

MATLAB EXPO

形式化方法赋能车规级芯片底层软件开发

胡乐华, *MathWorks*



您将了解到...

- 形式化方法及抽象解释法
- Polyspace对 ISO 26262 和 ISO/SAE 21434 支持
- 形式化方法验证用于性能提升
- 将 Polyspace 应用到开发流程中

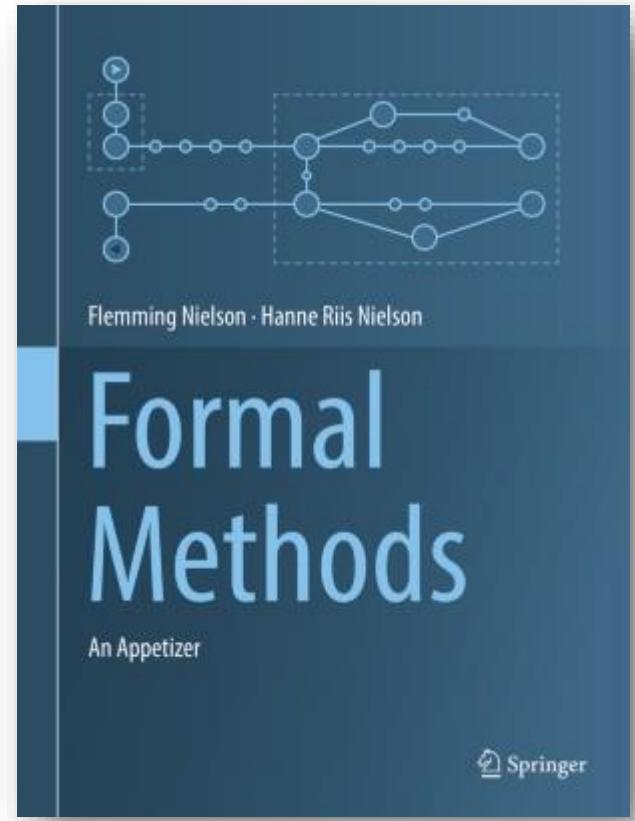
车规级芯片底层软件开发之挑战

- 功能安全和网络安全认证
 - ISO 26262, ISO/SAE 21434
 - CWE, CERT-C
- 芯片运行更多的程序
 - 高效运行驱动程序
 - 权衡防御性编程带来额外的ROM开销
- 更快的交付，响应市场需求
 - 更复杂，开发量更大
 - 协同要求更高

什么是形式化方法和抽象解释法？

形式化方法

Formal methods are system design techniques that use rigorously specified mathematical models to build software and hardware systems. In contrast to other design systems, formal methods use mathematical proof as a complement to system testing in order to ensure correct behavior. As systems become more complicated, and safety becomes a more important issue, the formal approach to system design offers another level of insurance.



抽象解释法

PRINCIPLES OF
ABSTRACT INTERPRETATION



PATRICK COUSOT

*In computer science, **abstract interpretation** is a theory of **sound approximation** of the **semantics of computer programs**, based on monotonic functions over ordered sets, especially lattices. It can be viewed as a partial execution of a computer program which gains information about its semantics (e.g., control-flow, data-flow) without performing all the calculations.*

抽象解释法：证明软件的健壮性

```
1 int new_position(int sensor_pos1, int sensor_pos2)
2 {
3     int actuator_position;
4     int x, y, tmp, magnitude;
5
6     actuator_position = 2; /* default */
7     tmp = 0;                /* values */
8     magnitude = sensor_pos1 / 100;
9     y = magnitude + 5;
10
11    while (actuator_position < 10)
12    {
13        actuator_position++;
14        tmp += sensor_pos2 / 100;
15        y += 3;
16    }
17    if ((3*magnitude + 100) > 43)
18    {
19        magnitude++;
20        x = actuator_position;
21        actuator_position = x / (x - y);
22    }
23    return actuator_position*magnitude + tmp; /* new value */
24 }
```

该行是否有软件bug?
如何证明没有bug?

Polyspace Code Prover 使用抽象解释法证明软件的健壮性

```
int new_position(int sensor_pos1, int sensor_pos2)
{
    int actuator_position;
    int x, y, tmp_pos, magnitude;

    actuator_position = 2; /* default */
    tmp_pos = 0;           /* values */
    magnitude = sensor_pos1 / 100;
    y = magnitude ± 5;
    x = actuator_position;

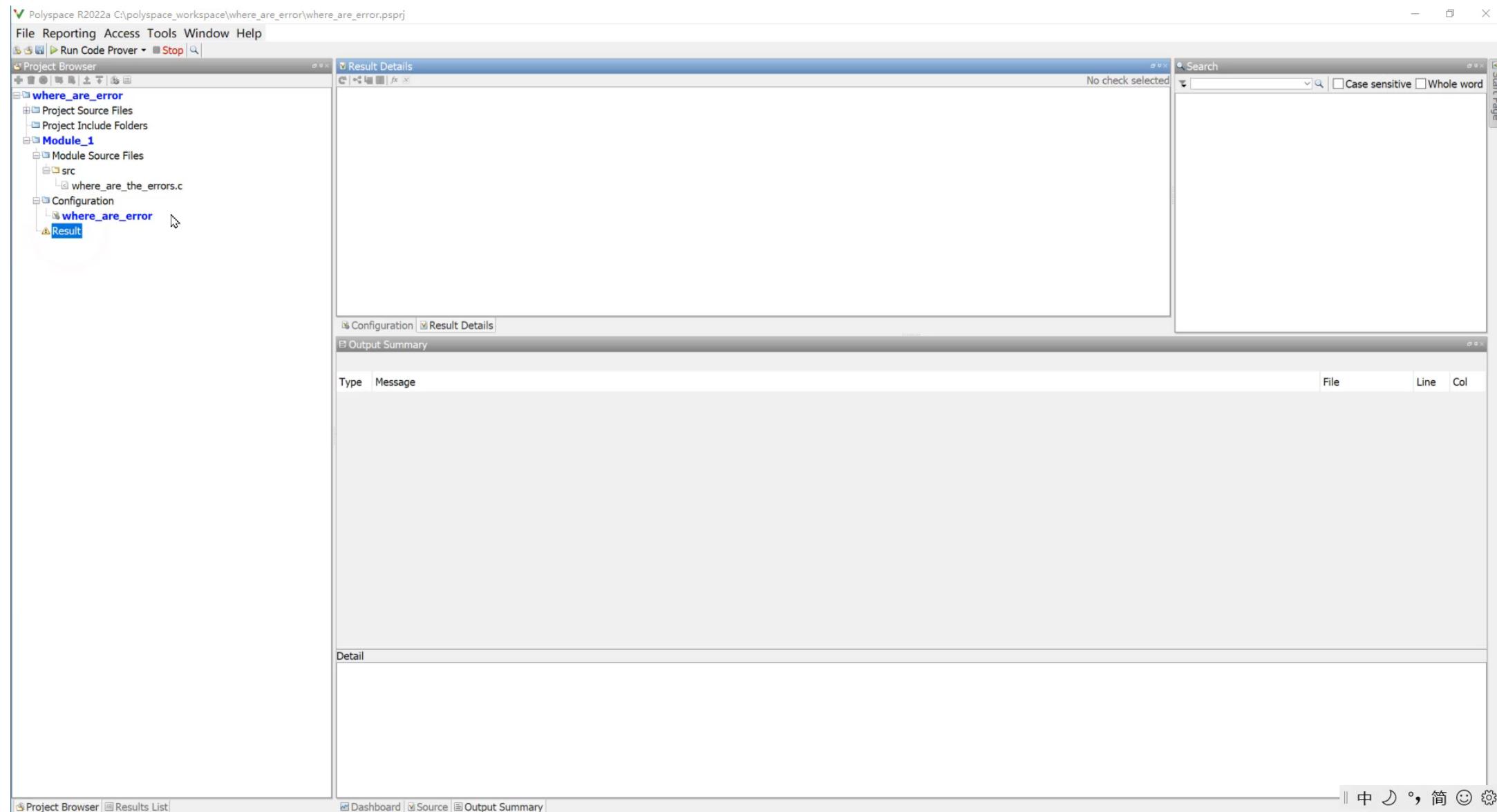
    while (actuator_position < 10)
    {
        actuator_position++;
        tmp_pos += sensor_pos2 / 100;
        y ±= 3;
    }
    if ((3 * magnitude ± 100) ≥ 43)
    {
        magnitude++;
        x = actuator_position;
        actuator_position = x / (x ± y);
    }
    return actuator_position + tmp_pos;
}
```

operator - on type int 32
left: 10
right: [-11 .. 21474865 (0x147AE31)]
result: [-21474855 .. -1]
(result is truncated)

证明除法安全

除数永不为零

示例：Polyspace Code Prover



Polyspace 代码着色让问题暴露无遗



Green: reliable
safe pointer access

Red: faulty
out of bounds error

Gray: dead
unreachable code

Orange: unproven
may be unsafe for some
conditions

Purple: violation
MISRA-C/C++ or JSF++
code rules

Range data
tool tip

```
static void pointer_arithmetic (void) {
    int array[100];
    int *p = array;
    int i;

    for (i = 0; i < 100; i++) {
        *p = 0;
        p++;
    }

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

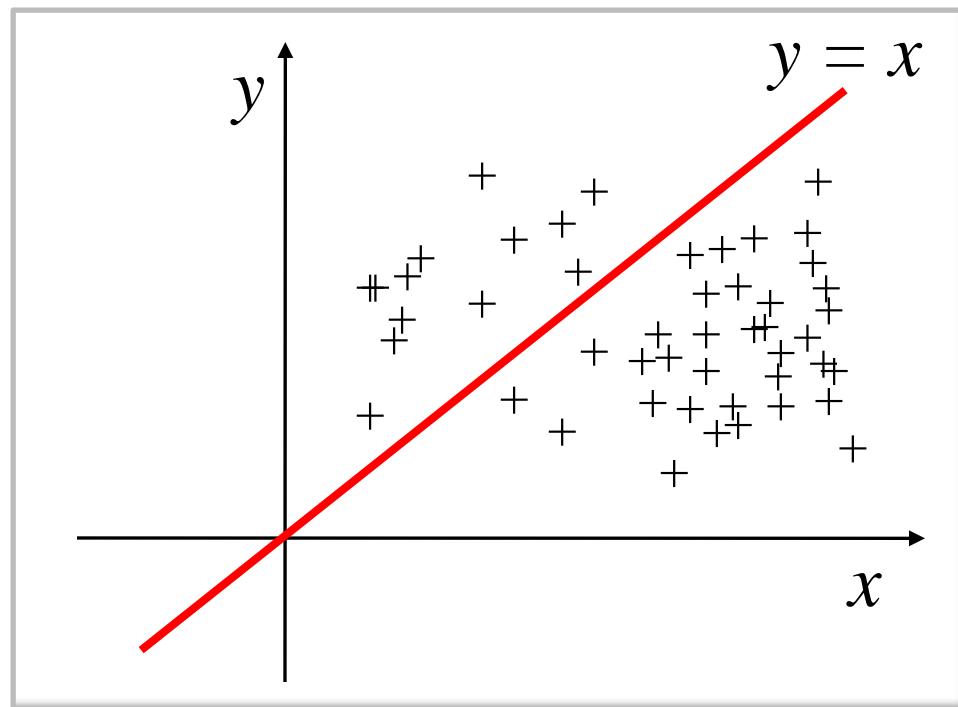
    i = get_bus_status();

    if (i >= 0) {
        *(p - i) = 10;
    }
}
```

variable 'i' (int32): [0 .. 99]
assignment of 'i' (int32): [1 .. 100]

