Mining Programs

Andreas Zeller
Saarland University
Saarbrücken
Saarbrücken
Mining
The First Bug
(September 9, 1947)

1545

First actual case of bug being found.

Relay #70 Panel F
(moth) in relay.
```c
void list() { // Display this
    delete list; (List *) 0x804df80
    delete list->next;
    delete list;
}

// Test
void list() {
    list = new List(a_global + start++);
}

void ref() {
    date = DeleteDate(date);
}
```

**DDD Tip of the Day #5**

If you made a mistake, try Edit→Undo. This will undo the most recent debugger command and redisplay the previous program state.
How to Debug
(Sommerville 2004)

Locate error → Design error repair → Repair error → Re-test program
WHERE'S WALDO?
A GCC State
A Guide to Systematic Debugging

WHY PROGRAMS FAIL

“And the definitive book on debugging”

– WALTER F. TICHY

TU Karlsruhe

A Guide to Systematic Debugging

ANDREAS ZELLER

dpunkt
What is the cause of this failure?
Isolating Causes

Actual world  Alternate world

Test  Mixed world

Locating Defects

Failing run  Passing run

Detecting Anomalies

Properties  Properties

Differences correlate with failure

Mining Models

Failing Tests  Passing Tests

Differences correlate with failures
Isolating Causes

Actual world
Alternate world

Test
Mixed world

Locating Defects

Failing run
Passing run

<PLUS node>
<PLUS node>

Detecting Anomalies

Properties Properties

Differences correlate with failure

Mining Models

Failing Tests
Passing Tests

-Different
isEmpty
add()
remove()
isEmpty()
<init>

-Different
isEmpty
add()
clear()
clear()
isEmpty()
<init>

Differences correlate with failures
double bug(double z[], int n) {
    int i, j;

    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}
What is the cause of this failure?
Causality

What does it mean that “A causes B”?

• **Counterfactual approach:** If A had not occurred, then B would not have occurred.

• **Statistical approach:** A results in an increase in the probability of B.
Counterfactual Causality

- Actual world
- Effect does occur

- Alternate world
- Effect does not occur

Causes
double bug(double z[], int n) {
    int i, j;
    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}
double bug(double z[], int n) {
    int i, j;
    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}
Causes as Differences

Actual world

empty.c: GCC works fine

bug.c: GCC crashes

Cause: bug.c

Alternate world
Actual Causes

“The” cause (actual cause) is a minimal difference
Isolating Causes

double bug(double z[], int n) {
    int i, j;
    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}
double bug(double z[], int n) {
    int i, j;
    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}
double bug(double z[], int n) {
    int i, j;
    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}
Isolating Causes

double bug(double z[], int n) {
    int i, j;

    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}

Actual cause narrowed down
double bug(double z[], int n) {
    int i, j;
    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}
double bug(double z[], int n) {
    int i, j;
    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}
Isolating Causes

double bug(double z[], int n) {
    int i, j;

    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }

    return z[n];
}

Actual cause of the GCC crash
Isolating Causes

Actual world

Alternate world

Test

Mixed world
Isolating Causes

Actual world → "+ 1.0" ← Alternate world

Test

Mixed world

✔

✘

?”
Configuration

Circumstance

\( \delta \)

All circumstances

\[ C = \{ \delta_1, \delta_2, \ldots \} \]

Configuration

\[ c \subseteq C \]

\[ c = \{ \delta_1, \delta_2, \ldots \delta_n \} \]
Tests

Testing function

\[
test(c) \in \{\checkmark, \times, ?\}
\]

Initial configurations

\[
\begin{align*}
test(c_{\checkmark}) &= \checkmark \\
test(c_{\times}) &= \times
\end{align*}
\]
Minimal Difference

Goal: Subsets $c'_x$ and $c'_\checkmark$

$\emptyset = c'_{\checkmark} \subseteq c'_\checkmark \subset c'_{\times} \subseteq c_x$

Difference

$\Delta = c'_x \setminus c'_\checkmark$

Difference is 1-minimal

$\forall \delta_i \in \Delta \cdot test(c'_\checkmark \cup \{\delta_i\}) \neq \checkmark \land test(c'_x \setminus \{\delta_i\}) \neq \times$
Isolating

Input

\[ test(c_x) = \times \]

\[ \Delta = c_x \setminus c' \]

Failure Cause

\[ test(c_\checkmark) = \checkmark \]
Algorithm Sketch

• Minimize the difference between two sets — with working sets $c'$ and $c'$

• Compute subsets

\[ \Delta_1 \cup \Delta_2 \cup \cdots \cup \Delta_n = \Delta = c' \setminus c' \]

• For each subset, test
  
  • the addition $c' \cup \Delta_i$
  
  • the removal $c' \setminus \Delta_i$
### Test Outcomes

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action 1</th>
<th>Action 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{test}(c'_x \setminus \Delta_i)$</td>
<td>$c'_x := c'_x \setminus \Delta_i$</td>
<td>$c'<em>\checkmark := c'</em>\checkmark \setminus \Delta_i$</td>
</tr>
<tr>
<td>$\text{test}(c'_\checkmark \cup \Delta_i)$</td>
<td>$c'<em>\checkmark := c'</em>\checkmark \cup \Delta_i$</td>
<td>$c'<em>\checkmark := c'</em>\checkmark \cup \Delta_i$</td>
</tr>
<tr>
<td>otherwise</td>
<td>increase granularity</td>
<td></td>
</tr>
</tbody>
</table>

most valuable outcomes
dd in a Nutshell

\[ dd(c_\checkmark, c_\times) = (c_\checkmark', c_\times') \quad \Delta = c_\times' \setminus c_\checkmark' \text{ is 1-minimal} \]

\[ dd(c_\checkmark, c_\times) = dd'(c_\checkmark, c_\times, 2) \]

