Defect Detection

Thomas Zimmermann
The First Bug
September 9, 1947

First actual case of bug being found.
More Bugs
More Bugs

The application Keynote has unexpectedly quit.
The system and other applications have not been affected.

Would you like to submit a bug report to Apple?

Submit Report…  Cancel
More Bugs

The application Keynote has unexpectedly quit.
The system and other applications have not been affected.
Would you like to submit a bug report to Apple?

Application "gedit" (process 25321) has crashed due to a fatal error.
(Segmentation fault)

Please visit the GNOME Application Crash page for more information
More Bugs

![Error messages from various applications showing crashes and bugs.](image)
More Bugs
More Bugs
More Bugs
More Bugs
More Bugs
More Bugs
More Bugs
More Bugs
Facts on Debugging

• Software bugs are costing ~60 bln US$/yr
• Improvements could reduce cost by 30%
• Validation (including debugging) can easily take up to 50-75% of the development time
• When debugging, some people are three times as efficient than others
How to Debug
(Sommerville 2004)

1. Locate error
2. Design error repair
3. Repair error
4. Re-test program
The Traffic Principle
The Traffic Principle
The Traffic Principle

rack the problem

T R A F F I C
The Traffic Principle

- Track the problem
- Reproduce
The Traffic Principle

Track the problem
Reproduce
Automate

T R A F F I C
The Traffic Principle

T rack the problem
R eproduce
A utomate
F ind Origins
C IC
The Traffic Principle

Track the problem
Reproduce
Automate
Find Origins
Focus

T R A F I C
The Traffic Principle

T rack the problem
R eproduce
A utomate
F ind Origins
F ocus
I solate
C
The Traffic Principle

T rack the problem
R eproduce
A utomate
F ind Origins
F ocus
I solate
C orrect
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*.

**From Defect to Failure**

<table>
<thead>
<tr>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>t</td>
</tr>
</tbody>
</table>
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.
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*.
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*. 

From Defect to Failure
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.
• Not every defect causes a failure!

• *Testing can only show the presence of errors — not their absence.*

(Dijkstra 1972)
• Every failure can be traced back to some infection, and every infection is caused by some defect.

• Debugging means to relate a given failure to the defect – and to remove the defect.
Defect detection

Program → Tool → List of defects
Defect localization

Program + Failure → Tool → Location (Defect)
## Outline

<table>
<thead>
<tr>
<th>Redundancies</th>
<th>FindBugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Engler et al.)</td>
<td>(Pugh et al.)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Models</th>
<th>Statistical</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Invited talk: A. Wasylkowski)</td>
<td>(Liblit et al.)</td>
</tr>
</tbody>
</table>
Dawson Engler
Co-founder of Coverity

redundancies
flag errors
Idempotent operations

Flag idempotent operations.

- variable is assigned to itself: \( x = x \)
- variable is divided by itself: \( x / x \)
- variable is bitwise or’d with itself: \( x | x \)
- variable is bitwise and’d with itself: \( x & x \)
Idempotent operations

/* 2.4.1/net/appletalk/aarp.c:aarp_rcv */

... else { /* We need to make a copy of the entry. */
    da.s_node = sa.s_node;
    da.s_net = da.s_net;
    ...

System | Bugs | Minor | False
-------|------|-------|-------
Linux   | 7    | 6     | 3     
Redundant assignments

Flag cases where a value assigned to a variable is subsequently not used.

