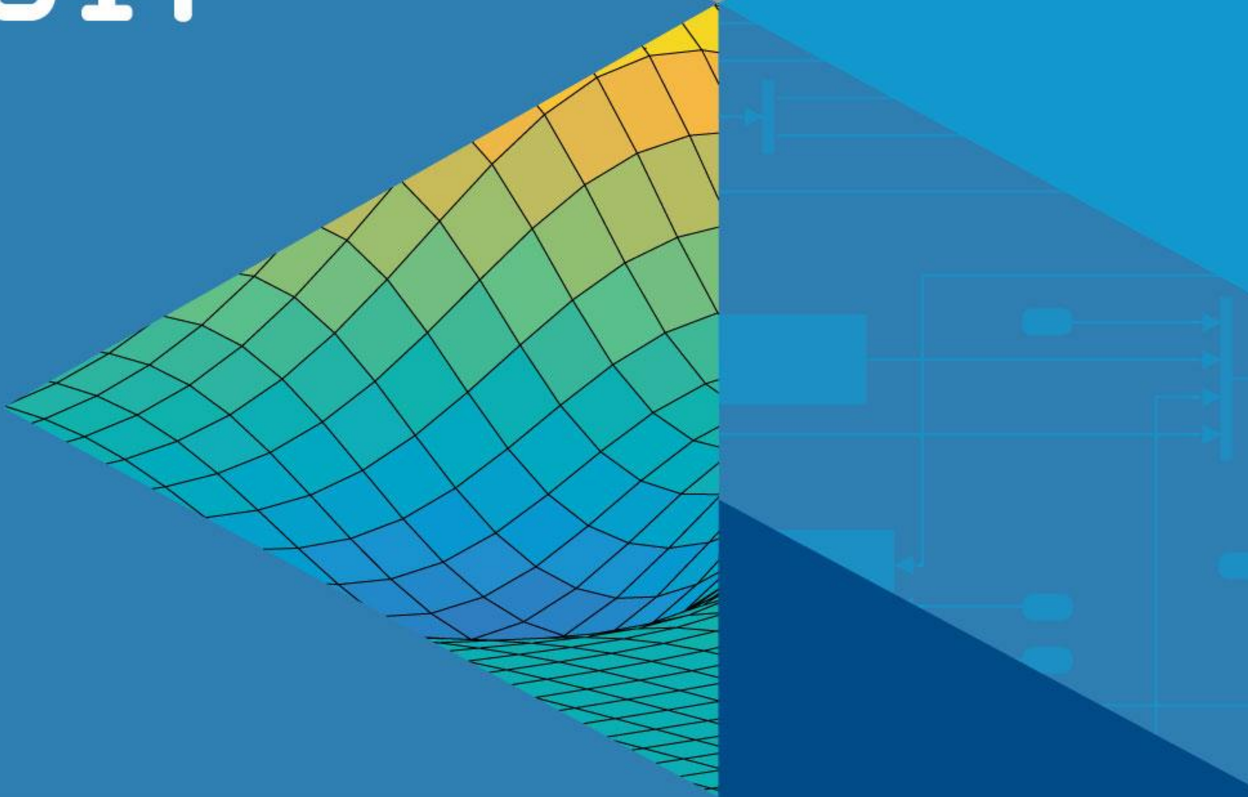
MATLAB EXPO 2017
KOREA

4월 27일, 서울

등록 하기 matlabexpo.co.kr



Polyspace를 활용한 MISRA C:2012 및 실행시간 오류 검사

Introduction to Polyspace with MISRA C:2012 and RTE

유용출 과장

Gary.Ryu@mathworks.co.kr

Agenda

- Why do we check MISRA C and Runtime errors?

- Polyspace Introduction
  - How to check MISRA C:2012 violations
  - How to verify Runtime errors

Why do we check **MISRA C** or **Runtime error** ?

The intention was to provide a "**restricted subset of a standardized structured language**" as required in the 1994 MISRA Guidelines for automotive systems being developed to **meet the requirements of functional safety standards like ISO 26262**.

