

MathWorks
**AUTOMOTIVE
CONFERENCE 2023**
North America

Find C/C++ Bugs as You Code Integrate Polyspace into Your IDE

John Boyd, MathWorks

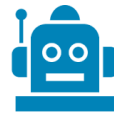


Comprehensive static analysis tools for increased efficiency



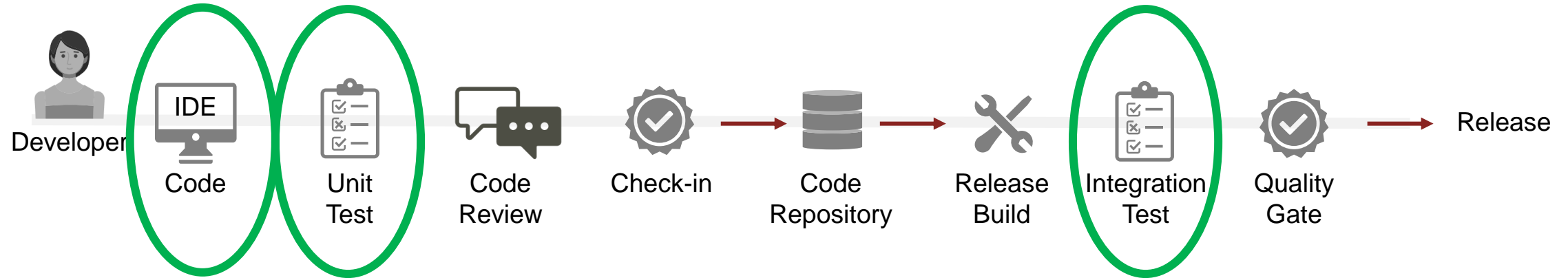
Catch and fix bugs
while you code

Automate with CI
Workflows



Collaborate with
Team Members

Leverage static analysis components at critical points in the workflow



1. Repeatability

2. Faster delivery

3. Higher quality

Successful use of tool suite at Volvo

Volvo Cars Software Factory Increases Pace and Quality of Development with Polyspace

“With Polyspace, we can ensure software security and quality by identifying and fixing critical run-time errors before every code merge.”