\[
\begin{align*}
dd'(c_\checkmark', c_\times', n) = \\
\begin{cases}
(c_\checkmark', c_\times') & \text{if } |\Delta| = 1 \\
\quad \text{if } \exists i \in \{1..n\} \cdot test(c_\times \setminus c_\checkmark') = \checkmark \\
\quad \text{if } \exists i \in \{1..n\} \cdot test(c_\checkmark' \cup \Delta_i) = \times \\
\quad \text{else if } \exists i \in \{1..n\} \cdot test(c_\checkmark' \cup \Delta_i) = \checkmark \\
\quad \text{else if } \exists i \in \{1..n\} \cdot test(c_\times \setminus \Delta_i) = \times \\
\quad \text{else if } n < |\Delta| \text{ ("increase granularity")}
\end{cases}
\end{align*}
\]
def dd(c_pass, c_fail):
    n = 2
    while 1:
        delta = listminus(c_fail, c_pass)
        deltas = split(delta, n); offset = 0; j = 0
        while j < n:
            i = (j + offset) % n
            next_c_pass = listunion(c_pass, deltas[i])
            next_c_fail = listminus(c_fail, deltas[i])
            if test(next_c_fail) == FAIL and n == 2:
                c_fail = next_c_fail; n = 2; offset = 0; break
            elif test(next_c_fail) == PASS:
                c_pass = next_c_fail; n = 2; offset = 0; break
            elif test(next_c_pass) == FAIL:
                c_fail = next_c_pass; n = 2; offset = 0; break
            elif test(next_c_fail) == FAIL:
                c_fail = next_c_fail; n = max(n - 1, 2); offset = i; break
            elif test(next_c_pass) == PASS:
                c_pass = next_c_pass; n = max(n - 1, 2); offset = i; break
            else:
                j = j + 1
        if j >= n:
            if n >= len(delta):
                return (delta, c_pass, c_fail)
            else:
                n = min(len(delta), n * 2)
Properties

number of tests $t$ – worst case:

$$t = |\Delta|^2 + 7|\Delta| \quad \text{where} \quad \Delta = c_x \setminus c_✓$$

number of tests $t$ – best case (no unresolved outcomes):

$$t \leq \log_2(\Delta)$$

size of difference – no unresolved outcomes

$$|c'_x \setminus c'_✓| = 1$$
# Applications

<table>
<thead>
<tr>
<th>Input</th>
<th>Code Changes</th>
<th>Schedules</th>
</tr>
</thead>
</table>

Mozilla Bug #24735

Ok the following operations cause mozilla to crash consistently on my machine

-> Start mozilla
-> Go to bugzilla.mozilla.org
-> Select search for bug
-> Print to file setting the bottom and right margins to .50
   (I use the file /var/tmp/netscape.ps)
-> Once it's done printing do the exact same thing again on
   the same file (/var/tmp/netscape.ps)
-> This causes the browser to crash with a segfault
Isolating Input

<SELECT NAME="priority" MULTIPLE SIZE=7> ✗

Difference narrowed down

<SELECT NAME="priority" MULTIPLE SIZE=7> ✔

<SELECT NAME="priority" MULTIPLE SIZE=7> ✔

<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
Isolating Input

<SELECT NAME="priority" MULTIPLE SIZE=7>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<SELECT NAME="priority" MULTIPLE SIZE=7>
When using DDD with GDB 4.16, the run command correctly uses any prior command-line arguments, or the value of "set args". However, when I switched to GDB 4.17, this no longer worked: If I entered a run command in the console window, the prior command-line options would be lost. [...]
Wie finden wir die alternative Welt?

Version Differences

New version

Program fails

Program works

Old version

Causes
What was Changed

$ diff -r gdb-4.16 gdb-4.17
diff -r gdb-4.16/COPYING gdb-4.17/COPYING
5c5
< 675 Mass Ave, Cambridge, MA 02139, USA
---
> 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
282c282
< Appendix: How to Apply These Terms to Your New Programs
---
> How to Apply These Terms to Your New Programs

…and so on for 178,200 lines (8,721 locations)
Challenges

- Granularity – within some large change, only a few lines may be relevant
- Interference – some (later) changes rely on other (earlier) changes
- Inconsistency – some changes may have to be combined to produce testable code

Delta debugging handles all this
General Plan

• Decompose diff into changes per location (= 8,721 individual changes)
• Apply subset of changes, using PATCH
• Reconstruct GDB; build errors mean unresolved test outcome
• Test GDB and return outcome
Isolating Changes

Delta Debugging Log

- GDB with ddm log algorithm
  - with dd algorithm
  - plus scope information

- Result after 98 tests (= 1 hour)
The Failure Cause

diff -r gdb-4.16/gdb/infcmd.c gdb-4.17/gdb/infcmd.c
1239c1278
< "Set arguments to give program being debugged when it is started."
---
> "Set argument list to give program being debugged when it is started."

- Documentation becomes GDB output
- DDD expects Arguments, but GDB outputs Argument list
Optimizations

- History – group changes by creation time
- Reconstruction – cache several builds
- Grouping – according to scope
- Failure Resolution – scan error messages for possibly missing changes
Thread Schedules

The behavior of a multi-threaded program can depend on the thread schedule:

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Thread A</th>
<th>Thread B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>open(&quot;.htpasswd&quot;)</td>
<td>open(&quot;.htpasswd&quot;)</td>
</tr>
<tr>
<td></td>
<td>read(...)</td>
<td>read(...)</td>
</tr>
<tr>
<td></td>
<td>modify(...)</td>
<td>modify(...)</td>
</tr>
<tr>
<td></td>
<td>write(...)</td>
<td>write(...)</td>
</tr>
<tr>
<td></td>
<td>close(...)</td>
<td>close(...)</td>
</tr>
</tbody>
</table>

A’s updates get lost!
DEJAVU captures and replays program runs deterministically:

- DEJAVU recorded schedule
- DEJAVU replay

DEJAVU allows simple reproduction of schedules and induced failures.
Schedules as Input

Using DEJAVU, we can consider the schedule as an input which determines whether the program passes or fails.

