26: Statistical fault localization
## Logistics

<table>
<thead>
<tr>
<th>Date</th>
<th>Activity</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>05/24</td>
<td>L: Fault Location</td>
<td></td>
</tr>
<tr>
<td>05/25</td>
<td>P: R3</td>
<td></td>
</tr>
<tr>
<td>05/26</td>
<td>LX: Fault Location</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>WEEK</strong> 10</td>
<td></td>
</tr>
<tr>
<td>05/29</td>
<td>H: MEM-DAY</td>
<td></td>
</tr>
<tr>
<td>05/30</td>
<td>T: R3</td>
<td></td>
</tr>
<tr>
<td>05/31</td>
<td>L: Program Analysis</td>
<td></td>
</tr>
<tr>
<td>06/01</td>
<td>P: IR</td>
<td></td>
</tr>
<tr>
<td>06/02</td>
<td>LX: PA (extra-cred)</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>WEEK</strong> 11</td>
<td></td>
</tr>
<tr>
<td>06/06</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Final Release (R3)**
- **DUE: R3!!!**
- **Individual Reflexion (IR)**
- **DUE: IR!!!**
Effective debugging
Software testing vs. software debugging

Testing: is there a bug?

```java
public void testAvg() {
    double nums = new double[]{1.0, 2.0, 3.0};
    double actual = Math.avg(nums);
    double expected = 2.0;
    assertEquals(expected, actual, EPS);
}
```

testAvg failed: 2.0 != 18.0

Starting point: a failing (bug-triggering) test.
Software testing vs. software debugging

Testing: is there a bug?
@Test
public void testAvg() {
  double[] nums = {1.0, 2.0, 3.0};
  double actual = Math.avg(nums);
  double expected = 2.0;
  assertEquals(expected, actual, EPS);
}

testAvg failed: 2.0 != 18.0

Debugging: where is the bug?
how to fix the bug?
Software testing vs. software debugging

Testing: is there a bug?

```java
@Test
public void testAvg() {
    double nums = new double[]{1.0, 2.0, 3.0};
    double actual = Math.avg(nums);
    double expected = 2.0;
    assertEquals(expected, actual, EPS);
}
```

testAvg failed: 2.0 != 18.0

Debugging: where is the bug? how to fix the bug?

What testing practices support effective debugging?
Testing best practices

- Naming: proper names for tests (clear link to tested class/method)
- Output: meaningful failure messages
- Atomicity: one test per behavior
- Style: one test, one assertion vs. one test, multiple assertions
Statistical fault localization
What is statistical fault localization?

**Program**

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

**Test suite**

- **Passing tests**
- **Failing tests**
What is statistical fault localization?

Program

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

Fault localization technique

Test suite

- Passing tests
- Failing tests

Where is the bug?
What is statistical fault localization?

**Program**

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

**Test suite**

- **Passing tests**: ✔
- **Failing tests**: ✗

**Statement ranking**

```java
double avg(double[] nums) {
    double sum = 0;
    int n = nums.length;
    for(int i=0; i<n; ++i) {
        sum -= nums[i];
    }
    return sum / n;
}
```
What is statistical fault localization?

Program

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

Test suite

- Passing tests
- Failing tests

Statement ranking

- Most suspicious
- Least suspicious

What are the key “ingredients” for suspiciousness?
Statistical fault localization: how it works

Program

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

Program

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

- Run all tests
  - t1 passes
Statistical fault localization: how it works

**Program**

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

- Run all tests
  - t1 passes
  - t2 passes
Statistical fault localization: how it works

Program

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

- Run all tests
  - t1 passes
  - t2 passes
  - t3 passes
Statistical fault localization: how it works

Program

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

Program

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

- Run all tests
  - t1 passes
  - t2 passes
  - t3 passes
  - t4 fails
  - t5 fails

Which line(s) seem(s) most suspicious?
### Spectrum-based fault localization

**Program**

```java
double avg(double[] nums) {
    double sum = 0;
    int n = nums.length;
    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!
Spectrum-based fault localization

Program

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

Spectrum-based FL (SBFL)

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

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

Visualization: the key idea behind Tarantula.

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

Jones et al., *Visualization of test information to assist fault localization*, ICSE’02
Statistical fault localization: live example
Testing best practices revisited

- Naming: proper names for tests (clear link to tested class/method)
- Output: meaningful failure messages
- **Atomicity:** one test per behavior
- Style: one test, one assertion vs. one test, multiple assertions
Mutation-based fault localization

Program

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

Mutants

```java
double avg(double[] nums) {
    double sum = 0;
    int n = nums.length;
    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!
Common structure of SBFL and MBFL

For each element

\[ \lambda \]

Statements, expressions, mutants, etc.

<table>
<thead>
<tr>
<th>Elem</th>
<th>Susp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

collect

<table>
<thead>
<tr>
<th>Line#</th>
<th>Susp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>0.0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

sort

<table>
<thead>
<tr>
<th>Line#</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

weighting factors

Suspiciousness formula

(identity for SBFL)
Effectiveness of SBFL and MBFL

<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 \textit{(best SBFL)}</td>
<td>30%</td>
<td>39%</td>
<td>82%</td>
</tr>
<tr>
<td>Metallaxis \textit{(best MBFL)}</td>
<td>29%</td>
<td>39%</td>
<td>77%</td>
</tr>
</tbody>
</table>

Percentage of buggy statements found when inspecting the top-n suspicious statements.
Effectiveness of SBFL and MBFL

<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>

- Top-10 useful for practitioners\(^1\).
- Top-200 useful for automated patch generation\(^2\).

What assumptions underpin these results? Are they realistic?

\(^1\)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
Spectrum-based fault localization

Program

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

Visualization & Hotspots: the key benefit!

Jones et al., Visualization of test information to assist fault localization, ICSE’02
Automated patch generation
Automatic patch generation (program repair)

Generate-and-validate Approaches

What are the main components of a (generate-and-validate) patch generation approach?
Automatic patch generation (program repair)

Generate-and-validate Approaches

Main components:
- Fault localization
- Mutation + fitness evaluation
- Patch validation
Reminder: fault localization framework

Questions, please!