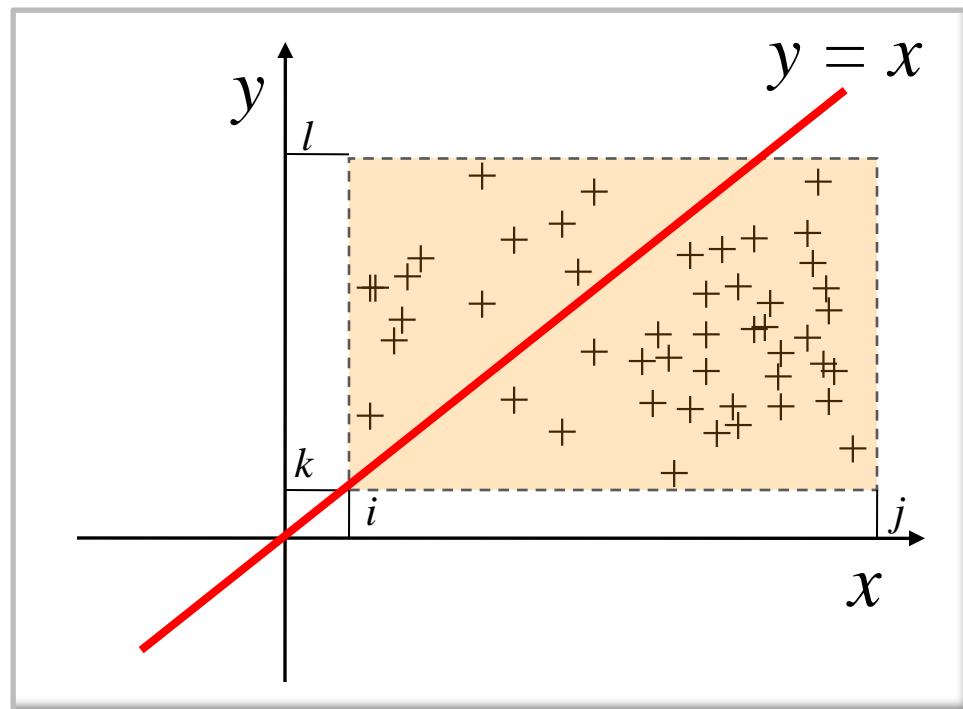
理解抽象解释法

x 和 y 可以是任何值



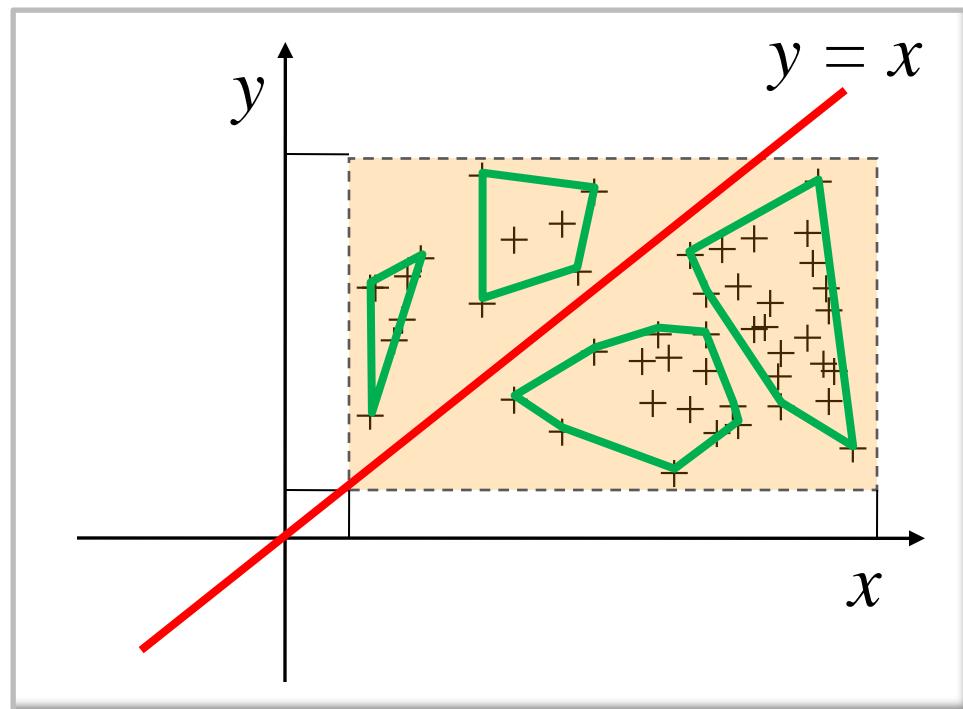
理解抽象解释法

类型分析，界定范围



理解抽象解释法

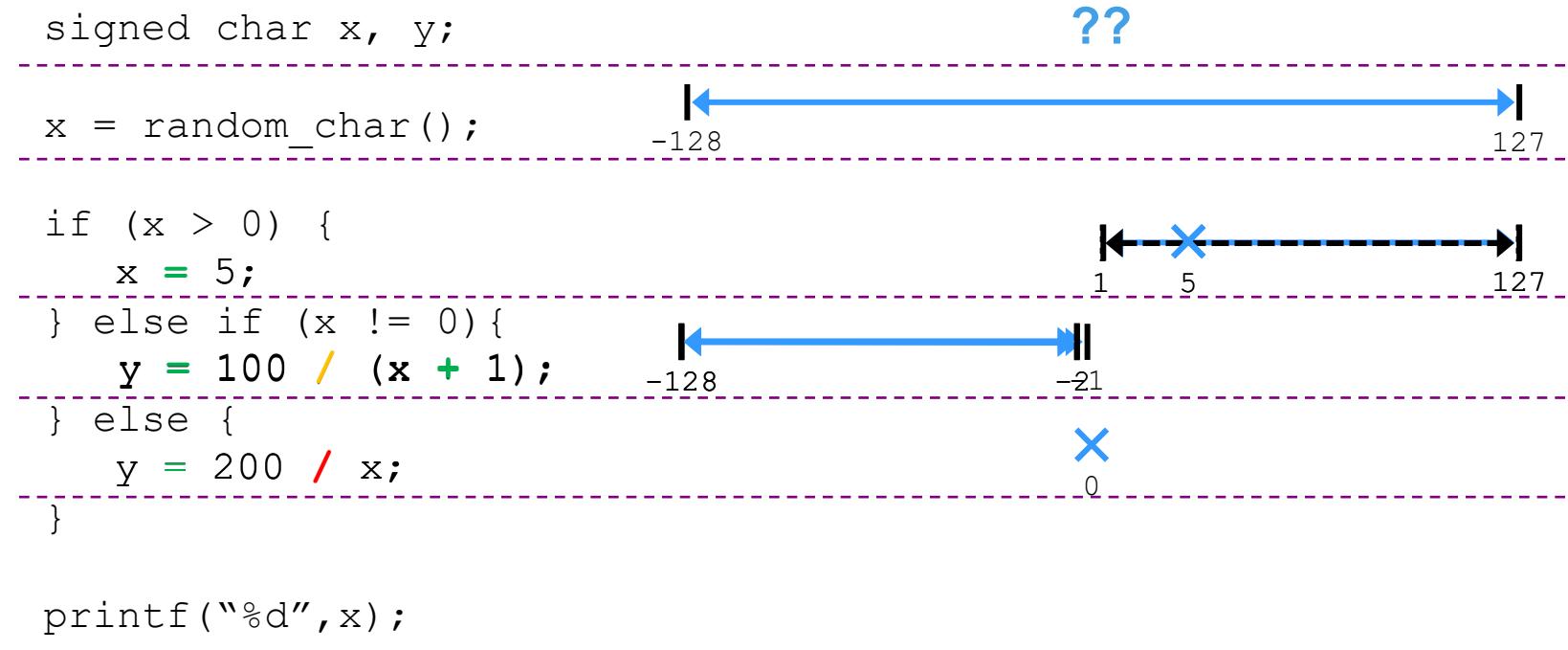
抽象解释法



不运行
无需测试用例

理解抽象解释法

抽象解释法考虑所有可能输入，所有可能执行路径来证明是否有缺陷。



完整的静态分析能力

Bug Finder



→ High Quality, Secure, Compliant Code:

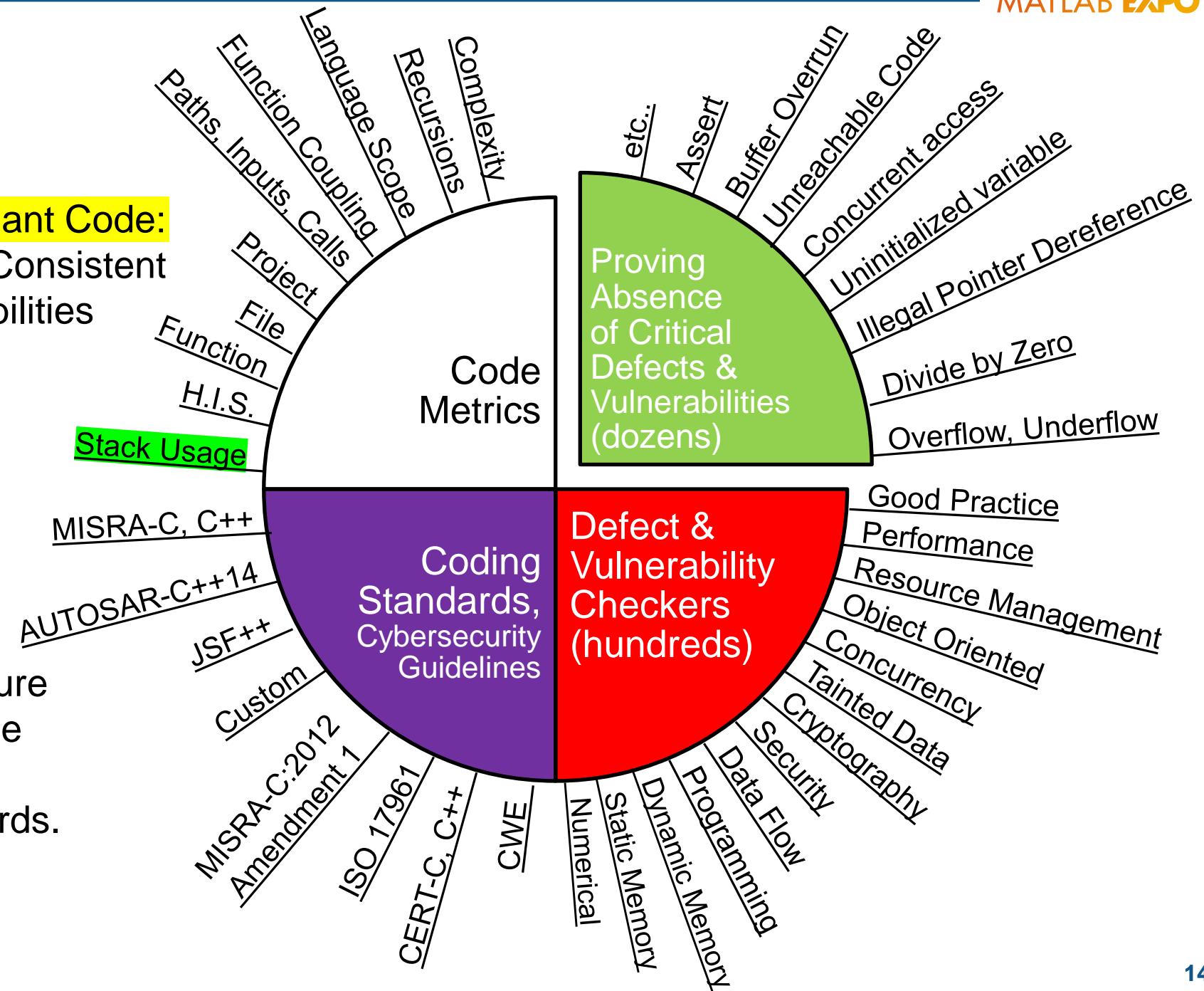
- Measurable, Maintainable, Consistent
- Very few defects or vulnerabilities
- Credits for functional safety, cybersecurity standards.

Code Prover



→ Fully Trusted Components:

- Reliable, Robust, Safe, Secure
- Proven free of critical runtime defects and vulnerabilities
- Additional credits for standards.



Polyspace 支持功能安全 ISO 26262 最高等级认证

Polyspace Bug Finder 对ISO 26262 Part6要求的覆盖

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

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

Table 3 – Principles for Software Architectural Design

	Topics	ASIL			
		A	B	C	D
1i	Appropriate management of shared resources	‡	‡	‡	‡
(...)					
1e	Loose coupling between software components	+	‡	‡	‡
(...)					
1c	Restricted size of interfaces	+	+	+	‡
1b	Restricted size and complexity of software components	‡	‡	‡	‡
(...)					

Table 6 – Design Principles for Software Unit Design and Implementation

	Topics	ASIL			
		A	B	C	D
1j	No recursions	+	+	‡	‡
1i	No unconditional jumps	‡	‡	‡	‡
1h	No hidden data flow or control flow	+	‡	‡	‡
1g	No implicit type conversions	+	‡	‡	‡
1f	Restricted use of pointers	+	‡	‡	‡
1e	Avoid global variables or else justify their usage	+	+	‡	‡
1d	No multiple use of variable names	‡	‡	‡	‡
1c	Initialization of variables	‡	‡	‡	‡
1b	No dynamic objects or variables, or else online test during their creation	+	‡	‡	‡
1a	One entry and one exit point in subprograms and functions	‡	‡	‡	‡

Table 7 – Methods for Software Unit Verification

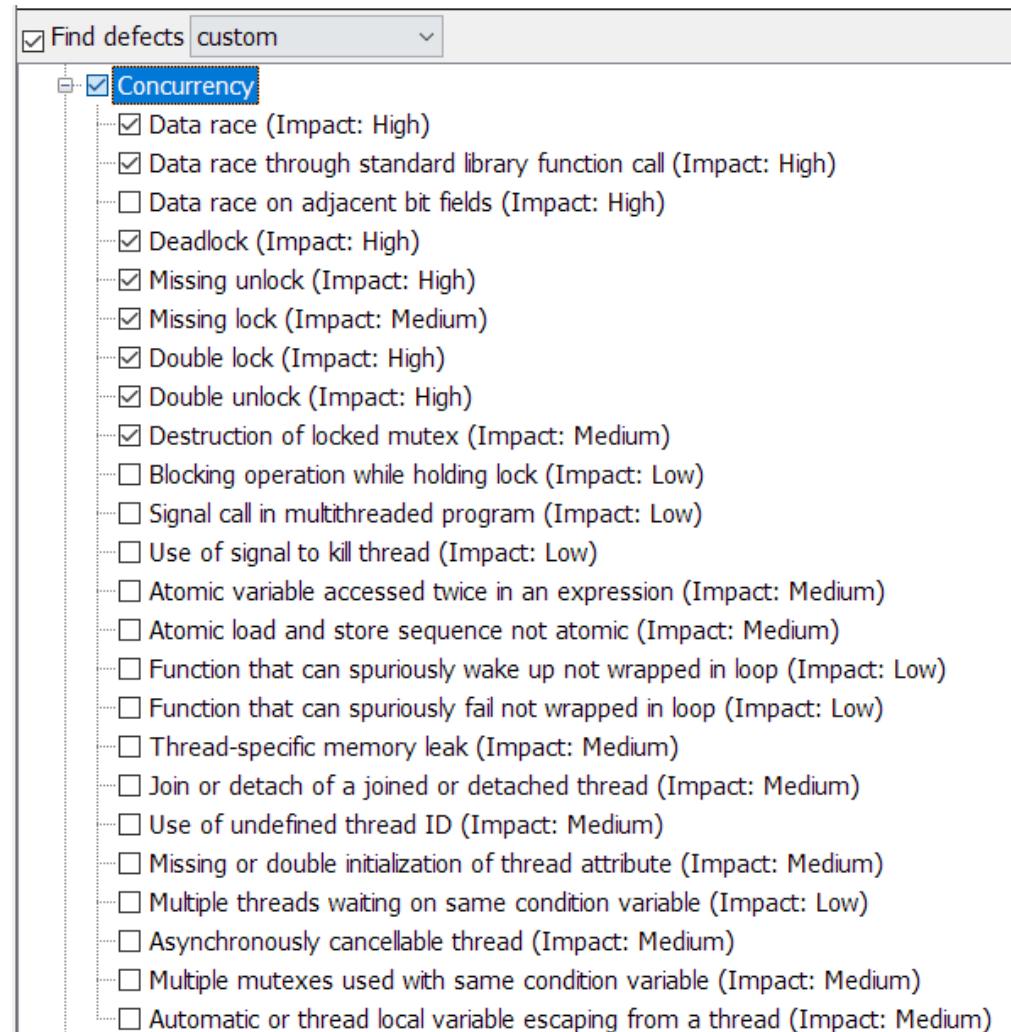
	Topics	ASIL			
		A	B	C	D
(...)					
1h	Static code analysis	‡	‡	‡	‡
(...)					
1a	Walk-through	‡	+	0	0

Table 10 – Methods for Verification of Software Integration

	Topics	ASIL			
		A	B	C	D
(...)					
1g	Static code analysis	‡	‡	‡	‡
(...)					

