MATLAB EXPO
中国

Polyspace代码静态分析
保障嵌入式软件可靠性

Lehua Hu, MathWorks

MathWorks

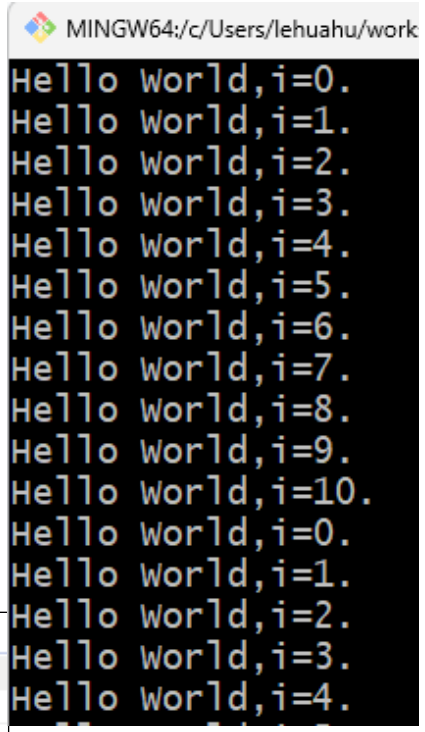
引子

```
int i = 0;
int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
for (i = 0; i <= 12; i++)
{
    arr[i] = 0;
    printf("Hello World,i=%d.\n",i);
}
```

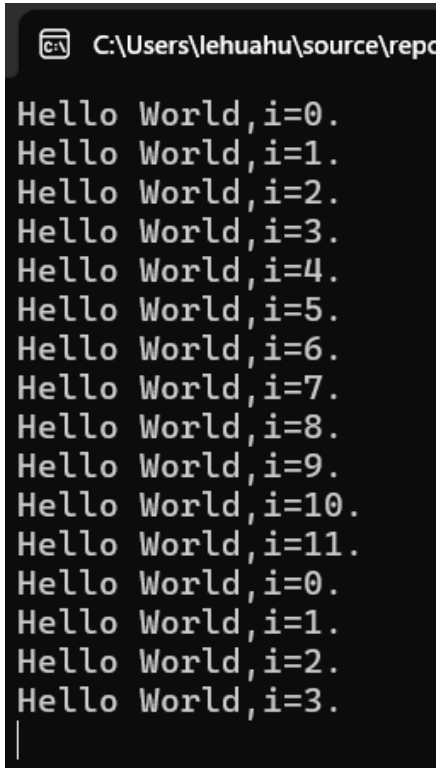
引子

```
int i = 0;
int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
for (i = 0; i <= 12; i++)
{
    arr[i] = 0;
    printf("Hello World,i=%d.\n",i);
}
```

Name	Value
i	11
arr	0x008ffad0 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
[0]	0
[1]	0
[2]	0
[3]	0
[4]	0
[5]	0
[6]	0
[7]	0
[8]	0
[9]	0
arr[10]	0
arr[11]	0
arr[12]	11



gcc

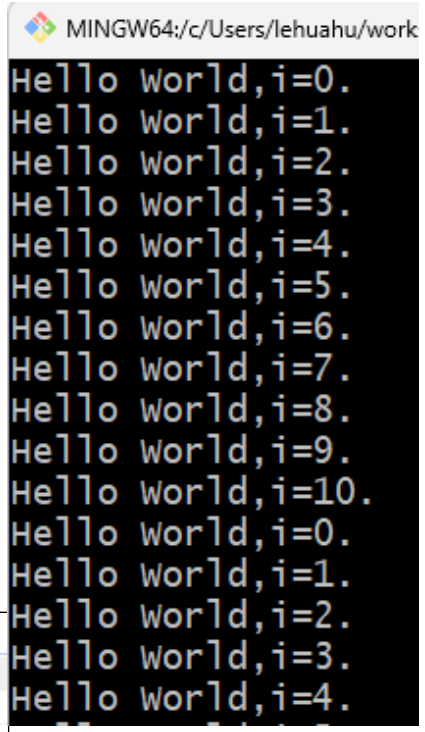


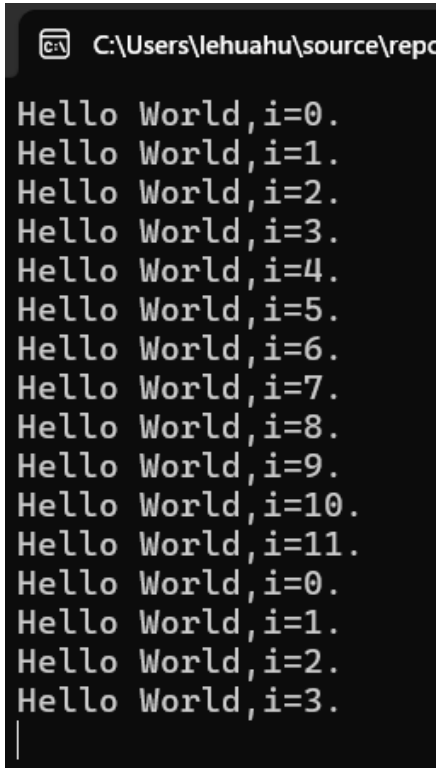
vc++

引子

```
int i = 0;
int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
for (i = 0; i <= 12; i++)
{
    arr[i] = 0;
    printf("Hello World,i=%d.\n",i);
}
```

Name	Value
i	11
arr	0x008ffad0 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
[0]	0
[1]	0
[2]	0
[3]	0
[4]	0
[5]	0
[6]	0
[7]	0
[8]	0
[9]	0
arr[10]	0
arr[11]	0
arr[12]	11
&arr[12]	0x008ffb00 {11}
	11
&i	0x008ffb00 {11}
	11



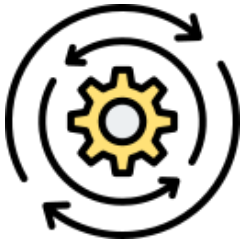


这段代码呢？

```
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 }
```

内容提纲

形式化证明与软件可靠性



集成Polyspace开发流程中

支持功能安全和信息安全的形式化方法使用

高可靠性的敌人：运行时错误

- A runtime error is error that occurs while a program is **running** and typically caused by issues such as **division by zero**, attempting to **access memory that is not allocated**, or using a **variable without initializing** it.

- Runtime errors can cause a program to **terminate** or **behave unexpectedly**. Unlike syntax errors, runtime errors are can **ONLY** detected while the program is executing, and compilers **CANNOT** detect runtime error.

嵌入式软件可靠性的挑战

未检测出的软件bug所导致的灾难性后果

爱国者防空导弹
“拦截失败，误伤自军”

weapons control error.
28 American soldiers killed
100+ soldiers wounded

Accumulating Error

USS 约克镇号
无法航行

Propulsion system repeatedly shut down.

Divide-by-zero error

Therac 25
严重剂量过大

Patients severely overdosed.
6 Killed.

Race Condition
Overflow Error

传统测试方法在应对关键运行时错误问题上能力不足

动态测试

完全依赖测试用例，覆盖率有局限
时间成本、人力成本高
测试用例，尤其是边界/覆盖率测试，设计困难
测试用例随应用复杂急剧增加
问题反馈之后
查出问题浮于表面
在目标机上运行费时费力

基于需求的功能测试

一般静态分析

主要针对规则和语法类问题
语义分析对复杂代码理解有限
动态特性和运行时行为无法捕捉
深度bug漏报的问题

MISRA 规则检查和基于规则的缺陷

Polyspace 产品家族

形式化工具 Code Prover

- Prove Software Runtime Correctness
- Architecture Analysis

静态分析工具 Bug Finder

- Detect Safety and Security Vulnerabilities
- Check Adherence to Coding Standards
- Compute Source Code Metrics

动态测试工具 PSTest

- Test Authoring
- Test Execution, Coverage and Profiling
- Test Management

评审中心 PSAccess

- Online review and collaboration on Polyspace results
- IDE Plugin for checking local coding rules and issues