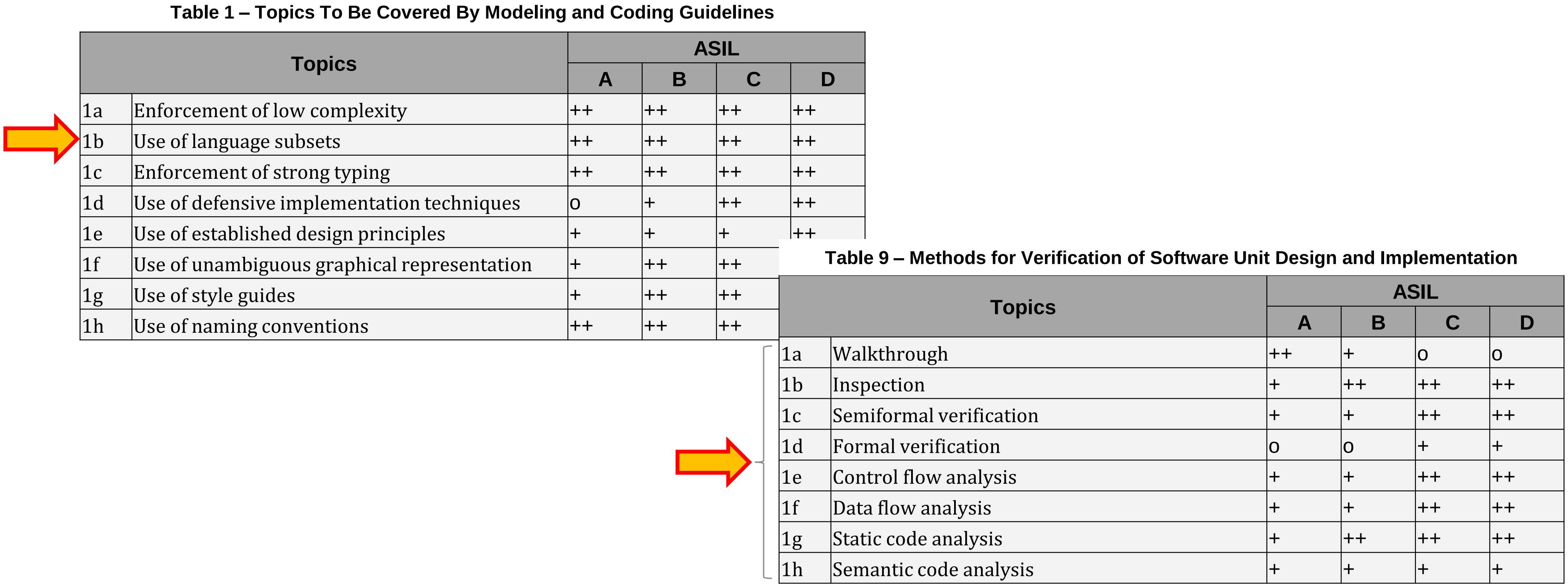


Table 1 – Topics To Be Covered By Modeling and Coding Guidelines

Topics		ASIL			
		A	B	C	D
1a	Enforcement of low complexity	++	++	++	++
1b	Use of language subsets	++	++	++	++
1c	Enforcement of strong typing	++	++	++	++
1d	Use of defensive implementation techniques	o	+	++	++
1e	Use of established design principles	+	+	+	++
1f	Use of unambiguous graphical representation	+	++	++	
1g	Use of style guides	+	++	++	
1h	Use of naming conventions	++	++	++	

Table 9 – Methods for Verification of Software Unit Design and Implementation

Topics		ASIL			
		A	B	C	D
1a	Walkthrough	++	+	o	o
1b	Inspection	+	++	++	++
1c	Semiformal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis	+	+	++	++
1f	Data flow analysis	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis	+	+	+	+

Ref) ISO 26262 – Mapping Table (Certification Kit 2017a)

Why **restricted subset**?

- There are several drawbacks with the C language
  - ISO Standard language definition is *incomplete* …
    - *Undefined* behavior
    - *Unspecified* behavior
    - *Implementation-defined* behavior
  - Misuse language
  - Misunderstanding language
  - Lack of Runtime error checking

- One of solution is ***MISRA C*** and ***RTE detection*** with ***Static Analysis***

Why **restricted subset**?

- There are several drawbacks with the C language
  - ISO Standard language definition is *incomplete* …
    - *Undefined* behavior
    - *Unspecified* behavior
    - *Implementation-defined* behavior
  - Misuse language
  - Misunderstanding language
  - Lack of Runtime error checking
- One of solu … c ***Analysis***

```
int  foo  (int  arg)  {
       return  arg  +  1;
}

void  main  (void)  {
       int  var  =  0;
       printf (“var :  %d  and  %d\n”,  var++,  foo(var));
}
```

```
Output with …
-   gcc 5.4.0                : var :  0  and  1
-   Visual Studio 2013       : var :  0  and  2
```

Why **restricted subset**?

- There are several drawbacks with the C language
  - ISO Standard language definition is *incomplete* …
    - *Undefined* behavior
    - *Unspecified* behavior
    - *Implementation-defined* behavior
  - Misuse language
  - Misunderstanding language
  - Lack of Runtime error checking

- One of solution is ***MISRA C*** and ***RTE detection*** with ***Static Analysis***

Brief History of MISRA C

- ***MISRA C:2012***
  - Compatible with **ISO/IEC 9899:1999 (C99)**
  - published in 2013
  - 159 Guidelines
    - 16 Directives
    - 143 Rules

→

- 173 Guidelines
  - 17 Directives
  - 156 Rules

More guidelines for Security (April, 2016)

- MISRA C:2004
  - Compatible with ISO/IEC 9899:1990 (C90)

- MISRA C:1998
  - Compatible with ISO/IEC 9899:1990 (C90)

What is MISRA C:2012

- Directives

  Guidelines for which it is not possible to provide the full description necessary to perform a check for compliance. Static analysis tools may be able to assist in checking compliance. For example, items are checked with design documents or requirements specification.

