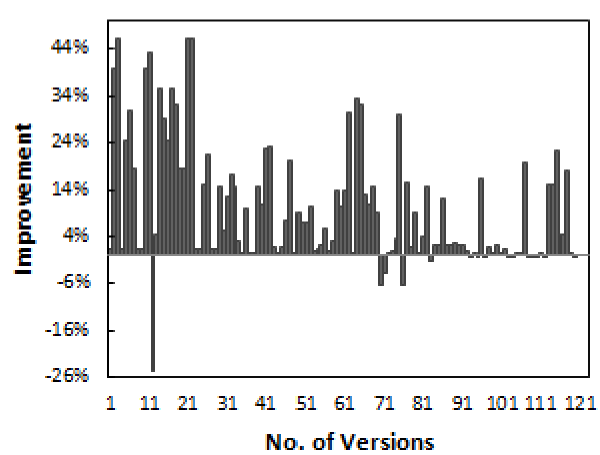


☹ *f+ requires 30% more debugging cost than f on faulty version 2*

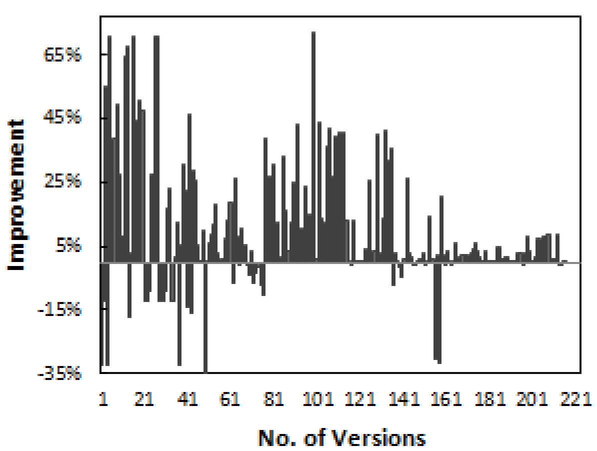
Improvement of TALK on Existing Methods



Ochiai+ vs Ochiai



Jaccard+ vs Jaccard



Tarantula+ vs Tarantula

Pair-wised T-test shows the improvements are statistically significant at 95% interval.