Table 1 Table 2 Table 3 Table 4 Table 5 Table 6 Table 7 Table 8 Table 9 Table 10 Table 11 Table 12 Table 13 Table 14 Tables

Polyspace Bug Finder 解决并发问题



Polyspace Bug Finder 死锁问题检查

The screenshot shows the Polyspace Bug Finder software interface. On the left is the Project Browser with a tree view of the project structure. The central area is the Configuration window for the 'Bug_Finder_Example' configuration. The 'Target & Compiler' tab is selected, showing settings for Source code language (C), C standard version (defined-by-compiler), Compiler (gnu8.x), Target processor type (x86_64), and various Compiler Behavior options. Below the configuration is a Source code editor window displaying 'concurrency.c'. Line 127 contains the code 'acquire_printer();', which is highlighted with a blue selection bar. The right side of the interface features a large pane for 'Find defects custom' with a tree view of detected issues under the 'Concurrency' category. A detailed list of findings is visible, including items like 'Data race (Impact: High)', 'Deadlock (Impact: High)', and 'Double lock (Impact: Medium)'. At the bottom, there are tabs for Project Browser, Results List, Dashboard, Output Summary, and Source.

File Reporting Access Tools Window Help

Run Bug Finder Stop

Project Browser

Bug_Finder_Example x Configuration

Target & Compiler

- Macros
- Environment Settings
- Inputs & Stubbing
- Multitasking
- Coding Standards & Code Metrics
- Bug Finder Analysis
- Code Prover Verification
- Verification Assumptions
- Check Behavior
- Precision
- Scaling
- Reporting
- Run Settings
- Advanced Settings

Target Language

Source code language: C

C standard version: defined-by-compiler

Target Environment

Compiler: gnu8.x

Target processor type: x86_64

Compiler Behavior

Division round down

Pack alignment value: defined-by-compiler

Ignore pragma pack directives

Enum type definition: defined-by-compiler

Signed right shift: Arithmetical

Management of size_t: defined-by-compiler

Management of wchar_t: defined-by-compiler

Source

concurrency.c x

```

123 int global_var1;
124
125 void bug_deadlock_task1(void) {
126     acquire_sensor();
127     acquire_printer();
128     global_var1 += 1;
129     release_printer();
130     release_sensor();
131 }

```

Find defects custom

Concurrency

- Data race (Impact: High)
- Data race through standard library function call (Impact: High)
- Data race on adjacent bit fields (Impact: High)
- Deadlock (Impact: High)
- Missing unlock (Impact: High)
- Missing lock (Impact: Medium)
- Double lock (Impact: High)
- Double unlock (Impact: High)

act: Medium
k (Impact: Low)
n (Impact: Low)
: Low)
an expression (Impact: Medium)
ot atomic (Impact: Medium)
up not wrapped in loop (Impact: Low)
t wrapped in loop (Impact: Low)
act: Medium
kct: Medium)
read attribute (Impact: Medium)
condition variable (Impact: Low)
(Impact: Medium)
condition variable (Impact: Medium)
escaping from a thread (Impact: Medium)

Project Browser Results List

Dashboard Output Summary Source

18

Polyspace Code Prover ISO 26262 Part 6进一步覆盖要求和增强...

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

Topics	ASIL			
	A	B	C	D
1i Concurrency aspects (more...)	+	+	+	+
(...)				

Table 4 – Methods for the Verification of the Software Architectural Design

Topics	ASIL			
	A	B	C	D
(...)				
1g Data flow analysis	+	+	‡	‡
1f Control flow analysis	+	+	‡	‡
1e Formal verification	o	o	+	+
(...)				

Table 6 – Design Principles for Software Unit Design and Implementation

Topics	ASIL			
	A	B	C	D
1j No recursions (more...)	+	+	‡	‡
(...)				
1g No implicit type conversions (more...)	+	‡	‡	‡
1f Restricted use of pointers (more...)	+	‡	‡	‡
1e Avoid global variables or else justify their usage (more...)	+	+	‡	‡
(...)				
1c Initialization of variables (more...)	‡	‡	+	‡
(...)				

Table 7 – Methods for Software Unit Verification

Topics	ASIL			
	A	B	C	D
(...)				
1g Data flow analysis	+	+	‡	‡
1f Control flow analysis	+	+	‡	‡
1e Formal verification	o	o	+	+
(...)				
1a Walk-through (more...)	‡	+	o	o

Table 10 – Methods for Verification of Software Integration

Topics	ASIL			
	A	B	C	D
1h Static analysis based on abstract interpretation	+	+	+	+
(...)				
1f Verification of the control flow and data flow	+	+	‡	‡
(...)				

Table 15 – Methods for Deriving Test Cases for Software Integration Testing

Topics	ASIL			
	A	B	C	D
(...)				
1e Analysis of functional dependencies	+	+	‡	‡
(...)				

Key:
 Bug Finder Credit
 Code Prover Credit
 Code Prover Enhances Bug Finder

Table 1 Table 2 Table 3 Table 4 Table 5 Table 6 Table 7 Table 8 Table 9 Table 10 Table 11 Table 12 Table 13 Table 14 Table 15

Tables

Polyspace Code Prover 证明变量访问冲突

V Polyspace R2022a - Code_Prover_Example C:\Workspace\Polyspace\Examples\R2022a\Code_Prover_Example\Module_1\CP_Result_1

File Reporting Access Tools Window Help

Run Code Prover Stop | Search

Results List

All results New Showing 323/323

Family	Information	File
Run-time Check	5 6 22 267	
Red Check	5	
Gray Check	6	
Orange Check	22	
Green Check	267	
Global Variable	1 3 2	
Shared	3 2	
Not shared	1	

Result Details

No check selected

Search - Source

Matches File Line

Start Page

Configuration Result Details

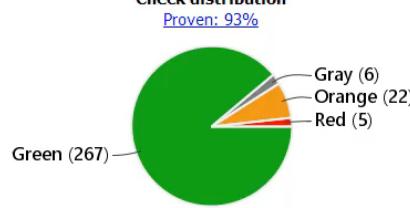
Dashboard C:\Workspace\Polyspace\Examples\R2022a\Code_Prover_Example\Module_1\CP_Result_1

Code_Prover_Example version 1.0 (31/05/2022) - Author: polyspace

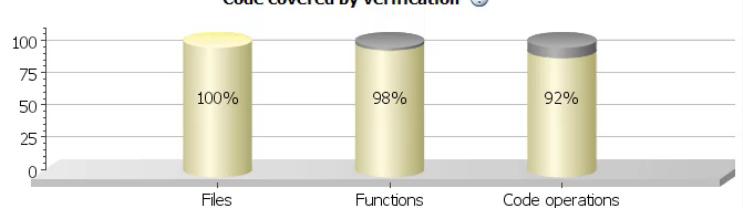
Analysis information: Configuration - Unreachable functions - Analysis assumptions - Concurrency modeling