<table>
<thead>
<tr>
<th>System</th>
<th>Bugs</th>
<th>False</th>
<th>Uninspected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>129</td>
<td>26</td>
<td>1840</td>
</tr>
<tr>
<td>xgcc</td>
<td>13</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
do {
    ...
    if (signal_pending(current)) {
        err = -ERESTARTSYS;
        break;
    }
    SOCK_SLEEP_PRE(sk)
    if (scp->state != DN_RUN)
        schedule();
    SOCK_SLEEP_POST(sk)
} while(scp->state != DN_RUN);
return 0;
/* 2.4.1/net/decnet/af_decnet.c:dn_wait_run */
do {
    ...
    if (signal_pending(current)) {
        err = -ERESTARTSYS; /* BUG: lost value */
        break;
    }
    SOCK_SLEEP_PRE(sk)
    if (scp->state != DN_RUN)
        schedule();
    SOCK_SLEEP_POST(sk)
} while(scp->state != DN_RUN);
return 0;

/* 2.4.1/net/decnet/af_decnet.c:dn_wait_run */
Redundant assignments

/* 2.4.1/net/atm/lec.c:lec_addr_delete: */

for (entry=priv->lec_arp_tables[i];
    entry != NULL;
    entry=next) {
    next = entry->next;
    if (...) {
        lec_arp_remove(priv->lec_arp_tables, entry);
        kfree(entry);
    }
    lec_arp_unlock(priv);
    return 0;
}
for(entry=priv->lec_arp_tables[i];
  entry != NULL;
  entry=next) { /* BUG: never reached */
  next = entry->next;
  if (...) {
    lec_arp_remove(priv->lec_arp_tables, entry);
    kfree(entry);
  }
  lec_arp_unlock(priv);
  return 0;
}
Dead code

Flag dead code (i.e., code that is never executed).

<table>
<thead>
<tr>
<th>System</th>
<th>Bugs</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>66</td>
<td>26</td>
</tr>
</tbody>
</table>
for (cnt = 0; cnt < min(name1_len, name2_len); ++cnt) {
    c1 = le16_to_cpu(*name1++);
    c2 = le16_to_cpu(*name2++);
    if (ic) {
        if (c1 < upcase_len)
            c1 = le16_to_cpu(upcase[c1]);
        if (c2 < upcase_len)
            c2 = le16_to_cpu(upcase[c2]);
    }

    if (c1 < 64 && legal_ansi_char_array[c1] & 8);
    return err_val;
    if (c1 < c2)
        return -1;
...

for (cnt = 0; cnt < min(name1_len, name2_len); ++cnt) {
    c1 = le16_to_cpu(*name1++);
    c2 = le16_to_cpu(*name2++);
    if (ic) {
        if (c1 < upcase_len)
            c1 = le16_to_cpu(upcase[c1]);
        if (c2 < upcase_len)
            c2 = le16_to_cpu(upcase[c2]);
    }
    /* [META] stray terminator! */
    if (c1 < 64 && legal_ansi_char_array[c1] & 8)
        return err_val;
    if (c1 < c2)
        return -1;
...
Redundant conditionals

Flag redundant branch conditionals from
(1) branch statements with non-constant conditionals
that always evaluate to either true or false
(2) switch statements with impossible cases

<table>
<thead>
<tr>
<th>System</th>
<th>Bugs</th>
<th>False</th>
<th>Uninspected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>49</td>
<td>52</td>
<td>169</td>
</tr>
</tbody>
</table>
Redundant conditionals

if ((login_state == NODE_LOGGED_IN) ||
    (login_state == NODE_PROCESS_LOGGED_IN)) {
    ...
}
else
if (login_state == NODE_LOGGED_OUT)
    tx_adisc(fi, ELS_ADISC, node_id,
              OX_ID_FIRST_SEQUENCE);
else /* BUG: redundant conditional */
if (login_state == NODE_LOGGED_OUT)
    tx_logi(fi, ELS_PLOGI, node_id);
Redundant conditionals

```c
/* 2.4.1/drivers/fc/iph5526.c: rscn_handler */

if ((login_state == NODE_LOGGED_IN) ||
    (login_state == NODE_PROCESS_LOGGED_IN)) {
    ...
}
else
    if (login_state == NODE_LOGGED_OUT)
        tx_adisc(fi, ELS_ADISC, node_id,
                 OX_ID_FIRST_SEQUENCE);
else /* BUG: redundant conditional */
    if (login_state == NODE_LOGGED_OUT)
        tx_logi(fi, ELS_PLOGI, node_id);

Overly cautious programming style (confused programmer)

Single iteration loop

Cut-and-paste errors
```
Correlation to hard bugs

