Evaluating and Improving Fault Localization

Spencer Pearson†, José Campos*, René Just, Gordon Fraser*, Rui Abreu‡, Michael D. Ernst†, Deric Pang†, Benjamin Keller†

University of Massachusetts
†University of Washington
*University of Sheffield
‡University of Lisbon

May 25, 2017
Fault localization: an important problem

Two use cases: developers and automated program repair

Many techniques and evaluations

<table>
<thead>
<tr>
<th>Prior studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>(winner &gt; loser)</td>
</tr>
<tr>
<td>Ochiai &gt; Tarantula [NLR11, LTL, WDGL14, XM14, LLT15]</td>
</tr>
<tr>
<td>Barinel &gt; Ochiai [AZVG09]</td>
</tr>
<tr>
<td>Barinel &gt; Tarantula [AZVG09]</td>
</tr>
<tr>
<td>Op2 &gt; Ochiai [NLR11]</td>
</tr>
<tr>
<td>Op2 &gt; Tarantula [NLR11, MKKY14]</td>
</tr>
<tr>
<td>DStar &gt; Ochiai [WDGL14, LLT15]</td>
</tr>
<tr>
<td>DStar &gt; Tarantula [WDGL14, JJC+14, LLT15]</td>
</tr>
<tr>
<td>Metallaxis &gt; Ochiai [PLT15]</td>
</tr>
<tr>
<td>MUSE &gt; Op2 [MKKY14]</td>
</tr>
<tr>
<td>MUSE &gt; Tarantula [MKKY14]</td>
</tr>
</tbody>
</table>
Fault localization: an important problem

Two use cases: developers and automated program repair

Many techniques and evaluations

<table>
<thead>
<tr>
<th>Prior studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>(winner &gt; loser)</td>
</tr>
<tr>
<td>Ochiai &gt; Tarantula [NLR11], [LTL], [WDD10]</td>
</tr>
<tr>
<td>Barinel &gt; Ochiai [AZVG09]</td>
</tr>
<tr>
<td>Barinel &gt; Tarantula [AZVG09]</td>
</tr>
<tr>
<td>Op2 &gt; Ochiai [NLR11]</td>
</tr>
<tr>
<td>Op2 &gt; Tarantula [NLR11], [MKKY14]</td>
</tr>
<tr>
<td>DStar &gt; Ochiai [WDGL14], [LLT15]</td>
</tr>
<tr>
<td>DStar &gt; Tarantula [WDGL14], [JJC+14], [XHM13]</td>
</tr>
<tr>
<td>Metallaxis &gt; Ochiai [PLT15]</td>
</tr>
<tr>
<td>MUSE &gt; Op2 [MKKY14]</td>
</tr>
<tr>
<td>MUSE &gt; Tarantula [MKKY14]</td>
</tr>
</tbody>
</table>

Do these results hold for real world programs?
Fault localization: an important problem

Two use cases: developers and automated program repair

Many techniques and evaluations

<table>
<thead>
<tr>
<th>Prior studies</th>
<th>(winner &gt; loser)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ochiai</td>
<td>Tarantula [NLR11], [LTL], [WDC11]</td>
</tr>
<tr>
<td>Barinel</td>
<td>Ochiai [AZVG09]</td>
</tr>
<tr>
<td>Barinel</td>
<td>Tarantula [AZVG09]</td>
</tr>
<tr>
<td>Op2</td>
<td>Ochiai [NLR11]</td>
</tr>
<tr>
<td>Op2</td>
<td>Tarantula [NLR11], [MKKY14]</td>
</tr>
<tr>
<td>DStar</td>
<td>Ochiai [WDGL14], [LLT15]</td>
</tr>
<tr>
<td>DStar</td>
<td>Tarantula [WDGL14], [JJC+14]</td>
</tr>
<tr>
<td>Metallaxis</td>
<td>Ochiai [PLT15]</td>
</tr>
<tr>
<td>MUSE</td>
<td>Op2 [MKKY14]</td>
</tr>
<tr>
<td>MUSE</td>
<td>Tarantula [MKKY14]</td>
</tr>
</tbody>
</table>

Do these results hold for real world programs? **NO!**

Why?
1. Unrealistic evaluations (artificial faults)
2. Negligible or small effect sizes
3. Unrealistic evaluation metrics
What is fault localization?
What is fault localization?