完整的静态解决方案

静态分析工具Bug Finder

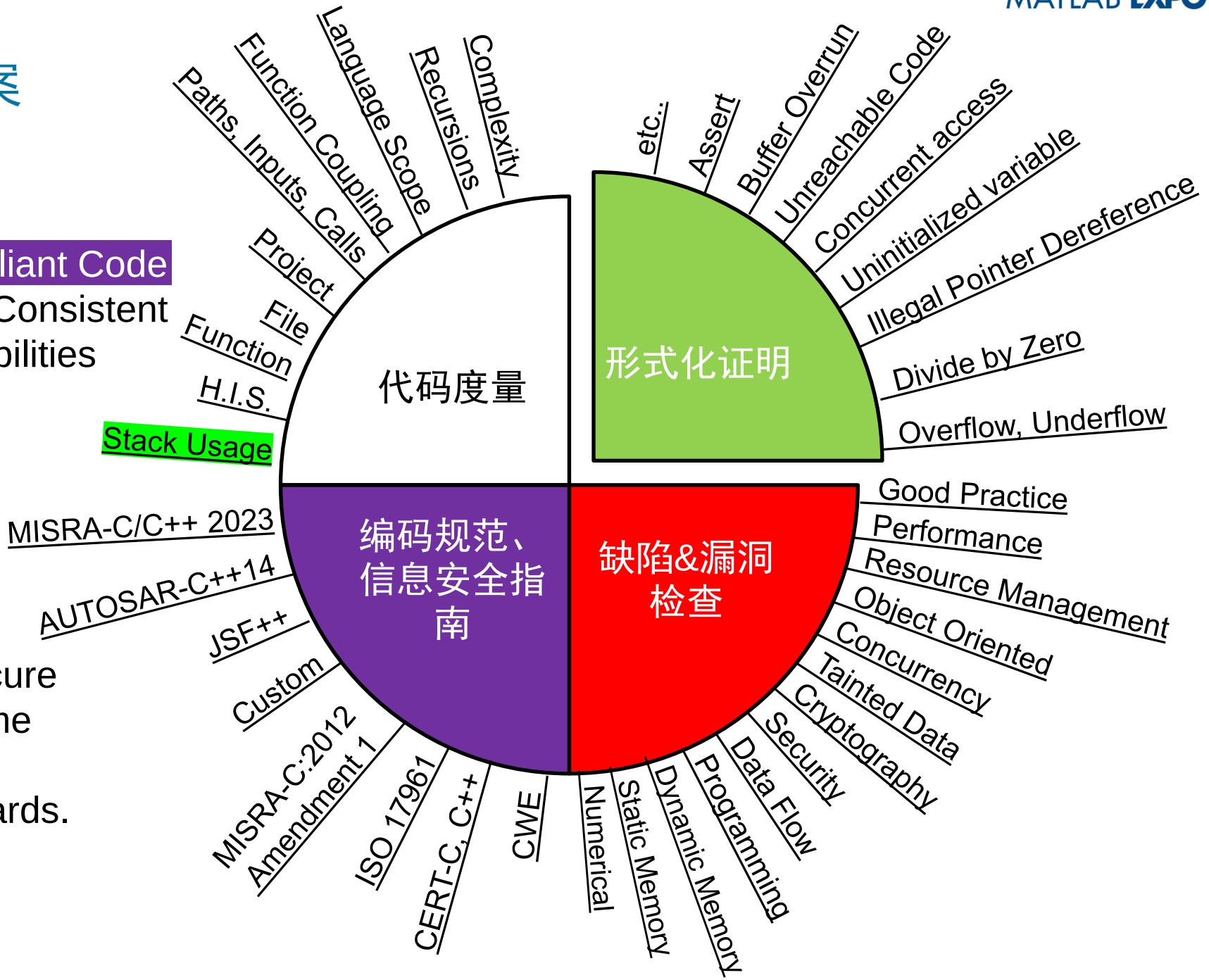
→High Quality, Secure, Compliant Code

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

形式化工具Code Prover

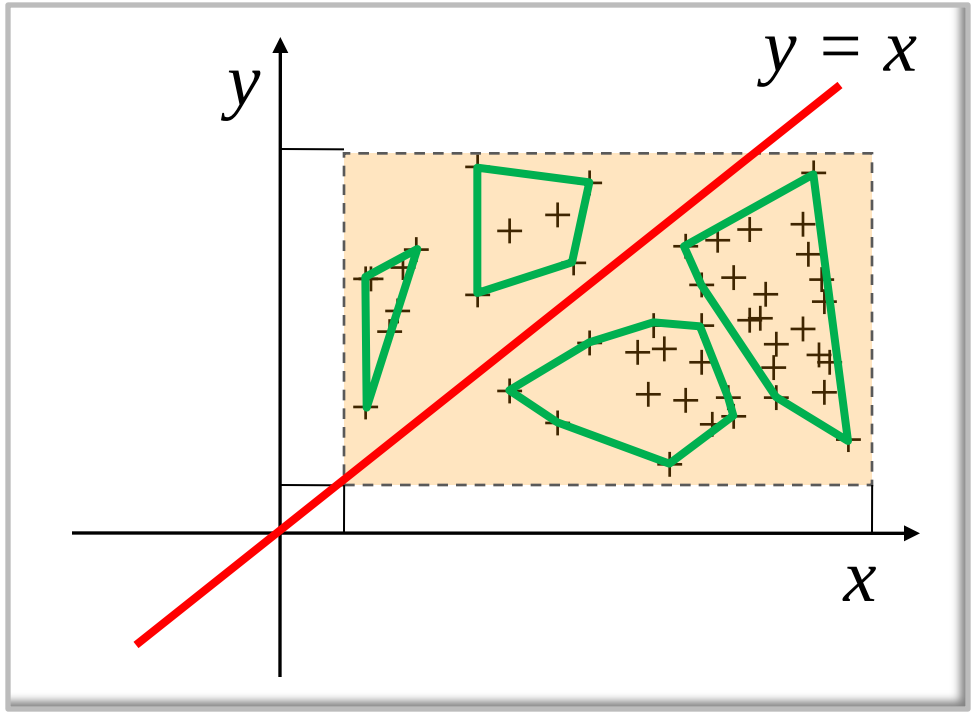
→Fully Trusted Components

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



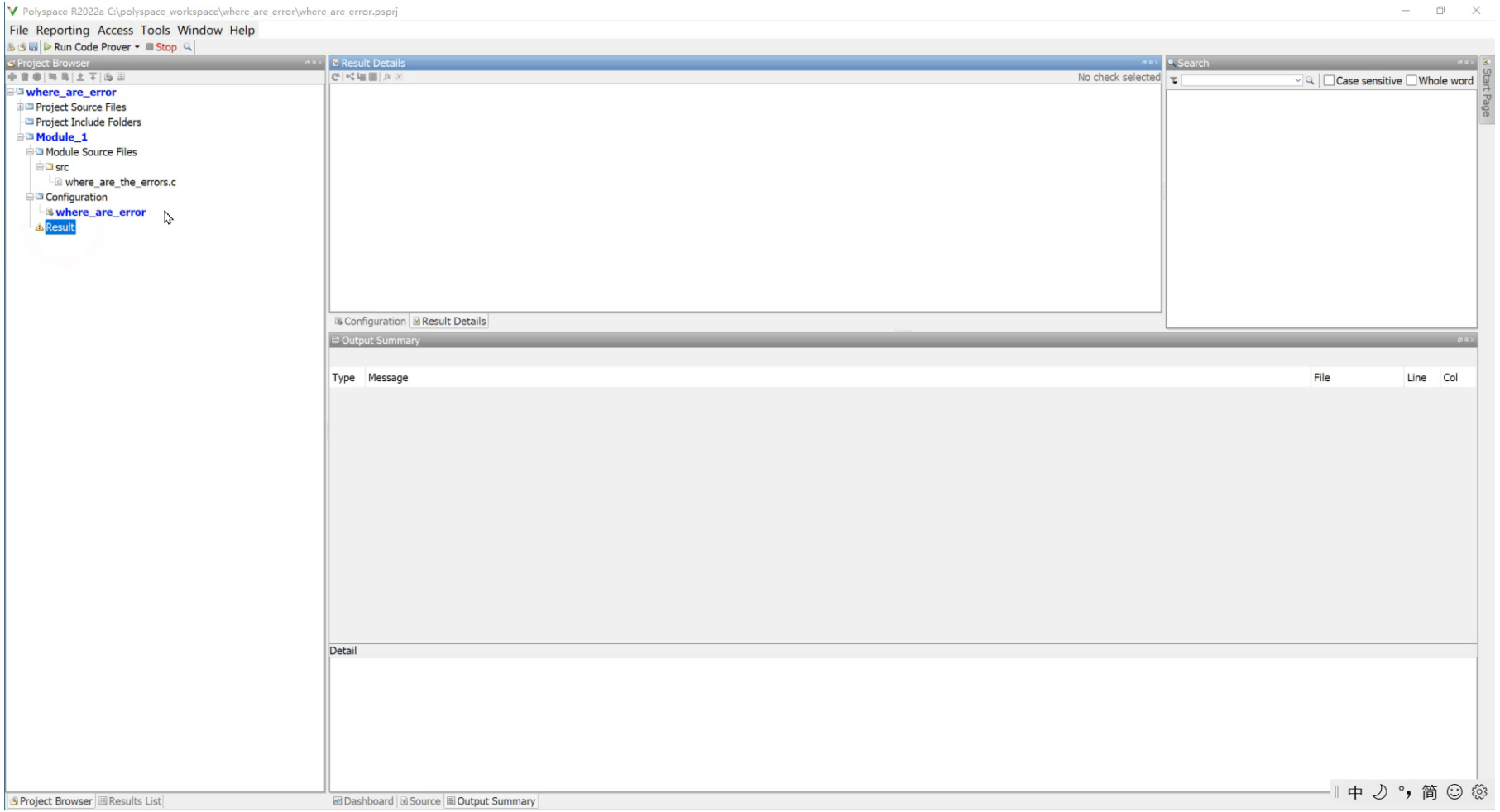
抽象解释法的形式化

验证 y=x/(x-y) 的除零