- Rules

  Guidelines for which a complete description has been provided. It is possible to check compliance with source code without any other information.

What is MISRA C:2012

- Directives
  - 17 Directives
    - 10 Required directives
    - 7 Advisory directives
- Rules
  - 156 Rules
    - 16 Mandatory rules
    - 108 Required rules
    - 32 Advisory rules

Mandatory:

- Deviation from this guidelines is not permitted.

Required:

- Formal deviation is required.

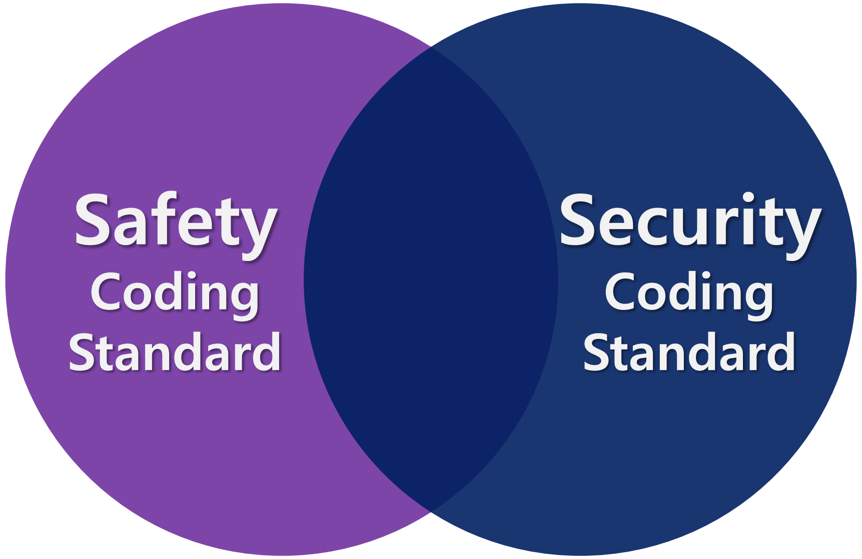
Advisory:

- Formal deviation is not necessary, but alternative arrangements should be made.

*** Any guideline can be treated as required/mandatory guideline.**

New Security guidelines of MISRA C:2012

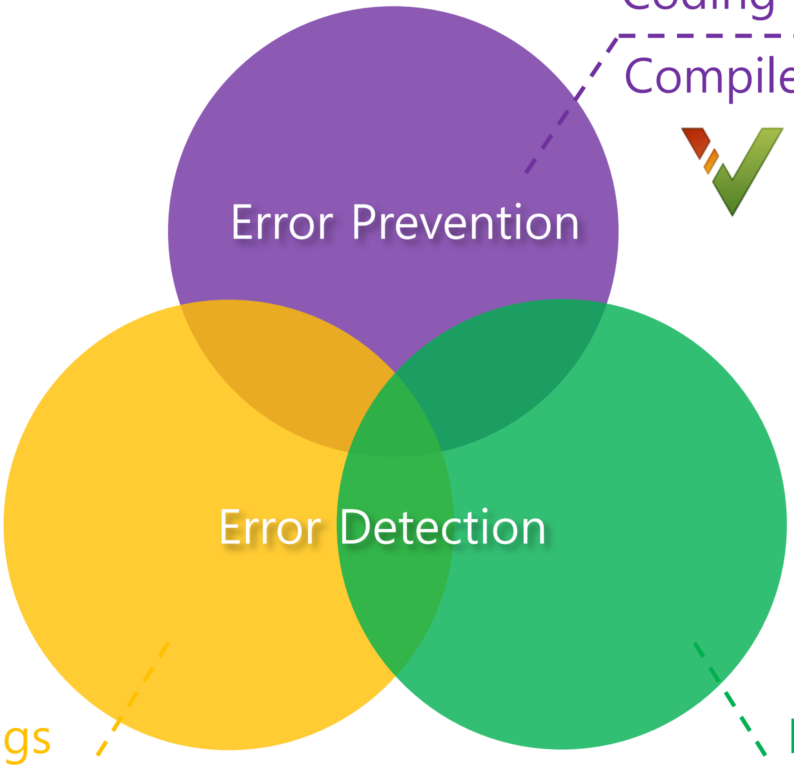
- are to improve the coverage of the **security concerns** highlighted by **ISO/ IEC 17961:2013**



- MISRA C has evolved…

  from automotive standard to industry-wide standard!

