Improved Debugging Using Automatic Fault-localization Techniques

Mary Jean Harrold
ADVANCE Professor of Computing
Georgia Institute of Technology

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

My Research Group

Postdoc
• Carsten Görg

Graduate students
• Ahmed Ghiduk
• Saswat Anand
• Term Apiwattanapong
• George Baah
• Jim Jones
• Irina Kishiniovski
• Paul Li
• Raul Santelices
• Hina Shah
• Yanbing Yu

Research Scientists
• Rashmitha Bupathi
• Pavan Kumar Chittimali

Aristotle Research Group
Software bugs are costing the U.S. economy an estimated $59.5 billion each year. Improvements in testing and debugging could reduce this cost by about a third, or $22.5 billion.

(from NIST Estimated Planning Report 02-3)
Fault Localization

Usage scenarios
Nightly-build process
• Run set tests (regression, breadth) each night
• Report pass/fail status
Test-driven development
Create and run tests (regression, breadth) after changes
• Report pass/fail status
Regression testing
• Run regression tests after changes
• Report pass/fail status

Outline of Talk

• Introduction
• Fault localization—Tarantula
  • General technique
  • Visualization and demo
  • Test-case clustering to improve efficiency
• Related work
• Conclusion
General Technique—Tarantula

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)
            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);
}
**General Technique—Tarantula**

```plaintext
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)
            5: m = y;
        6: else if (x < z)
            7: m = y;
    8: else
        9: if (x > y)
            10: m = z;
        11: else if (x > z)
            12: m = x;
    13: print("Middle number is: ", m);
}
```

<table>
<thead>
<tr>
<th></th>
<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>1</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
</tr>
<tr>
<td>2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>3</td>
<td>3.6</td>
<td>3.6</td>
<td>3.6</td>
<td>3.6</td>
<td>3.6</td>
<td>3.6</td>
<td>3.6</td>
<td>3.6</td>
<td>3.6</td>
<td>3.6</td>
</tr>
<tr>
<td>4</td>
<td>1.4</td>
<td>1.4</td>
<td>1.4</td>
<td>1.4</td>
<td>1.4</td>
<td>1.4</td>
<td>1.4</td>
<td>1.4</td>
<td>1.4</td>
<td>1.4</td>
</tr>
<tr>
<td>5</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
</tr>
<tr>
<td>6</td>
<td>2.1</td>
<td>2.1</td>
<td>2.1</td>
<td>2.1</td>
<td>2.1</td>
<td>2.1</td>
<td>2.1</td>
<td>2.1</td>
<td>2.1</td>
<td>2.1</td>
</tr>
<tr>
<td>7</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
</tr>
<tr>
<td>8</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
</tr>
<tr>
<td>9</td>
<td>5.4</td>
<td>5.4</td>
<td>5.4</td>
<td>5.4</td>
<td>5.4</td>
<td>5.4</td>
<td>5.4</td>
<td>5.4</td>
<td>5.4</td>
<td>5.4</td>
</tr>
<tr>
<td>10</td>
<td>5.2</td>
<td>5.2</td>
<td>5.2</td>
<td>5.2</td>
<td>5.2</td>
<td>5.2</td>
<td>5.2</td>
<td>5.2</td>
<td>5.2</td>
<td>5.2</td>
</tr>
<tr>
<td>11</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
</tr>
<tr>
<td>12</td>
<td>5.3</td>
<td>5.3</td>
<td>5.3</td>
<td>5.3</td>
<td>5.3</td>
<td>5.3</td>
<td>5.3</td>
<td>5.3</td>
<td>5.3</td>
<td>5.3</td>
</tr>
<tr>
<td>13</td>
<td>5.5</td>
<td>5.5</td>
<td>5.5</td>
<td>5.5</td>
<td>5.5</td>
<td>5.5</td>
<td>5.5</td>
<td>5.5</td>
<td>5.5</td>
<td>5.5</td>
</tr>
</tbody>
</table>

**Pass/fail Status**

- **P** - Passed
- **F** - Failed
- **P** - Passed
- **P** - Passed
- **P** - Passed
- **F** - Failed
- **F** - Failed
- **F** - Failed
- **F** - Failed

**Suspiciousness**

- **P** - Passed
- **P** - Passed
- **P** - Passed
- **P** - Passed
- **P** - Passed
- **F** - Failed
- **F** - Failed
- **F** - Failed
- **F** - Failed

