Program Analysis & Software Engineering

Hira Agrawal
hira@research.telcordia.com

Telcordia Technologies
Formerly Bellcore... Performance from Experience
Motivation

- Common Scenarios
  - No idea why the program fails for a certain input
  - No idea which modules implement function X
  - No time to achieve desired block coverage
  - No time to run even half of the regression tests
Outline

- Program Analysis
- Static v/s Dynamic
- Program Analysis in Software Engineering
  - Efficient Coverage Testing
  - Smart Debugging
  - Incremental Regression Testing
- A Short Demo
- Future Directions
Program Analysis

• Compiler Optimizations
  • Common Subexpression Elimination
  • Constant Folding
  • Code Motion
  • etc.
Static Program Analysis

• Control Flow Analysis
  • Basic Blocks
  • Dominator Trees
  • Natural Loops
  • etc.
• Data Flow Analysis
  • Def-Use Chains
  • Live Variables
  • Available Expressions
  • etc.
Dynamic Program Analysis

- Which paths were traversed?
- Which def-use associations were exercised?
- Which functions got invoked from where?
- Which statements were exercised?
- Which statement should I try to cover next?
- etc.
### Static v/s Dynamic

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>What <em>could</em> happen?</td>
<td>What <em>did</em> happen?</td>
</tr>
<tr>
<td>Foresight</td>
<td>Hindsight</td>
</tr>
<tr>
<td>Conservative</td>
<td>Precise</td>
</tr>
</tbody>
</table>
Program Analysis & Software Engineering

- Traditional Applications
  - Software Metrics
  - Coverage Testing
  - Execution Profiling
- Newer Applications
  - Efficient Coverage Testing
  - Incremental Regression Testing
  - Smart Debugging
  - Program Understanding
  - White Box Reliability Engineering
  - Probabilistic Optimizations
An Example

```plaintext
exp_1;
while (exp_2) {
    switch (exp_3) {
        case 1: exp_4;
        break;
        case 2: exp_5;
        while (exp_6) exp_7;
        default:
            if (exp_8) {
                exp_9;
                continue;
            }
            do exp_10; while (exp_11);
    }
    exp_12;
}
exp_13;
exp_14;
```
Efficient Coverage Testing

```plaintext
def ef 1;
while (exp 2) {
    switch (exp 3) {
        case 1: exp 4;
            break;
        case 2: exp 5;
            while (exp 6) exp 7;
        default:
            if (exp 8) {
                exp 9;
                continue;
            }
            do exp 10; while (exp 11);
            exp 12;
        }
    exp 13;
}
exp 14;
```
Control Flow Graph

entry 1

2

3

4

5

6

7

8

9

10

11

12

13

14

exit
Predominator Tree
Control Flow Graph

entry 1

2

3

4

5

6

7

8

9

10

11

12

13

14

exit
Predominator Tree (cont’d)
Postdominator Tree
Control Flow Graph
Postdominator Tree (cont’d)
Dominator Graph

union of pre- & post-dominator trees
Dominator Graph (cont’d)
Condensed Dominator Graph
Super Block Dominator Graph

1, 2, 14

3

8

9

5, 6

7

13

10, 11, 12

4
Super Blocks v/s Basic Blocks

- A super block contains one or more basic blocks.
- A super block need not be contiguous.
- If any statement in a super block is visited, then all statements in it must be visited, provided the execution terminates on that input.
- We only need one instrument per super block.
Weighted Super Block Dominator Graph

1, 2, 14

3

8

9

5, 6

10, 11, 12

13

4

9

6

7

8

6

9

6
Cover the Heaviest Leaf First
Recompute Weights

Preferred Order:
<10, 7, 9, 4>
Coverage Rate

![Graph showing overall block coverage and basic block coverage order with percentages (7%, 14%, 21%, 29%) at different points.](image-url)
## Experimental Results

<table>
<thead>
<tr>
<th>program</th>
<th>basic blocks</th>
<th>blocks that need to be covered</th>
<th>probes required</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort</td>
<td>455</td>
<td>138</td>
<td>152</td>
</tr>
<tr>
<td>spiff</td>
<td>1266</td>
<td>361</td>
<td>404</td>
</tr>
<tr>
<td>mgr</td>
<td>3848</td>
<td>1043</td>
<td>1233</td>
</tr>
<tr>
<td>ion</td>
<td>4886</td>
<td>1280</td>
<td>1507</td>
</tr>
<tr>
<td>atac</td>
<td>8737</td>
<td>2574</td>
<td>2971</td>
</tr>
<tr>
<td>odin</td>
<td>9870</td>
<td>2344</td>
<td>2944</td>
</tr>
<tr>
<td>xlib</td>
<td>15580</td>
<td>5111</td>
<td>6016</td>
</tr>
<tr>
<td>tvo</td>
<td>17680</td>
<td>6267</td>
<td>8150</td>
</tr>
</tbody>
</table>
read (a, b, c);
class := scalene;
if a = b || b = a
  class := isosceles;
if a*a = b*b + c*c
  class := right;
if a = b && b = c
  class := equilateral;
case class of
  right : area := b*c / 2;
equilatral : area := a*a * sqrt(3)/4;
otherwise : s := (a+b+c)/2;
            area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
### Test Suite

<table>
<thead>
<tr>
<th>Test case</th>
<th>Input</th>
<th>Output</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>$T_1$</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$T_2$</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$T_3$</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$T_4$</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>$T_5$</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
## Failure Detected!