Polyspace PRODUCTS



Coding Rules, Code Metrics

Compiler Warnings

✓ Polyspace Bug Finder
Polyspace Code Prover

Bug Findings

(False negative)

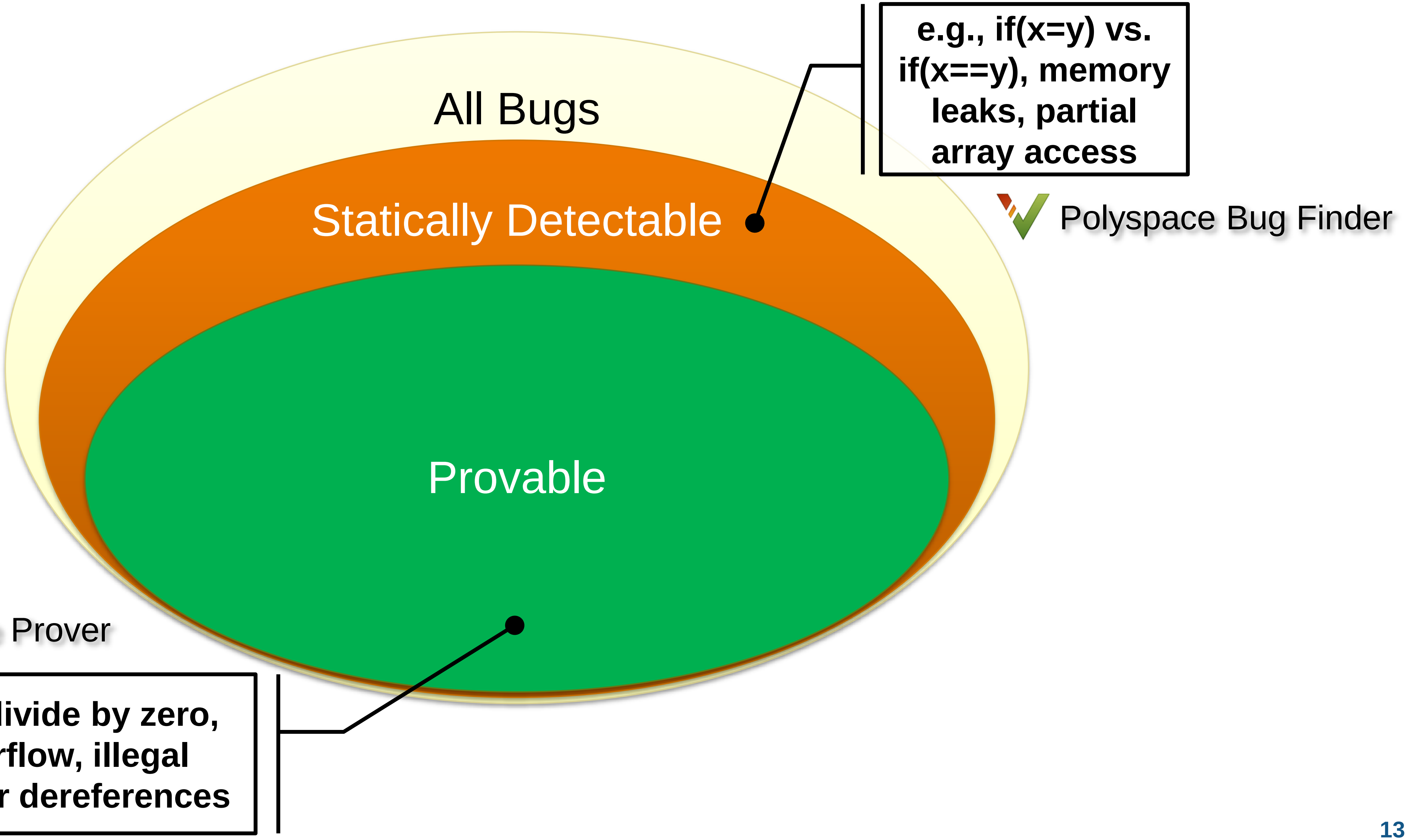
✓ Polyspace Bug Finder

Formal Methods

(No False negative)