Review Scope: All results - View all results in this scope

Check distribution
Proven: 93%



Code covered by verification



Top 5 orange sources
Total: 15 check(s) caused by orange sources

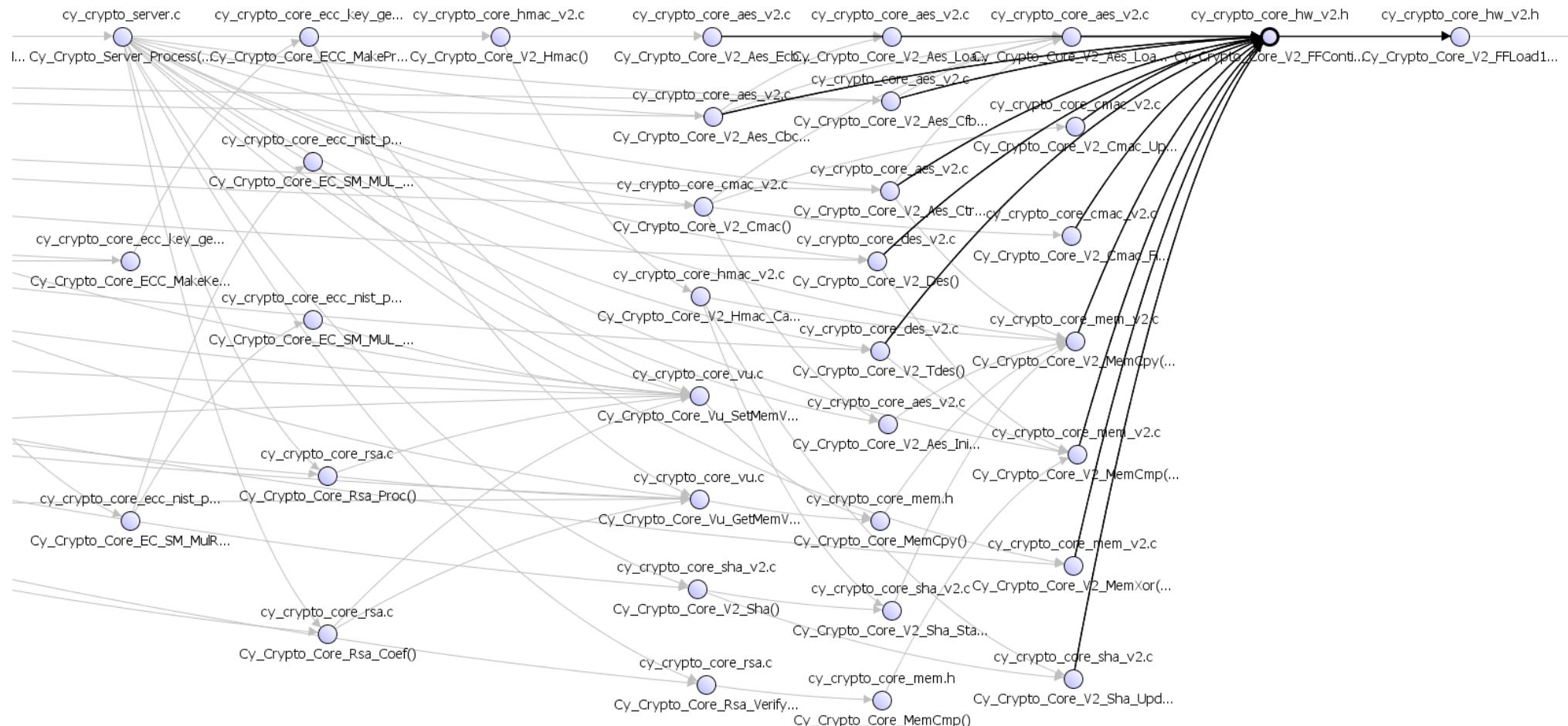


Project Browser Results List

Dashboard Source Output Summary Graph

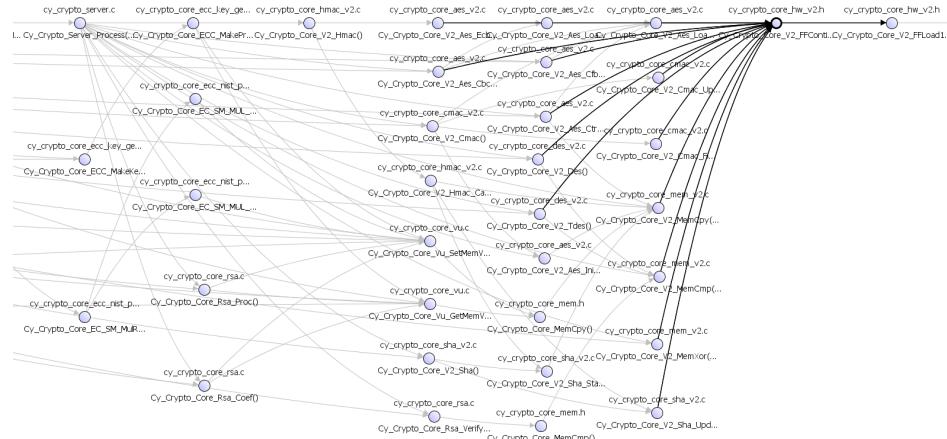
中 ° 简 ☰

从 Code Prover 提供控制流和数据流分析



问题追溯树

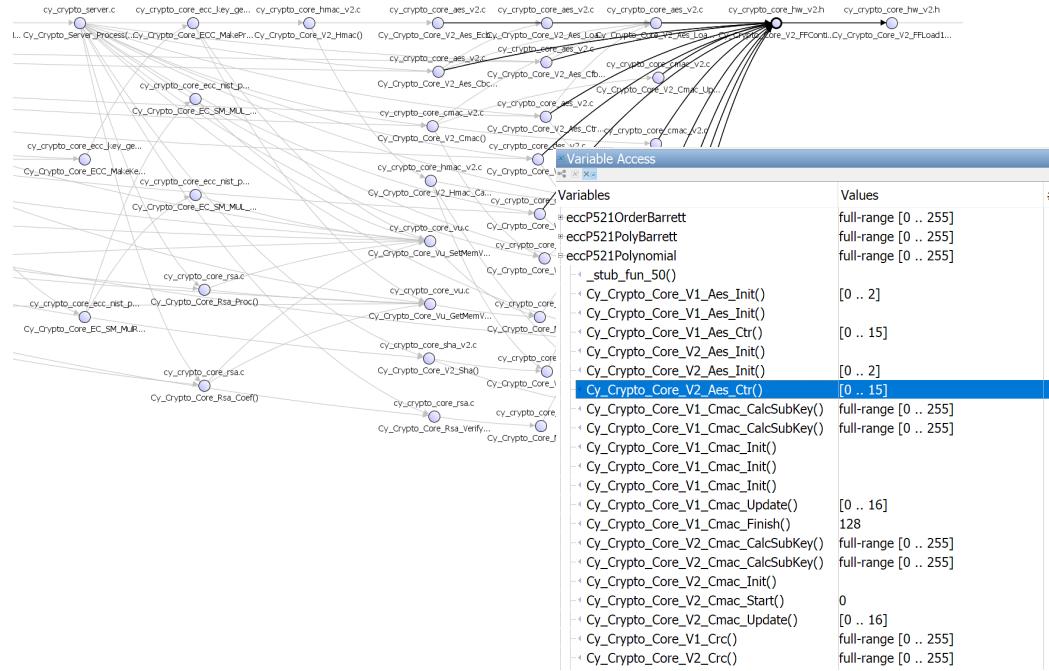
从 Code Prover 提供控制流和数据流分析



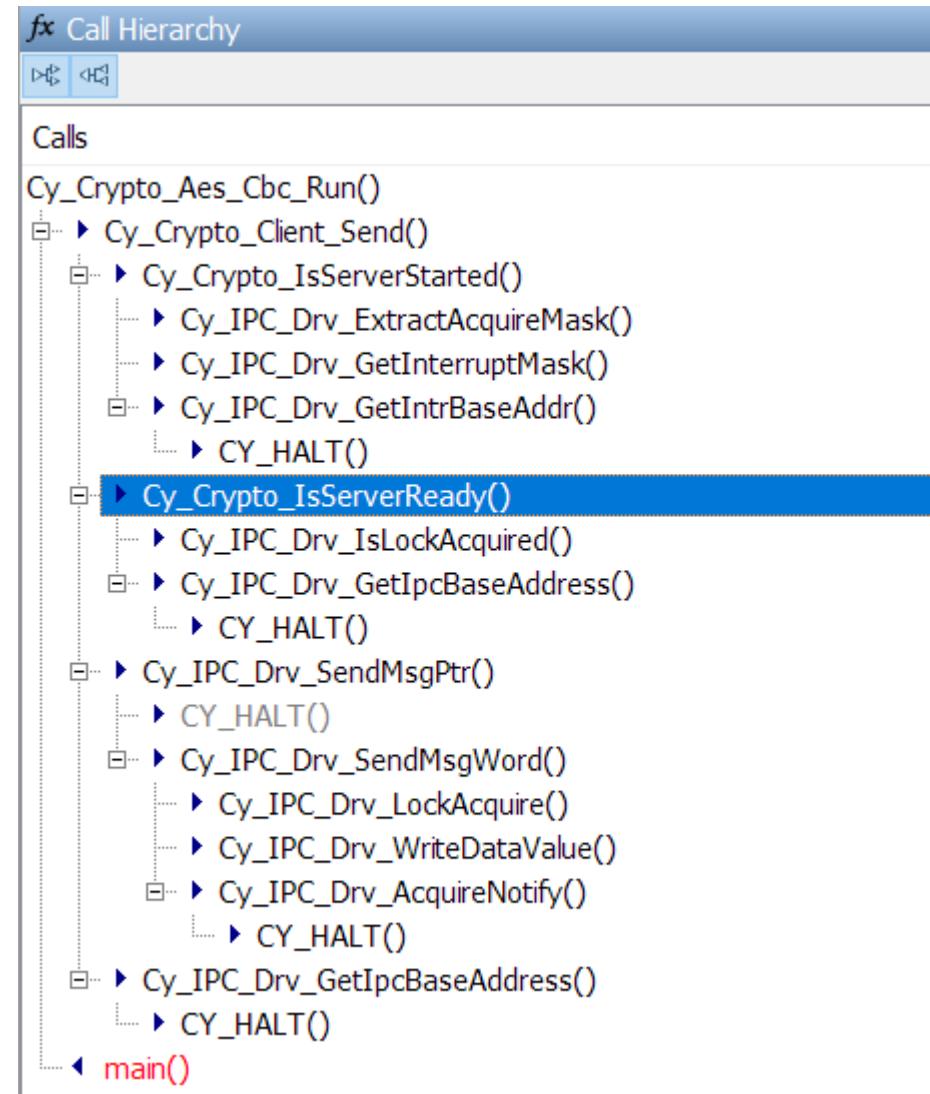
变量访问树

Variable Access		
Variables	Values	#
eccP521OrderBarrett	full-range [0 .. 255]	
eccP521PolyBarrett	full-range [0 .. 255]	
eccP521Polynomial	full-range [0 .. 255]	
_stub_fun_50()		
Crypto_Core_V1_Aes_Init()	[0 .. 2]	
Crypto_Core_V1_Aes_Init()	[0 .. 15]	
Crypto_Core_V1_Aes_Ctr()	[0 .. 2]	
Crypto_Core_V2_Aes_Init()	[0 .. 15]	
Crypto_Core_V2_Aes_Ctr()	[0 .. 15]	
Crypto_Core_V1_Cmac_CalcSubKey()	full-range [0 .. 255]	
Crypto_Core_V1_Cmac_CalcSubKey()	full-range [0 .. 255]	
Crypto_Core_V1_Cmac_Init()		
Crypto_Core_V1_Cmac_Init()		
Crypto_Core_V1_Cmac_Init()		
Crypto_Core_V1_Cmac_Update()	[0 .. 16]	
Crypto_Core_V1_Cmac_Finish()	128	
Crypto_Core_V2_Cmac_CalcSubKey()	full-range [0 .. 255]	
Crypto_Core_V2_Cmac_CalcSubKey()	full-range [0 .. 255]	
Crypto_Core_V2_Cmac_Init()		
Crypto_Core_V2_Cmac_Start()	0	
Crypto_Core_V2_Cmac_Update()	[0 .. 16]	
Crypto_Core_V1_Crc()	full-range [0 .. 255]	
Crypto_Core_V2_Crc()	full-range [0 .. 255]	

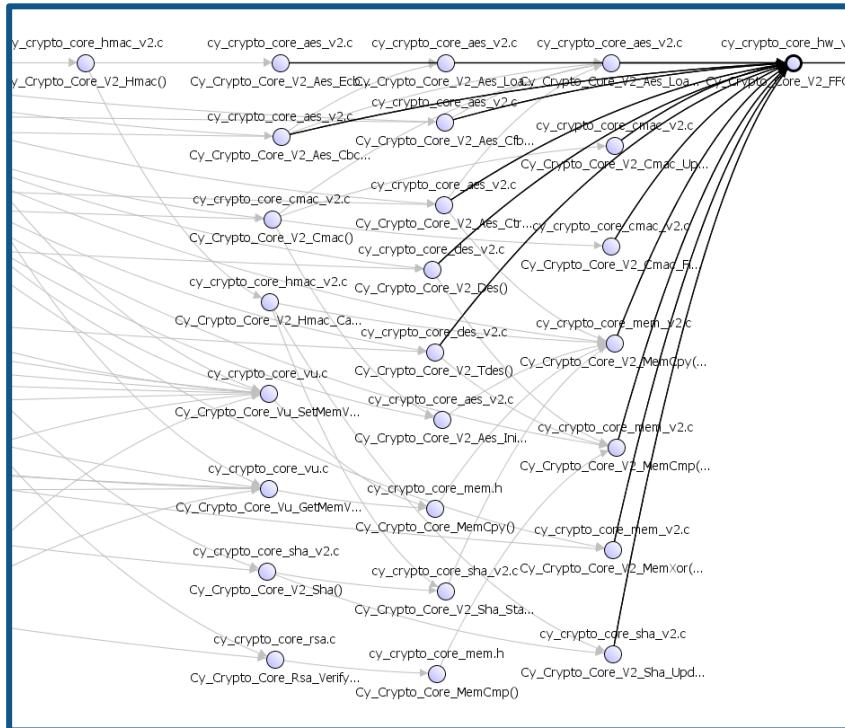
从 Code Prover 提供控制流和数据流分析



函数调用树



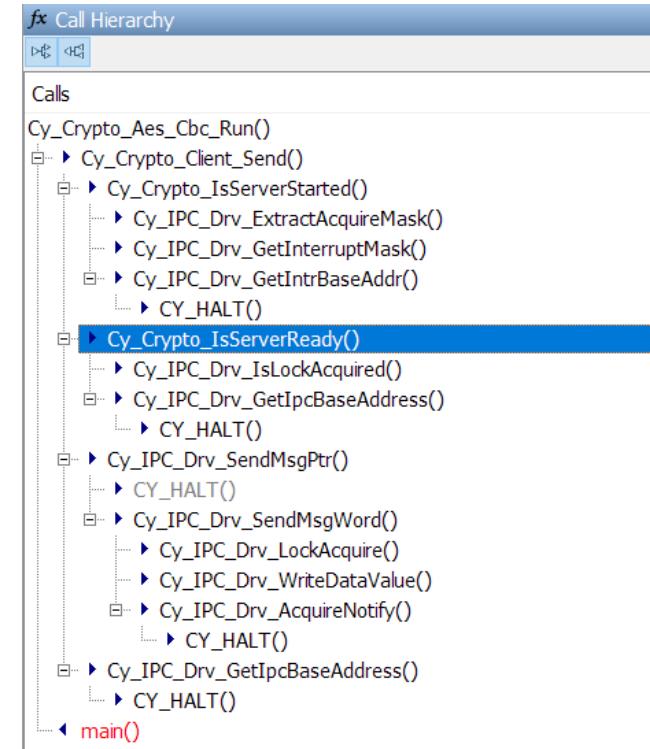
从 Code Prover 提供控制流和数据流分析



问题追溯树

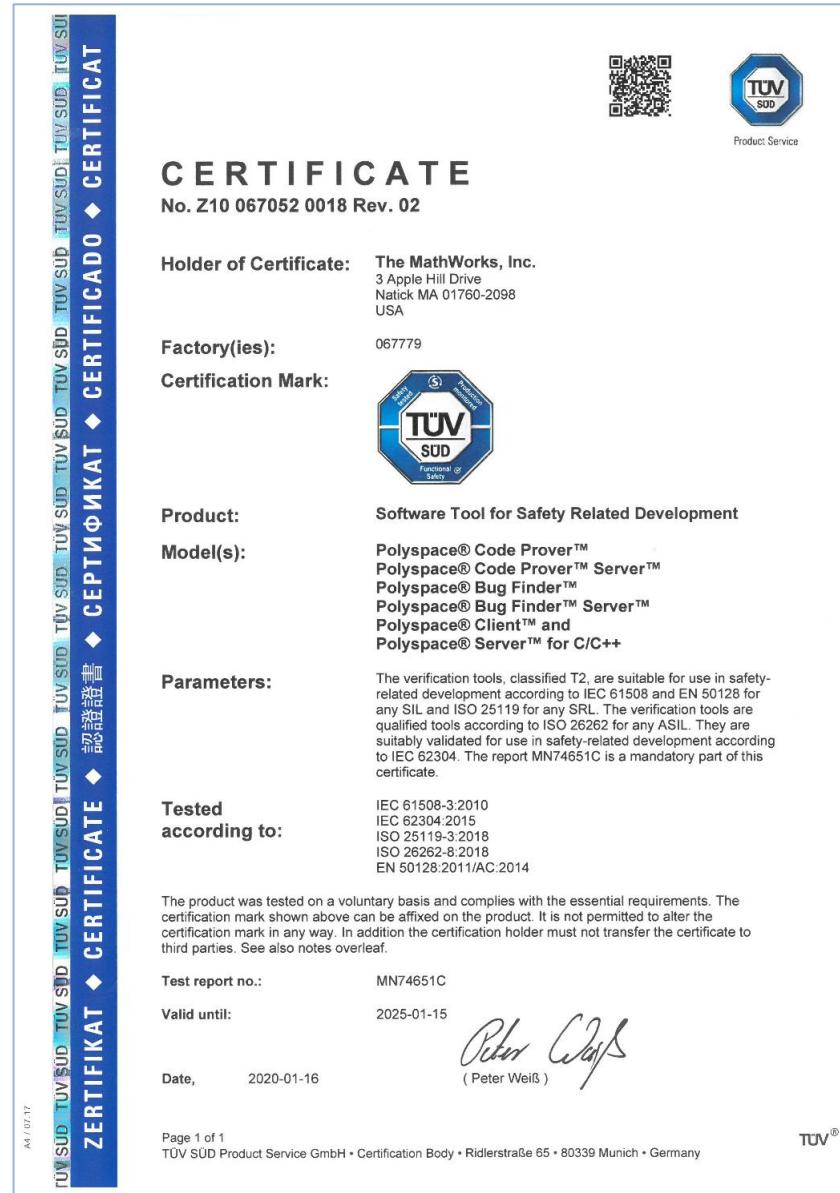
Variables	Values
eccP521OrderBarrett	full-range [0 .. 255]
eccP521PolyBarrett	full-range [0 .. 255]
eccP521Polynomial	full-range [0 .. 255]
_stub_fun_50()	[0 .. 2]
Cy_Crypto_Core_V1_Aes_Init()	[0 .. 15]
Cy_Crypto_Core_V1_Aes_Init()	[0 .. 15]
Cy_Crypto_Core_V1_Aes_Ctr()	[0 .. 15]
Cy_Crypto_Core_V2_Aes_Init()	[0 .. 2]
Cy_Crypto_Core_V2_Aes_Ctr()	[0 .. 15]
Cy_Crypto_Core_V1_Cmac_CalcSubKey()	full-range [0 .. 255]
Cy_Crypto_Core_V1_Cmac_CalcSubKey()	full-range [0 .. 255]
Cy_Crypto_Core_V1_Cmac_Init()	[0 .. 16]
Cy_Crypto_Core_V1_Cmac_Init()	[0 .. 16]
Cy_Crypto_Core_V1_Cmac_Init()	[0 .. 16]
Cy_Crypto_Core_V1_Cmac_Update()	[0 .. 16]
Cy_Crypto_Core_V1_Cmac_Finish()	128
Cy_Crypto_Core_V2_Cmac_CalcSubKey()	full-range [0 .. 255]
Cy_Crypto_Core_V2_Cmac_CalcSubKey()	full-range [0 .. 255]
Cy_Crypto_Core_V2_Cmac_Init()	0
Cy_Crypto_Core_V2_Cmac_Start()	[0 .. 16]
Cy_Crypto_Core_V2_Cmac_Update()	full-range [0 .. 255]
Cy_Crypto_Core_V1_Crc()	full-range [0 .. 255]
Cy_Crypto_Core_V2_Crc()	full-range [0 .. 255]