**MASPLAS 2007**
General Technique—Tarantula

\[
suspiciousness(s) = \frac{\text{failed}(s)}{\text{total failed}} + \frac{\text{passed}(s)}{\text{total passed}}
\]

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)
            5: m = y;
        6: else if (x < z)
            7: m = y;
    8: else
        9:   if (x > y)
            10: m = z;
        11: else if (x > z)
            12: m = x;
    13: print("Middle number is: ", m);
}

Pass/fail Status

MASPLAS 2007
General Technique—Tarantula

```c
mid() {
    int x,y,z,m;
    1:read("Enter 3 integers: ",x,y,z);  * * * * * * * * * * 0.50 5
    2:m = z;                             * * * * * * * * * * 0.50 5
    3:if (y<z)                           * * * * * * * * * * 0.50 5
        4: if (x<y)                     * * * * * * * * 0.43 10
            5: m = y;                 * 0.00 11
            6: else if (x<z)         * * * * 0.60 2
            7: m = y;                 * * * * 0.60 2
        8:else
            9: if (x>y)               * * * * 0.60 2
            10: m = z;                * * * * 0.75 1
            11: else if (x>z)        * 0.00 11
            12: m = x;                * 0.00 11
    13:print("Middle number is ", m);  * * * * * * * * * * 0.50 5
    }
    Pass/fail Status P P P P F F F F F
    suspiciousness 0.50
    rank 5
}
```

General Technique—Tarautula

```c
mid() {
    int x,y,z,m;
    suspiciousness(s) = \frac{\text{failed}(s)}{\text{total failed}} \cdot \frac{\text{passed}(s)}{\text{total passed}} + \frac{\text{failed}(s)}{\text{total failed}} 
    1:read("Enter 3 integers: ",x,y,z);  * * * * * * * * * * 0.50 5
    2:m = z;                             * * * * * * * * * * 0.50 5
    3:if (y<z)                           * * * * * * * * * * 0.50 5
        4: if (x<y)                     * * * * * * * * 0.43 10
            5: m = y;                 * 0.00 11
            6: else if (x<z)         * * * * 0.60 2
            7: m = y;                 * * * * 0.60 2
        8:else
            9: if (x>y)               * * * * 0.60 2
            10: m = z;                * * * * 0.75 1
            11: else if (x>z)        * 0.00 11
            12: m = x;                * 0.00 11
    13:print("Middle number is ", m);  * * * * * * * * * * 0.50 5
    }
    Pass/fail Status P P P P F F F F F
    suspiciousness 0.50
    rank 5
}
```
General Technique—Tarantula

\[ \text{suspiciousness}(s) = \frac{\text{failed}(s)}{\text{total\ failed}} + \frac{\text{failed}(s)}{\text{total\ passed}} \]

mid() {
    int x, y, z, m;
    read("Enter 3 integers:", 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);
}
General Technique—Tarantula

\[
\text{suspiciousness}(s) = \frac{\text{failed}(s)}{\text{total passed} + \text{failed}(s)}
\]

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

<table>
<thead>
<tr>
<th>Measure</th>
<th>Percentage of program to be examined to find fault</th>
</tr>
</thead>
</table>

### Subjects

#### Siemens Suite

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Faulty Versions (single fault)</th>
<th>Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print_tokens</td>
<td>472</td>
<td>7</td>
<td>4056</td>
</tr>
<tr>
<td>Print_tokens_2</td>
<td>399</td>
<td>10</td>
<td>4071</td>
</tr>
<tr>
<td>Replace</td>
<td>512</td>
<td>32</td>
<td>5542</td>
</tr>
<tr>
<td>Schedule</td>
<td>292</td>
<td>9</td>
<td>2650</td>
</tr>
<tr>
<td>Schedule_2</td>
<td>301</td>
<td>10</td>
<td>2680</td>
</tr>
<tr>
<td>Tcas</td>
<td>141</td>
<td>41</td>
<td>1578</td>
</tr>
<tr>
<td>Tot_info</td>
<td>440</td>
<td>23</td>
<td>1054</td>
</tr>
<tr>
<td>Space</td>
<td>6000</td>
<td>33</td>
<td>13585</td>
</tr>
</tbody>
</table>
Empirical Study

Method

- For each program and test suite, compute suspiciousness of each statement using Tarantula
- Compute percentage of program examined to find fault, using suspiciousness to order search
- Use results of published studies on same subjects

Techniques compared

- Tarantula [Jones, Harrold, Stasko, ICSE02, ASE05]
- Set-based, Nearest Neighbor [Renieris, Reiss, ASE03]
- Cause Transitions [Cleve, Zeller, ICSE05]
- Statistical [Liblit et al., PLDI05]

