Reducing Testing Effort with Static Code Verification

유용출 과장 (gary.ryu@mathworks.com)
Agenda

- Misconceptions lead Testing effort high
- Polyspace helps you to reduce testing effort
- Conclusion
Misconceptions lead Testing Efforts High

1. Our expert programmers don’t make mistakes

2. I need Dynamic Testing only, I do not need Static Analysis

3. I do Static Analysis as the last quality gate before release
1. Our expert programmers don’t make mistakes

- They are infallible! Aren’t they?

- Look at this table,

<table>
<thead>
<tr>
<th>Group</th>
<th>Average Bugs per KLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 1% developers</td>
<td>11.2</td>
</tr>
<tr>
<td>Top 10% developers</td>
<td>28.9</td>
</tr>
<tr>
<td>Top 25% developers</td>
<td>61.9</td>
</tr>
<tr>
<td>All</td>
<td>120.8</td>
</tr>
</tbody>
</table>

Defect Injection Ranges for 810 Experienced Software Developers
The Ganssle Group, Derived from articles written by Watts Humphrey (Father of Software Quality)
1. Our expert programmers don’t make mistakes

- They are infallible! Aren’t they?

- Look at this table,

<table>
<thead>
<tr>
<th></th>
<th>Defect Injection Ranges for 810 Experienced Software Developers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The Ganssle Group, Derived from articles written by Watts Humphrey (Father of Software Quality)</td>
</tr>
<tr>
<td>Top 1% developers</td>
<td>11.2</td>
</tr>
<tr>
<td>Top 10% developers</td>
<td>28.9</td>
</tr>
<tr>
<td>Top 25% developers</td>
<td>61.9</td>
</tr>
<tr>
<td>All</td>
<td>120.8</td>
</tr>
</tbody>
</table>

Finding Bugs with Right Tool is Must-Do-activity!
2. I need Dynamic Testing only, I do not need Static Analysis

- I do sufficient testing: Unit testing, Integration testing, Field testing …
- It’s redundant to do both of dynamic testing and static analysis

### DEFECT REMOVAL EFFICIENCY CASE 4 (Worst)

**No Inspections; No static analysis**

**DEVELOPMENT DEFECTS REMOVED**

<table>
<thead>
<tr>
<th>Type</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static analysis</td>
<td>0</td>
</tr>
<tr>
<td>Inspections</td>
<td>0</td>
</tr>
<tr>
<td>Testing</td>
<td>850</td>
</tr>
<tr>
<td><strong>Subtotal</strong></td>
<td>850</td>
</tr>
</tbody>
</table>

**USER-REPORTED DEFECTS IN FIRST 90 DAYS**

<table>
<thead>
<tr>
<th>Type</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid unique defects</td>
<td>150</td>
</tr>
</tbody>
</table>

**TOTAL DEFECT VOLUME**

<table>
<thead>
<tr>
<th>Type</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defect totals</td>
<td>1,000</td>
</tr>
</tbody>
</table>

**DEFECT REMOVAL EFFICIENCY**

\[
\frac{\text{Dev. (850)}}{\text{Total (1,000)}} = 85.0\%
\]

### DEFECT REMOVAL EFFICIENCY CASE 1 (BEST)

**Inspections + static analysis + testing**

**DEVELOPMENT DEFECTS REMOVED**

<table>
<thead>
<tr>
<th>Type</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static analysis</td>
<td>350</td>
</tr>
<tr>
<td>Inspections</td>
<td>390</td>
</tr>
<tr>
<td>Testing</td>
<td>250</td>
</tr>
<tr>
<td><strong>Subtotal</strong></td>
<td>990</td>
</tr>
</tbody>
</table>

**USER-REPORTED DEFECTS IN FIRST 90 DAYS**

<table>
<thead>
<tr>
<th>Type</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid unique defects</td>
<td>10</td>
</tr>
</tbody>
</table>

**TOTAL DEFECT VOLUME**

<table>
<thead>
<tr>
<th>Type</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defect totals</td>
<td>1,000</td>
</tr>
</tbody>
</table>

**DEFECT REMOVAL EFFICIENCY**

\[
\frac{\text{Dev. (990)}}{\text{Total (1,000)}} = 99.0\%
\]
Quiz: How many tests to achieve 100% MCDC

```
#include "simple_lookup_table.h"

const double x[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
const double y[10] = {3, 6, 9, 12, 15, 18, 21, 24, 27, 30};

double simple_lookup_tbl (double in1, sint8_t * stat)
{
    double out;
    double f;
    uint32_t i;

    if (in1 < x[0])
    {
        /* lower saturation */
        out = y[0];
        *stat = -1;
    }
    else if (in1 > x[9])
    {
        /* upper saturation */
        out = y[9];
        *stat = -1;
    }
    else
    {
        for (i = 4; in1 < x[i]; i--);
        if (i > 2 && in1 > 5.0F)
        {
            f = 1.0F;
        }
        else
        {
            for (i = 4; in1 < x[i]; i--);
            if (i > 2 && in1 > 5.0F)
            {
                f = 1.0F;
            }
        }
        out = (y[i + 1U] - y[i]) * f + y[i];
        *stat = 0;
    }
    return out;
}
```

- Automatically-generated test for input = 5.9566, 1.9184, -5.4067, 9.8142, 11, 4

Answer: 6 tests
Let’s see the coverage report of the hand-written code

Don’t we miss something dangerous?
Runtime error lurking in the code

Polyspace Code Prover finds this runtime error
- Array \( x \) allocated for 10 elements
- Range of index is 1 .. 10
- Potential to access \( x[10] \), out of bounds array

Improve SW quality with Polyspace!
3. I do Static Analysis as the last quality gate before release

- Efficient Defect Reduction
  - Cost less in early stage

- Accumulated Technical Debt
  - Too many things to review or fix

![Software Development Lifecycle Phase](image)

**HOW QUALITY AFFECTS SOFTWARE COSTS**

3. I do Static Analysis as the last quality gate before release

- **Efficient Defect Reduction**
  - Cost less in early stage

- **Accumulated Technical Debt**
  - Too many things to review or fix

![Effort & Cost Graph](image)

**EARLY & OFTEN!**

---

Lessons learned from misconceptions

Verify Software
with Right tools
Early & Often
Lessons learned from misconceptions
Polyspace helps you to …

**Bug Finder**  
- High Quality, Secure, Compliant Code:  
  - Various defects or vulnerabilities  
  - Credits for functional safety, cybersecurity standards.

**Code Prover**  
- Fully Trusted Components:  
  - Proven free of critical runtime defects and vulnerabilities  
  - Additional credits for standards.
Save-time analysis workflow in Eclipse
Unit-proving of AUTOSAR components

- Detect runtime errors early in AUTOSAR design process
- Detect inconsistencies between AUTOSAR specification and Source code
Unit-proving of AUTOSAR components
Ease of configuring projects

- **Create project automatically** (DIAB, TASKING, GreenHills, IAR, CodeWarrior, TI CCS, GCC, Visual Studio)
When/Who using Polyspace products?
Conclusion

<table>
<thead>
<tr>
<th>Group</th>
<th>Avg Bugs per KLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 1% developers</td>
<td>11.2</td>
</tr>
<tr>
<td>Top 10% developers</td>
<td>28.9</td>
</tr>
<tr>
<td>Top 25% developers</td>
<td>61.9</td>
</tr>
<tr>
<td>All</td>
<td>120.8</td>
</tr>
</tbody>
</table>

- Efficient Defect Reduction
  - Cost less in early stage

- Accumulated Technical Debt
  - Too many things to review or fix

Verify Software with Right tools Early & Often