— Johannes Foufas, Volvo Cars

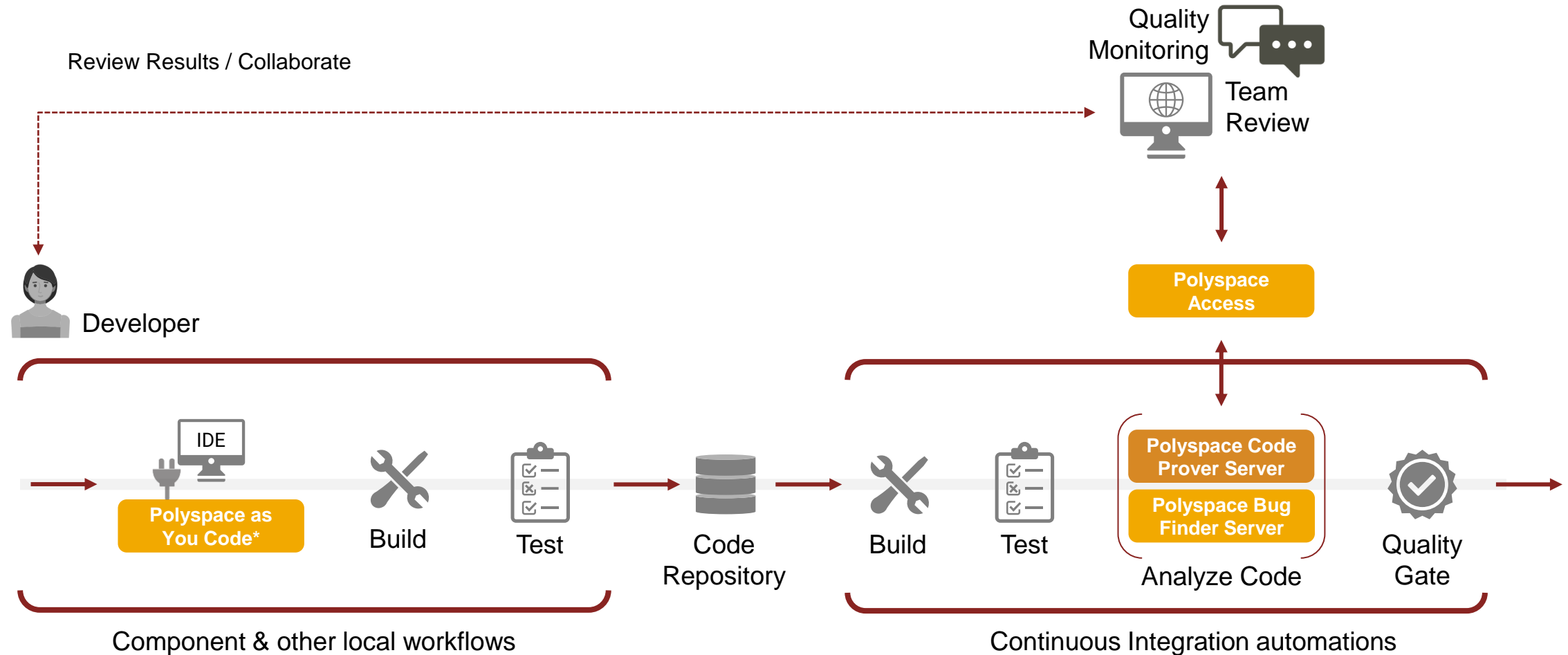


Volvo Cars uses Polyspace for static code checking throughout the development lifecycle.

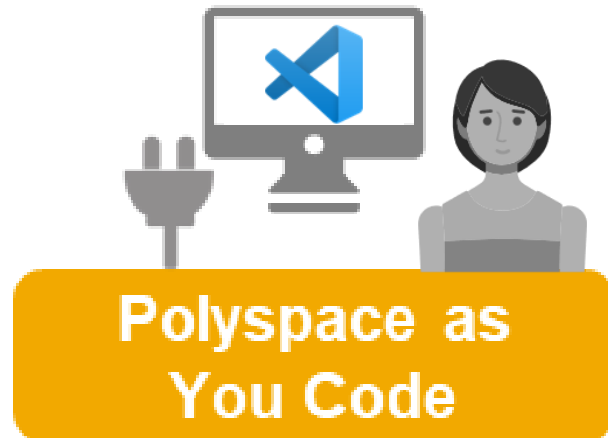
With the Polyspace as You Code plugin available in Polyspace Access™, several teams check adherence to CERT® C, CERT C++, MISRA C®, and AUTOSAR C++14 coding guidelines while they are coding in their IDEs. Before submitting their code modifications, developers run Polyspace Bug Finder™ and Polyspace Code Prover™ on their local computers to prequalify their changes.

When developers push their changes to the source code repository, it automatically triggers Polyspace Bug Finder Server™ and Polyspace Code Prover Server™ analysis. The Polyspace results are integrated into Gerrit to support code reviews. The CI system employs strict gating: every proposed change is verified before a code merge and is promoted into the central Git™ repository only if it meets safety and security requirements.

Polyspace as You Code: static analysis integrated in your environment



First opportunity to fix bugs...



Also supported:



and custom integrations

- ...on demand.
- ...before committing.
- ...before running tests.
- ...while you remember the code.
- ...when it's easiest.
- ...when it's least expensive.

+ Help develop good habits

Find Bugs and Enforce Coding Standards



Defect Types

- ❖ Numerical
- ❖ Tainted Data
- ❖ Security
- ❖ Cryptography
- ❖ Data Flow
- ❖ Concurrency
- ❖ Static Memory
- ❖ Dynamic Memory
- ❖ Good Practice
- ❖ Performance
- ❖ Resource Mgmt.
- ❖ Programming



Coding Standards

- ❖ MISRA C:2004
- ❖ MISRA C:2012
- ❖ CERT C
- ❖ MISRA C++:2008
- ❖ AUTOSAR C++-14
- ❖ CERT C++
- ❖ Naming Rules
- ❖ JSF AV C++
- ❖ ISO/IEC TS 17961