The schedule difference causes the failure!
Finding Differences

- We start with runs ✓ and ✗
- We determine the differences $\Delta_i$ between thread switches $t_i$:
  - $t_1$ occurs in ✓ at “time” 254
  - $t_1$ occurs in ✗ at “time” 278
  - The difference $\Delta_1 = |278 - 254|$ induces a statement interval: the code executed between “time” 254 and 278
  - Same applies to $t_2$, $t_3$, etc.
Isolating Differences

We use Delta Debugging to isolate the relevant differences:

- The entire difference $\Delta_1$ is applied
- Half of the difference $\Delta_2$ is applied
- $\Delta_3$ is not applied at all

DEJAVU executes the debuggee under this generated schedule; an automated test checks if the failure occurs.
Isolating Differences

The Isolation Process

Delta Debugging systematically narrows down the difference!

Dejavu replays the generated schedule

Test outcome
Example: Raytracer

- Raytracer program from Spec JVM98 suite
- Injected a simple race condition
- Set up automated test + random schedules
- Obtained passing and failing schedule
- 3,842,577,240 differences, each moving a thread switch by ±1 yield point (time unit)
no unresolved outcomes: complexity is $O(\log_2 n)$
The Failure Cause

```java
public class Scene {
    private static int ScenesLoaded = 0;
    (more methods...)

    private int LoadScene(String filename) {
        int OldScenesLoaded = ScenesLoaded;
        (more initializations...)
        infile = new DataInputStream(...);
        (more code...)
        ScenesLoaded = OldScenesLoaded + 1;
        System.out.println("" + ScenesLoaded + " scenes loaded.");
        ...
    }
}
```
Isolating Causes

Actual world
Alternate world

Test
Mixed world

Locating Defects

Failing run
Passing run

Locating Defects

Locating Defects

Locating Defects

Detecting Anomalies

Detecting Anomalies

Detecting Anomalies

Detecting Anomalies

Detecting Anomalies

Mining Models

Mining Models

Mining Models

Mining Models

Mining Models

Mining Models

Mining Models

Mining Models

Mining Models

Mining Models
Isolating Causes

Alternate world
Actual world

Test
Mixed world

Locating Defects

Failing run
Passing run

<PLUS node>
<PLUS node>

Detecting Anomalies

Properties
Properties

Differences correlate with failure

Mining Models

Failing Tests
Passing Tests

<init>
add()
remove()
isEmpty()
isEmpty()
isEmpty()
<init>
add()
clear()
clear()
isEmpty()
isEmpty()
Differences correlate with failures
What is the cause of this failure?
Isolating Causes

double bug(double z[], int n) {
    int i, j;

    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }

    return z[n];
}
1. The programmer creates a defect – an error in the code.

2. When executed, the defect creates an infection – an error in the state.

3. The infection propagates.

4. The infection causes a failure.

This infection chain must be traced back – and broken.
Causes in State

Infected state

Sane state

- 41,000 variables
- 42,000 references
- 1 reference is wrong
- Which one?

The difference causes GCC to crash!
Search in Space

Infected state

Sane state

Test

Mixed state

✔

✘
Delta Debugging

Delta Debugging Log

cpass
cfail

first_loop_store_insn \rightarrow fld[1].rtx \rightarrow fld[1].rtx \rightarrow fld[3].rtx \rightarrow fld[1].rtx \rightarrow code == PLUS
Search in Space

Infected state

Mixed state

Sane state

Test

Mixed state

?
Search in Space

Infected state

Sane state

<PLUS node>
Search in Time

Failing run

<PLUS node>

Passing run

<PLUS node>

<PLUS node>

<PLUS node>

\[ t \]
Search in Time

Failing run

Passing run

\[ \text{link} \rightarrow \text{fld}[0].rtx \rightarrow \text{fld}[0].rtx == \text{link} \]
Why Transitions?

• Each failure cause in the program state is caused by some statement
• These statements are executed at cause transitions
• Cause transitions thus are statements that cause the failure
## All GCC Transitions

<table>
<thead>
<tr>
<th>Location</th>
<th>New cause at transition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Start&gt;</code></td>
<td><code>argv[3]</code></td>
</tr>
<tr>
<td>toplev.c:4755</td>
<td><code>name</code></td>
</tr>
<tr>
<td>toplev.c:2909</td>
<td><code>dump_base_name</code></td>
</tr>
<tr>
<td>c-lex.c:187</td>
<td><code>finput → _IO_buf_base</code></td>
</tr>
<tr>
<td>c-lex.c:1213</td>
<td><code>nextchar</code></td>
</tr>
<tr>
<td>c-lex.c:1213</td>
<td><code>yyssa[41]</code></td>
</tr>
<tr>
<td>c-typeck.c:3615</td>
<td><code>yyssa[42]</code></td>
</tr>
<tr>
<td>c-lex.c:1213</td>
<td><code>last_insn → fld[1].rtx → ... → fld[1].rtx.code</code></td>
</tr>
<tr>
<td>c-decl.c:1213</td>
<td><code>sequence_result[2] → ... → fld[1].rtx.code</code></td>
</tr>
<tr>
<td>combine.c:4271</td>
<td><code>x → fld[0].rtx → fld[0].rtx</code></td>
</tr>
</tbody>
</table>
if (GET_CODE (XEXP (x, 0)) == PLUS {
    x = apply_distributive_law
        (gen_binary (PLUS, mode,
                     gen_binary (MULT, mode,
                                 XEXP (XEXP (x, 0), 0),
                                 XEXP (x, 1)),
                     gen_binary (MULT, mode,
                                 XEXP (XEXP (x, 0), 1),
                                 XEXP (x, 1))));

if (GET_CODE (x) != MULT)
    return x;
}
## Implementations

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web service + command line</td>
<td>Eclipse plug-in</td>
<td>Module</td>
<td></td>
</tr>
<tr>
<td>24 months</td>
<td>12 months</td>
<td>2 days</td>
<td></td>
</tr>
</tbody>
</table>