General Technique—Tarantula

\[
\text{suspiciousness}(s) = \frac{\text{failed}(s)}{\text{total failed}} + \frac{\text{passed}(s)}{\text{total passed}}
\]

```
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)
            5:      m = y;
        6:   else if (x < z)
            7:      m = y;
    8: else
        9:   if (x > y)
            10:      m = z;
        11:   else if (x > z)
            12:      m = x;
    13: print("Middle number is: ", m);
}
```

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

MASPLAS 2007
Reporting Technique

% of program to be examined to find fault

% of faulty versions

Ideal technique

Worst technique

Results on Siemens

% of program to be examined to find fault

% of faulty versions

Set based
Nearest Neighbor (1)
Nearest Neighbor (2)
Cause Transitions
Statistical
Tarantula

MASPLAS 2007
Threats to Validity

- Generalization

![](image)

- Single fault (more later)

Outline of Talk

- Introduction
- Fault localization—Tarantula
  - General technique
  - Visualization and demo
    - Test-case clustering to improve efficiency
- Related work
- Conclusion
Visualization

For statement $s$:

Hue (color)
summarizes pass/fail results of test
cases that executed $s$

Coloring Statements

```
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)
          5: m = y;
       6: else if (x<z)
          7: m = y;
       else
          9: if (x>y)
             10: m = z;
            else if (x>z)
               12: m = x;
            print("Middle number is:", m);
            return;
}
```
File-level View

SeeSoft view

• each pixel represents a character in the source

```cpp
mid() {
    int x, y, z, m;
    read("Enter 3 integers:", x, y, z);
    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:", m);
}
```
System-level View

TreeMap view
• each node
  • represents a file
  • is divided into blocks representing color of statements

MASPLAS 2007

Tarantula

MASPLAS 2007
Outline of Talk

- Introduction
- Automatic fault localization—Tarantula
  - General technique
  - Visualization
  - Test-case clustering to improve efficiency
- Related work
- Conclusion

Improving Fault-localization Efficiency

- Are all failing tests caused by the same fault?
- Can we associate groups of tests with different faults?
- Can we reduce debugging effort by considering these groups individually?
- Can we reduce debugging effort by considering these groups simultaneously?
Improving Fault-localization Efficiency

```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)
            5:     m = y;
        6:   else if (x < z)
            7:     m = y;
    8: else
        9:   if (x > y)
            10:     m = z;
        11:   else if (x > z)
            12:     m = x;
    13: print("Middle number is:", m);
}
```

Pass/fail Status

MASPLAS 2007
Debugging Process

P → Execute failed tests → Debug → P'' → Execute failed tests → Debug → Execute → all tests pass

P → Execute some failed tests → Debug → P' → Execute some failed tests → Debug → Execute → all tests pass

P → Execute → Debug → P' → Execute → Debug → P'' → Execute → all tests pass

P is failure-free

P' is failure-free

Some failed tests

Some failed tests

Some failed tests

P is failure-free

P' is failure-free

MASPLAS 2007
Debugging Process

Potential costs
- Overhead to partition test cases
- Multiple debuggers (developers)

Potential benefits
- Reduced time to failure-free program
- Less “noise” in locating each fault
- Better utilization of developer effort

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

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

• Find faults one at a time using specialized test suites

• Find faults at the same time (in parallel) using specialized test suites
Fault-focusing Clusters

Clustering by behavior models
- Dynamic information
  - profiles (branch, method-method, ...)
  - only failed tests
- Statistical analysis, machine learning
  - generate models for each execution
  - cluster models
- Fault-localization for stopping point
Most difficult problem of clustering is determining a good stopping criterion?

Clustering Behavior Models

• Models: discrete-time Markov chains (DTMCs) from profiles (branch, method,...)
• Clustering: iterative with two most similar according to $Sim_1$

$Sim_1$: sum of absolute difference between matching transitions in DTMCs being compared
Fault Localization for Stopping Point

\[ Sim_2 = \frac{|A \cap B|}{|A \cup B|} \]
Fault Localization for Stopping Point

\[ Sim2 = \frac{|A \cap B|}{|A \cup B|} \]

MASPLAS 2007

Tarantula: Fault Localization

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

MASPLAS 2007
Fault Localization for Stopping Point

\[ Sim2 = \frac{|A \cap B|}{|A \cup B|} \]