Program

def double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    for(int i=0; i<n; ++i) {
        sum += nums[i];
    }
    return sum * n;
}

Test suite

- Passing tests
- Failing tests
What is fault localization?

Program

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    for(int i=0; i<n; ++i) {
        sum += nums[i];
    }
    return sum * n;
}

Test suite

Passing tests ✔
Failing tests ❌

Statement ranking

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    for(int i=0; i<n; ++i) {
        sum += nums[i];
    }
    return sum * n;
}
Fault localization: how it works

Program

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    for(int i=0; i<n; ++i) {
        sum += nums[i];
    }
    return sum * n;
}
Spectrum-based fault localization

Program

```java
double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    for(int i=0; i<n; ++i) {
        sum += nums[i];
    }
    return sum * n;
}
```

Spectrum-based FL (SBFL)

- **Compute** suspiciousness per statement
- **Example:**

\[
S(s) = \frac{\text{failed}(s)/\text{totalfailed}}{\text{failed}(s)/\text{totalfailed} + \text{passed}(s)/\text{totalpassed}}
\]

- Statement **covered** by failing test
- Statement **covered** by passing test

More → statement is more suspicious!

Jones et al., *Visualization of test information to assist fault localization*, ICSE’02
Mutation-based fault localization

Program

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    for(int i=0; i<n; ++i) {
        sum += nums[i];
    }
    return sum * n;
}

Mutants

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    for(int i=0; i<n; ++i) {
        sum += nums[i];
    }
    return sum + n;
}

Mutation-based FL (MBFL)

- **Compute** suspiciousness per mutant
- **Aggregate** results per statement
- Example:

\[ S(s) = \max_{m \in \text{mut}(s)} \frac{\text{failed}(m)}{\sqrt{\text{total failed} \cdot (\text{failed}(m) + \text{passed}(m))}} \]

Mutant affects failing test outcome
Mutant breaks passing test
More ▲→ mutant is more suspicious!

Papadakis and Traon, *Metallaxis-FL: mutation-based fault localization*, STVR’15
Outline and contributions

● **How to evaluate** fault localization techniques?

● **Empirical study** on artificial and real faults:
  ○ Do the results agree with prior work?
  ○ Do the results agree on artificial and real faults?
  ○ *No!* Explain why not.

● **What design decisions matter** (on real faults)?

● **How to improve** fault localization?
Evaluating fault localization techniques

Program

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    for(int i=0; i<n; ++i) {
        sum += nums[i];
    }
    return sum * n;
}

Statement ranking

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    for(int i=0; i<n; ++i) {
        sum += nums[i];
    }
    return sum * n;
}
Evaluating fault localization techniques

Program

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    for(int i=0; i<n; ++i) {
        sum += nums[i];
    }
    return sum * n;
}

Statement ranking

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    for(int i=0; i<n; ++i) {
        sum += nums[i];
    }
    return sum * n;
}

**EXAM score**: relative rank of the defective statement (e.g., \(3/5 = 0.6\)).

Smaller EXAM scores are better!
Evaluating fault localization techniques

Not straightforward for real faults:
- Multi-line defects (localize 1 or all lines?)
- Non-executable code (declarations)
- Fault of omission (>1 possible location)

Details in the paper

**EXAM score**: relative rank of the defective statement (e.g., \(3/5 = 0.6\)).

Smaller EXAM scores are better!
Empirical study on artificial and real faults

Experimental design

- 7 widely studied FL techniques
  - SBFL: Barinel, D*, Ochiai, Op2, and Tarantula
  - MBFL: Metallaxis and Muse
- 310 real faults (5 times as many as prior studies combined)
- 2995 artificial faults (more than prior studies combined)
- 100,000 CPU hours (MBFL is expensive)

http://www.defects4j.org  http://www.mutation-testing.org
## Results of prior studies

<table>
<thead>
<tr>
<th>Prior studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>(winner &gt; loser)</td>
</tr>
<tr>
<td>Ochiai &gt; Tarantula</td>
</tr>
<tr>
<td>Barinel &gt; Ochiai</td>
</tr>
<tr>
<td>Barinel &gt; Tarantula</td>
</tr>
<tr>
<td>Op2 &gt; Ochiai</td>
</tr>
<tr>
<td>Op2 &gt; Tarantula</td>
</tr>
<tr>
<td>DStar &gt; Ochiai</td>
</tr>
<tr>
<td>DStar &gt; Tarantula</td>
</tr>
</tbody>
</table>