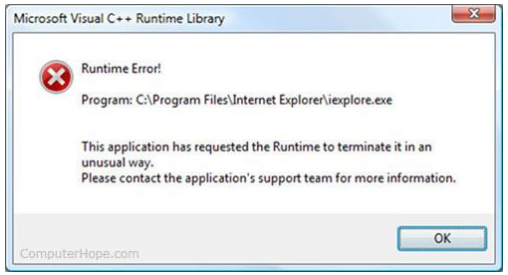


- 无需测试用例
- 无需执行代码
- 静态分析

使用 Polyspace Code Prover 证明代码安全无虞



如何衡量软件内部质量



运行时错误

缺陷&漏洞

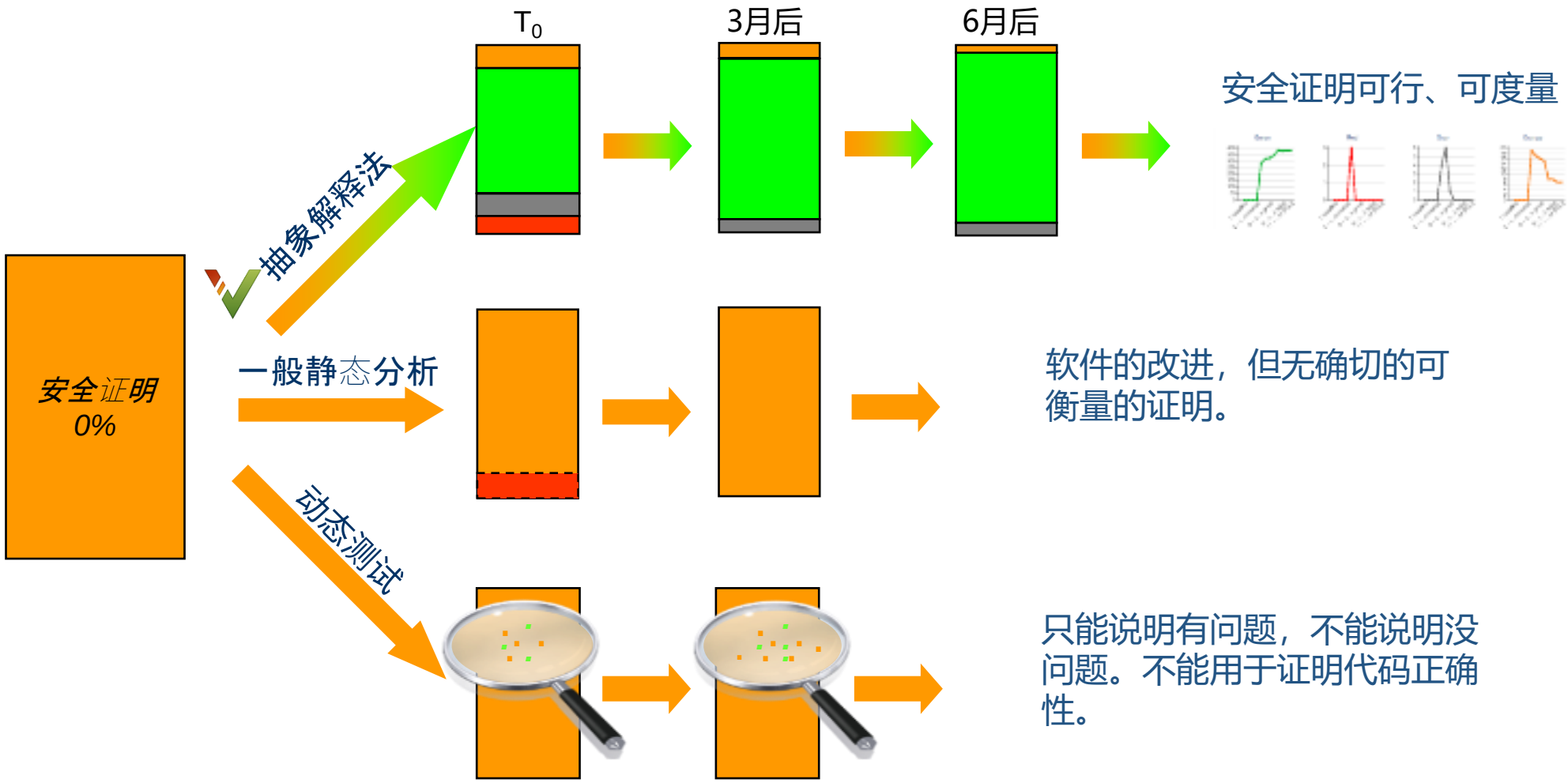
编码规范

代码度量

为什么这些是好的衡量方式？

- 市场最佳实践
- 工具结合评审
- 符合行业标准

Code Prover 有何独到之处？



应用 polyspace 到开发流程

规格/任务书

管理人员

- 软件质量趋势
- 缺陷任务分配
- 问题分布

设计

实现

软件工程师

- 查找并修复本地缺陷
- 查找并修复MISRA等规范问题

单元测试

软件工程师
质量工程师
测试工程师

- 修复或解释运行时错误
- 证明模块中无运行时错误

集成测试