Fault-focusing Cluster 1

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

Pass/fail Status: P P P P P P F F F
Fault Localization for Stopping Point

\[ \text{Sim}^2 = \frac{|A \cap B|}{|A \cup B|} \]

\[ \begin{align*}
\text{rank } & \quad 10 \quad 9 \quad 8 \quad 7 \\
\text{t07-09-08-10} & \quad \text{t07-09} & \quad \text{t08-10} & \quad \text{t07-09-08-10}
\end{align*} \]

MASPLAS 2007
Fault-focusing Cluster 2

```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)
       5:      m = y;
       6:   else if (x<z)
       7:      m = y;
    8:else
       9:   if (x>y)
      10:      m = z;
      11:   else if (x>z)
      12:      m = x;
    13:print(“Middle number is:”, m);
}
```

Pass/fail Status

```
P P P P P | F
```

Fault Localization for Stopping Point

```
Sim2 = \frac{|A \cap B|}{|A \cup B|}
```

Fault-focusing Cluster 2
Fault Localization for Stopping Point

\[ Sim2 = \frac{|A \cap B|}{|A \cup B|} \]

t07

t08

t09

t10

\[ Sim2 = \frac{|A \cap B|}{|A \cup B|} \]

t07

t08

t09

t10

Rank

\begin{array}{cccc}
7 & 6 & 4 & 1 \\
01 & 02 & 03 & 04 \\
05 & 06 & 07 & 08 \\
\end{array}

Rank

\begin{array}{cccc}
7 & 6 & 4 & 1 \\
01 & 02 & 03 & 04 \\
05 & 06 & 07 & 08 \\
\end{array}

MASPLAS 2007
Fault Localization for Stopping Point

\[ Sim2 = \frac{|A \cap B|}{|A \cup B|} \]

\[ Sim2 = \frac{4}{4} \]

MASPLAS 2007
• Composite is similar (above threshold) to both of its constituents so clustering stops at this level
• Result is two clusters: \{t07, t09\}, \{t08, t10\}

MASPLAS 2007
Fault-focusing Cluster 1

```
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)
            5: m = y;
        6:   else if (x<z)
            7: m = y;
    8:else
        9:   if (x>y)
            10: m = z;
        11:   else if (x>z)
            12: m = x;
    13:print("Middle number is:", m);
}
```

Pass/fail Status

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>P</th>
<th>P</th>
<th>P</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
</table>

Fault-focusing Cluster 2

```
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)
            5: m = y;
        6:   else if (x<z)
            7: m = y;
    8:else
        9:   if (x>y)
            10: m = z;
        11:   else if (x>z)
            12: m = x;
    13:print("Middle number is:", m);
}
```

Pass/fail Status

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>P</th>
<th>P</th>
<th>P</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
</table>

MASPLAS 2007
Visualization of Specialized Test Suites

Cluster 1

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

Cluster 2

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

Metrics for Evaluation

**E:** Expense (developer effort): number of statements examined to find fault

**D:** Total developer effort (total person hours): sum of developer effort for each fault in the program

**FF:** Total effort to failure-free program (time to failure-free): sum of maximum developer effort at each iteration (critical path to failure-free)

**R:** rank of statement
Metrics for Evaluation

**E: Expense (developer effort):**
number of statements examined to find fault

**D: Total developer effort (total person hours):**
sum of developer effort for each fault in the program

**FF: Total effort to failure-free program (time to failure-free):**
sum of maximum developer effort at each iteration (critical path to failure-free)

Fault 1  Fault 2  Fault 3

MASPLAS 2007
Metrics for Evaluation

**E: Expense (developer effort):**
- Number of statements examined to find fault

**D: Total developer effort (total person hours):**
- Sum of developer effort for each fault in the program

**FF: Total effort to failure-free program (time to failure-free):**
- Sum of maximum developer effort at each iteration (critical path to failure-free)

<table>
<thead>
<tr>
<th>Fault</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault 1</td>
<td>1</td>
</tr>
<tr>
<td>Fault 2</td>
<td>2</td>
</tr>
<tr>
<td>Fault 3</td>
<td>3</td>
</tr>
<tr>
<td>Fault 4</td>
<td>4</td>
</tr>
<tr>
<td>Fault 5</td>
<td>5</td>
</tr>
<tr>
<td>Fault 6</td>
<td>6</td>
</tr>
</tbody>
</table>
## Metrics for Evaluation

### E: Expense (developer effort):
- Number of statements examined to find fault

### D: Total developer effort (total person hours):
- Sum of developer effort for each fault in the program

### FF: Total effort to failure-free program (time to failure-free):
- Sum of maximum developer effort at each iteration (critical path to failure-free)
### Metrics for Evaluation