Hard bugs can crash a system (use of freed memory, dereferences of null pointers, potential deadlocks, unreleased locks, and security violations)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>$o_{11}$</td>
<td>$o_{1.}$</td>
</tr>
<tr>
<td>False</td>
<td>$o_{21}$</td>
<td>$o_{2.}$</td>
</tr>
<tr>
<td>Totals</td>
<td>$n_{1}$</td>
<td>$n_{2}$</td>
</tr>
</tbody>
</table>

Chi-Square test

Null hypothesis: “A and B are mutually independent”
Correlation to hard bugs

<table>
<thead>
<tr>
<th>Redundant Assignments</th>
<th>Hard Bugs</th>
<th></th>
<th>Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>345</td>
<td>435</td>
<td>780</td>
</tr>
<tr>
<td>No</td>
<td>206</td>
<td>1069</td>
<td>1275</td>
</tr>
<tr>
<td>Totals</td>
<td>551</td>
<td>1504</td>
<td>2055</td>
</tr>
</tbody>
</table>

\[ T = 194.37, \ p\text{-value} = 0.00 \]
Correlation to hard bugs

<table>
<thead>
<tr>
<th>Redundant Assignments</th>
<th>Hard Bugs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Totals</td>
</tr>
<tr>
<td>Yes</td>
<td>345</td>
<td>435</td>
<td>780</td>
</tr>
<tr>
<td>No</td>
<td>351</td>
<td>1504</td>
<td>1855</td>
</tr>
<tr>
<td>Totals</td>
<td>596</td>
<td>1939</td>
<td>2525</td>
</tr>
</tbody>
</table>

\[ T = 194.37, \ p\text{-value} = 0.00 \]
Correlation to hard bugs

<table>
<thead>
<tr>
<th>Redundant Assignments</th>
<th>Hard Bugs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Totals</td>
</tr>
<tr>
<td>Yes</td>
<td>345</td>
<td>485</td>
<td>830</td>
</tr>
<tr>
<td>No</td>
<td>265</td>
<td>165</td>
<td>1275</td>
</tr>
<tr>
<td>Totals</td>
<td>551</td>
<td>1504</td>
<td>2055</td>
</tr>
</tbody>
</table>

\[ T = 194.37, \ p\text{-value} = 0.00 \]

<table>
<thead>
<tr>
<th>Redundant Conditionals</th>
<th>Hard Bugs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Totals</td>
</tr>
<tr>
<td>Yes</td>
<td>75</td>
<td>49</td>
<td>124</td>
</tr>
<tr>
<td>No</td>
<td>475</td>
<td>1423</td>
<td>1901</td>
</tr>
<tr>
<td>Totals</td>
<td>551</td>
<td>1504</td>
<td>2055</td>
</tr>
</tbody>
</table>

\[ T = 40.65, \ p\text{-value} = 0.00 \]
Correlation to hard bugs

<table>
<thead>
<tr>
<th>Redundant Assignments</th>
<th>Hard Bugs</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>345</td>
<td>435</td>
<td>780</td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>265</td>
<td>1065</td>
<td>1275</td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td>551</td>
<td>1504</td>
<td>2055</td>
<td></td>
</tr>
</tbody>
</table>

\[ T = 194.37, \ p\text{-value} = 0.00 \]

<table>
<thead>
<tr>
<th>Redundant Conditionals</th>
<th>Hard Bugs</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>75</td>
<td>9</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>475</td>
<td>1425</td>
<td>1901</td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td>551</td>
<td>1504</td>
<td>2055</td>
<td></td>
</tr>
</tbody>
</table>

\[ T = 40.65, \ p\text{-value} = 0.00 \]