Guidelines checks for software metrics

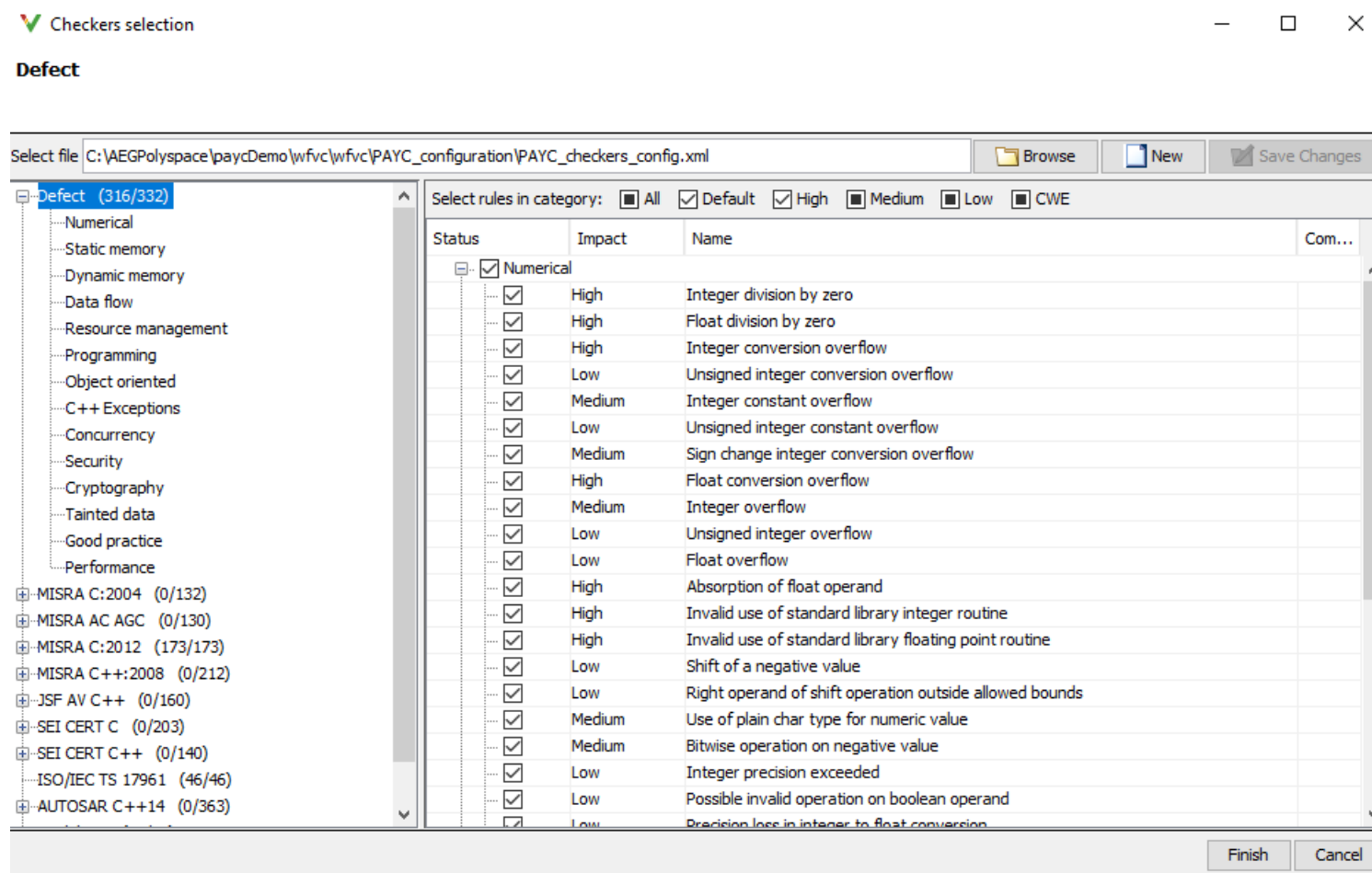


Guidelines

- ❖ Complexity
- ❖ Recursions
- ❖ Language Scope
- ❖ Function Coupling
- ❖ HIS
- ❖ Paths, Inputs, Calls
- ❖ Project
- ❖ File
- ❖ Function

Configuration

- File based configuration can be centralized and shared across teams
- Can also import from Polyspace Desktop or use options text files



Polyspace as You Code for all the C[++] code you write



Monitor findings for key files as you code

See finding details with a traceback

Also supported:



and custom integrations

File Edit Selection View Go Run Terminal Help

Polyspace

QUALITY MONITORING

- sut.c src 9+
- test_driver.c src 3

RESULT DETAILS

- ARR38-C Guarantee that library fu...
 - sut.c [77, 5]
 - Data size argument to 'memcpy' is n...
 - SEI CERT C:ARR38-C
 - Traceback