Stability
Isolating Relevant Calls

Step 1: Record

Event log contains 32 interactions

Vector()
add()
add()
remove()
remove()
remove()
Isolating Relevant Calls

Step 2: Replay

Vector

add

dd

remove

remove

remove

JINSI

ev: Vector

Event Log
Isolating Relevant Calls

Step 3: Simplify

Event log contains 32 interactions

JINSI

Vector() → add() → add() → remove() → remove() → remove() → remove() → v:Vector

Event Log
Isolating Relevant Calls
Step 4: Create Unit Test

testVector()
{
    Vector v = new Vector();
    v.remove(obj);
}
Columba ContactModel

- ContactModel()
- setSortString()
- setFormattedName()
- setNickName()
- setFamilyName()
- setGivenName()
- and 18732 more…
Columba ContactModel

```
ContactModel()
getPreferredEmail()
c: ContactModel
```
Unit Test

testContactModel()
{
    ContactModel c = new ContactModel();
    String s = c.getPreferredEmail();
}
public String getPreferredEmail() {
    Iterator it = getEmailIterator();

    // get first item
    IEmailModel model = (IEmailModel) it.next();

    // backwards compatibility
    // -> its not possible anymore to create a
    // contact model without email address
    if (model == null) {
        return null;
    }

    return model.getAddress();
}
Failure Cause
Conclusion

- Delta debugging on *states* is nice, but cumbersome
- Alternative: delta debugging on *object interaction*
- First promising results with JINSI prototype
- Combined solution for reproducing + simplifying
Isolating Causes

Actual world
Alternate world

Test
Mixed world

Locating Defects

Failing run

<PLUS node>

Passing run

<PLUS node>

Detecting Anomalies

Properties
Properties

Differences correlate with failure

Mining Models

Failing Tests

<init> add() isEmpty()

-isEmpty remove()

Passing Tests

<init> add() isEmpty()

-isEmpty clear()

Differences correlate with failures
FAIL

WHY PROGRAMS FAIL
A Guide to Systematic Debugging

ANDREAS ZELLER

“The definitive book on debugging”
– WALTER F. TICHY
TU Karlsruhe

On Display!
#1 in
Software Engineering Books
Algorithms Books
Software Design Books
WHY PROGRAMS FAIL: A Guide to Systematic Debugging

"Today every computer program written is also debugged, but debugging is not a widely studied or taught skill. Few books, beyond this one, present a systematic approach to finding and fixing programming errors."
— from the foreword by JAMES LARUS, Microsoft Research

WHY PROGRAMS FAIL is a book about bugs in computer programs, how to reproduce them, how to find them, and how to fix them such that they do not occur anymore. This book teaches a number of techniques that allow you to debug any program in a systematic, and sometimes even elegant way. Moreover, the techniques can widely be automated, which allows you to let your computer do most of the debugging.

Learn more about the book, its author, or its contents.

News

2006-08-24: John Lam of Dr. Dobbs writes: "This is a practical book where you find excellent discussions, everything from tracking defects to debugging. If you want to write better software, read this book."

2006-03-17: Why Programs Fail has won a Software Development Jolt Productivity Award! This is a great honor, and I am deeply grateful to the judges and organizers for this result. My editor, Tim Cox, has been able to attend the ceremony and accept the award.
Isolating Causes

Actual world

Alternate world

Test

Mixed world

Locating Defects

Failing run

<PLUS node>

<PLUS node>

Passing run

Detecting Anomalies

Properties

Properties

Differences correlate with failure

Mining Models

Failing Tests

<init>

add()

-isEmpty()

remove()

Differences correlate with failures

Passing Tests

<init>

add()

-isEmpty()

clear()
Isolating Causes

Actual world

Alternate world

Mixed world

Test

Locating Defects

Failing run

Passing run

Detecting Anomalies

Mining Models

Properties

Properties

Differences correlate with failure

Differences correlate with failures
Detecting Anomalies

Properties

Differences correlate with failure
<table>
<thead>
<tr>
<th>Coverage</th>
<th>Sequences</th>
<th>Returns</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Coverage</th>
<th>Sequences</th>
<th>Returns</th>
</tr>
</thead>
</table>

Properties
Comparing Coverage

1. Every failure is caused by an infection, which in turn is caused by a defect

2. The defect must be executed to start the infection

3. Code that is executed in failing runs only is thus likely to cause the defect
The middle program

$ middle 3 3 5
middle: 3

$ middle 2 1 3
middle: 1
int main(int argc, char *argv[])
{
    int x = atoi(argv[1]);
    int y = atoi(argv[2]);
    int z = atoi(argv[3]);
    int m = middle(x, y, z);
    printf("middle: %d\n", m);
    return 0;
}
int middle(int x, int y, int z) {
    int 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;
    }
    return m;
}
Obtaining Coverage
for C programs
if __name__ == "__main__":
    sys.settrace(tracer)
    x = sys.argv[1]
    y = sys.argv[2]
    z = sys.argv[3]
    m = middle(x, y, z)

    print "middle:", m
def tracer(frame, event, arg):
    code = frame.f_code
    function = code.co_name
    filename = code.co_filename
    line = frame.f_lineno
    print filename + "::" + `line` + \\
    "::" + function + "(O):", \\
    event, arg
    return tracer
Obtaining Coverage for Python programs

$ ./middle.py 3 3 5
./middle.py:13:middle(): call None
./middle.py:14:middle(): line None
./middle.py:15:middle(): line None
./middle.py:16:middle(): line None
./middle.py:18:middle(): line None
./middle.py:19:middle(): line None
./middle.py:26:middle(): line None
./middle.py:26:middle(): return 3
middle: 3
```c
int middle(int x, int y, int z) {
    int 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;
    }
    return m;
}
```
Discrete Coloring