✓ Polyspace Code Prover

Not all bugs can be statically proven

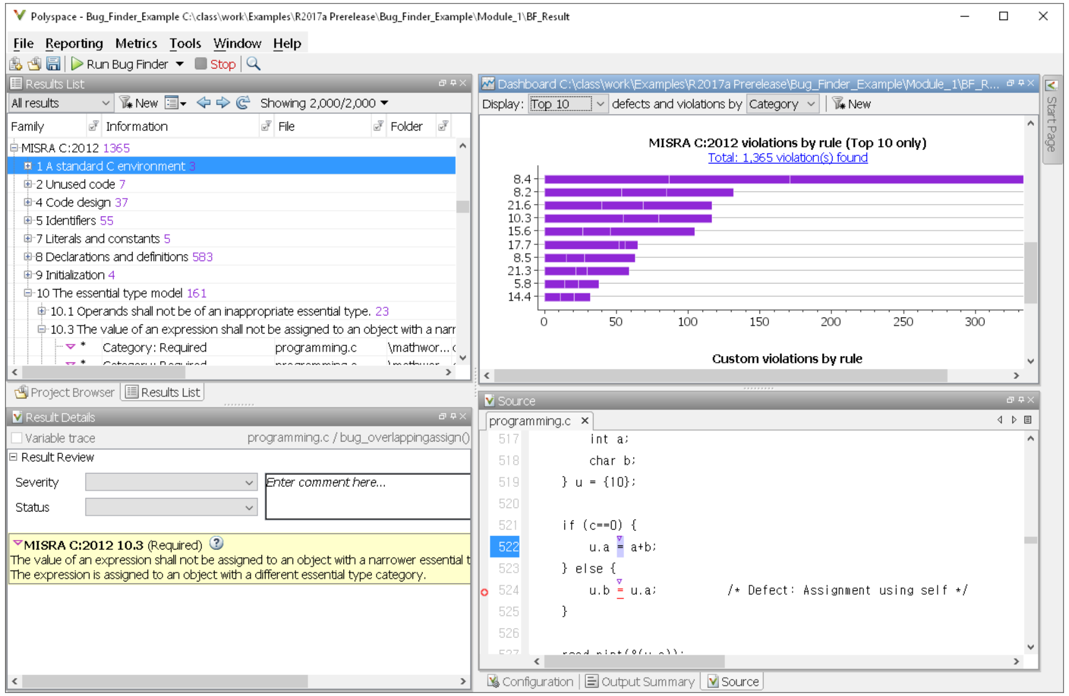


e.g., if(x=y) vs. if(x==y), memory leaks, partial array access

e.g., divide by zero, overflow, illegal pointer dereferences

Polyspace supports for Coding Rules Compliance

- **<u>MISRA C:2012</u>**
  - 11 Directives supported
  - 156 rules supported
  - 6 directives not enforceable
- MISRA C++:2008
  - 185 of the 228 rules supported
- JSF++:2005
  - 157 of 234 rules supported



Polyspace - Bug_Finder_Example C:\class\work\Examples\R2017a Prerelease\Bug_Finder_Example\Module_1\BF_Result

MISRA C:2012 1365
- 1 A standard C environment 3
- 2 Unused code 7
- 4 Code design 37
- 5 Identifiers 55
- 7 Literals and constants 5
- 8 Declarations and definitions 583
- 9 Initialization 4
- 10 The essential type model 161
  - 10.1 Operands shall not be of an inappropriate essential type. 23
  - 10.3 The value of an expression shall not be assigned to an object with a narr
    - * Category: Required programming.c \mathwor...

Project Browser
Results List
Dashboard C:\class\work\Examples\R2017a Prerelease\Bug_Finder_Example\Module_1\BF_R...
Display: Top 10
defects and violations by Category
New
Start Page
MISRA C:2012 violations by rule (Top 10 only)
Total: 1,365 violation(s) found
8.4
8.2
21.6
10.3
15.6
17.7
8.5
21.3
5.8
14.4
0
50
100
150
200
250
300
Custom violations by rule
Result Details
Variable trace
programming.c / bug_overlappingassign()
Result Review
Severity
Status
Enter comment here...

MISRA C:2012 10.3 (Required)
The value of an expression shall not be assigned to an object with a narrower essential t
The expression is assigned to an object with a different essential type category.

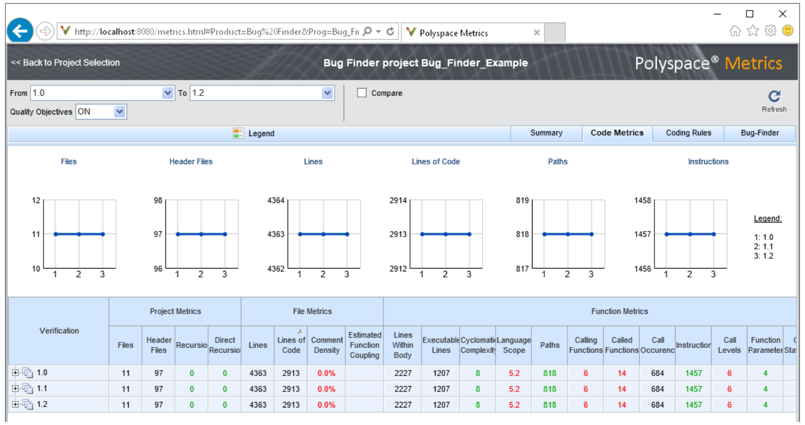
```
        int a;
        char b;
    } u = {10};

    if (c==0) {
        u.a = a+b;
    } else {
        u.b = u.a;          /* Defect: Assignment using self */
    }

```

Polyspace supports for various Code Metrics

- **Project Metrics**
  - Direct Recursions
  - Header Files
  - Files
  - Recursions
- **File Metrics**
  - Comment Density
  - Estimated Function Coupling
  - Lines
  - Lines without comment