BASELINE

- Mode: Show local findings only

CONFIGURATION

- Build setting: Build options file not req...
- Build options generation not needed
- Checkers file: my_payc_default.xml

```

src > C sut.c > cpy_data(BUF_MEM *)
63     size_t max; /* size of buffer */
64     } BUF_MEM;
65
66     extern BUF_MEM beta;
67
68     int cpy_data(BUF_MEM *alpha)
69     {
70         BUF_MEM *os = alpha;
71         int num, length;
72
73         if (alpha == 0x0) return 0;
74         num = 0;
75
76         length = *(unsigned short *)os->
77         memcpy(8(beta.data[num]), os->data
78
79         return(1);
80     }
81
82     void absolute_difference(int32_t x, int32_t y);
83     void absolute_difference(int32_t x, int32_t y)
84     {
85         int32_t lx;
86         if (x > y)
87         {

```

PROBLEMS 52

- DCL00-C Const-qualify immutable objects polyspace(SEI CERT C:DCL00-C) [Ln 70, Col 14]
- INT00-C Understand the data model used by your implementation(s) polyspace(SEI CERT C:INT00-C) [Ln 71, Col 5]
- EXP19-C Use braces for the body of an if, for, or while statement polyspace(SEI CERT C:EXP19-C) [Ln 73, Col 5]
- INT00-C Understand the data model used by your implementation(s) polyspace(SEI CERT C:INT00-C) [Ln 76, Col 16]
- EXP39-C Do not access a variable through a pointer of an incompatible type polyspace(SEI CERT C:EXP39-C) [Ln 76, Col 35]
- ARR38-C Guarantee that library functions do not form invalid pointers polyspace(SEI CERT C:ARR38-C) [Ln 77, Col 5]
- Non-initialized variable polyspace(Defect:NON_INIT_VAR) [Ln 134, Col 8]

Learn about checks, why they matter, and examples with fixes

Easy shortcuts for details and in-code justification

CERT C: Rule ARR38-C

Guarantee that library functions do not form invalid pointers

Description

Rule Definition

Guarantee that library functions do not form invalid pointers.¹

Polyspace Implementation

This checker checks for these issues:

- Mismatch between data length and size.
- Invalid use of standard library memory routine.
- Possible misuse of sizeof.
- Buffer overflow from incorrect string format specifier.
- Invalid use of standard library string routine.
- Destination buffer overflow in string manipulation.
- Destination buffer underflow in string manipulation.

Examples

- Mismatch between data length and size
 - Issue
 - Mismatch between data length and size looks for memory copying functions such as memcpy, memcpy_s, and strncpy. These functions do not match the length argument and data buffer argument properly. Bug Finder raises a defect.
 - Risk
 - If an attacker can manipulate the data buffer or length argument, the attacker can cause buffer overflow. If the actual data size is smaller than the length, the attacker can copy memory past the data buffer to a new location. This mismatch in length allows the attacker to copy memory past the data buffer to a new location. This new location contains sensitive information, the attacker can now access that data.
 - This defect is similar to the SSL Heartbleed bug.
 - Fix
 - When copying or manipulating memory, compute the length argument directly from the data so that the length argument matches the data buffer argument.
 - Example - Copy Buffer of Data

```

#include <stdlib.h>
#include <string.h>

typedef struct buf_mem_st {
    char *data;
    size_t max; /* size of buffer */
} BUF_MEM;

extern BUF_MEM beta;

int cpy_data(BUF_MEM *alpha)
{
    BUF_MEM *os = alpha;

```

Fast local analysis on save and on-demand

The screenshot shows an IDE window with a C code editor. The code is as follows:

```
91 (overall_
92 (void)printf("-----
93 if (overall_status ==
94 {
95     for (i = 0; i < LARC
96     {
97         printf("LargeArray
98     }
99     printf("\n"); /* po
100 }
101
102 return 1;
```

A context menu is open over the code, listing various actions. The 'Run Polyspace Analysis' option, with the keyboard shortcut Ctrl+Shift+Alt+A, is highlighted. Below the code editor, the 'PROBLEMS' panel shows four error messages for the file 'test_driver.c':

- Vulnerable pseudo-random num
- Dead code polyspace(Defect:DE
- 17.7 The value returned by a fur
- 17.7 The value returned by a fur

Two callout boxes provide additional information:

- A callout pointing to the 'Run Polyspace Analysis' option in the context menu says: "Run ad-hoc analysis when you need it".
- A callout pointing to the 'Add file to the Polyspace Quality Monitoring list' option in the context menu says: "Add files to the quality monitoring panel for continuous observation".