- Executed only in failing runs
  - Highly suspect

- Executed in passing and failing runs
  - Ambiguous

- Executed only in passing runs
  - Likely correct
```c
int middle(int x, int y, int z) {
    int 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;
    }
    return m;
}
```
```c
int middle(int x, int y, int z) {
    int 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;
    }
    return m;
}
```
Continuous Coloring

- executed only in failing runs
- passing and failing runs
- executed only in passing runs
Hue

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

0% passed 100% passed
Brightness

\[ bright(s) = \max(\%passed(s), \%failed(s)) \]

frequently executed

rarely executed
int middle(int x, int y, int z) {
    int 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;
    }
    return m;
}
(*grampexc_ptr)->PQEXP_PTR = NULL;

Line 3754
Executions: 34 / 300
Passed: 5 / 267
Failed: 29 / 33

Source: Jones et al., ICSE 2002
Evaluation

How well does comparing coverage detect anomalies?

- How green are the defects? *(false negatives)*
- How red are non-defects? *(false positives)*
Space

- 8000 lines of executable code
- 1000 test suites with 156 to 4700 test cases
- 20 defective versions with one defect each (corrected in subsequent version)
18 of 20 defects are correctly classified in the “reddest” portion of the code.

Source: Jones et al., ICSE 2002
The “reddest” portion is at most 20% of the code

Source: Jones et al., ICSE 2002
Siemens Suite

- 7 C programs, 170–560 lines
- 132 variations with one defect each
- 108 all yellow (i.e., useless)
- 1 with one red statement (at the defect)

Source: Renieris and Reiss, ASE 2003
Nearest Neighbor

Run ✔
Run ✔
Run ✔
Run ✔
Run ✔
Run ✘
Run ✘
Nearest Neighbor

Compare with the single run that has the most similar coverage
Locating Defects

- Nearest Neighbor
  - Renieris+Reiss (ASE 2003)

- Intersection
  - Jones et al. (ICSE 2002)

Results obtained from Siemens test suite; cannot be generalized.
<table>
<thead>
<tr>
<th>Coverage</th>
<th>Sequences</th>
<th>Returns</th>
</tr>
</thead>
</table>

Properties
Sequences

Sequences of locations can correlate with failures:

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>open() read() close()</code></td>
<td>✔</td>
</tr>
<tr>
<td><code>open() close() read()</code></td>
<td>✘</td>
</tr>
<tr>
<td><code>close() open() read()</code></td>
<td>✘</td>
</tr>
</tbody>
</table>

…but all locations are executed in both runs!
$ ajc Test3.aj
$ java test.Test3

test.Test3@b8df17.x Unexpected Signal : 11 occurred at PC=0xFA415A00
Function name=(N/A) Library=(N/A) ...
Please report this error at http://java.sun.com/...
$

Coverage Differences

• Compare the failing run with passing runs
• BcelShadow.getThisJoinPointVar() is invoked in the failing run only
• Unfortunately, this method is correct
Sequence Differences

This *sequence* occurs only in the failing run:

\[
\langle \text{ThisJoinPointVisitor.isRef()}, \text{ThisJoinPointVisitor.canTreatAsStatic()}, \text{MethodDeclaration.traverse()}, \text{ThisJoinPointVisitor.isRef()}, \text{ThisJoinPointVisitor.isRef()} \rangle
\]

Defect location
Collecting Sequences

Trace

| mark | read | read | skip | read | read | skip | read |

Sequences

InputStream

<table>
<thead>
<tr>
<th>mark</th>
<th>read</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip</td>
<td>read</td>
</tr>
</tbody>
</table>

Sequence Set

<table>
<thead>
<tr>
<th>read</th>
<th>read</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>skip</td>
</tr>
<tr>
<td>skip</td>
<td>read</td>
</tr>
</tbody>
</table>
Ingoing vs. Outgoing

- **aProducer**
- **aConsumer**
- **aQueue**
- **aLinkedList**
- **aLogger**

- **incoming calls**
- **outgoing calls**

- add
- isEmpty
- get
- isEmpty
- add
- add
- add

- add
- size
- get
- size
- add
- add
- add
Anomalies

weights

passing run

0.5

1.0

0.5

failing run

0.5

0.5

1.0

0.5

passing run

0.5

1.0

0.5

ranking by average weight

0.60

0.50

0.40
NanoXML

- Simple XML parser written in Java
- 5 revisions, each with 16–23 classes
- 33 errors discovered or seeded
Locating Defects

AMPLE/window size 8
Dallmeier et al. (ECOOP 2005)

Results obtained from NanoXML; cannot be generalized

On average 0.5 classes less than window size 1
public boolean visit(MessageSend call, BlockScope scope) {
    Expression receiver = call.receiver;
    if (isRef(receiver, thisJoinPointDec)) {
        if (canTreatAsStatic(new String(call.selector))) {
            if (replaceEffectivelyStaticRefs) {
                replaceEffectivelyStaticRef(call);
            } else {
                //System.err.println("has static reg");
                hasEffectivelyStaticRef = true;
                if (call.arguments != null) {
                    int argumentsLength = call.arguments.length;
                    for (int i = 0; i < argumentsLength; i++) {
                        call.arguments[i].traverse(this, scope);
                    }
                }
            }
        }
        return false;
    }
    return super.visit(call, scope);
}

private MethodBinding getEquivalentStaticBinding(MethodBinding template) {
    ReferenceBinding b = (ReferenceBinding)thisJoinPointStaticPartDec.type;
    return b.getExactMethod(template.selector, template.parameters);
}

private void replaceEffectivelyStaticRef(MessageSend call) {

<table>
<thead>
<tr>
<th>Coverage</th>
<th>Sequences</th>
<th>Returns</th>
</tr>
</thead>
</table>

Properties
Detecting Anomalies

How do we collect data in the field?