架构工程师
软件工程师
质量工程师

- 查找并修复集成问题
- 函数/变量声明不匹配
- 共享变量访问冲突

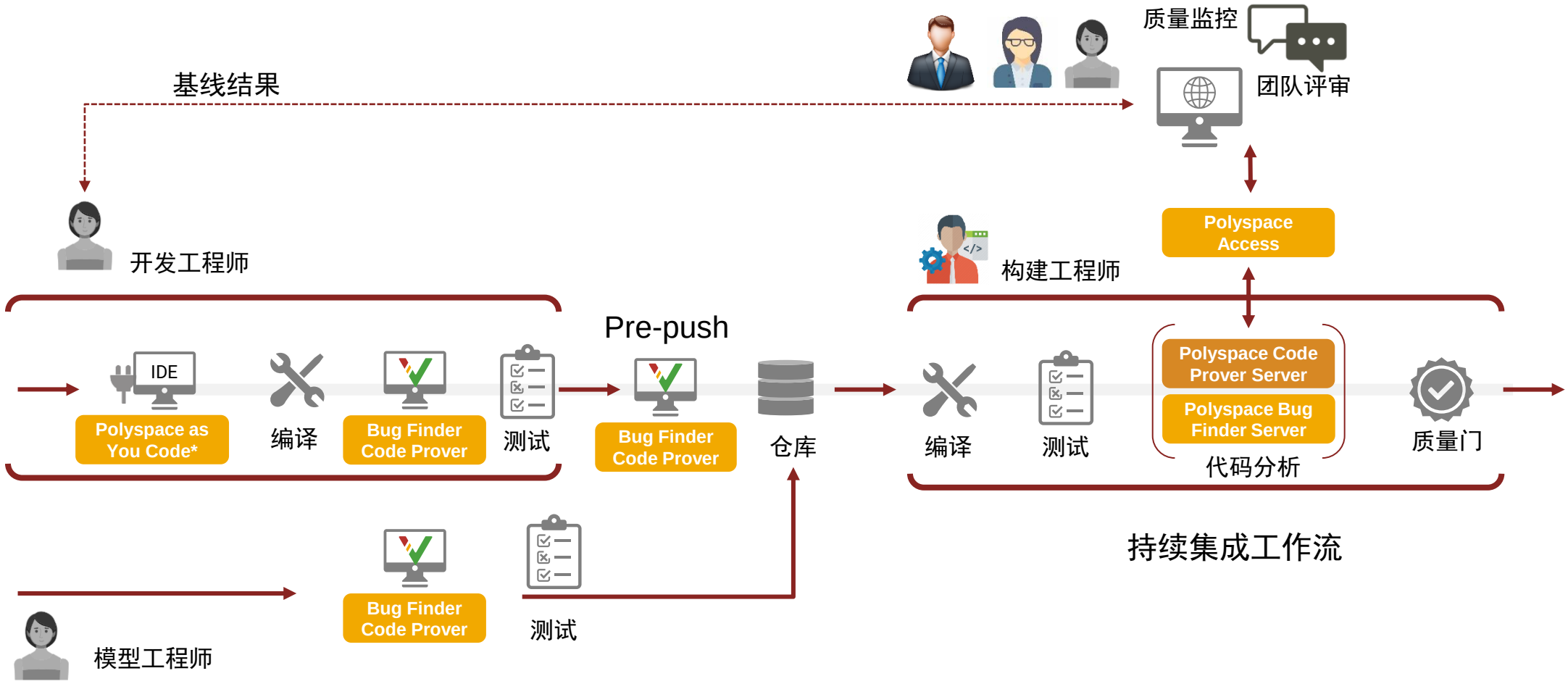
验收测试

质保工程师

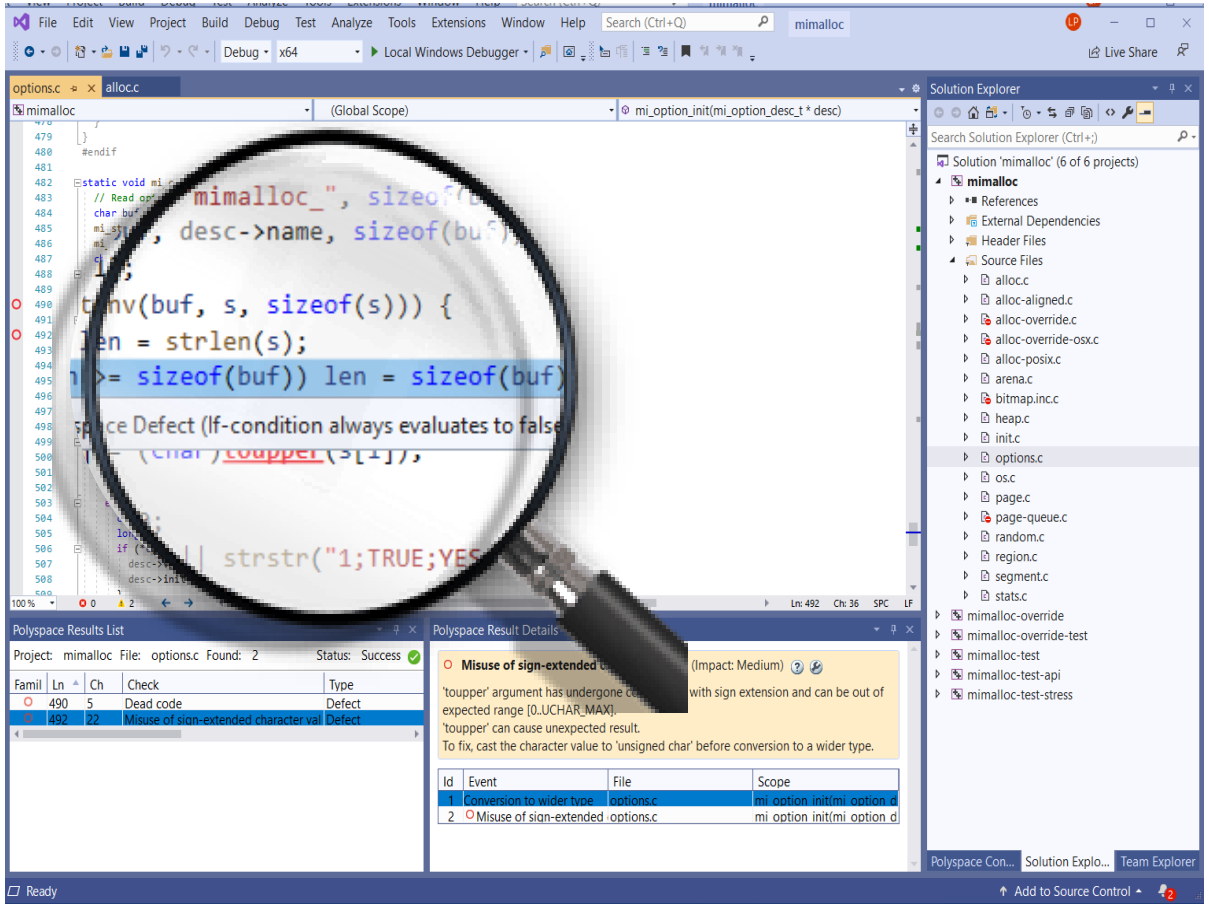
- 软件质量度量
- 验证编码规范符合性
- 质量报告生成
- 全局变量访问安全

Bug Finder
Code Prover
Access
As You Code

应用 polyspace 到开发流程

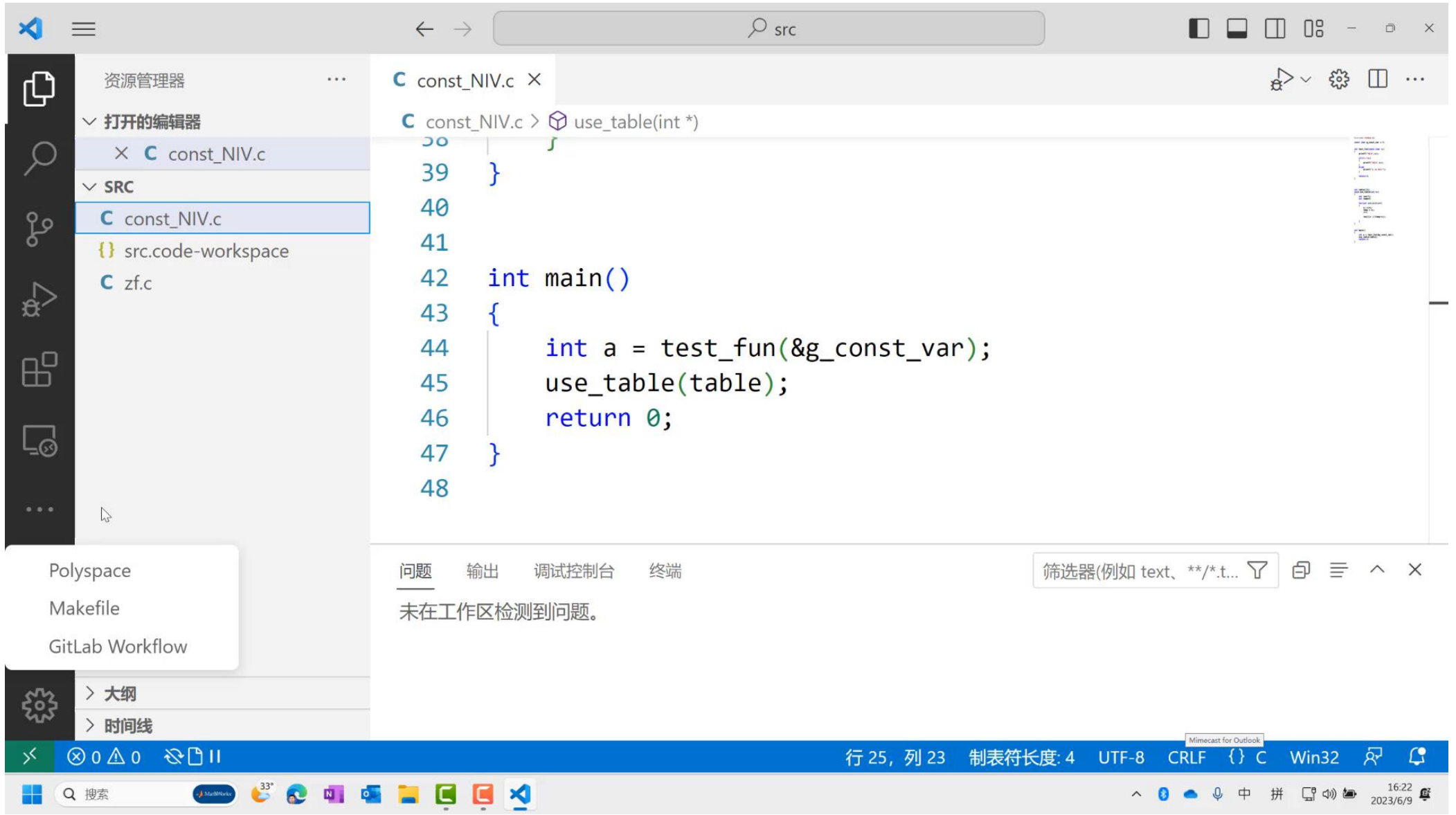


编码时即检测和修复缺陷



- 在IDE中编码的早期发现问题安全隐患
- 常见缺陷和漏洞、编码规范等
- 支持Eclipse、VS 和 VS Code
- 提供与代码编辑器和IDE的集成 API