<table>
<thead>
<tr>
<th>Test case</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>class</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>area</td>
</tr>
<tr>
<td>$T_1$</td>
<td>2</td>
<td>equilateral</td>
</tr>
<tr>
<td>$T_2$</td>
<td>4</td>
<td>isosceles</td>
</tr>
<tr>
<td>$T_3$</td>
<td>5</td>
<td>right</td>
</tr>
<tr>
<td>$T_4$</td>
<td>6</td>
<td>scalene</td>
</tr>
<tr>
<td>$T_5$</td>
<td>3</td>
<td>equilateral</td>
</tr>
<tr>
<td>$T_6$</td>
<td>4</td>
<td>scalene</td>
</tr>
</tbody>
</table>

*Failure!*
Where’s the Bug?

read (a, b, c);
class := scalene;
if a = b || b = a
  class := isosceles;
if a*a = b*b + c*c
  class := right;
if a = b && b = c
  class := equilateral;
case class of
  right : area := b*c / 2;
equilateral : area := a*a * sqrt(3)/4;
otherwise : s := (a+b+c)/2;
  area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
Execution Slice w.r.t. $T_6 = (4 3 3)$

```pascal
read (a, b, c);
class := scalene;
if a = b || b = a
  class := isosceles;
if a*a = b*b + c*c
  class := right;
if a = b && b = c
  class := equilateral;
case class of
  right       : area := b*c / 2;
equilateral : area := a*a * sqrt(3)/4;
otherwise    : s := (a+b+c)/2;
                        area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```
## Failure Detected!

<table>
<thead>
<tr>
<th>Test case</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>class</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>area</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>$T_1$</td>
<td>2</td>
<td>equilateral</td>
</tr>
<tr>
<td>$T_2$</td>
<td>4</td>
<td>isosceles</td>
</tr>
<tr>
<td>$T_3$</td>
<td>5</td>
<td>right</td>
</tr>
<tr>
<td>$T_4$</td>
<td>6</td>
<td>scalene</td>
</tr>
<tr>
<td>$T_5$</td>
<td>3</td>
<td>equilateral</td>
</tr>
<tr>
<td>$T_6$</td>
<td>4</td>
<td>scalene</td>
</tr>
</tbody>
</table>

Failure Detected!
Execution Slice w.r.t. Failed Test (T₆)

```pascal
read (a, b, c);
class := scalene;
if a = b || b = a
  class := isosceles;
if a*a = b*b + c*c
  class := right;
if a = b && b = c
  class := equilateral;
case class of
  right : area := b*c / 2;
equilateral : area := a*a * sqrt(3)/4;
otherwise : s := (a+b+c)/2;
  area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```
Execution Slice w.r.t. Passed Test, \((T_2)\)