Properties

Differences correlate with failure
Liblit’s Sampling

- We want properties of runs in the field
- Collecting all this data is too expensive
- Would a *sample* suffice?
- Sampling experiment by Liblit et al. (2003)
Return Values

- Hypothesis: *function return values* correlate with failure or success
- Classified into positive / zero / negative
CCRYPT fails

- CCRYPT is an interactive encryption tool
- When CCRYPT asks user for information before overwriting a file, and user responds with EOF, CCRYPT crashes
- 3,000 random runs
- Of 1,170 predicates, only `file_exists() > 0` and `xreadline() == 0` correlate with failure
Liblit’s Sampling

Properties

• Can we apply this technique to remote runs, too?
• 1 out of 1000 return values was sampled
• Performance loss <4%
After 3,000 runs, only five predicates are left that correlate with failure.
Web Services

- Sampling is first choice for web services
- Have 1 out of 100 users run an instrumented version of the web service
- Correlate instrumentation data with failure
- After sufficient number of runs, we can automatically identify the anomaly
Locating Defects

- NN (Renieris + Reiss, ASE 2003)
- CT (Cleve + Zeller, ICSE 2005)
- SD (Liblit et al., PLDI 2005)
- SOBER (Liu et al, ESEC 2005)

Results obtained from Siemens test suite; cannot be generalized.

- 5,542 runs
- 2 runs

% of failing tests vs. source code to examine

0%  <10%  <20%  <30%
Anomalies and Causes

• An anomaly is not a cause, but a correlation
• Although correlation \(\neq\) causation, anomalies can be excellent hints
• Future belongs to those who exploit
  • Correlations in *multiple runs*
  • Causation in *experiments*
Concepts

★ Comparing coverage (or other features) shows anomalies correlated with failure

★ Nearest neighbor or sequences locate errors more precisely than just coverage

★ Low overhead + simple to realize
Isolating Causes

Actual world

Alternate world

Mixed world

Test

Locating Defects

Failing run

<PLUS node>

<PLUS node>

Passing run

Detecting Anomalies

Properties

Properties

Differences correlate with failure

Mining Models

Failing Tests

<init> isEmpty() add() isEmpty() remove()

Differences correlate with failures

Passing Tests

<init> isEmpty() add() isEmpty() clear()
Isolating Causes

Actual world

Alternate world

Mixed world

Test

Failing Defects

Failing run

Passing run

Locating Defects

Detecting Anomalies

Properties

Differences correlate with failure

Mining Models

Failing Tests

Passing Tests

Differences correlate with failures
Dynamic Invariants

At \( f() \), \( x \) is odd

At \( f() \), \( x = 2 \)
Daikon

- Determines *invariants* from program runs
- Written by Michael Ernst et al. (1998–)
- C++, Java, Lisp, and other languages
- analyzed up to 13,000 lines of code
public int ex1511(int[] b, int n) {
    int s = 0;
    int i = 0;
    while (i != n) {
        s = s + b[i];
        i = i + 1;
    }
    return s;
}

- Run with 100 randomly generated arrays of length 7–13

**Precondition**
- \( n = \text{size}(b[]) \)
- \( b \neq \text{null} \)
- \( n \leq 13 \)
- \( n \geq 7 \)

**Postcondition**
- \( b[] = \text{orig}(b[]) \)
- return == sum(b)
Daikon

Run

get trace

filter invariants

Postcondition

\[ b[] = \text{orig}(b[]) \]
\[ \text{return} == \text{sum}(b) \]
Getting the Trace

- Records all variable values at all function entries and exits
- Uses VALGRIND to create the trace
Filtering Invariants

- Daikon has a library of *invariant patterns* over variables and constants
- Only matching patterns are preserved
Method Specifications

using *primitive data*

<table>
<thead>
<tr>
<th>$x = 6$</th>
<th>$x \in {2, 5, -30}$</th>
<th>$x &lt; y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = 5x + 10$</td>
<td>$z = 4x + 12y + 3$</td>
<td>$z = \text{fn}(x, y)$</td>
</tr>
</tbody>
</table>

using *composite data*

<table>
<thead>
<tr>
<th>A subseq B</th>
<th>$x \in A$</th>
<th>sorted(A)</th>
</tr>
</thead>
</table>

cHECKED AT METHOD ENTRY + EXIT
### Object Invariants

<table>
<thead>
<tr>
<th>Invariant</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>string.content[string.length] = '\0'</code></td>
</tr>
<tr>
<td><code>node.left.value ≤ node.right.value</code></td>
</tr>
<tr>
<td><code>this.next.last = this</code></td>
</tr>
</tbody>
</table>

checked at entry + exit of public methods
public int ex1511(int[] b, int n) {
    int s = 0;
    int i = 0;
    while (i != n) {
        s = s + b[i];
        i = i + 1;
    }
    return s;
}
## Matching Invariants

<table>
<thead>
<tr>
<th>Pattern Variables</th>
<th>s</th>
<th>size(b[])</th>
<th>sum(b[])</th>
<th>orig(n)</th>
<th>ret</th>
</tr>
</thead>
<tbody>
<tr>
<td>A == B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>✘</td>
<td></td>
<td>✘</td>
<td>✘</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td></td>
<td>✘</td>
<td></td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>size(b[])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum(b[])</td>
<td></td>
<td></td>
<td>✘</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orig(n)</td>
<td></td>
<td></td>
<td></td>
<td>✘</td>
<td></td>
</tr>
<tr>
<td>return</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✘</td>
</tr>
</tbody>
</table>

**Variables**

s  size(b[])  sum(b[])  orig(n)  n

**Run 1**

```
A == B
```
Matching Invariants

<table>
<thead>
<tr>
<th>Pattern Variables</th>
<th>A == B</th>
</tr>
</thead>
<tbody>
<tr>
<td>s size(b[])</td>
<td>✘</td>
</tr>
<tr>
<td>sum(b[])</td>
<td>✘</td>
</tr>
<tr>
<td>orig(n)</td>
<td>✘</td>
</tr>
<tr>
<td>n</td>
<td>✘</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>n</th>
<th>size(b[])</th>
<th>sum(b[])</th>
<th>orig(n)</th>
<th>ret</th>
</tr>
</thead>
<tbody>
<tr>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
</tbody>
</table>