Tsinghua University

SMU SINGAPORE MANAGEMENT UNIVERSITY

Experiment

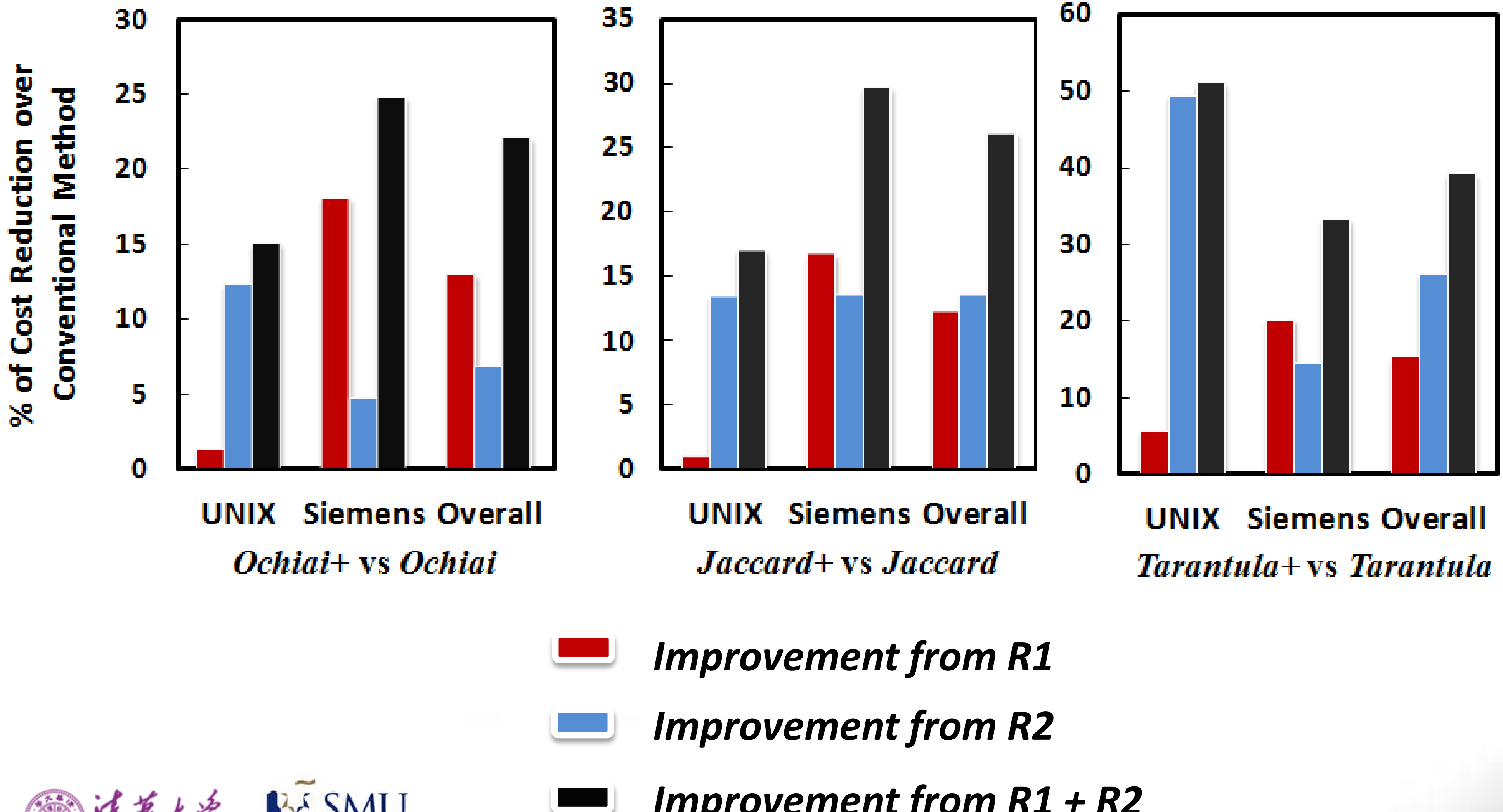
Research Questions

Research Questions Investigated:

- Is user feedback helpful for improving fault localization accuracy?
- What is the relative effectiveness of the two rules to improve fault localization?

Contributions of *R1* and *R2*

on Improving Fault Localization Effectiveness



Tsinghua University

SMU SINGAPORE MANAGEMENT UNIVERSITY

Related Works

Introduction

Fault Localization *(state-of-the-arts)*

- ✓ *WHITHER[Renieris and Reiss]*
- ✓ *Liblit05[Liblit]*
- ✓ *Delta Debugging[Zeller]*
- ✓ *Tarantula[Harrold], Ochiai etc.*

Interactive Fault Localization

Hao et al. propose an technique **[JCST]**

- ✓ **Record sequential execution trace**
- ✓ **Judge** whether the fault is executed **before or after the checking point**.

Lucia et al. adopt user feedback for clone-based bug detection approaches **[ICSE 12]**

Insa et al. propose a strategy for **algorithmic debugging** which asks user questions concerning program state. **[ASE 11]**

Tsinghua University

SMU SINGAPORE MANAGEMENT UNIVERSITY

Threats to *Validity*

Construct Validity *(Evaluation Metric)*

We use a cost metric that has been utilized to evaluate past fault localization techniques. We believe this is a fair and well-accepted metric.

External Validity *(Generalizability)*

All of subjects are written in C. In the future, we plan to investigate more programs written in various programming languages.

What if user provides a wrong feedback?

Tsinghua University

SMU
SINGAPORE MANAGEMENT UNIVERSITY

Threats to *Validity*

What if user provides a **wrong feedback**?

Since we use simple feedbacks, mistakes can only be:

- *Clean Statement* labeled as *Faulty*

In this case, when developers try to fix the "bug", they will realize their mistake.

- *Faulty Statement* labeled as *Clean*

*Most fault localization techniques are evaluated by assuming a user is **always correct** when ascertaining if an element is buggy or correct.*

In future: *Allow users to **rollback** their feedback if they made mistakes.*

Tsinghua University

SMU SINGAPORE MANAGEMENT UNIVERSITY

Thank you!

- Any questions?

• • •