Visualization for Fault Localization

James A. Jones, Mary Jean Harrold, John T. Stasko
Georgia Institute of Technology

Supported by Boeing Commercial Airplane Group, National Science Foundation, and the Yamacraw Project
Motivation

- Improve software quality
- Reduce the number of delivered faults
- Locate faults after failures (debugging)
- Reduce the time and cost necessary to debug
Outline

• Approaches
  – Discrete
  – Continuous

• Tarantula
  – Demo

• Preliminary Evaluation
• Open Questions
Discrete Approach

- **Input**
  - Source code
  - For each test case
    - its pass/fail status
    - statements that it executes

- **Display statements in program according to the test cases that execute them**

 Statements executed by:
- Only failed test cases
- Both passed & failed test cases
- Only passed test cases
Example

```c
mid() {
    int x, y, z, m;
    1: read("Enter 3 numbers:", x, y, z);
    2: m = z;
    3: if (y<z)
        4: if (x<y)
            5: m = y;
        6: else if (x<z)
            7: m = y;
        8: else
            9: if (x>y)
                10: m = y;
            11: else if (x>z)
                12: m = x;
            13: print("Middle number is:", m);
    } }```

Test Cases

<table>
<thead>
<tr>
<th>Pass Status:</th>
<th>3,3,5</th>
<th>1,2,3</th>
<th>3,2,1</th>
<th>5,5,5</th>
<th>5,3,4</th>
<th>2,1,3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pass Status: P P P P P P F
Example

```c
mid() {
    int x, y, z, m;
    1: read(“Enter 3 numbers:”, x, y, z);
    2: m = z;
    3: if (y<z)
        4: if (x<y)
            5: m = y;
        6: else if (x<z)
            7: m = y;
        8: else
            9: if (x>y)
                10: m = y;
            11: else if (x>z)
                12: m = x;
    13: print(“Middle number is:”, m);
}
```
Continuous Approach

- Distribute statements executed by both passed and failed test cases over spectrum
- Indicate the relative success rate of each statement by its hue
Hue

• If a statement is executed by 10 failed test cases and 20 passed test cases, it would be colored mostly green.

• However, if the test suite contains 10 failed test cases and 200 passed test cases, it would be colored mostly red.

\[
hue(s) = \text{low hue (red)} + \frac{\%\text{passed}(s)}{\%\text{passed}(s) + \%\text{failed}(s)} \times \text{hue range}
\]

\[
\frac{10 \text{ failed test cases}}{10} = 100\%
\]

\[
m = y;
\]

\[
\frac{20 \text{ passed test cases}}{200} = 10 \%
\]
Hue

\[
\text{hue}(s) = \text{low hue (red)} + \frac{\%\text{passed}(s)}{\%\text{passed}(s) + \%\text{failed}(s)} \times \text{hue range}
\]

10 failed test cases = 100%
\[
\frac{10}{10}
\]
0 passed test cases = 0%
\[
\frac{0}{10}
\]

10 failed test cases = 100%
\[
\frac{10}{10}
\]
341 failed test cases = 10%
\[
\frac{341}{10200}
\]
100 passed test cases = 0%
\[
\frac{100}{10200}
\]

10 failed test cases = 100%
\[
\frac{10}{10}
\]
10 passed test cases = 100%
\[
\frac{10}{10}
\]
0 passed test cases = 0%
\[
\frac{0}{10}
\]

1 failed test cases = 10%
\[
\frac{1}{10}
\]
0 passed test cases = 0%
\[
\frac{0}{10}
\]

\[
\text{a = b;}
\]
\[
\text{m = y;}
\]
\[
\text{c = d;}
\]
Brightness

10 failed test cases = 100%
\[
\frac{10}{10}
\]
0 passed test cases = 0%
\[
\frac{0}{10}
\]

\[m = y;\]

1 failed test cases = 10%
\[
\frac{1}{10}
\]
0 passed test cases = 0%
\[
\frac{0}{10}
\]

\[m = y;\]

10 failed test cases = 100%
\[
\frac{10}{10}
\]

20 passed test cases = 10%
\[
\frac{20}{200}
\]
Brightness

- Using total percentage of test cases that execute a statement may cause important statements to be overlooked.
- Instead we use the higher of the two independent percentages.

\[
\text{bright}(s) = \max(\%\text{ passed}(s), \%\text{ failed}(s))
\]
Example

```c
mid() {
    int x, y, z, m;
    1: read("Enter 3 numbers:", x, y, z);
    2: m = z;
    3: if (y<z)
        4: if (x<y)
            5: m = y;
        6: else if (x<z)
            7: m = y;
        8: else
            9: if (x>y)
                10: m = y;
            11: else if (x>z)
                12: m = x;
    13: print("Middle number is:", m);
}
```

Test Cases

<table>
<thead>
<tr>
<th></th>
<th>3,3,5</th>
<th>1,2,3</th>
<th>3,2,1</th>
<th>5,5,5</th>
<th>5,3,4</th>
<th>2,1,3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass Status:</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>F</td>
</tr>
</tbody>
</table>
Scalability

- Large programs difficult to display
- Use the line-of-pixels, SeeSoft, view
- Each character in the source is displayed as a pixel

```c
mid() {
  int x, y, z, m;
  read(“Enter 3 numbers:”, x, y, z);
  m = z;
  if (y < z)
    if (x < y)
      m = y;
    else if (x < z)
      m = y;
  else
    if (x > y)
      m = y;
    else if (x > z)
      m = x;
  print(“Middle number is:”, m);
}
```

[Eick, Steffen, Sumner, TSE 1992]
Preliminary Evaluation

- Two preliminary studies
  - How red are the faulty statements?
  - How red are the non-faulty statements?
- Subject program: Space
  - 8000 lines of executable code
  - 1000 coverage-based test suites of size 156-4700 test cases
  - 20 faulty versions
Open Questions

- Are the faults typically colored red?
- If there are faults that are not colored red, why and what are they?
- Are non-faulty red statements near the fault?
- What other views and analyses would be useful?
- What is the maximum practical number of faults for which this technique works?