编码时即检测和修复缺陷



使用桌面版扩大分析

声明不匹配

修饰符不匹配

类型别名冲突

死锁/重复锁等

证明无运行时错误
健壮性分析

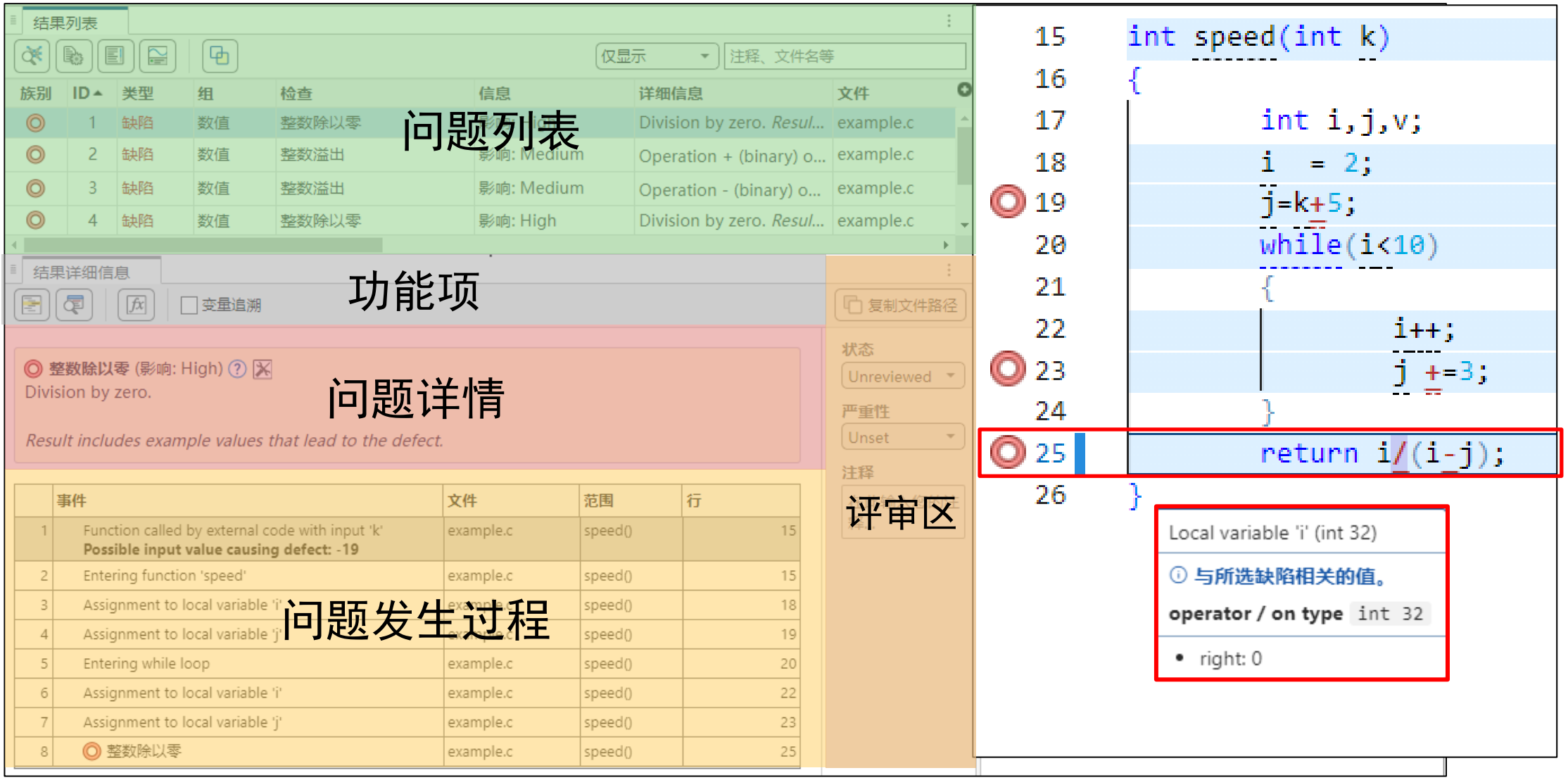
上溢/下溢
除零
数组越界
缓存访问越界
非法指针解引
并发访问保护证明

数据流分析

控制流分析

堆栈统计

快速定位根本原因


结果列表
仅显示
注释、文件名等
族别 ID 类型 组 检查 信息 详细信息 文件
1 缺陷 数值 整数除以零 Division by zero. Resul... example.c
2 缺陷 数值 整数溢出 影响: Medium Operation + (binary) o... example.c
3 缺陷 数值 整数溢出 影响: Medium Operation - (binary) o... example.c
4 缺陷 数值 整数除以零 影响: High Division by zero. Resul... example.c
问题列表
结果详细信息
变量追溯
功能项
复制文件路径
整数除以零 (影响: High)
Division by zero.
Result includes example values that lead to the defect.
问题详情
状态
Unreviewed
严重性
Unset
注释
评审区
事件 文件 范围 行
1 Function called by external code with input 'k' Possible input value causing defect: -19 example.c speed() 15
2 Entering function 'speed' example.c speed() 15
3 Assignment to local variable 'i' example.c speed() 18
4 Assignment to local variable 'j' example.c speed() 19
5 Entering while loop example.c speed() 20
6 Assignment to local variable 'i' example.c speed() 22
7 Assignment to local variable 'j' example.c speed() 23
8 整数除以零 example.c speed() 25
问题发生过程
15 int speed(int k)
16 {
17 int i,j,v;
18 i = 2;
19 j=k+5;
20 while(i<10)
21 {
22 i++;
23 j +=3;
24 }
25 return i/(i-j);
26 }
Local variable 'i' (int 32)
与所选缺陷相关的值。
operator / on type int 32
right: 0

生成验证Bug Finder检查项的测试用例

☑ 运行更严格的检查并考虑所有的系统输入值

运行更严格的检查并考虑所有的系统输入值 (-checks-using-system-input-values)
考虑以下各项的所有可能的值:
- 全局变量
- 易失变量的读取值
- 插桩函数的返回值
- 使用 "考虑这些函数的输入" 指定的函数输入

更严格的静态分析可检测极端数值情况可能导致的问题，
并提供可能导致检测到的缺陷的示例值。

对使用选项 "考虑这些函数的输入" (-system-input -from)指定的函数执行更严格的分析。

◎ **整数除以零** (影响: High) ⓘ
Division by zero.

Result includes example values that lead to the defect.

	事件	文件	范围
1	Function called by external code with input 'k' **Possible input value causing defect: -19**	example.c	speed()
2	Entering function 'speed'	example.c	speed()
3	Assignment to local variable 'i'	example.c	speed()
4	Assignment to local variable 'j'	example.c	speed()
5	Entering while loop	example.c	speed()
6	Assignment to local variable 'i'	example.c	speed()
7	Assignment to local variable 'j'	example.c	speed()
8	◎ 整数除以零	example.c	speed()

ⓘ **与所选缺陷相关的值。**

Parameter k (int 32): -19

ⓘ **与所选缺陷相关的值。**

Assignment to local variable 'j' (int 32): -14

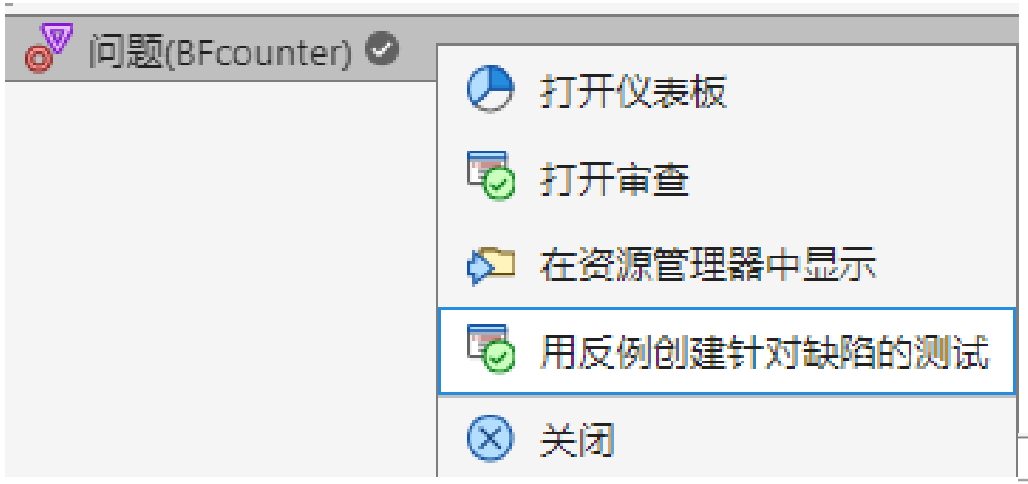
```
15  int speed(int k)
16  {
        int i,j,v
        i  = 2;
        j=k+5;
20      while(i<1
21      {
22          i
23          j
24      }
25      return i/(i-j);
26  }
```