Productivity enhancing auto-fix feature

The screenshot displays a code editor window for a file named `test_driver.c`. The code contains several comments and preprocessor directives. A yellow lightbulb icon is positioned over the `#include "header_file.h"` line, which has triggered a "Quick Fix" menu. The menu lists several options for resolving the "Defect:USELESS_INCLUDE" error, with "Fix USELESS_INCLUDE: remove useless #include directive.." selected. A callout bubble points to this menu with the text "Auto-fix options for straightforward corrections".

```
7  * Created on: April 10, 2021
8  * Last Update: July 15, 2021
9  * Author: jchapple
10 * Purpose /* extra comment in comment
11 * This provides unit tests for the
12 * function "my_function" in sut.c (the "source under test")
13 * *****
14 */
15
16
17 /* polyspace-begin MISRA-C3:21.6 */
18 // local includes
19 #include "header_file.h"
20 #include "unused_header.h"
21 //
22 // Quick Fix...
23 // ⚡ Show details about Defect:USELESS_INCLUDE finding
24 // ⚡ Justify Defect:USELESS_INCLUDE with annotation
25 // ⚡ Justify all Defect:USELESS_INCLUDE findings in this file with annotation
26 #: ⚡ Fix USELESS_INCLUDE: comment out useless #include directive.
27 #: ⚡ Fix all USELESS_INCLUDE instances in current file: comment out useless #inclu...
28 #: ⚡ Fix USELESS_INCLUDE: remove useless #include directive..
29 #: ⚡ Fix all USELESS_INCLUDE instances in current file: remove useless #include dir...
30 #:
31
32 #define START 2
33 #define END 10
```

Auto-fix options for straightforward corrections

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, **/*.ts, !*...)

- ✗ 3.1 The character sequences `/*` and `//` shall not be used within a comment. polyspace(MISRA C:2012:3.1:Required) [Ln 10, Col 13]
- 💡 Useless include polyspace(Defect:USELESS_INCLUDE) [Ln 20, Col 1]
- ✗ Useless include polyspace(Defect:USELESS_INCLUDE) [Ln 29, Col 1]
- ✗ 338 Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) polyspace(CWE:338) [Ln 64, Col 2]
- ✗ 676 Use of Potentially Dangerous Function polyspace(CWE:676) [Ln 64, Col 2]
- ✗ Dead code polyspace(Defect:DEAD_CODE) [Ln 74, Col 5]
- ✗ 571 Expression is Always True polyspace(CWE:571) [Ln 74, Col 5]
- ✗ 570 Expression is Always False polyspace(CWE:570) [Ln 74, Col 5]
- ✗ 561 Dead Code polyspace(CWE:561) [Ln 74, Col 5]

Demo

Come by the booth or visit our website

Get Polyspace as You Code

- https://www.mathworks.com/downloads/csd/Polyspace/23a/PaYC_Installer_win64.zip
- https://www.mathworks.com/downloads/csd/Polyspace/23a/PaYC_Installer_glnxa64.zip

Training: Polyspace for C/C++ Code Verification

After this 2-day course you will be able to:

- Create a verification project
- Review verification results
- Emulate target execution environments
- Handle missing functions and data
- Manage unproven code
- Apply MISRA-C[®] rules
- Report verification results via Polyspace Access

```
27     y = a[2];  
28     if(z == 6)  
29     {  
30         y = b[3];  
31     }  
32     else if(z == 7)  
33     {  
34         y = b[4];  
35     }  
36  
37     return res/z;  
38 }
```

```
27     if(z == 6)  
28     {  
29         y = b[3];  
30     }  
31     else if(z == 7)  
32     {  
33         y = b[4];  
34     }  
35     else  
36     {  
37         y = a[2];  
38     }  
39  
40     return (res/z)+y;  
41 }
```

Upcoming offerings: May 2-3, June 22-23