变量访问树



函数调用树

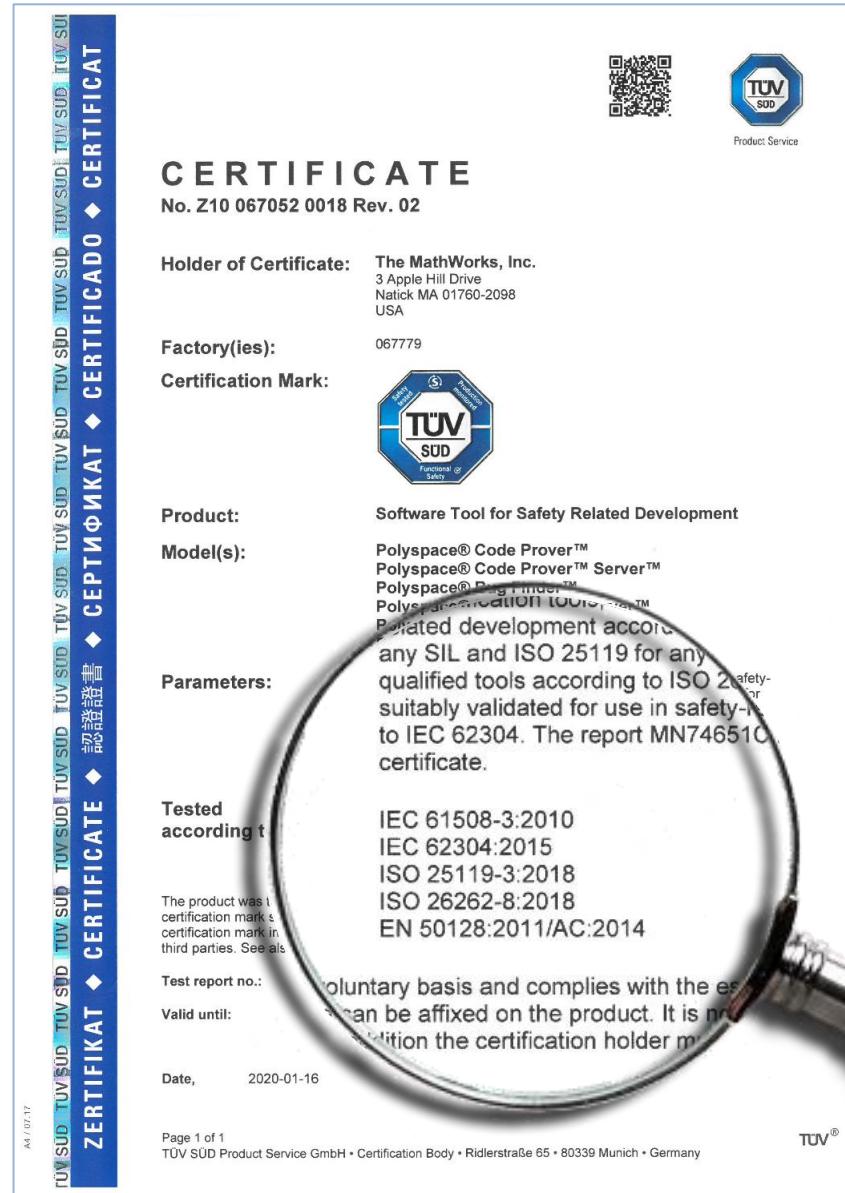
Polyspace 满足认证最高等级要求



Polyspace 满足认证最高等级要求



Polyspace 满足认证最高等级要求



Polyspace 满足认证最高等级要求



Rail

5.7.5 Summary

All Polyspace® tools versions listed in the subsequent table are qualified for all ASILs according to ISO 26262 up to a maximum tool confidence level of TCL2 for Polyspace® Bug Finder™ and TCL3 for Polyspace® Code Prover™. The review of the tool classification and the assessment of the results of the measures applied to qualify the software tools were carried out by TÜV SÜD.

伊必 Elektrobit：使用Polyspace 移除运行时错误 挑战和问题

无运行时错误来保
证车辆安全

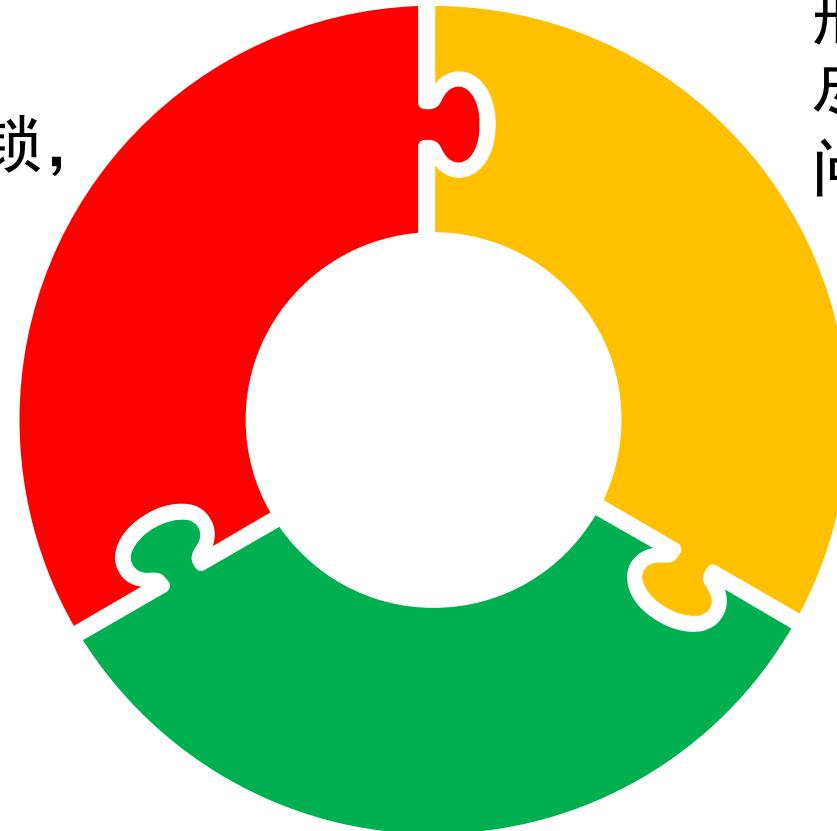


数千种系统配置

边界检查
需测试用例和运行

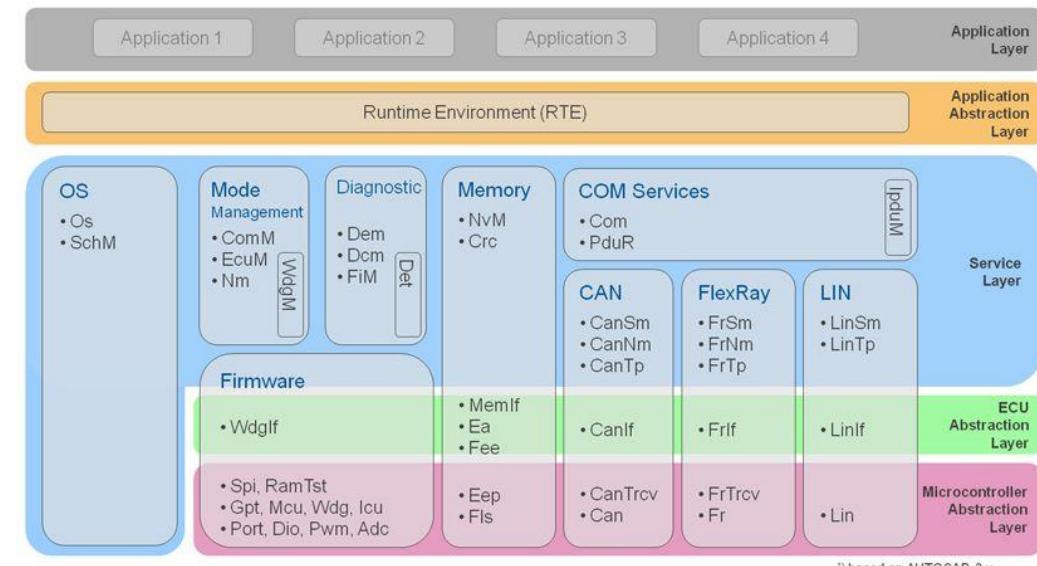
伊必 Elektrobit：使用Polyspace 移除运行时错误 Polyspace 如何解决

减少非必要互斥锁，
提高性能



形式化方法穷
尽分析，聚焦
问题

AUTOSAR Layered Architecture ¹⁾



MISRA C和HIS 代码度量

伊必 Elektrobit：使用Polyspace 移除运行时错误 成果总结



发现了诸如溢出、变量未初始化、并发等问题



代码证明比例超过97%，无 RTE



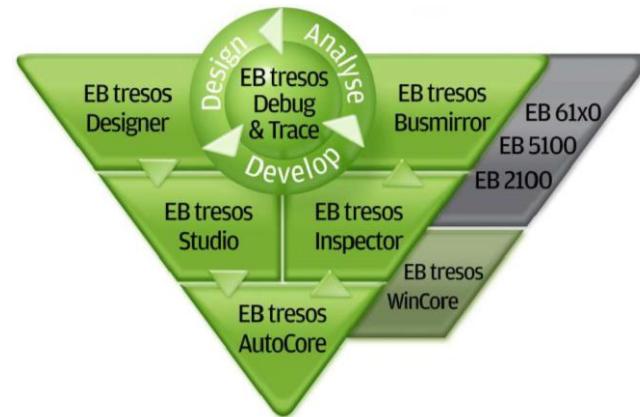
大幅度缩小验证时间



通过了ISO 26262 认证



使用持续测试技术，每次分析检测



Polyspace code verifiers enable us to demonstrate conclusively that the software we deliver is **free of certain run-time errors**. More importantly, they enable us to do so **faster, more thoroughly, and with less manual review** than was previously possible.

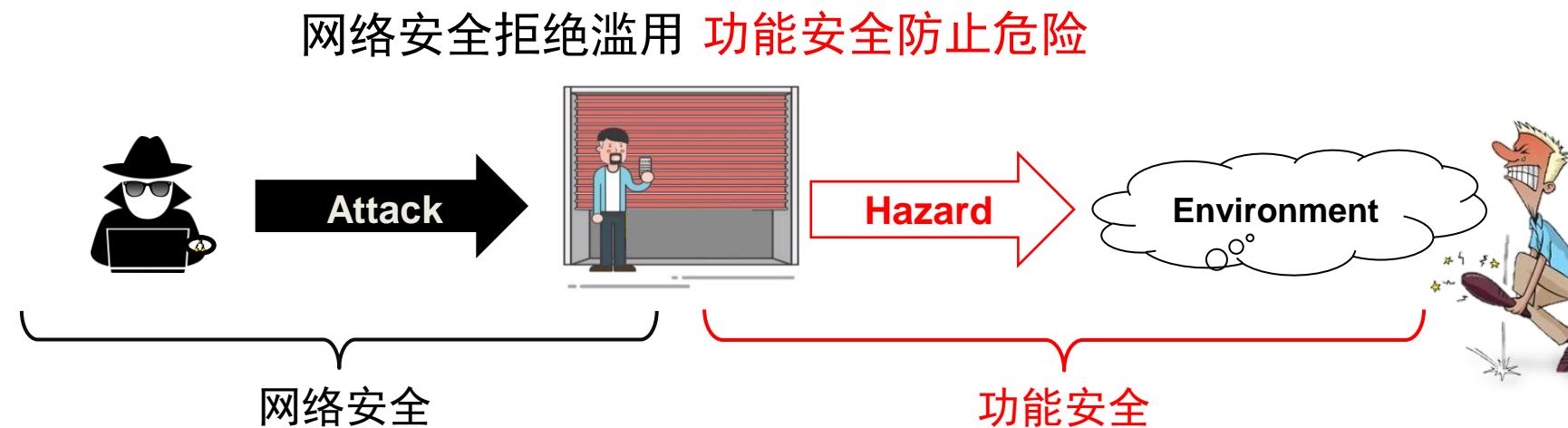
[案例链接](#)

Polyspace 为车联网之网络安全保驾护航

Safety 依赖 Security, Security 促进 Safety

Security: “在受攻击时，保证可靠的运行，Reliable computing in the presence of adversaries”

Safety: “没有危险，Absence of hazard”



使用 Polyspace 进行网络安全检查和证明

检查安全编码规范和实践指南



证明没有安全漏洞

CERT

CWE

MISRA

Defect distribution by category

<input checked="" type="checkbox"/> Check SEI CERT-C	all	View
<input type="checkbox"/> Check ISO/IEC TS 17961	all-rules	View
<input type="checkbox"/> Check Guidelines	publish-2016	View
<input type="checkbox"/> Check custom rules	all	View
	from-file	View

Bug Finder Analysis

Find defects

Defects

CWE

- default
- CWE**
- all
- custom

Results List

Family	Information	File	Class
Run-time Check			
<input type="checkbox"/> Gray Check	19	25	703
<input type="checkbox"/> Orange Check		25	
<input type="checkbox"/> Green Check		703	
<input type="checkbox"/> Absolute address usage		1	
<input type="checkbox"/> Division by zero		3	
<input type="checkbox"/> Function not returning value		14	
<input type="checkbox"/> Illegally dereferenced pointer		20	

Result Details

focVelocityEncoder_F28069.c / focVelocityEncoder_F28069_step0()

Result Review

Status: Unreviewed | Severity: Unset | Enter comment here...