Variable

run 2

s size(b[]) orig(n) return...
## Matching Invariants

### Pattern

\[
A == B
\]

### Variables

- \( s \)
- \( \text{size(b[])} \)
- \( \text{sum(b[])} \)
- \( \text{orig(n)} \)
- \( \text{ret} \)

### Table

<table>
<thead>
<tr>
<th>==</th>
<th>s</th>
<th>n</th>
<th>size(b[])</th>
<th>sum(b[])</th>
<th>orig(n)</th>
<th>ret</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>n</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>size(b[])</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>sum(b[])</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>orig(n)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

**run 3**
# Matching Invariants

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>size(b[])</th>
<th>sum(b[])</th>
<th>orig(n)</th>
<th>ret</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>n</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>size(b[])</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>sum(b[])</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>orig(n)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>ret</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

- $s == \text{sum}(b[])$
- $s == \text{ret}$
- $n == \text{size}(b[])$
- $\text{ret} == \text{sum}(b[])$
public int ex1511(int[] b, int n) {
    int s = 0;
    int i = 0;
    while (i != n) {
        s = s + b[i];
        i = i + 1;
    }
    return s;
}
Enhancing Relevance

• Handle polymorphic variables
• Check for derived values
• Eliminate redundant invariants
• Set statistical threshold for relevance
• Verify correctness with static analysis
• As long as some property can be observed, it can be added as a pattern
• Pattern vocabulary determines the invariants that can be found ("sum()", etc.)
• Checking all patterns (and combinations!) is expensive
• Trivial invariants must be eliminated
Why mine executions?

- We can also mine other program code!
Program

Usage Models

iter.hasNext ()
iter.next ()

Temporal Properties

hasNext < next
hasNext < hasNext
next < hasNext
next < next

Anomalies

hasNext < next ✓
hasNext < hasNext ✓

Patterns

hasNext < next
hasNext < hasNext
OP-Miner

**Usage Models**

iter.hasNext ()  iter.next ()

**Temporal Properties**

hasNext ≺ next
hasNext ≺ hasNext
next ≺ hasNext
next ≺ next

**Anomalies**

hasNext < next
✓ hasNext < hasNext
✓ hasNext < hasNext
✗ hasNext < next
✗ hasNext ≺ hasNext

**Patterns**

hasNext < next
hasNext < hasNext
public Stack createStack () {
    Random r = new Random ();
    int n = r.nextInt ();
    Stack s = new Stack ();
    int i = 0;
    while (i < n) {
        s.push (rand (r));
        i++;
    }
    s.push (-1);
    return s;
}
public Stack createStack () {
    Random r = new Random ();
    int n = r.nextInt ();
    Stack s = new Stack ();
    int i = 0;
    while (i < n) {
        s.push (rand (r));
        i++;
    }
    s.push (-1);
    return s;
}
public Stack createStack () {
    Random r = new Random ();
    int n = r.nextInt ();
    Stack s = new Stack ();
    int i = 0;
    while (i < n) {
        s.push (rand (r));
        i++;
    }
    s.push (-1);
    return s;
}
public Stack createStack () {
    Random r = new Random ();
    int n = r.nextInt ();
    Stack s = new Stack ();
    int i = 0;
    while (i < n) {
        s.push (rand (r));
        i++;
    }
    s.push (-1);
    return s;
}
public Stack createStack () {
    Random r = new Random ();
    int n = r.nextInt ();
    Stack s = new Stack ();
    int i = 0;
    while (i < n) {
        s.push (rand (r));
        i++;
    }
    s.push (-1);
    return s;
}
public Stack createStack () {
    Random r = new Random ();
    int n = r.nextInt ();
    Stack s = new Stack ();
    int i = 0;
    while (i < n) {
        s.push (rand (r));
        i++;
    }
    s.push (-1);
    return s;
}
Random r = new Random();
int n = r.nextInt();
Stack s = new Stack();
int i = 0;

i < n
s.push (-1);

i < n
s.push (rand (r));

i++;

Usage Models
Usage Models

Stack s = new Stack ();

s.push (-1);
s.push (rand (r));
Usage Models

s.<init>()

s.push (_)
Random r = new Random ();
int n = r.nextInt ();
Stack s = new Stack ();
int i = 0;

while (i < n) {
    s.push (-1);
    s.push (rand (r));
    i++;
}
Random r = new Random ();
int n = r.nextInt ();
s.push (rand (r));
Usage Models

```java
r.<init> ()

r.nextInt ()

Utils.rand (r)
```
JPanel.add()
ASTNode.reapPropertyList()
Resource.getFlags()
OP-Miner

Usage Models

iter.hasNext () iter.next ()

Temporal Properties

hasNext ≺ next
hasNext ≺ hasNext
next ≺ hasNext
next ≺ next

Anomalies

hasNext < next
✓ hasNext < hasNext

Patterns

hasNext < next
hasNext < hasNext
OP-Miner

**Usage Models**

```
iter.hasNext ()  iter.next ()
```

**Temporal Properties**

```
hasNext ≺ next
hasNext ≺ hasNext
next ≺ hasNext
next ≺ next
```

**Patterns**

```
hasNext ≺ next
hasNext ≺ hasNext
```

**Anomalies**

```
✓ hasNext ≺ next
✓ hasNext ≺ hasNext
✗ hasNext ≺ next
✗ hasNext≺= hasNext
```
## Methods vs. Properties

