Automation to Assist Software Debugging

James A. Jones
College of Computing
Georgia Institute of Technology

Supported by Tata Consultancy Services, NSF, NASA
Boeing Commercial Airplanes
Debugging Phases

• Make changes to the program
• Test
• Observe failure
• Locate bugs
• Identify possible fixes
• Choose best fix
• Implement fix
mid() {
    int x, y, z, m;
    1: read("Enter 3 integers:");
    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);
}
Tarantula

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; // bug  
        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>Test Case</th>
<th>Pass Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,3,5</td>
<td>P</td>
</tr>
<tr>
<td>1,2,3</td>
<td>P</td>
</tr>
<tr>
<td>3,2,1</td>
<td>P</td>
</tr>
<tr>
<td>5,5,5</td>
<td>P</td>
</tr>
<tr>
<td>5,3,4</td>
<td>P</td>
</tr>
<tr>
<td>2,1,3</td>
<td>F</td>
</tr>
</tbody>
</table>

Coloring statements

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; // bug  
        8: else  
            9: if (x>y)  
                10: m = y;  
            11: else if (x>z)  
                12: m = x;  
    13: print("Middle number is:", m);
}
Tarantula

Results

Tarantula on Space
(~6000 LOC)

% of program to be examined to find bug

% of bugs
Overview

Tarantula: Fault Localization

```c
mid() {
    int x,y,z,m;
    1: read("Enter 3 integers: ",x,y,z);
    2: m = z;
    3: if (y<z) { y = z; }
    4:   if (x<y) { m = y; }
    5:   else if (x<z) { m = y; }
    6:     else if (x<y) { m = y; }
    7:   else if (x>y) { m = z; }
    8:     else if (x>z) { m = x; }
    9:     else if (x<y) { m = y; }
    10:    else if (x>y) { m = z; } // bug; correct m=y
    11:    else if (x>z) { m = x; }
    12:    else if (x>y) { m = z; }
    13: print("Middle number is: ", m);
}
Pass/fail Status
P P P P P P F F F F
```
Tarantula: Fault Localization

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

```
3,3,5
1,2,3
3,2,2
5,5,5
1,1,4
5,3,4
```

Pass/fail Status: P P P P P P F P F

Tarantula: Fault Localization

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

```
3,2,1
2,1,3
5,4,2
5,2,6
```

Pass/fail Status: P P P P P P F P F
Tarantula: Fault Localization

```c
mid() {
    int x, y, z, m;
    1: read("Enter 3 integers:", x, y, z);
    2: m = z;
    3: if (y<z) //fixed
        4:   if (x<y) //fixed
            5:       m = y;
        6:   else if (x<z)
            7:       m = x;
    8: else //fixed
        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>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
<th>t7</th>
<th>t8</th>
<th>t9</th>
<th>t10</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Pass/fail Status

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
</tbody>
</table>

Debugging Process

- Are all failing tests caused by the same fault?
- Can we associate groups of tests with different faults?
- Can we search for different faults simultaneously?
Tarantula: Fault Localization

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

Pass/fail Status

<table>
<thead>
<tr>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
<th>t7</th>
<th>t8</th>
<th>t9</th>
<th>t10</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,3,5</td>
<td>3,2,2</td>
<td>5,5,5</td>
<td>5,2,4</td>
<td>5,1,3</td>
<td>5,4,2</td>
<td>5,2,6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Debugging Process

Sequential Debugging:
- Execute
- Tests fail
- Debug
- Repeat

Parallel Debugging:
- Execute
- Tests fail
- Debug
- Parallel Debugging
- Repeat
- All tests pass

P' is failure-free
P'' is failure-free
Debugging Process

Crucial problem:
- Partitioning failed tests into groups of similar behavior—focus on different faults
- fault-focusing clusters

Potential benefits:
- Reduced time to failure-free program
- Better utilization of developer effort
- Fewer tests executed
- Early indication of the number of faults
- Less "noise" in locating each fault

Crucial problem:
- Partitioning failed tests into groups of similar behavior—focus on different faults
- fault-focusing clusters

Parallel Debugging

Fault-focusing Clusters—Overview

Test Cases

Fault-focusing clusters:
- Clusters of failing test cases
- Clusters failing in similar way
- Each cluster targeting a different fault
Specialized Test Suites

**Specialized test suites:**
Fault-focusing clusters combined with passing test cases

Give each specialized test suite to different developer (debugger)

Visualization of Each Result

```
mid() {
  int x, y, z, m;
  read("Enter 3 integers:");
  m = z;
  if (y<z) {
    if (x<y) {
      m = y;
    } else if (x<z) {
      m = y;
    } else {
      if (x>y) {
        m = z;
      } else if (x>z) {
        m = x;
      }
      print("Middle number is:"...)
    }
  }
}
```

```
mid() {
  int x, y, z, m;
  read("Enter 3 integers:");
  m = z;
  if (y<z) {
    if (x<y) {
      m = y;
    } else if (x<z) {
      m = y;
    } else {
      if (x>y) {
        m = z;
      } else if (x>z) {
        m = x;
      }
      print("Middle number is:"...)
    }
  }
}
```
Fault-focusing Clusters

Failed test cases execution information → Execution Clustering

Specialized test suites → Fault Localization

Fault location prediction

- Clustering Technique 1: Less aggressive
- Clustering Technique 2: More aggressive

Summary of Results

- Parallel less expensive even if debugging sequential
- CT1 if time to delivery is most important (low salaries, high urgency)
- CT2 if developer cost is most important (high salaries, low urgency)
Conclusion

• Technique and visualization to provide support for finding faults
• New approach to debugging—parallelized debugging
• Technique to get fault-focused clusters to create specialized test suites
• Benefits, supported by studies
  • less time to failure-free program
  • better utilization of developer effort
  • less running of the test suite
  • indication of the number of faults
  • less “noise” in locating each fault

Tarantula: Fault Localization

• Jones, Harrold, Stasko, Workshop on Software Visualization 2001
• Jones, Harrold, Stasko, International Conference on Software Engineering 2002
• Jones, Orso, Harrold, Software Visualization 2003
• Jones, ICSE 2004 Doctoral Workshop
• Jones, Orso, Harrold, Journal of Information Visualization 2004
• Jones, Harrold, Automated Software Engineering 2005