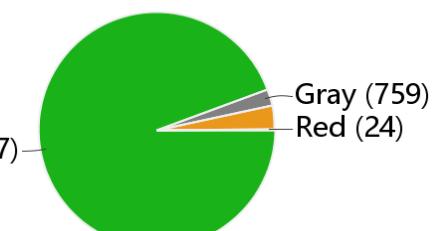
Illegally dereferenced pointer

Pointer is within its bounds

Dereference of local pointer 'meminddst' (pointer to unsigned int 16, size: 16 bits):
 Pointer is not null.
 Points to 1 bytes at offset 0 in buffer of 1 bytes, so is within bounds (if memory is allocated).
 Pointer may point to variable or field of variable:
 'focVelocityEncoder_F28069_B'.

Check distribution

Proven: 97%



```

932 }
933
934 /* S-Function (memcpy): '<Root>/QEP_Index_Pulse_Status'
935 {
936     uint16_T *memindsr = (uint16_T *)memindsr;
937     boolean_T *meminddst = (boolean_T *)meminddst;
938     (&focVelocityEncoder_F28069_B);
939     *(boolean_T *)meminddst) = *(uint16_T *)memindsr;
940 }

```

proof

IEC Certification Kit 提供 ISO/SAE 21434:2021 实践指南

IEC Certification Kit

Model-Based Design for ISO/SAE
21434:2021 Road vehicles —
Cybersecurity engineering

ISO/SAE 21434:2021 Clause	Applicable Requirements ID and Summary	Applicable Model-Based Design Tools and Processes	Comments
8. Continual cybersecurity activities	[RQ-08-05] Weaknesses evaluation	Polyspace Code Prover	You can use these MathWorks products to analyze system design and implementation for weakness that can contribute to vulnerabilities:
	[RQ-08-07] Vulnerability management	Polyspace Code Prover Server Polyspace Bug Finder Polyspace Bug Finder Server Requirements Toolbox	<ul style="list-style-type: none">• Polyspace tools can be used as Static Application Security Testing (SAST) tools to analyze source code for numerous defects related to security weaknesses and vulnerabilities, such as weak cryptographic algorithms.• Requirements Toolbox identifies requirements weakness through the traceability, implementation, and verification status)

形式化方法助力高效运行、低ROM占用的底层软件

高安全代码常用编码实践：防御性编程

```
3 int new_position_protected(int sensor_pos1, int sensor_pos2)
4 {
5     int actuator_position;
6     int x, y, tmp_pos, magnitude;
7
8     actuator_position = 2; /* default */
9     tmp_pos = 0;           /* values */
10    magnitude = sensor_pos1 / 100;
11    y = magnitude + 5;
12    x = actuator_position;
13
14    while (actuator_position < 10)
15    {
16        actuator_position++;
17        tmp_pos += sensor_pos2 / 100;
18        y += 3;
19    }
20
21    if ((3 * magnitude + 100) > 43)
22    {
23        magnitude++;
24        x = actuator_position;
25        if (x != y)
26        {
27            actuator_position = x / (x - y);
28        }
29        else
30        {
31            warning("Actuator position close to infinite, roudning by 1 (sensor1: %d, sensor2: %d)", sensor_pos1, sensor_pos2);
32            y = magnitude - 3; // update y to value from 1 cycle less
33            actuator_position = x / (x - y);
34        }
35    }
36    return actuator_position + tmp_pos; /* new value */
37 }
```

形式化方法证明，除法安全，无需防御

```
3 int new_position_protected(int sensor_pos1, int sensor_pos2)
4 {
5     int actuator_position;
6     int x, y, tmp_pos, magnitude;
7
8     actuator_position = 2; /* default */
9     tmp_pos = 0;           /* values */
10    magnitude = sensor_pos1 / 100;
11    y = magnitude + 5;
12    x = actuator_position;
13
14    while (actuator_position < 10)
15    {
16        actuator_position++;
17        tmp_pos += sensor_pos2 / 100;
18        y += 3;
19    }
20
21    if ((3 * magnitude + 100) > 43)
22    {
23        magnitude++;
24        x = actuator_position;
25        if (x != y)
26        {
27            actuator_position = x / (x - y);
28        }
29        else
30        {
31            warning("Actuator position close to infinite, roudning by 1 (sensor1: %d, sensor2: %d)", sensor_pos1, sensor_pos2);
32            y = magnitude - 3; // update y to value from 1 cycle less
33            actuator_position = x / (x - y);
34        }
35    }
36    return actuator_position + tmp_pos; /* new value */
37 }
```

防御性代码导致资源的浪费

The image shows the Compiler Explorer interface with two panes. The left pane displays a C source code snippet, and the right pane shows the generated ARM assembly code. Red boxes highlight specific sections of both the C code and the assembly code.

C Source Code:

```

1 // Type your code here, or load an example.
2 int new_position(int sensor_pos1, int sensor_pos2)
3 {
4     int actuator_position;
5     int x, y, tmp_pos, magnitude;
6
7     actuator_position = 2; /* default */
8     tmp_pos = 0;           /* values */
9     magnitude = sensor_pos1 / 100;
10    y = magnitude + 5;
11    x = actuator_position;
12
13    while (actuator_position < 10)
14    {
15        actuator_position++;
16        tmp_pos += sensor_pos2 / 100;
17        y += 3;
18    }
19    if ((3*magnitude + 100) > 43)
20    {
21        magnitude++;
22        x = actuator_position;
23        if (x != y){
24            actuator_position = x / (x - y);
25        } else {
26            warning("Actuator position close to infinite, rounding by 1 (sensor1: %d, sensor2: %d)", sensor_pos1, sensor_pos2);
27            actuator_position = x / (1 + x - y);
28        }
29    }
30    return actuator_position + tmp_pos; /* new value */
31 }

```

Generated Assembly Code:

```

ARM gcc 11.2 (linux) (C, Editor #1, Compiler #1)
56 adds r3, r3, #1
57 str r3, [r7, #16]
58 ldr r3, [r7, #28]
59 str r3, [r7, #12]
60 ldr r2, [r7, #12]
61 ldr r3, [r7, #24]
62 cmp r2, r3
63 beq .L5
64 ldr r2, [r7, #12]
65 ldr r3, [r7, #24]
66 subs r3, r2, r3
67 mov r1, r3
68 ldr r0, [r7, #12]
69 bl __aeabi_idiv
70 mov r3, r0
71 str r3, [r7, #28]
72 b .L4
73 .L5:
74 ldr r2, [r7]
75 ldr r1, [r7, #4]
76 movw r0, #:lower16:.LC0
77 movt r0, #:upper16:.LC0
78 bl warning
79 ldr r3, [r7, #12]
80 adds r2, r3, #1
81 ldr r3, [r7, #24]
82 subs r3, r2, r3
83 mov r1, r3
84 ldr r0, [r7, #12]
85 bl __aeabi_idiv
86 mov r3, r0
87 str r3, [r7, #28]
88 .L4:

```

A red arrow points from the text "防御性代码带来的汇编指令" to the assembly code, specifically highlighting the redundant code blocks at the bottom of the assembly listing.

防御性代码带来的汇编指令

删除防御性代码

```
int new_position(int sensor_pos1, int sensor_pos2)
{
    int actuator_position;
    int x, y, tmp_pos, magnitude;

    actuator_position = 2; /* default */
    tmp_pos = 0;           /* values */
    magnitude = sensor_pos1 / 100;
    y = magnitude + 5;
    x = actuator_position;

    while (actuator_position < 10)
    {
        actuator_position++;
        tmp_pos += sensor_pos2 / 100;
        y += 3;
    }
    if ((3 * magnitude + 100) > 43)
    {
        magnitude++;
        x = actuator_position;
        actuator_position = x / (x - y);
    }
    return actuator_position + tmp_pos; /* new value */
}
```

- 资源占用更少
- 性能更好
- 维护更容易
- 代码审查更简单

某IC驱动中不必要的条件或防御

```

uint32_t memResult = 1U;

if (size != 0U)
{
    Parameter 'size' (unsigned int 16): 8 or 15 or [19 .. 20] or 28 or 32
    C
    Conversion from unsigned int 16 to unsigned int 32
    right: 8 or 15 or [19 .. 20] or 28 or 32 or 48 or 64
    result: 8 or 15 or [19 .. 20] or 28 or 32 or 48 or 64

    /* I
     * Press 'F2' for focus
    Crypto_Run3ParamInstr(base,
        CRYPTO_V1_STR_MEMCMP_OPC,
        CRYPTO_RSRC0_SHIFT,
        CRYPTO_RSRC4_SHIFT,
        CRYPTO_RSRC8_SHIFT);

```

多余的判定条件

```

switch (mul_red_alg_select)
{
    case CRYPTO_NIST_P
        /* Curve-specific multiplication reduction algorithms */
        Crypto_Core_EC_CS_MulRed(base, z, x, size);
        break;
    case CRYPTO_NIST_P_SHIFT_MUL_RED_ALG:
        /* Shift-multiply, curve-specific multiplication reduction algorithms */
        Crypto_Core_EC_SM_MulRed(base, z, x, size);
        break;
    default:
        /* Generic Barrett modular reduction */
        Crypto_Core_EC_Bar_MulRed(base, z, x, size);
        break;
}

```

不可用的分支

```

/* Verify operation result */
if (timeout > 0U)
{
    Local variable 'timeout' (unsigned int 32): 0
    USBF
    retStatus = USBFS_DEV_DRV_SUCCESS;
}

```

多余的防御操作

更多关于软件性能的检查项

Bug Finder Analysis

Find defects custom

- Good practice
- Performance
 - std::endl may cause an unnecessary flush (Impact: Low)
 - Empty destructors may cause unnecessary data copies (Impact: Low)
 - Const return values may cause unnecessary data copies (Impact: Low)
 - Const parameter values may cause unnecessary data copies (Impact: Low)
 - Inefficient string length computation (Impact: Medium)
 - A move operation may throw (Impact: Low)
 - Expensive pass by value (Impact: Medium)
 - Expensive return by value (Impact: Medium)
 - Expensive copy in a range-based for loop iteration (Impact: Medium)
 - Const std::move input may cause a more expensive object copy (Impact: Medium)
 - std::move called on an unmovable type (Impact: Medium)
 - Missing constexpr specifier (Impact: Medium)
 - Expensive constant std::string construction (Impact: Medium)
 - Unnecessary use of std::string::c_str() or equivalent string methods (Impact: Medium)
 - Expensive use of std::string with empty string literal (Impact: Low)
 - Expensive use of std::string method instead of more efficient overload (Impact: Low)
 - Expensive use of non-member std::string operator+() instead of a simple append (Impact: Low)
 - Expensive use of substr() to shorten a std::string (Impact: Low)
 - Const rvalue reference parameter may cause unnecessary data copies (Impact: Low)
 - Expensive logical operation (Impact: Low)
 - Expensive local variable copy (Impact: Medium)
 - Expensive use of container's count method (Impact: Medium)
 - Expensive use of container's insertion method (Impact: Medium)
 - Expensive use of a standard algorithm when a more efficient method exists (Impact: Medium)
 - Expensive use of string functions from the C standard library (Impact: Low)
 - Use of new or make_unique instead of more efficient make_shared (Impact: Low)
 - Unnecessary padding (Impact: Medium)
 - Inefficient use of sprintf (Impact: Medium)
 - Expensive post-increment operation (Impact: Low)
 - Expensive dynamic_cast (Impact: Low)
 - Move operation uses copy (Impact: Medium)
 - Expensive return of a const object (Impact: Low)
 - Inefficient use of for loop (Impact: Low)
 - Expensive allocation in loop (Impact: Medium)

Expensive allocation in loop (Impact: Medium) [?](#) [🔗](#)
malloc and *free* called in a loop with a constant buffer size.
 Moving these calls outside the loop will avoid unnecessary work.