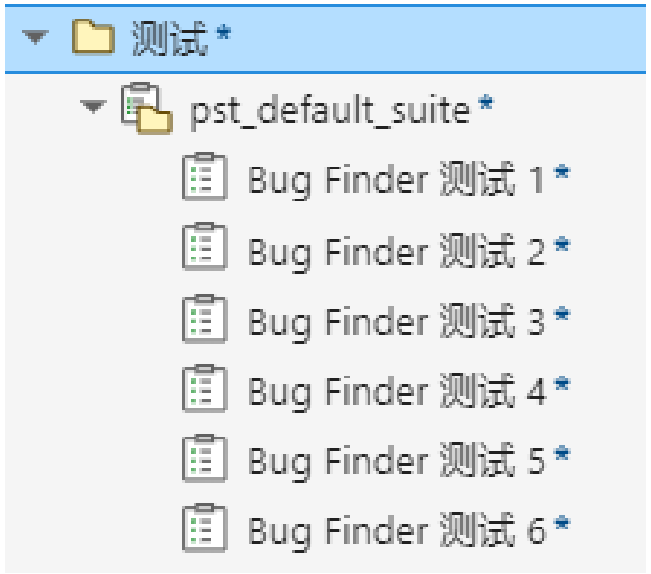
ⓘ **与所选缺陷相关的值。**

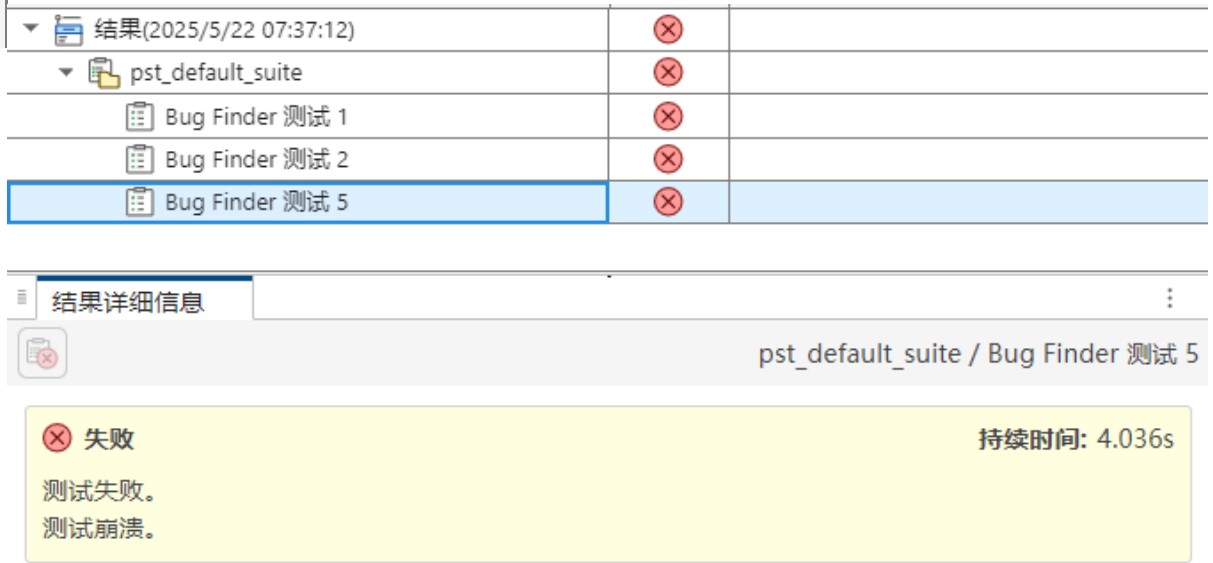
operator - on type `int 32`

- left: 10
- right: 10
- result: 0
- (result is wrapped)

生成验证Bug Finder检查项的测试用例







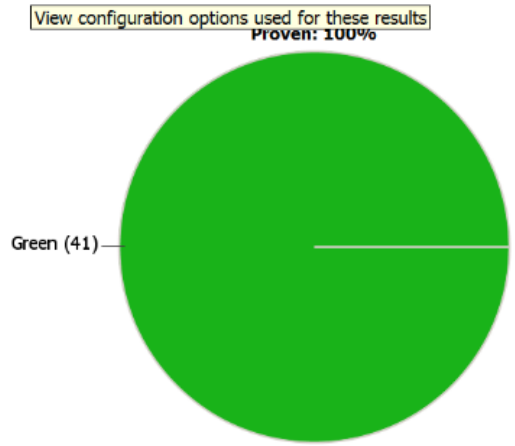
形式化交付安全的单元软件

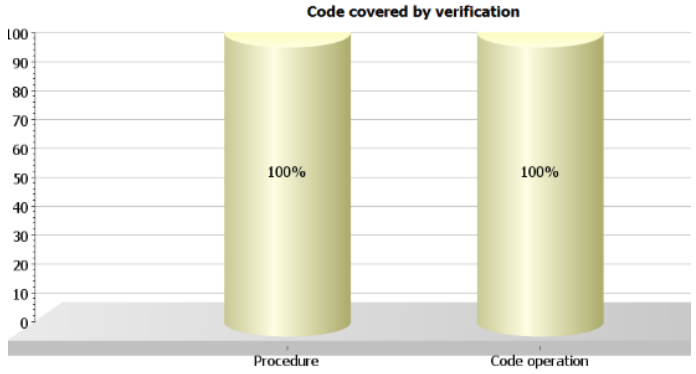
where_are_the_errors.c

```
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 }
```

operator / on type int 16
left: 10
right: [-346 .. -1]
result: [-10 .. 0]

return (int 16): [-5896 .. 2786]





Is my configuration correct?

