Visually Encoding Program Test Information to Find Faults in Software

James Eagan, Mary Jean Harrold, James A. Jones, John T. Stasko

College of Computing / GVU Center
Georgia Institute of Technology

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

- Software creation & maintenance
- Locate faults after failures (debugging)
- Reduce the time and cost necessary to debug
Particular Focus

- Testing of large software systems
- Large suites of tests
  - Able to characterize whether a program
    - Passes on a test (Execution is judged correct)
    - Fails on a test (Execution is judged incorrect)
Test Data

- Execution summary on test suite
  - For each test case
    - Its pass/fail status
    - Statements that it executes

<table>
<thead>
<tr>
<th>Test #</th>
<th>Status</th>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P</td>
<td>12 13 14 24 25 27 28 ...</td>
</tr>
<tr>
<td>2</td>
<td>P</td>
<td>12 13 14 15 16 ...</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>12 13 124 125 126 ...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Problem

• Analyzing one failed test (traditional)
  – Just one data point
  – Neglects context of all other tests

• Analyzing whole suite
  – Huge data set
  – Locating bug by analyzing this textual summary is at best tedious and difficult
  – Desire a better way of representing the summary of the test suite execution
One Approach

- Visualize the execution of all tests
- 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>P</th>
<th>P</th>
<th>P</th>
<th>P</th>
<th>P</th>
<th>P</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,3,5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,2,3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3,2,1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5,5,5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5,3,4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2,1,3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

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);
}

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>1</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>2</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>3</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>4</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>•</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td>•</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</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
Our Approach

- Distribute statements executed by both passed and failed test cases over spectrum
- Use hue and brightness to communicate status more clearly

Discrete Approach:
- Only failed test cases
- Both passed & failed test cases
- Only passed test cases

Continuous Approach:
Hue

- Compare ratios of passed and failed tests through statement
- Slide toward higher percentage

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

- Encodes ratio of passed or failed (whichever is higher) through statement
- Higher percentage makes statement brighter, lower makes it darker

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

<table>
<thead>
<tr>
<th>Test Cases</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>
</tbody>
</table>

Pass Status: P P P P P P F
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]
Tarantula
Initial Evaluation

• Two questions to ask
  – How red are the faulty statements?
  – How red are the non-faulty statements?

• Preliminary tests on one system appear promising
  – Faults are typically red
  – Non-faults aren’t red very much
  – Submitted ICSE paper
Future Work

- Further evaluation
- Examining perception of colors better
- Understanding bug-finding process
  - What other views and analyses would be useful?
  - How to incorporate system in larger software engineering context