<table>
<thead>
<tr>
<th>Dead Code</th>
<th>Hard Bugs</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>133</td>
<td>115</td>
<td>248</td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>131</td>
<td>1309</td>
<td>1787</td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td>551</td>
<td>1504</td>
<td>2055</td>
<td></td>
</tr>
</tbody>
</table>

\[ T = 81.74, \ p\text{-value} = 0.00 \]
Correlation to hard bugs

<table>
<thead>
<tr>
<th>Redundant Assignments</th>
<th>Hard Bugs</th>
<th>Yes</th>
<th>No</th>
<th>Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>345</td>
<td>485</td>
<td></td>
<td>830</td>
</tr>
<tr>
<td>No</td>
<td>200</td>
<td>1063</td>
<td></td>
<td>1275</td>
</tr>
<tr>
<td>Totals</td>
<td>551</td>
<td>1504</td>
<td></td>
<td>2055</td>
</tr>
</tbody>
</table>

$T = 194.37$, $p$-value = 0.00

<table>
<thead>
<tr>
<th>Redundant Conditionals</th>
<th>Hard Bugs</th>
<th>Yes</th>
<th>No</th>
<th>Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>75</td>
<td>19</td>
<td></td>
<td>94</td>
</tr>
<tr>
<td>No</td>
<td>175</td>
<td>1225</td>
<td></td>
<td>1901</td>
</tr>
<tr>
<td>Totals</td>
<td>551</td>
<td>1504</td>
<td></td>
<td>2055</td>
</tr>
</tbody>
</table>

$T = 40.65$, $p$-value = 0.00

<table>
<thead>
<tr>
<th>Dead Code</th>
<th>Hard Bugs</th>
<th>Yes</th>
<th>No</th>
<th>Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>133</td>
<td>135</td>
<td></td>
<td>268</td>
</tr>
<tr>
<td>No</td>
<td>173</td>
<td>1309</td>
<td></td>
<td>1787</td>
</tr>
<tr>
<td>Totals</td>
<td>551</td>
<td>1504</td>
<td></td>
<td>2055</td>
</tr>
</tbody>
</table>

$T = 81.74$, $p$-value = 0.00

<table>
<thead>
<tr>
<th>Aggregate</th>
<th>Hard Bugs</th>
<th>Yes</th>
<th>No</th>
<th>Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>372</td>
<td>535</td>
<td></td>
<td>907</td>
</tr>
<tr>
<td>No</td>
<td>796</td>
<td>1033</td>
<td></td>
<td>1110</td>
</tr>
<tr>
<td>Totals</td>
<td>551</td>
<td>1504</td>
<td></td>
<td>2055</td>
</tr>
</tbody>
</table>

$T = 140.48$, $p$-value = 0.00
Eclipse
FindBugs

Bill Pugh
FindBugs: Bug patterns

FindBugs recognizes 284 different bug patterns

http://findbugs.sourceforge.net/
bugDescriptions.html
FindBugs: Infinite Loops

- Students are good bug generators:
  public WebSpider() {
    WebSpider w = new WebSpider();
  }

- Five infinite loops in JDK1.6.0-b13, 27 across all versions of JDK, 31 in Google’s Java code

Use of history

- **Track warnings across releases**
  Jaime Spacco, David Hovemeyer, William Pugh: Tracking defect warnings across versions. MSR 2006: 133-136

- **Rank warnings with historic data**

  Sunghun Kim, Michael D. Ernst: "Which Warnings Should I Fix First?" ESEC/FSE 2007, to appear
Meet & Greet

Andrzej Wasylkowski

Slides will be available on the lecture web-page.
Statistical bug isolation

Ben Liblit

References

- Yichen Xie, Dawson R. Engler: Using redundancies to find errors. SIGSOFT FSE 2002: 51-60