### SBFL vs. SBFL

<table>
<thead>
<tr>
<th>Metallaxis &gt; Ochiai</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUSE &gt; Op2</td>
</tr>
<tr>
<td>MUSE &gt; Tarantula</td>
</tr>
</tbody>
</table>
Our results on artificial faults

<table>
<thead>
<tr>
<th>Prior studies (winner &gt; loser)</th>
<th>Ours (artificial faults)</th>
<th>Replicated</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ochiai &gt; Tarantula</td>
<td>yes</td>
<td>small</td>
<td></td>
</tr>
<tr>
<td>Barinel &gt; Ochiai</td>
<td>no</td>
<td>small</td>
<td></td>
</tr>
<tr>
<td>Barinel &gt; Tarantula</td>
<td>yes</td>
<td>negligible</td>
<td></td>
</tr>
<tr>
<td>Op2 &gt; Ochiai</td>
<td>yes</td>
<td>negligible</td>
<td></td>
</tr>
<tr>
<td>Op2 &gt; Tarantula</td>
<td>yes</td>
<td>small</td>
<td></td>
</tr>
<tr>
<td>DStar &gt; Ochiai</td>
<td>yes</td>
<td>negligible</td>
<td></td>
</tr>
<tr>
<td>DStar &gt; Tarantula</td>
<td>yes</td>
<td>small</td>
<td></td>
</tr>
<tr>
<td>Metallaxis &gt; Ochiai</td>
<td>yes</td>
<td>negligible</td>
<td></td>
</tr>
<tr>
<td>MUSE &gt; Op2</td>
<td>no</td>
<td>negligible</td>
<td></td>
</tr>
<tr>
<td>MUSE &gt; Tarantula</td>
<td>no</td>
<td>negligible</td>
<td></td>
</tr>
</tbody>
</table>

Results agree with most prior studies on artificial faults but only 3 effect sizes are not negligible.
### Our results on real faults

<table>
<thead>
<tr>
<th>Prior studies (winner &gt; loser)</th>
<th>Ours (artificial faults)</th>
<th>Ours (real faults)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Replicated</td>
<td>Effect</td>
</tr>
<tr>
<td><strong>SBFL vs. SBFL</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ochiai &gt; Tarantula</td>
<td>yes</td>
<td>small</td>
</tr>
<tr>
<td>Barinel &gt; Ochiai</td>
<td>no</td>
<td>small</td>
</tr>
<tr>
<td>Barinel &gt; Tarantula</td>
<td>yes</td>
<td>negligible</td>
</tr>
<tr>
<td>Op2 &gt; Ochiai</td>
<td>yes</td>
<td>negligible</td>
</tr>
<tr>
<td>Op2 &gt; Tarantula</td>
<td>yes</td>
<td>small</td>
</tr>
<tr>
<td>DStar &gt; Ochiai</td>
<td>yes</td>
<td>negligible</td>
</tr>
<tr>
<td>DStar &gt; Tarantula</td>
<td>yes</td>
<td>small</td>
</tr>
<tr>
<td><strong>MBFL vs. SBFL</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metallaxis &gt; Ochiai</td>
<td>yes</td>
<td>negligible</td>
</tr>
<tr>
<td>MUSE &gt; Op2</td>
<td>no</td>
<td>negligible</td>
</tr>
<tr>
<td>MUSE &gt; Tarantula</td>
<td>no</td>
<td>negligible</td>
</tr>
</tbody>
</table>

**Results disagree** with all prior studies on real faults.
Results on artificial vs. real faults

<table>
<thead>
<tr>
<th>Artificial faults</th>
<th>Real faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metallaxis (MBFL)</td>
<td>Ochiai</td>
</tr>
<tr>
<td>MUSE (MBFL)</td>
<td>Barinel</td>
</tr>
<tr>
<td></td>
<td>D*</td>
</tr>
<tr>
<td></td>
<td>Op2</td>
</tr>
<tr>
<td></td>
<td>Tarantula</td>
</tr>
</tbody>
</table>
All SBFL techniques are equally good

<table>
<thead>
<tr>
<th>Artificial faults</th>
<th>Real faults</th>
</tr>
</thead>
</table>

For SBFL, results on artificial faults do not predict results on real faults!
MBFL is only better than SBFL on artificial faults

For MBFL, results on artificial faults do not predict results on real faults!
Why these differences?