From 1.0 To 1.2

☐ Compare

Quality Objectives ON

Verification	Project Metrics				File Metrics				Function Metrics												
	Files	Header Files	Recursio	Direct Recursio	Lines	Lines of Code	Comment Density	Estimated Function Coupling	Lines Within Body	Executable Lines	Cyclomati Complexity	Language Scope	Paths	Calling Functions	Called Functions	Call Occurenc	Instruction	Call Levels	Function Parameter	C Sta
1.0	11	97	0	0	4363	2913	0.0%		2227	1207	8	5.2	818	6	14	684	1457	6	4	
1.1	11	97	0	0	4363	2913	0.0%		2227	1207	8	5.2	818	6	14	684	1457	6	4	
1.2	11	97	0	0	4363	2913	0.0%		2227	1207	8	5.2	818	6	14	684	1457	6	4	

- **Function Metrics**
  - Cyclomatic Complexity
  - Higher Estimate of Local Variable Size
  - Lower Estimate of Local Variable Size
  - Language Scope
  - Call Levels
  - Call Occurrences
  - Called Functions
  - Calling Functions
  - Executable Lines
  - Function Parameters
  - Goto Statements
  - Instructions
  - Lines Within Body
  - Local Non-Static Variables
  - Local Static Variables
  - Paths
  - Return Statements

Types of Defects detected by Polyspace Bug Finder

Numerical

- Division by zero, Overflow
- Invalid use of standard library integer/floating point routine
- …

Static memory

- Array access out of bounds
- Null pointer
- …

Dynamic memory

- Memory leaks
- Use of previously freed pointer
- …

Dataflow

- Write without further read
- Non-initialized variable
- …

Concurrency

- Data races (atomic, non-atomic)
- Deadlocks
- …

Resource management

- Resource leak
- Writing to read-only resource
- …

Programming

- Invalid use of = or == operator
- Declaration mismatch
- …

Good Practice

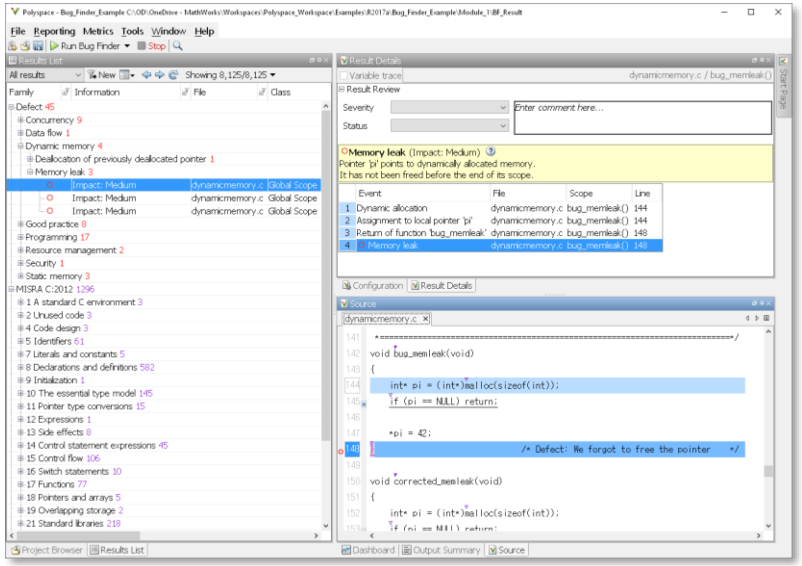
- Unused parameter
- Large pass-by-value argument
- …

Security

- Unsafe standard function
- Use of non-secure temporary file
- …

Tainted data

- Array access with tainted index
- Tainted sign change conversion
- …



Family	Information	File	Class
Defect 45			
Concurrency 9			
Data flow 1			
Dynamic memory 4			
Deallocation of previously deallocated pointer 1			
Memory leak 3			
○	Impact: Medium	dynamicmemory.c	Global Scope
○	Impact: Medium	dynamicmemory.c	Global Scope
○	Impact: Medium	dynamicmemory.c	Global Scope
Good practice 8			
Programming 17			
Resource management 2			
Security 1			
Static memory 3			
MISRA C:2012 1296			
1 A standard C environment 3			
2 Unused code 3			
4 Code design 3			
5 Identifiers 61			
7 Literals and constants 5			
8 Declarations and definitions 582			
9 Initialization 1			
10 The essential type model 145			
11 Pointer type conversions 15			
12 Expressions 1			
13 Side effects 8			
14 Control statement expressions 45			
15 Control flow 106			
16 Switch statements 10			
17 Functions 77			
18 Pointers and arrays 5			
19 Overlapping storage 2			
21 Standard libraries 218			

○ **Memory leak** (Impact: Medium)
Pointer 'pi' points to dynamically allocated memory.
It has not been freed before the end of its scope.