<table>
<thead>
<tr>
<th>Methods</th>
<th>Temporal Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>start (&lt;) lock (&lt;) eof (&lt;)</td>
</tr>
<tr>
<td>get()</td>
<td></td>
</tr>
<tr>
<td>open()</td>
<td></td>
</tr>
<tr>
<td>hello()</td>
<td></td>
</tr>
<tr>
<td>parse()</td>
<td></td>
</tr>
</tbody>
</table>
Methods vs. Properties

<table>
<thead>
<tr>
<th>Temporal Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>start (\prec)</td>
</tr>
<tr>
<td>stop</td>
</tr>
<tr>
<td>lock (\prec)</td>
</tr>
<tr>
<td>unlock</td>
</tr>
<tr>
<td>eof (\prec)</td>
</tr>
<tr>
<td>close</td>
</tr>
</tbody>
</table>

Methods

- get()
- open()
- hello()
- parse()
Discovering Anomalies

<table>
<thead>
<tr>
<th>Temporal Properties</th>
<th>start ≺ stop</th>
<th>lock ≺ unlock</th>
<th>eof ≺ close</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods</td>
<td>get()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>open()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>hello()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>parse()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The diagram shows the temporal properties of various methods, with 'hello()' having a marked constraint violation.
Discovering Anomalies

<table>
<thead>
<tr>
<th>Methods</th>
<th>Temporal Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>get()</td>
<td>start &lt; stop</td>
</tr>
<tr>
<td></td>
<td>lock &lt; unlock</td>
</tr>
<tr>
<td>open()</td>
<td>eof &lt; close</td>
</tr>
<tr>
<td>hello()</td>
<td></td>
</tr>
<tr>
<td>parse()</td>
<td></td>
</tr>
</tbody>
</table>
Case Study: AspectJ

- 2,954 classes
- 36,045 methods
- 1,154 methods with OP support $\geq 20$
- 300 violations found in 8 minutes
- Examined every single one
A Defect

```java
for (Iterator iter = itdFields.iterator();
     iter.hasNext();)
{
    ...
    for (Iterator iter2 = worthRetrying.iterator();
         iter.hasNext();)
    {
        ...
        should be iter2
    }
}
```
@interface A {}

aspect Test {
    declare @field : @A int var* : @A;
    declare @field : int var* : @A;

    interface Subject {}

    public int Subject.vara;
    public int Subject.varb;
}

class X implements Test.Subject {}
Another Defect

```java
public void visitNEWARRAY (NEWARRAY o) {
    byte t = o.getTypecode ();
    if (t != Constants.T_BOOLEAN ||
        t != Constants.T_CHAR ||
        ... ||
        t != Constants.T_LONG)) {
        constraintViolated (o, "(...)\'+t+\'(...)");
    }
}
```

should be double quotes
A False Positive

Name internalNewName (String[] identifiers)
...  
  for (int i = 1; i < count; i++) {
    SimpleName name = new SimpleName(this);
    name.internalSetIdentifier(identifiers[i]);
    ...  
  }
...
...

should stay as is
public String getRetentionPolicy ()
{
    ...
    for (Iterator it = ...; it.hasNext();)
    {
        ... = it.next();
        ...;
        return retentionPolicy;
    }
    ...
}
## More Results

<table>
<thead>
<tr>
<th>Program</th>
<th># Violations</th>
<th># Violations</th>
<th># Defects</th>
<th># Code Smells</th>
<th># False Positives</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT-RBot 0.8.2</td>
<td>25</td>
<td>25</td>
<td>2</td>
<td>13</td>
<td>10</td>
<td>60%</td>
</tr>
<tr>
<td>Apache Tomcat 6.0.16</td>
<td>55</td>
<td>55</td>
<td>0</td>
<td>9</td>
<td>46</td>
<td>16%</td>
</tr>
<tr>
<td>ArgoUML 0.24</td>
<td>305</td>
<td>28</td>
<td>0</td>
<td>12</td>
<td>16</td>
<td>43%</td>
</tr>
<tr>
<td>AspectJ 1.5.3</td>
<td>300</td>
<td>300</td>
<td>16</td>
<td>42</td>
<td>242</td>
<td>19%</td>
</tr>
<tr>
<td>Azureus 2.5.0.0</td>
<td>315</td>
<td>85</td>
<td>1</td>
<td>26</td>
<td>58</td>
<td>32%</td>
</tr>
<tr>
<td>Columba 1.2</td>
<td>57</td>
<td>57</td>
<td>4</td>
<td>15</td>
<td>38</td>
<td>33%</td>
</tr>
<tr>
<td>JEdit 4.2</td>
<td>11</td>
<td>11</td>
<td>0</td>
<td>4</td>
<td>7</td>
<td>36%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1,068</strong></td>
<td><strong>562</strong></td>
<td><strong>23</strong></td>
<td><strong>121</strong></td>
<td><strong>417</strong></td>
<td><strong>26%</strong></td>
</tr>
</tbody>
</table>
OP-Miner

★ OP-Miner learns *operational preconditions* 
  i.e., how to typically construct arguments

★ Learns from normal argument usage 
  for specific projects or across projects

★ Fully automatic

★ Found dozens of verified defects
A Guide to Systematic Debugging

WHY PROGRAMS FAIL

“A Guide to Systematic Debugging”
WALTER F. TICHY
TU Karlsruhe

On Display!

“The definitive book on debugging”

software development
16th annual productivity award

On Display!

ANDREAS ZELLER

MORGAN KAUFMANN
dpunkt.verlag
#1 in 
Software Engineering Books
Algorithms Books
Software Design Books
Isolating Causes

Actual world

Alternate world

Test

Mixed world

Locating Defects

Failing run

<PLUS node>

Passing run

<PLUS node>

Detecting Anomalies

Properties

Properties

Differences correlate with failure

Mining Models

Failing Tests

¬isEmpty

add()

¬isEmpty

isEmpty()

<init>

remove()

Differences correlate with failures

Passing Tests

¬isEmpty

add()

¬isEmpty

<init>

clear()

clear()