数据流和多任务共享变量访问的证明

保护方式

共享变量的保护方式或未保护

变量范围

变量在程序中所有位置的变化

橙色：未被证明

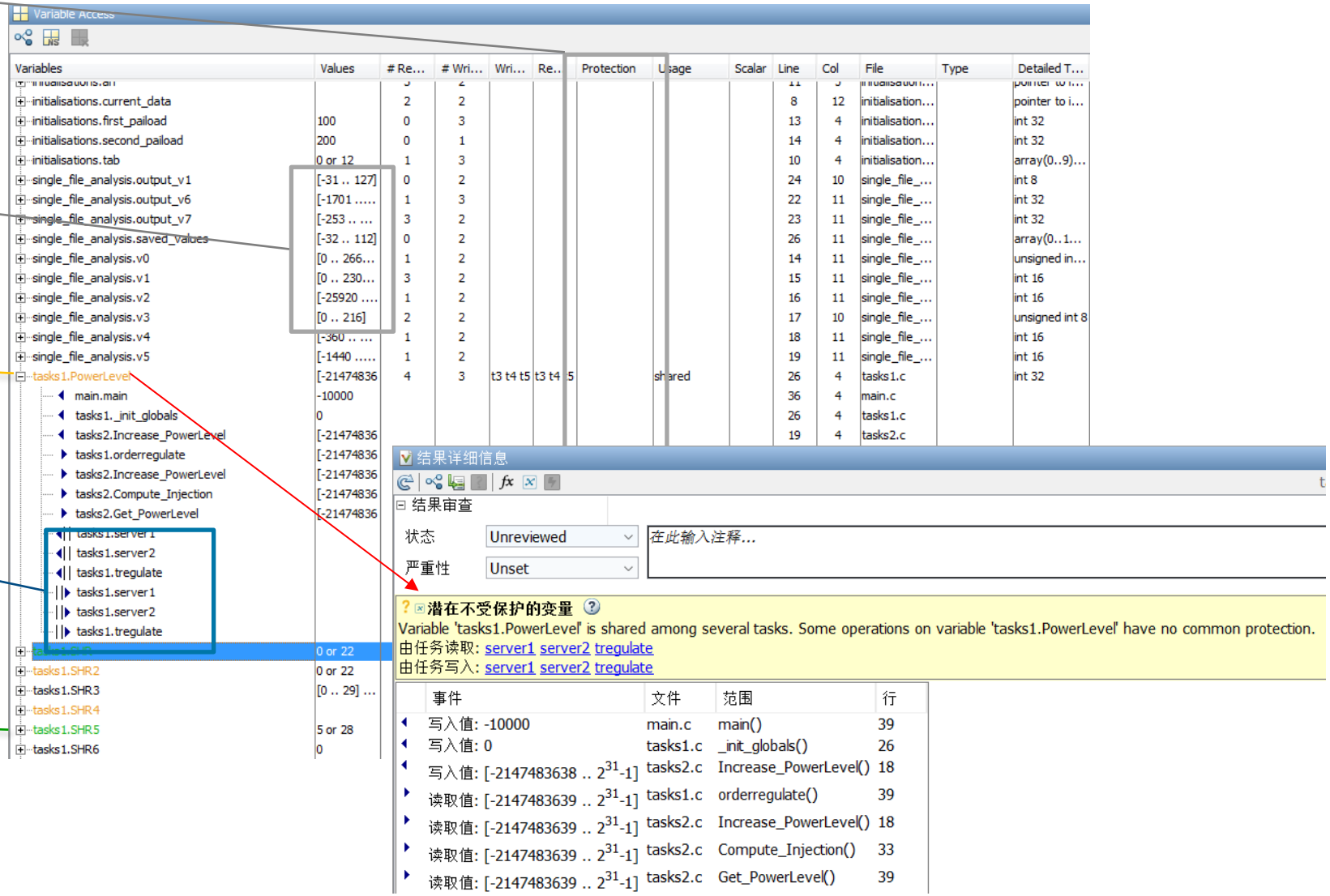
共享内存的访问可能不安全

任务和中断

对共享内存的读写操作

绿色：安全可靠

共享内存访问安全



推送前自动分析

以 GitLab 为例

分析流程化和强制化

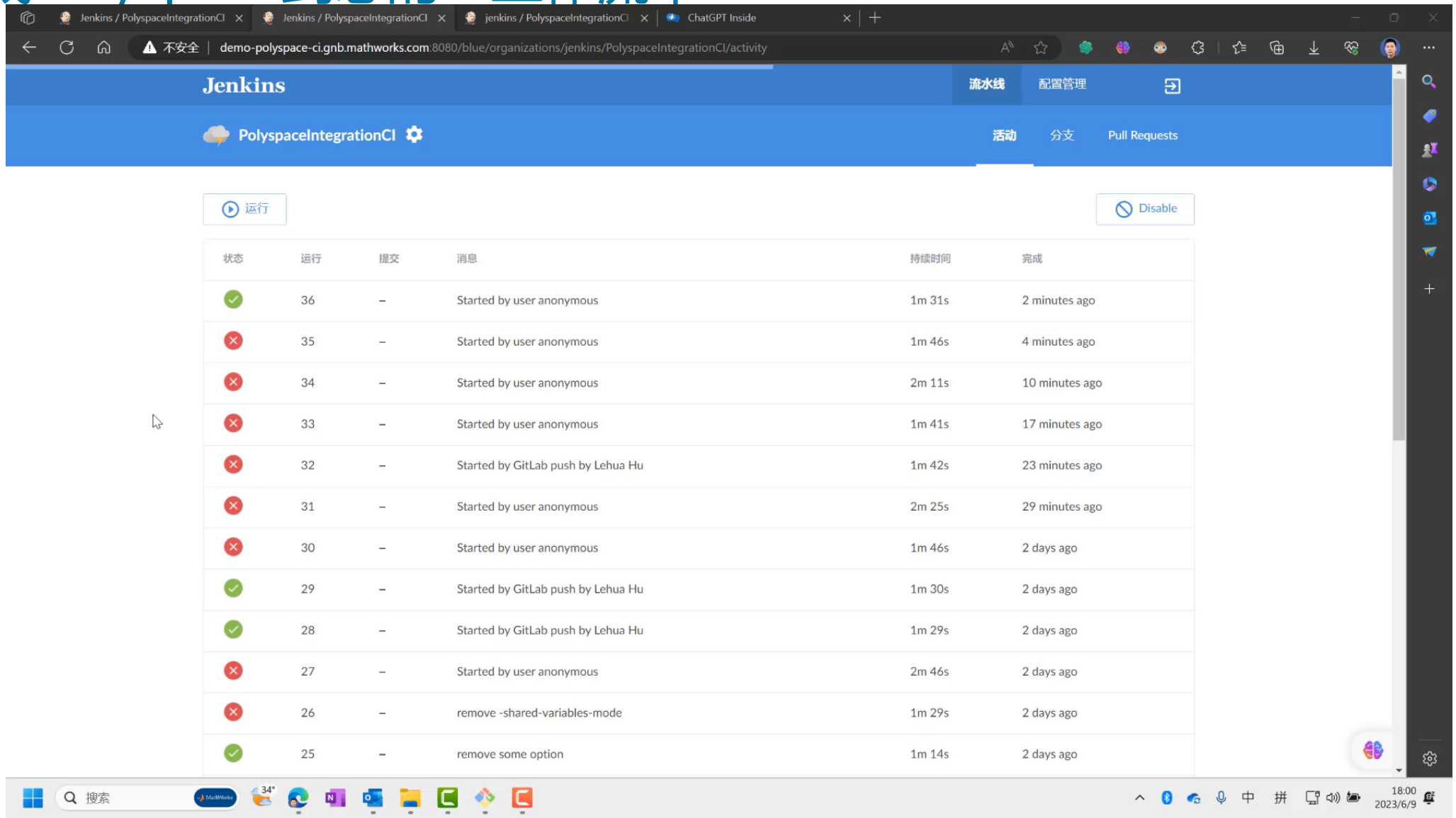
防止问题代码带入到控制库

受控库最后的把关

```
#!/bin/sh
#
# An polyspace verification example h
is about to be committed.
# Called by "git commit" with no argu
# exit with non-zero status after issuir
message if
# it wants to stop the commit.
#
# To enable this hook, rename this file
echo "Running pre-commit checks wi
export PATH=$PATH:"C:\Program
Files\Polyspace\R2023b\polyspace\bi
polyspace-bug-finder -options-file op
sources-list-file source_command.txt
polyspace-results-export -format csv
output-name result.tsv

results_num=$(cat result.tsv | tail -n
echo there are $results_num defects
```

集成Polyspace到您的CI工作流中



IEC 61508 Part 3 – Polyspace Bug Finder Supports…

Table A.2 – Software Design and Development – Software Architecture Design

	Technique / Measure	SIL 1	SIL 2	SIL 3	SIL 4
(…)					
14	Static resource allocation	-	R	HR	HR
(…)					

Table A.8 – Modification

	Technique / Measure	SIL 1	SIL 2	SIL 3	SIL 4
(…)					
3	Reverify affected software modules	R	HR	HR	HR
2	Reverify changed software module	HR	HR	HR	HR
(…)					

Table B.8 – Static Analysis

	Technique / Measure	SIL 1	SIL 2	SIL 3	SIL 4
(…)					
9	Static analysis of run time error behavior	R	R	R	HR
(…)					

Table A.3 – Software Design and Implementation – Support Tools and Programming Language

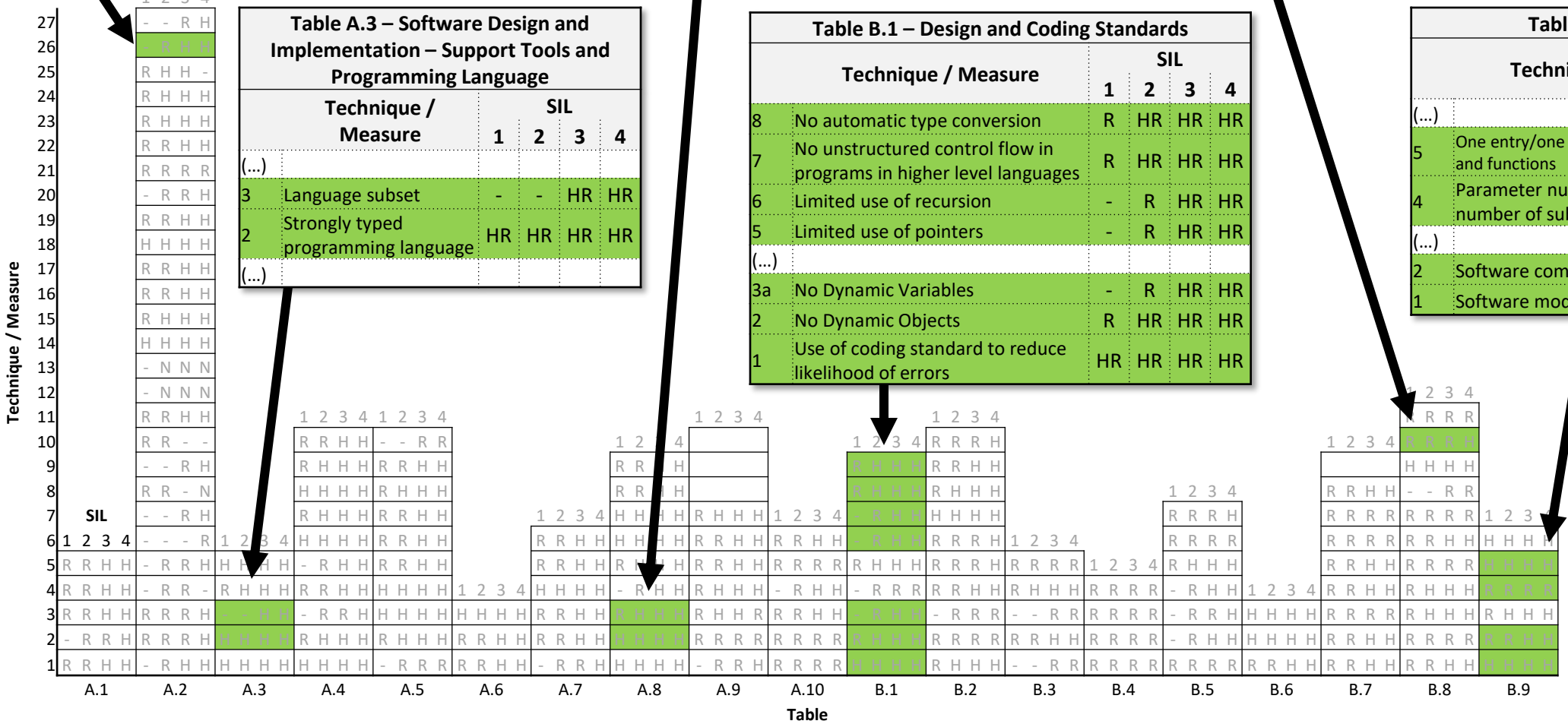
	Technique / Measure	SIL 1	SIL 2	SIL 3	SIL 4
(…)					
3	Language subset	-	-	HR	HR
2	Strongly typed programming language	HR	HR	HR	HR
(…)					