	Event	File	Scope	Line
1	Dynamic allocation	dynamicmemory.c	bug_memleak()	144
2	Assignment to local pointer 'pi'	dynamicmemory.c	bug_memleak()	144
3	Return of function 'bug_memleak'	dynamicmemory.c	bug_memleak()	148
4	○ Memory leak	dynamicmemory.c	bug_memleak()	148

```
 *=========================================================================*/
void bug_memleak(void)
{
    int* pi = (int*)malloc(sizeof(int));
    if (pi == NULL) return;

    *pi = 42;
}                                   /* Defect: We forgot to free the pointer    */

void corrected_memleak(void)
{
    int* pi = (int*)malloc(sizeof(int));
    if (pi == NULL) return;
```

www.mathworks.com/help/bugfinder/defect-reference.html

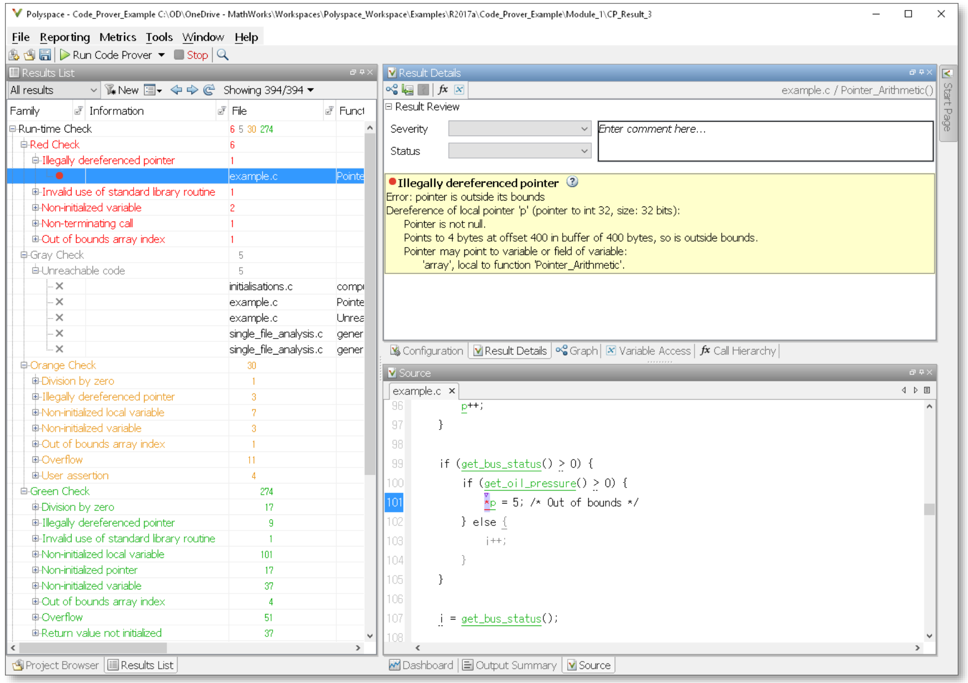
Full list of Runtime checks in Polyspace Code Prover

C run-time checks

- Unreachable Code
- Function not called
- Function not reachable
- Non-initialized local variable
- Non-initialized pointer
- Non-initialized variable
- Return value not initialized
- Division by zero
- Invalid operation on floats
- Invalid shift operations
- Overflow
- Subnormal float
- Absolute address usage
- Illegally dereferenced pointer
- Out of bound array index
- Non-terminating call
- Non-terminating loop
- Correctness condition (array conversion must not extend range, function pointer does not point to a valid function)
- Invalid use of standard library routine
- User assertion

Additional run-time checks for C++ only

- Incorrect object oriented programming
- Invalid C++ specific operations
- Function not returning value
- Null this-pointer calling method
- Uncaught exception



Family	Information	File	Funct
Run-time Check		6 5 30 274	
Red Check		6	
Illegally dereferenced pointer		1	
●		example.c	Pointe
Invalid use of standard library routine		1	
Non-initialized variable		2	
Non-terminating call		1	
Out of bounds array index		1	
Gray Check		5	
Unreachable code		5	
×		initialisations.c	compi
×		example.c	Pointe
×		example.c	Unrea
×		single_file_analysis.c	gener
×		single_file_analysis.c	gener
Orange Check		30	
Division by zero		1	
Illegally dereferenced pointer		3	
Non-initialized local variable		7	
Non-initialized variable		3	
Out of bounds array index		1	
Overflow		11	
User assertion		4	
Green Check		274	
Division by zero		17	
Illegally dereferenced pointer		9	
Invalid use of standard library routine		1	
Non-initialized local variable		101	
Non-initialized pointer		17	
Non-initialized variable		37	
Out of bounds array index		4	
Overflow		51	
Return value not initialized		37	