- MBFL does exceptionally well on “reversible” faults

<table>
<thead>
<tr>
<th>Correct</th>
<th>Faulty</th>
<th>MBFL mutates</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum / n</td>
<td>sum * n</td>
<td>sum + n</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sum / n</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sum - n</td>
</tr>
</tbody>
</table>
Why these differences?

- MBFL does exceptionally well on “reversible” faults
- Most real faults are not reversible
Why these differences?

- MBFL does exceptionally well on “reversible” faults
- Most real faults are not reversible
- Real faults often involve unmutable statements (e.g., break, continue, return)
Why these differences?

- MBFL does exceptionally well on “reversible” faults
- Most real faults are not reversible
- Real faults often involve unmutatable statements

**MBFL has pinpoint accuracy on artificial faults but poor performance on real faults.**
What design decisions matter on real faults?

Defined and explored a design space for SBFL and MBFL

- 4 design factors (e.g., formula)
What design decisions matter on real faults?

Defined and explored a design space for SBFL and MBFL

- 4 design factors (e.g., formula)
- 156 FL techniques
What design decisions matter on real faults?

Defined and explored a design space for SBFL and MBFL

- 4 design factors (e.g., formula)
- 156 FL techniques

Results

- Most design decisions don’t matter (in particular for SBFL)
- Definition of test-mutant interaction matters for MBFL
What design decisions matter on real faults?

Defined and explored a design space for SBFL and MBFL

- 4 design factors (e.g., formula)
- 156 FL techniques

Results

- Most design decisions don’t matter (in particular for SBFL)
- Definition of test-mutant interaction matters for MBFL
- Barinel, D*, Ochiai, and Tarantula are indistinguishable

Existing **SBFL techniques** perform **best**. **No breakthroughs** in the **MBFL/SBFL design space**.
How to improve fault localization?

Artificial faults

Real faults

better

better
How to improve fault localization?

Explored two options:
1. Make MBFL great again
2. Hybrid: Stronger together
How to improve fault localization?

<table>
<thead>
<tr>
<th>Artificial faults</th>
<th>Real faults</th>
</tr>
</thead>
</table>

Explored two options:
1. **Make MBFL great again**
2. **Hybrid: Stronger together**

**Hybrid technique is significantly better than all techniques in the MBFL/SBFL design space (small effect size).**
Only top-ranked results matter

- Top-10 useful for practitioners\(^1\).
- Top-200 useful for automated program repair\(^2\).

<table>
<thead>
<tr>
<th>Technique</th>
<th>Top-5</th>
<th>Top-10</th>
<th>Top-200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hybrid</td>
<td>36%</td>
<td>45%</td>
<td>85%</td>
</tr>
<tr>
<td>DStar ((best \ SBFL))</td>
<td>30%</td>
<td>39%</td>
<td>82%</td>
</tr>
<tr>
<td>Metallaxis ((best \ MBFL))</td>
<td>29%</td>
<td>39%</td>
<td>77%</td>
</tr>
</tbody>
</table>

\(^1\) Kochhar et al., *Practitioners’ Expectations on Automated Fault Localization*, ISSTA’16
\(^2\) Long and Rinard, *An analysis of the search spaces for generate and validate patch generation systems*, ICSE’16
Evaluating and improving fault localization

FL performance on artificial faults is not predictive for real faults.
- MBFL only better on artificial faults
- All SBFL techniques are equally good

Evaluating and improving fault localization

**FL performance on artificial faults is not predictive for real faults.**
- MBFL only better on artificial faults
- All SBFL techniques are equally good

**MBFL/SBFL design space exploration**
- Most design decisions don’t matter
- Existing SBFL techniques perform best
- No breakthroughs in the design space
  → *FL needs to employ more information*

Evaluating and improving fault localization

FL performance on artificial faults is not predictive for real faults.
- MBFL only better on artificial faults
- All SBFL techniques are equally good

MBFL/SBFL design space exploration
- Most design decisions don’t matter
- Existing SBFL techniques perform best
- No breakthroughs in the design space
  → FL needs to employ more information

A new hybrid FL technique
- Combines MBFL and SBFL techniques
- Outperforms all existing FL techniques