**E: Expense (developer effort):**
number of statements examined to find fault

**D: Total developer effort (total person hours):** sum of developer effort for each fault in the program

\[ \sum_{i=1}^{\text{faults}} E_i \]

**FF: Total effort to failure-free program (time to failure-free):** sum of maximum developer effort at each iteration (critical path to failure-free)

<table>
<thead>
<tr>
<th>Fault</th>
<th>Statement Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault 1</td>
<td>1</td>
</tr>
<tr>
<td>Fault 2</td>
<td>2</td>
</tr>
<tr>
<td>Fault 3</td>
<td>3</td>
</tr>
<tr>
<td>Fault 4</td>
<td>4</td>
</tr>
<tr>
<td>Fault 5</td>
<td>5</td>
</tr>
<tr>
<td>Fault 6</td>
<td>6</td>
</tr>
</tbody>
</table>

MASPLAS 2007
Metrics for Evaluation

**E: Expense (developer effort):**
number of statements examined to find fault

**D: Total developer effort (total person hours):** sum of developer effort for each fault in the program

**FF: Total effort to failure-free program (time to failure-free):** sum of maximum developer effort at each iteration (critical path to failure-free)

\[ \sum_{i=1}^{[\text{faults}]} E_i \]

MASPLAS 2007
Empirical Study

Variables
- NSTS: Finding faults using non-specialized test suites
- STS-S: Finding faults using specialized test suites
- STS-P: Finding faults using specialized test suites in parallel

Measures
- D: total developer effort
- FF: total effort to failure-free program

Subject
- SPACE
  - 6000 LOC
  - 100 8-fault versions; > 1000 derivative versions

Method
- For each of 100 8-fault versions, debug until failure-free, using
  - Non specialized test suite
  - Specialized test suite both sequential and parallel

Summary of Results

<table>
<thead>
<tr>
<th>M</th>
<th>NSTS</th>
<th>STS-S</th>
<th>STS-P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20.2</td>
<td>20.2</td>
<td>20.2</td>
</tr>
</tbody>
</table>

MASPLAS 2007
Summary of Results

- Using specialized test suites based on fault-focusing clusters reduces developer effort, on average, over not using specialized test suites.
- Benefit holds when performing:
  - fault localization sequentially (one developer)
  - fault localization in parallel (multiple developers)
Summary of Results

- Using specialized test suites based on fault-focusing cluster can reduce time to a failure-free program, on average, over not using specialized test suites
- Benefit holds when performing
  - fault localization sequentially (one developer)
  - fault localization in parallel (multiple developers)

Outline of Talk

- Introduction
- Automatic fault localization—Tarantula
  - General technique
  - Visualization and demo
  - Test-case clustering to improve efficiency
- Related work
- Conclusion
Related Work

Automated clustering
• Multivariate Projection
  [Podgurski et al. ICSE ’01, ICSE ’03]

Fault localization
• Set-based, and Nearest Neighbor [Renieris, Reiss ASE ’03]
• Statistical [Liblit et al. PLDI ‘03, ‘05]
• SOBER [Liu et al. ESEC/FSE ‘05]

Outline of Talk

• Introduction
• Automatic fault localization—Tarantula
  • General technique
  • Visualization and demo
  • Test-case clustering to improve efficiency
• Related work

Conclusion
Conclusion

- **Technique and visualization** to provide support for finding faults
- **Technique to get fault-focusing clusters to create specialized test suite**
- **New approach to debugging—parallel debugging**
- **Benefits, supported by studies**
  - general technique outperforms existing techniques (subjects)
  - less time to failure-free program
  - better utilization of developer effort
  - indication of the number of faults

Future Work

- **Additional experimentation and with more subject programs**
- **Development of a cost model** that accounts for
  - Number of developers that are available
  - Organizational parameters such as salary, testing costs, and urgency of software delivery
- **Development of a process for debugging** that
  - includes fault-localization techniques
  - guides multiple debuggers with tasks such as synchronization
For More Information

- Jones, Harrold, Automated Software Engineering (ASE) 2005
- Jones, Orso, Harrold, Journal of Information Visualization 2004
- Jones, Orso, Harrold, Software Visualization (SoftVis) 2003
- Jones, Harrold, Stasko, International Conference on Software Engineering (ICSE) 2002
- Jones, Harrold, Stasko, Workshop on Software Visualization 2001