Event	File	Scope	L...
1 This constant expression is passed to <i>malloc</i>	expensive_by_value.c	expensive_by_value.c	30
2 This <i>free</i> call releases a constant amount of memory every loop iteration	expensive_by_value.c	expensive_by_value.c	32
3 Expensive allocation in loop	expensive_by_value.c	Loop()	30

Configuration **Result Details**

Source

```
expensive_by_value.c x
20
21
22
23
24
25
26
27
28
29
30 void* buffer = malloc(10); // Defect
31 use_buffer(buffer);
32 free(buffer);
33 }
```

在循环中分配内存

更多关于软件性能的检查项

Bug Finder Analysis

Find defects custom

- Good practice
- Performance
 - std::endl may cause an unnecessary flush (Impact: Low)
 - Empty destructors may cause unnecessary data copies (Impact: Low)
 - Const return values may cause unnecessary data copies (Impact: Low)
 - Const parameter values may cause unnecessary data copies (Impact: Low)
 - Inefficient string length computation (Impact: Medium)
 - A move operation may throw (Impact: Low)
 - Expensive pass by value (Impact: Medium)
 - Expensive return by value (Impact: Medium)
 - Expensive copy in a range-based for loop iteration (Impact: Medium)
 - Const std::move input may cause a more expensive object copy (Impact: Medium)
 - std::move called on an unmovable type (Impact: Medium)
 - Missing constexpr specifier (Impact: Medium)
 - Expensive constant std::string construction (Impact: Medium)
 - Unnecessary use of std::string::c_str() or equivalent string methods (Impact: Medium)
 - Expensive use of std::string with empty string literal (Impact: Low)
 - Expensive use of std::string method instead of more efficient overload (Impact: Low)
 - Expensive use of non-member std::string operator+() instead of a simple append (Impact: Low)
 - Expensive use of substr() to shorten a std::string (Impact: Low)
 - Const rvalue reference parameter may cause unnecessary data copies (Impact: Low)
 - Expensive logical operation (Impact: Low)
 - Expensive local variable copy (Impact: Medium)
 - Expensive use of container's count method (Impact: Medium)
 - Expensive use of container's insertion method (Impact: Medium)
 - Expensive use of a standard algorithm when a more efficient method exists (Impact: Medium)
 - Expensive use of string functions from the C standard library (Impact: Low)
 - Use of new or make_unique instead of more efficient make_shared (Impact: Low)
 - Unnecessary padding (Impact: Medium)
 - Inefficient use of sprintf (Impact: Medium)
 - Expensive post-increment operation (Impact: Low)
 - Expensive dynamic_cast (Impact: Low)
 - Move operation uses copy (Impact: Medium)
 - Expensive return of a const object (Impact: Low)
 - Inefficient use of for loop (Impact: Low)
 - Expensive allocation in loop (Impact: Medium)

Expensive allocation in loop (Impact: Medium) malloc and free called in a loop with a constant buffer size. Moving these calls outside the loop will avoid unnecessary work.

Event	File	Scope
1 This constant expression is passed to malloc	expensive_by_value.c	expensive_by_value.c:30
2 This free call releases a constant amount of memory every loop iteration	expensive_by_value.c	expensive_by_value.c:32
3 Expensive allocation in loop		

Expensive pass by value (Impact: Medium) This input parameter can be passed by const pointer/reference. This change avoids a copy for each call.

Event	File	Scope
1 Expensive pass by value expensive_by_value.c printPlayer_by_value() 37		

Configuration **Result Details**

Source

```
expensive_by_value.c
31
32 typedef struct _Player {
33     char name[50];
34     size_t rank;
35 } Player;
36
37 void printPlayer_by_value(Player const player)
38 {
```

使用值传递大内存变量

更多关于软件性能的检查项

Bug Finder Analysis

Find defects custom

Good practice

Performance

- std::endl may cause an unnecessary flush (Impact: Low)
- Empty destructors may cause unnecessary data copies (Impact: Low)
- Const return values may cause unnecessary data copies (Impact: Low)
- Const parameter values may cause unnecessary data copies (Impact: Low)
- Inefficient string length computation (Impact: Medium)
- A move operation may throw (Impact: Low)
- Expensive pass by value (Impact: Medium)
- Expensive return by value (Impact: Medium)
- Expensive copy in a range-based for loop iteration (Impact: Medium)
- Const std::move input may cause a more expensive object copy (Impact: Medium)
- std::move called on an unmovable type (Impact: Medium)
- Missing constexpr specifier (Impact: Medium)
- Expensive constant std::string construction (Impact: Medium)
- Unnecessary use of std::string::c_str() or equivalent string methods (Impact: Medium)
- Expensive use of std::string with empty string literal (Impact: Low)
- Expensive use of std::string method instead of more efficient overload (Impact: Low)
- Expensive use of non-member std::string operator+() instead of a simple append (Impact: Low)
- Expensive use of substr() to shorten a std::string (Impact: Low)
- Const rvalue reference parameter may cause unnecessary data copies (Impact: Low)
- Expensive logical operation (Impact: Low)
- Expensive local variable copy (Impact: Medium)
- Expensive use of container's count method (Impact: Medium)
- Expensive use of container's insertion method (Impact: Medium)
- Expensive use of a standard algorithm when a more efficient method exists (Impact: Medium)
- Expensive use of string functions from the C standard library (Impact: Low)
- Use of new or make_unique instead of more efficient make_shared (Impact: Low)
- Unnecessary padding (Impact: Medium)
- Inefficient use of sprintf (Impact: Medium)
- Expensive post-increment operation (Impact: Low)
- Expensive dynamic_cast (Impact: Low)
- Move operation uses copy (Impact: Medium)
- Expensive return of a const object (Impact: Low)
- Inefficient use of for loop (Impact: Low)
- Expensive allocation in loop (Impact: Medium)

The screenshot shows the Bug Finder Analysis interface with several findings:

- Expensive allocation in loop (Impact: Medium)**: This finding highlights that `malloc` and `free` are called in a loop with a constant buffer size. It suggests moving these calls outside the loop to avoid unnecessary work.
- Expensive pass by value (Impact: Medium)**: This finding notes that an input parameter is passed by value, which can be costly. It suggests using a const pointer/reference instead.
- Unnecessary padding (Impact: Medium)**: This finding indicates that a class or struct contains unnecessary padding. It suggests re-ordering members to save memory.

The code editor shows the following snippet from `expensive_by_value.c`:

```

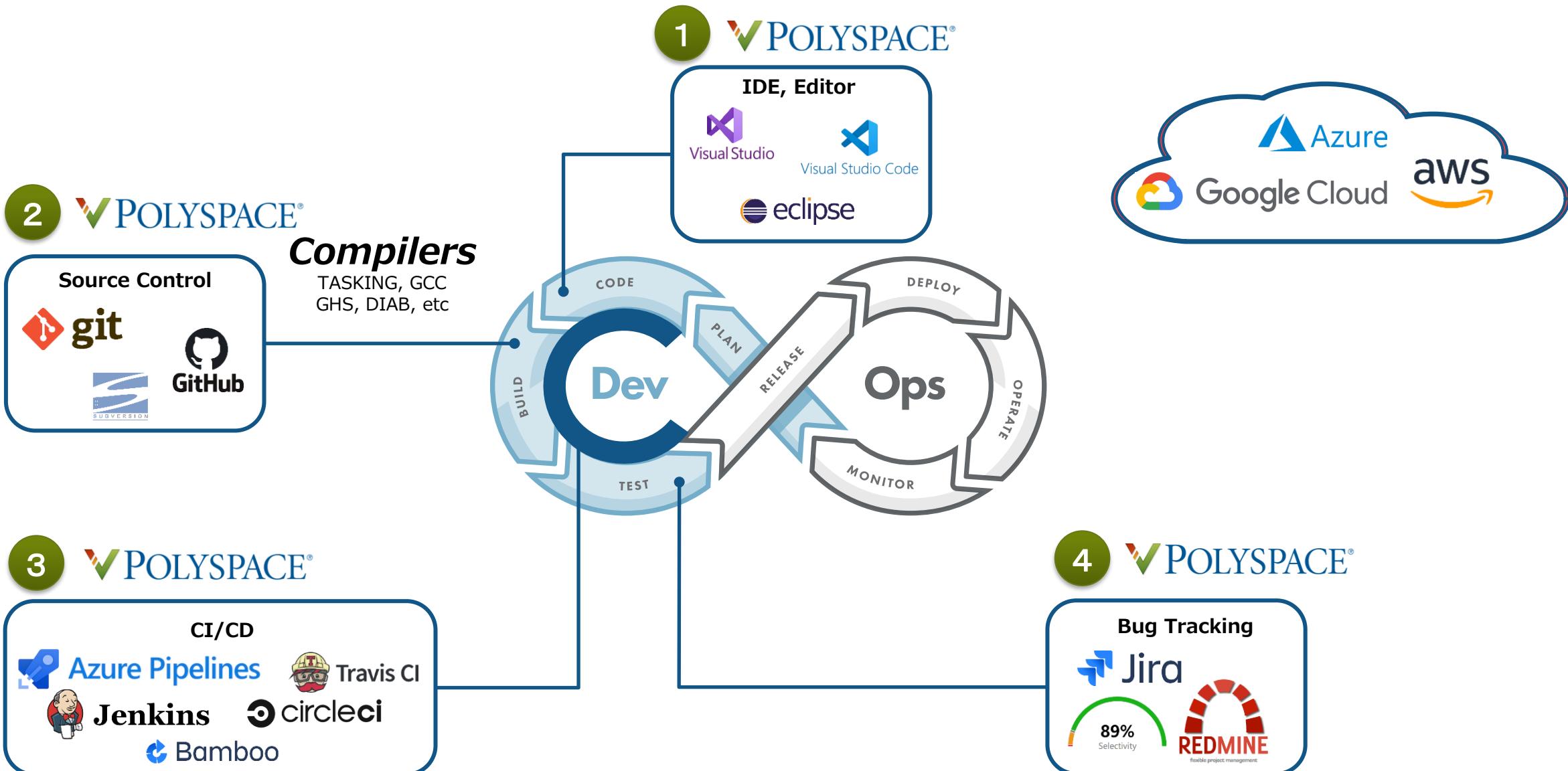
13
14 struct Array
15 {
16     uint8_t m1[5];
17     uint64_t m2;
18     uint8_t m3[3];
19 };

```

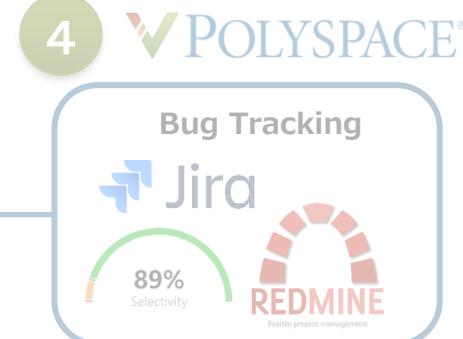
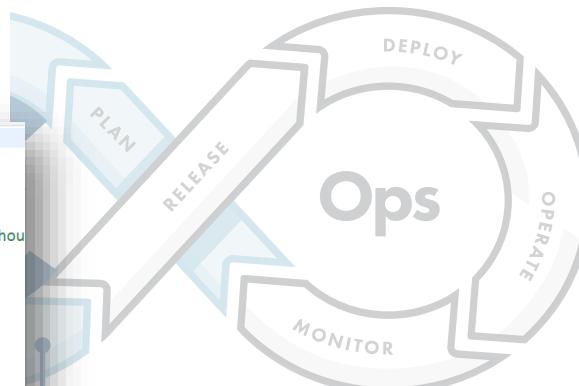
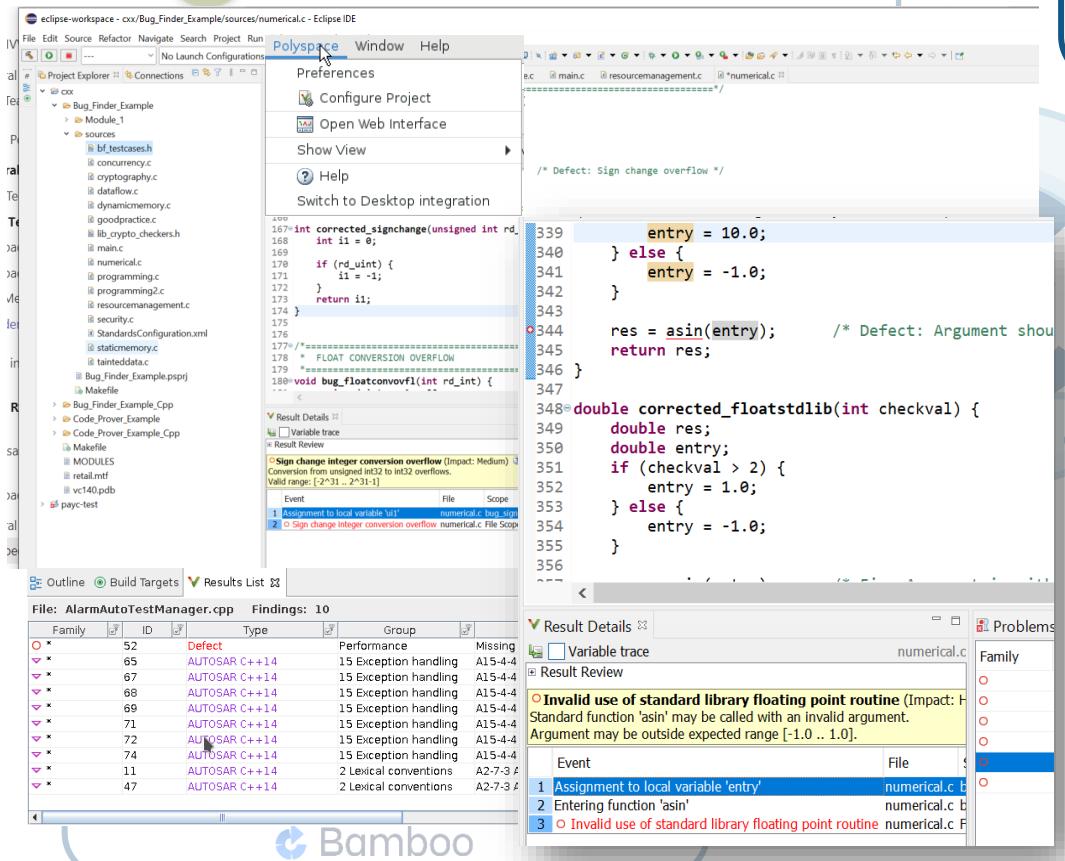
不合理的结构体成员顺序增加内存使用