● **Illegally dereferenced pointer**

```
Error: pointer is outside its bounds
Dereference of local pointer 'p' (pointer to int 32, size: 32 bits):
    Pointer is not null.
    Points to 4 bytes at offset 400 in buffer of 400 bytes, so is outside bounds.
    Pointer may point to variable or field of variable:
        'array', local to function 'Pointer_Arithmetic'.
```

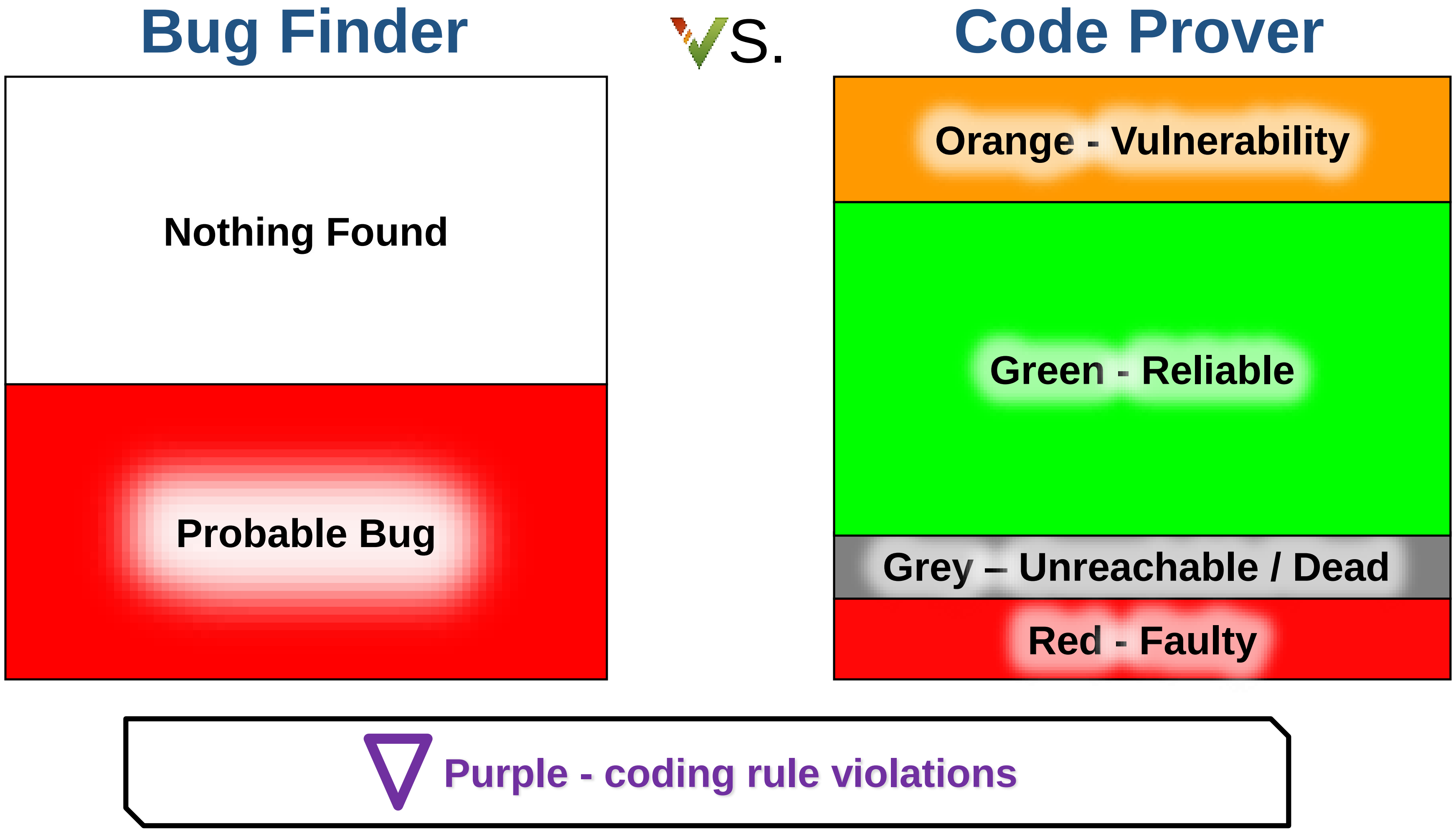
```
            p++;
        }

        if (get_bus_status() > 0) {
            if (get_oil_pressure() > 0) {
                *p = 5; /* Out of bounds */
            } else {
                i++;
            }
        }

        i = get_bus_status();
```

www.mathworks.com/help/codeprover/run-time-check-reference.html

How do Bug Finder results differ from Code Prover results?



Bug Finder vs. Code Prover

Bug Finder:

- Nothing Found
- Probable Bug

Code Prover:

- Orange - Vulnerability
- Green - Reliable
- Grey – Unreachable / Dead
- Red - Faulty

Purple - coding rule violations

Polyspace demonstration