\[
\text{read} (a, b, c); \\
\text{class} := \text{scalene}; \\
\text{if} \ a = b \ || \ b = c \\
\quad \text{class} := \text{isosceles}; \\
\text{if} \ a^2 = b^2 + c^2 \\
\quad \text{class} := \text{right}; \\
\text{if} \ a = b \ && \ b = c \\
\quad \text{class} := \text{equilateral}; \\
\text{case} \ \text{class} \ \text{of} \\
\quad \text{right} : \ \text{area} := \frac{b \times c}{2}; \\
\quad \text{equilateral} : \ \text{area} := \frac{a \times a \times \sqrt{3}}{4}; \\
\quad \text{otherwise} : \ s := \frac{a+b+c}{2}; \\
\quad \quad \text{area} := \sqrt{s(s-a)(s-b)(s-c)}; \\
\text{end}; \\
\text{write}(\text{class}, \text{area});
\]
Resulting Dice [Slice (4 3 3) – Slice (4 4 3)]

```plaintext
read (a, b, c);
class := scalene;
if a = b || b = a
    class := isosceles;
if a*a = b*b + c*c
    class := right;
if a = b && b = c
    class := equilateral;
case class of
    right      : area := b*c / 2;
equilateral : area := a*a * sqrt(3)/4;
otherwise   : s := (a+b+c)/2;
              area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```

*Bug! Should be:*  
\[ b = c \]
Slicing Experiment

• Unix Sort Program
• 508 Basic Blocks
• 25 Independently Seeded Faults
• 56 Tests
• 96% Block Coverage, 89% Branch Coverage
• 6 Faults Not Detected By Any Test
Slicing Experiment Results

<table>
<thead>
<tr>
<th>fault id</th>
<th># of dices</th>
<th># good</th>
<th>avg. size</th>
</tr>
</thead>
<tbody>
<tr>
<td>F117</td>
<td>159</td>
<td>84%</td>
<td>26%</td>
</tr>
<tr>
<td>F206</td>
<td>55</td>
<td>93%</td>
<td>11%</td>
</tr>
<tr>
<td>F336</td>
<td>783</td>
<td>100%</td>
<td>24%</td>
</tr>
<tr>
<td>F414</td>
<td>55</td>
<td>100%</td>
<td>22%</td>
</tr>
<tr>
<td>F439</td>
<td>208</td>
<td>94%</td>
<td>18%</td>
</tr>
<tr>
<td>F442</td>
<td>495</td>
<td>100%</td>
<td>14%</td>
</tr>
<tr>
<td>F507</td>
<td>684</td>
<td>50%</td>
<td>6%</td>
</tr>
<tr>
<td>F608</td>
<td>55</td>
<td>100%</td>
<td>31%</td>
</tr>
<tr>
<td>F631</td>
<td>423</td>
<td>100%</td>
<td>19%</td>
</tr>
<tr>
<td>F634</td>
<td>300</td>
<td>100%</td>
<td>23%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fault id</th>
<th># of dices</th>
<th># good</th>
<th>avg. size</th>
</tr>
</thead>
<tbody>
<tr>
<td>F691</td>
<td>735</td>
<td>100%</td>
<td>16%</td>
</tr>
<tr>
<td>F772</td>
<td>55</td>
<td>98%</td>
<td>19%</td>
</tr>
<tr>
<td>F782</td>
<td>384</td>
<td>79%</td>
<td>11%</td>
</tr>
<tr>
<td>F783</td>
<td>384</td>
<td>100%</td>
<td>11%</td>
</tr>
<tr>
<td>F806</td>
<td>55</td>
<td>96%</td>
<td>29%</td>
</tr>
<tr>
<td>F811</td>
<td>159</td>
<td>85%</td>
<td>23%</td>
</tr>
<tr>
<td>F851</td>
<td>768</td>
<td>100%</td>
<td>15%</td>
</tr>
<tr>
<td>F882</td>
<td>423</td>
<td>100%</td>
<td>14%</td>
</tr>
<tr>
<td>F910</td>
<td>255</td>
<td>86%</td>
<td>15%</td>
</tr>
<tr>
<td><strong>Avg.</strong></td>
<td><strong>338</strong></td>
<td><strong>93%</strong></td>
<td><strong>18%</strong></td>
</tr>
</tbody>
</table>
Incremental Regression Testing

```
read (a, b, c);
class := scalene;
if a = b || b = c
    class := isosceles;
if a*a = b*b + c*c
    class := right;
if a = b && b = c
    class := equilateral;
case class of
    right : area := b*c / 2;
equilateral : area := a*a * sqrt(3)/4;
otherwise : s := (a+b+c)/2;
            area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```
### Which Tests Should We Rerun?

<table>
<thead>
<tr>
<th>Test case</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>class</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>area</td>
</tr>
<tr>
<td>$T_1$</td>
<td>2</td>
<td>equilateral</td>
</tr>
<tr>
<td>$T_2$</td>
<td>4</td>
<td>isosceles</td>
</tr>
<tr>
<td>$T_3$</td>
<td>5</td>
<td>right</td>
</tr>
<tr>
<td>$T_4$</td>
<td>6</td>
<td>scalene</td>
</tr>
<tr>
<td>$T_5$</td>
<td>3</td>
<td>equilateral</td>
</tr>
<tr>
<td>$T_6$</td>
<td>4</td>
<td>scalene</td>
</tr>
</tbody>
</table>

Failure!
Execution Slice w.r.t. $T_2 = (4 \ 4 \ 3)$

```plaintext
read (a, b, c);
class := scalene;
if a = b || b = a
    class := isosceles;
if a*a = b*b + c*c
    class := right;
if a = b && b = c
    class := equilateral;
case class of
    right    : area := b*c / 2;
equilaterial : area := a*a * sqrt(3)/4;
otherwise  : s := (a+b+c)/2;
              area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```

Bug!
Execution Slice w.r.t. $T_4 = (6 \, 5 \, 4)$

```pascal
read (a, b, c);
class := scalene;
if a = b || b = a
    class := isosceles;
if a^2 = b^2 + c^2
    class := right;
if a = b && b = c
    class := equilateral;
end;
case class of
    right : area := b*c / 2;
equilateral : area := a^2 * sqrt(3)/4;
otherwise : s := (a+b+c)/2;
    area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```

Bug!
Which Test Cases Should We Rerun?