将 Polyspace 运用到开发流程中

集成到 DevOps 中



集成到 DevOps 中



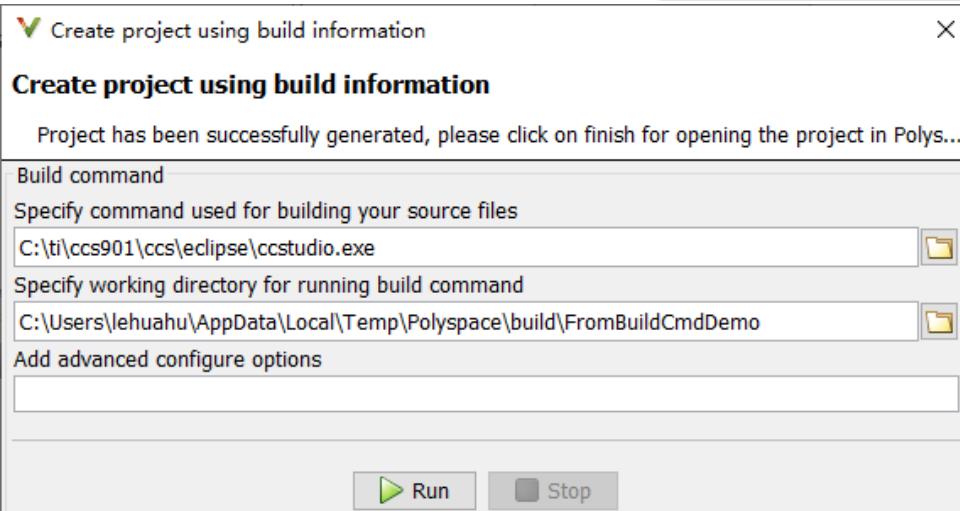
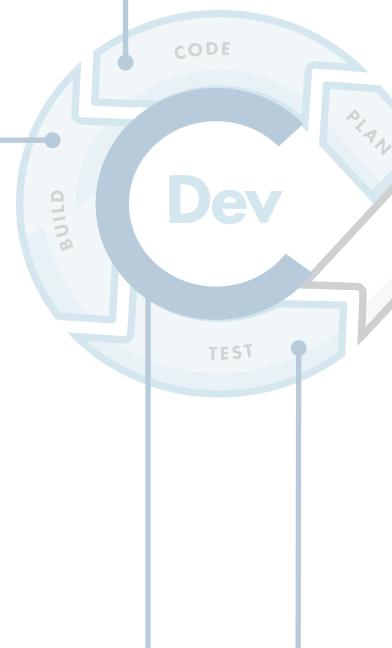
集成到 DevOps 中

2 POLYSPACE®



Compilers

TASKING, GCC
GHS, DIAB, etc



3 POLYSPACE®



```
polyspace-configure -allow-build-error -allow-overwrite -module -output-options-path options_path make -B
clude/ src/idl_options.proto

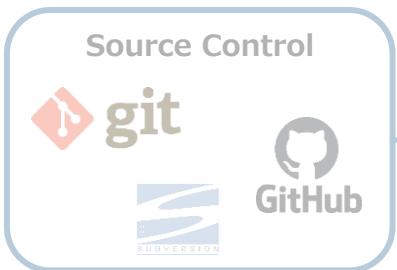
-DBTHREAD_USE_FAST_PTHREAD_MUTEX -D_const_= -D_GNU_SOURCE -DUSE_SYMBOLIZE -DNOCMALLOC -D_STDC_FORMAT_MACROS -D_
35\|2022-04-27T10:25:52+08:00\" -O2 -pipe -Wall -W -fPIC -fstrict-aliasing -Wno-invalid-offsetof -Wno-unused-parameter
.cpp -o src/mcpack2pb/generator.o
```

```
y/esp_authenticator.o src/brpc/policy/file_naming_service.o src/brpc/policy/hasher.o src/brpc/policy/consul_naming_service.o src/
c/brpc/policy/redis_protocol.o src/brpc/policy/ubrpc2pb_protocol.o src/brpc/policy/auto_concurrency_limiter.o -Xlinker "-")" -lgfl
t
> Copying to output/include
> Copying to output/lib
> Copying to output/bin
polyspace-configure: 416s: WARNING: Compilation units detected during your build in module libbrpc_a use different languages.
polyspace-configure: 416s: WARNING: Compilation units detected during your build in module libbrpc_so use different languages.
```

```
polyspace-configure: 117s: WARNING: Keeping potentially big build trace, remember to delete
polyspace-configure: 117s: WARNING: Keeping potentially big cache directory, remember to de
polyspace-configure: 0s: WARNING: Build command ignored (build deactivated)
polyspace-configure: 13s: INFO: Created project file C:\polyspace_workspace\FromBuildCmdD
polyspace-configure: 13s: WARNING: Keeping potentially big build trace, remember to delete it
```

集成到 DevOps 中

2 POLYSPACE®



Compilers

TASKING, GCC
GHS, DIAB, etc



流水线

定义

Pipeline script

脚本 ?

```

59
60      sh label: "Cleanup", script: "make clean"
61      sh label: "Polyspace configure", script: "$configure -allow-build-error -allow-overwrite -lang
62
63      }
64      stage ("Analyze")
65      {
66          sh label: "Polyspace analysis",
67          script: "$analyze -options-file ${PROG}.opts -misra3 all -code-metrics -results-dir $RESULT"
68
69
70
    // remove this paragraph if you do not use Polyspace Access
    withCredentials([usernamePassword(credentialsId: 'Access_UserName', passwordVariable: 'password', usernameVariable: 'username')])

```

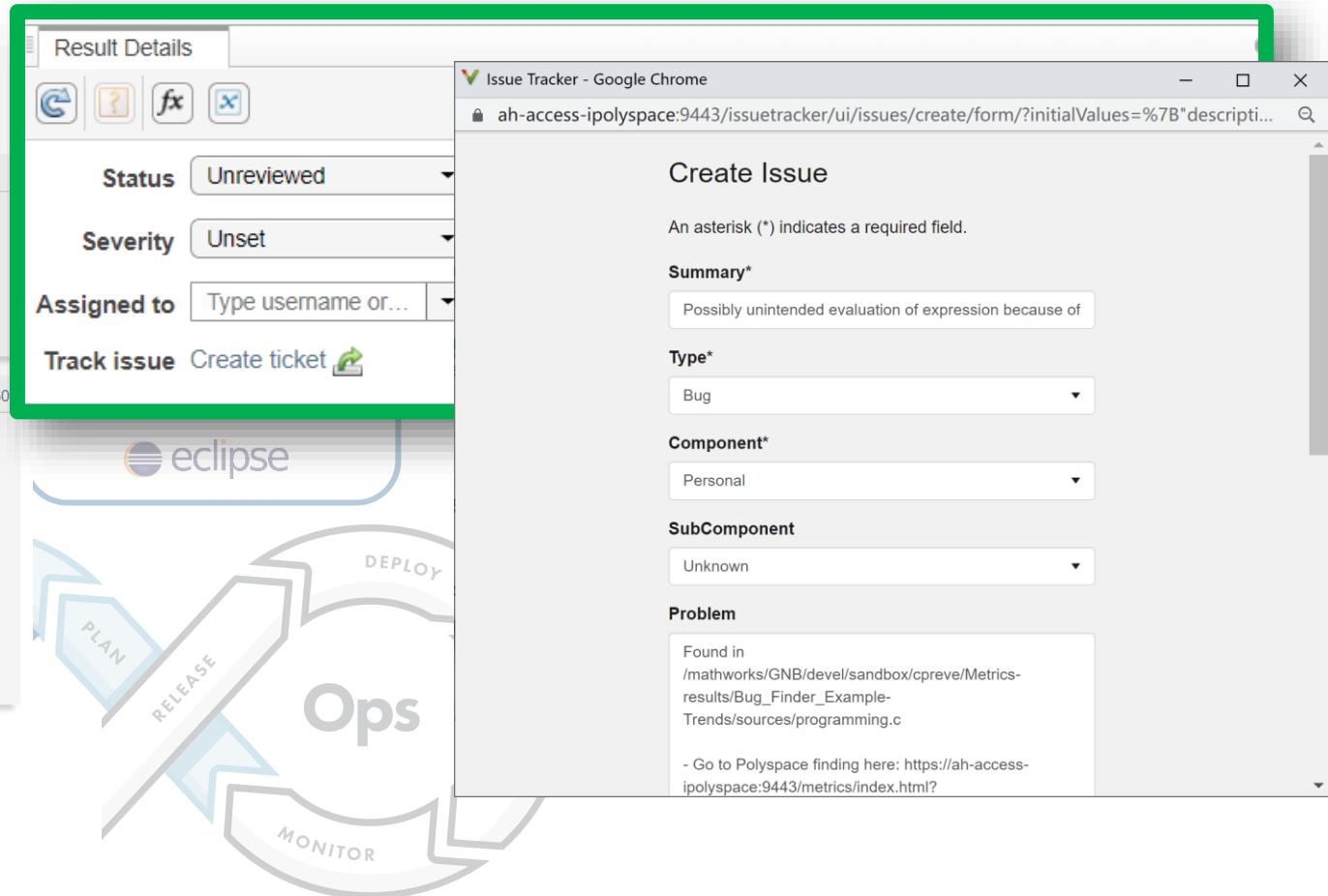
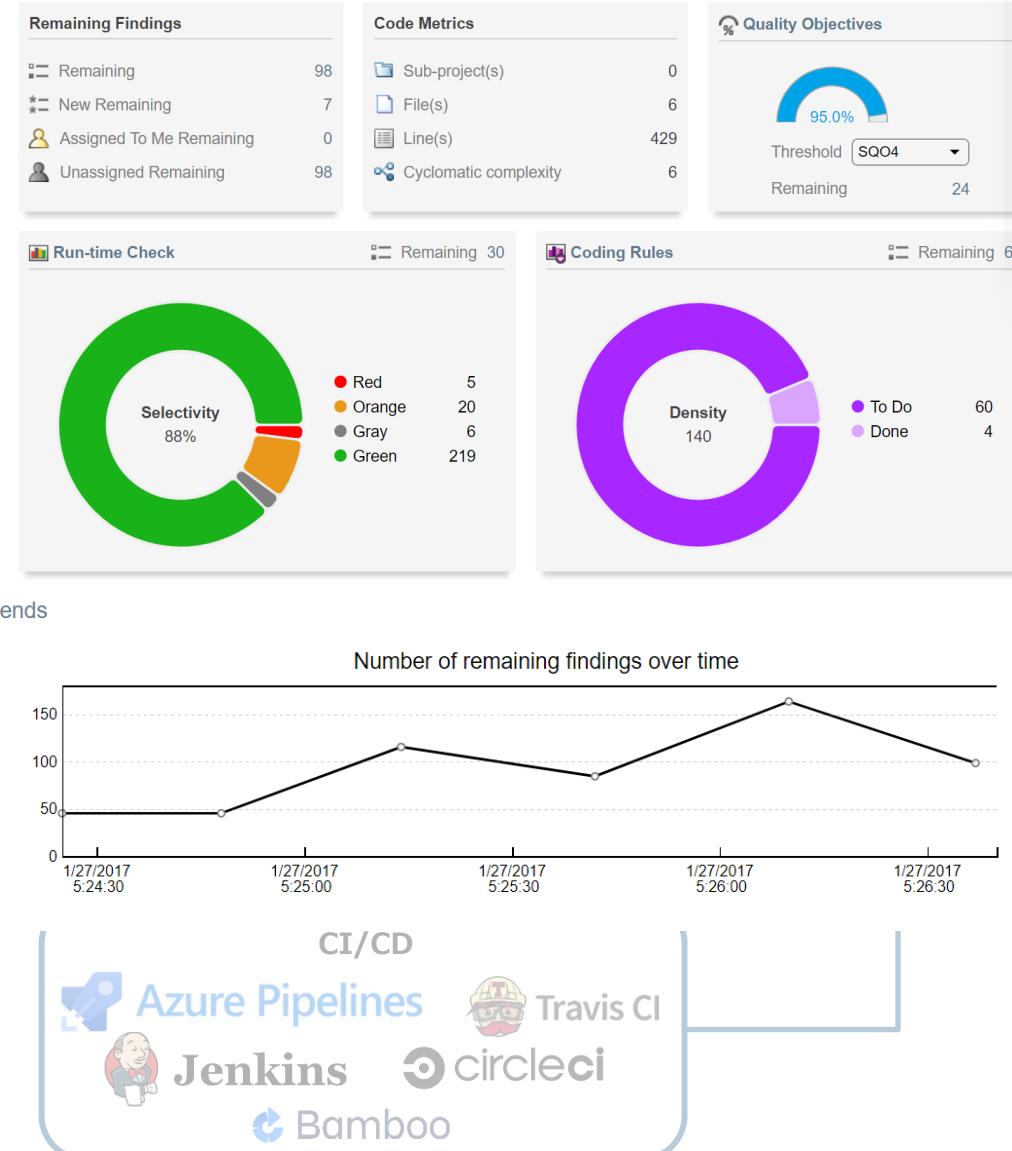
阶段视图

	Prepare	Checkout	Build	Polyspace Analysis	Upload to Access	Quality Gate	Email Notification
Average stage times: (Average full run time: ~40s)							
#41 Sep 13 15:35 No Changes	374ms	2s	9s	12s	4s	8s	875ms
#40 Sep 10 23:08 No Changes	323ms	2s	10s	13s	4s	8s	1s
#39 Sep 10 22:52 No Changes	303ms	1s	9s	11s	5s	8s	1s
	471ms	3s	10s	15s	4s	8s	1s

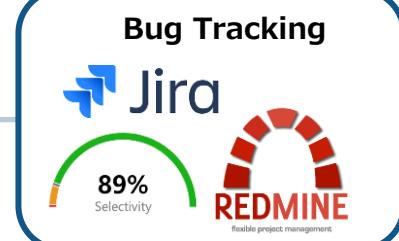
3 POLYSPACE®



集成到 DevOps 中



4 POLYSPACE®



要点回顾

- 形式化方法及抽象解释法
- Polyspace对 ISO 26262 和 ISO/SAE 21434 支持
- 形式化方法验证用于性能提升
- 将 Polyspace 应用到开发流程中

MATLAB EXPO

感谢您的聆听！



© 2022 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

