MATLAB EXPO 2018
KOREA
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>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>

Finding Bugs with Right Tool is Must-Do-activity!

Defect Injection Ranges for 810 Experienced Software Developers
The Ganssle Group, Derived from articles written by Watts Humphrey (Father of Software Quality)
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

### DEFACT REMOVAL EFFICIENCY CASE 4 (Worst)

- No Inspections; No static analysis
- Testing Only

#### DEVELOPMENT DEFECTS REMOVED

- Static analysis: 0
- Inspections: 0
- Testing: 850
- Subtotal: 850

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

- Valid unique defects: 150

#### TOTAL DEFECT VOLUME

- Defect totals: 1,000

#### DEFACT REMOVAL EFFICIENCY

- Dev. (850) / Total (1,000) = 85.0%

### DEFACT REMOVAL EFFICIENCY CASE 1 (BEST)

- Inspections + static analysis + testing

#### DEVELOPMENT DEFECTS REMOVED

- Static analysis: 350
- Inspections: 390
- Testing: 250
- Subtotal: 990

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

- Valid unique defects: 10

#### TOTAL DEFECT VOLUME

- Defect totals: 1,000

#### DEFACT REMOVAL EFFICIENCY

- Dev. (990) / Total (1,000) = 99.0%
Quiz: How many tests to achieve 100% MCDC

- 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

---

**HOW QUALITY AFFECTS SOFTWARE COSTS**

![Diagram showing cost variations across software development lifecycle phases]

- **Reduce effort & cost**
- Poor quality vs. High quality

---

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

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?

- Specification
- Design
- Implementation
- SW Unit Tests
- SW Integration tests
- SW Acceptance tests

{ DEV } { QE }
{ DEV } { QA }
{ DEV } { QE }
{ QE }
{ DEV }
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