<table>
<thead>
<tr>
<th>Test case</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>( T_1 )</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( T_2 )</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>( T_3 )</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>( T_4 )</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>( T_5 )</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>( T_6 )</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Failure!
Execution Slice w.r.t. $T_3 = (5 \ 4 \ 3)$

```plaintext
read (a, b, c);
class := scalene;
if a = b || b = a
  class := isosceles;
if a*a = b*b + c*c
  class := right;
if a = b && b = c
  class := equilateral;
case class of
  right : area := b*c / 2;
equilateral : area := a*a * sqrt(3)/4;
otherwise : s := (a+b+c)/2;
  area := sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```

*Bug!*

*scalene*  
*Failure!*
## Which Test Cases Should We Rerun?

<table>
<thead>
<tr>
<th>Test case</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a b c</td>
<td>class area</td>
</tr>
<tr>
<td>( T_1 )</td>
<td>2 2 2</td>
<td>equilateral 1.73</td>
</tr>
<tr>
<td>( T_2 )</td>
<td>4 4 3</td>
<td>isosceles 5.56</td>
</tr>
<tr>
<td>( T_3 )</td>
<td>5 4 3</td>
<td>right 6.00</td>
</tr>
<tr>
<td>( T_4 )</td>
<td>6 5 4</td>
<td>scalene 9.92</td>
</tr>
<tr>
<td>( T_5 )</td>
<td>3 3 3</td>
<td>equilateral 3.90</td>
</tr>
<tr>
<td>( T_6 )</td>
<td>4 3 3</td>
<td>scalene 4.47</td>
</tr>
</tbody>
</table>

*Failure!*
Regression Testing Experiment

- A Space Program from European Space Agency
- About 10,000 lines of C code
- 10 Actual Faults Obtained from Its Error Logs
- 1,000 Regression Tests With Operational Profile
## Regression Testing Experiment Results

<table>
<thead>
<tr>
<th>fault id</th>
<th># tests rerun</th>
<th>% tests rerun</th>
<th>minimum # tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>38</td>
<td>3.8%</td>
<td>26</td>
</tr>
<tr>
<td>F2</td>
<td>22</td>
<td>2.2%</td>
<td>16</td>
</tr>
<tr>
<td>F3</td>
<td>84</td>
<td>8.4%</td>
<td>36</td>
</tr>
<tr>
<td>F4</td>
<td>4</td>
<td>0.4%</td>
<td>4</td>
</tr>
<tr>
<td>F5</td>
<td>4</td>
<td>0.4%</td>
<td>4</td>
</tr>
<tr>
<td>F6</td>
<td>38</td>
<td>3.8%</td>
<td>32</td>
</tr>
<tr>
<td>F7</td>
<td>38</td>
<td>3.8%</td>
<td>32</td>
</tr>
<tr>
<td>F8</td>
<td>34</td>
<td>3.4%</td>
<td>6</td>
</tr>
<tr>
<td>F9</td>
<td>71</td>
<td>7.1%</td>
<td>46</td>
</tr>
<tr>
<td>F10</td>
<td>470</td>
<td>47%</td>
<td>320</td>
</tr>
</tbody>
</table>
\( \chi \)Suds ToolSuite

Diagram showing the integration of tools in the design and development process:
- \( \chi \)Design
- \( \chi \)Find
- \( \chi \)Vue
- \( \chi \)Diff
- \( \chi \)Regress
- \( \chi \)ATAC
- \( \chi \)Slice
- \( \chi \)Prof
- \( \chi \)SFIT

The tools are interconnected in a cycle, indicating a seamless workflow from design to testing and back.
Program Instrumentation

sort.c → cc → sort → output
Program Instrumentation (cont’d)

input

output

sort.c

atac cc

sort

sort.atac

sort.trace
Program Instrumentation (cont’d)

sort.c → atac cc → sort → output

sort.atac → χSuds

input

sort.trace

GUI & reports
A Short Demo

Design

χDesign
χATAC
χSlice
χProf
χSFIT
χRegress
χDiff
χFind
χVue
χSUDS

Design

Texting

Development

Maintenance

Mainenance

Testing

Development

Maintenance

Texting

Mainenance
For More Information:

- www.research.telcordia.com/demos/#xsuds
- xsuds.argreenhouse.com
- www.research.telcordia.com/papers/hira
Future Directions

- Smart Program Diffs and Incremental Analysis
- More Empirical Studies
- More Advanced Heuristics
- High Level User Interfaces
- Automatic Test Case Generation