Table B.1 – Design and Coding Standards

	Technique / Measure	SIL 1	SIL 2	SIL 3	SIL 4
8	No automatic type conversion	R	HR	HR	HR
7	No unstructured control flow in programs in higher level languages	R	HR	HR	HR
6	Limited use of recursion	-	R	HR	HR
5	Limited use of pointers	-	R	HR	HR
(…)					
3a	No Dynamic Variables	-	R	HR	HR
2	No Dynamic Objects	R	HR	HR	HR
1	Use of coding standard to reduce likelihood of errors	HR	HR	HR	HR

Table B.9 – Modular Approach

	Technique / Measure	SIL 1	SIL 2	SIL 3	SIL 4
(…)					
5	One entry/one exit point in subroutines and functions	HR	HR	HR	HR
4	Parameter number limit / fixed number of subprogram parameters	R	R	R	R
(…)					
2	Software complexity control	R	R	HR	HR
1	Software module size limit	HR	HR	HR	HR



IEC 61508 Part 3 – Polyspace Code Prover Adds…

Table A.3 – Software Design and Implementation – Support Tools and Programming Language

	Technique / Measure	SIL 1	2	3	4
(…)					
2	Strongly typed programming language	HR	HR	HR	HR
(…)					

Table A.8 – Modification

	Technique / Measure	SIL 1	2	3	4
(…)					
3	Reverify affected software modules	R	HR	HR	HR
2	Reverify changed software module	HR	HR	HR	HR
(…)					

Table B.1 – Design and Coding Standards

	Technique / Measure	SIL 1	2	3	4
8	No automatic type conversion	R	HR	HR	HR
(…)					
6	Limited use of recursion	-	R	HR	HR
5	Limited use of pointers	-	R	HR	HR
(…)					

Table A.5 – Software Design and Development – Software Module Testing and Integration

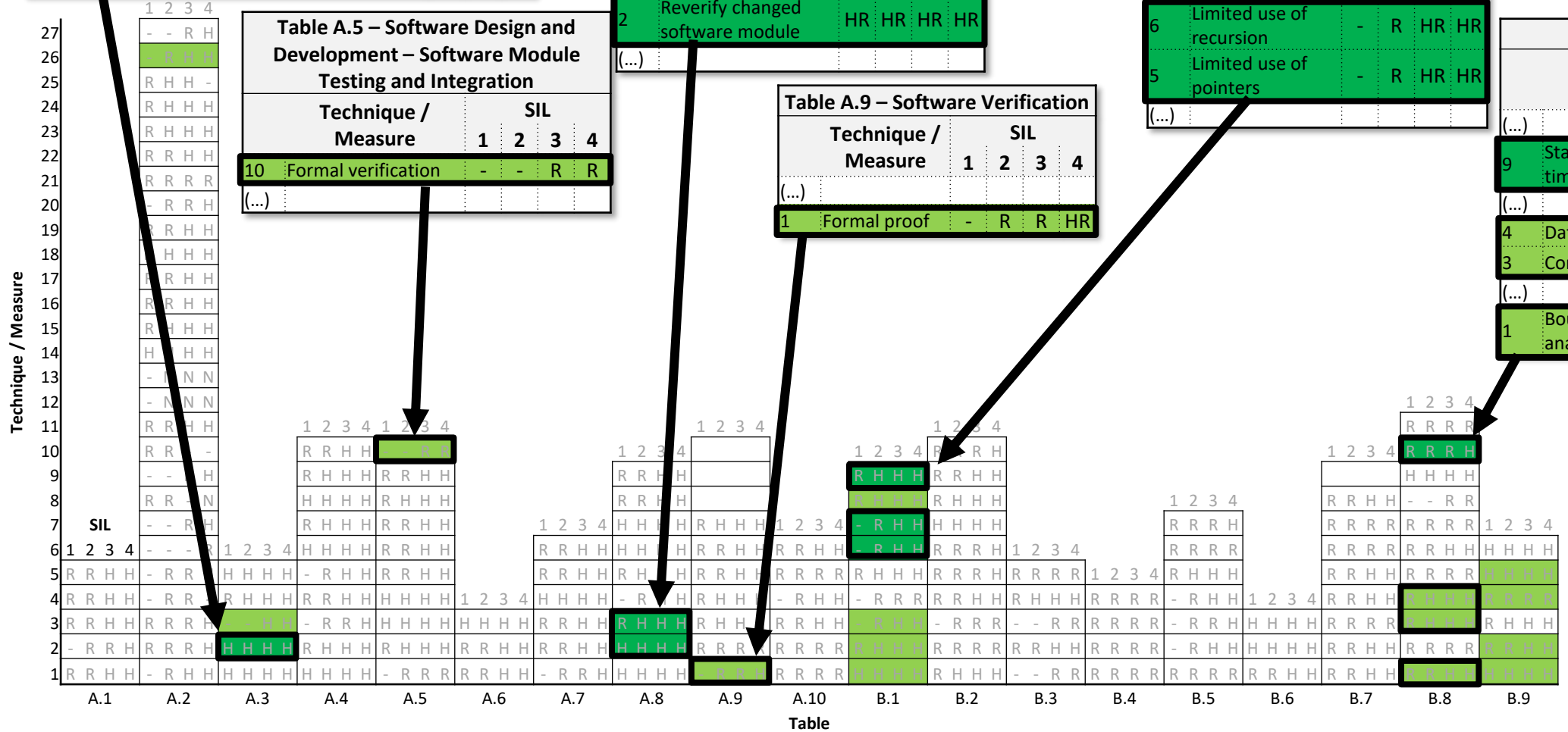
	Technique / Measure	SIL 1	2	3	4
10	Formal verification	-	-	R	R
(…)					

Table A.9 – Software Verification

	Technique / Measure	SIL 1	2	3	4
(…)					
1	Formal proof	-	R	R	HR

Table B.8 – Static Analysis

	Technique / Measure	SIL 1	2	3	4
(…)					
9	Static analysis of run time error behavior	R	R	R	HR
(…)					
4	Data flow analysis	R	HR	HR	HR
3	Control flow analysis	R	HR	HR	HR
(…)					
1	Boundary value analysis	R	R	HR	HR



产品开发阶段的需求（ISO 21434）

[RQ-10-04] If design, modelling or programming notations or languages are used for the cybersecurity specifications or their implementation, the following shall be considered when selecting such a notation or language:

f) resilience of the language against vulnerabilities due to its improper use.

EXAMPLE 4 Resilience against buffer overflows.

[RQ-10-05] Criteria (see [RQ-10-04]) for suitable design, modelling or programming languages for cybersecurity that are not addressed by the language itself shall be covered by design, modelling and coding guidelines, or by the development environment.

EXAMPLE 5 Use of MISRA C:2012 [17] or CERT C [18] for secure coding in the "C" programming language.

EXAMPLE 6 Criteria for suitable design, modelling and programming languages:

— use of language subsets;

— enforcement of strong typing; and/or

— use of defensive implementation techniques.

- Follow Security Coding Standards
- Avoid Vulnerabilities Due to Improper Use
- Dataflow/Control Flow
- Resources Analysis
- Unidentified Weakness Confirmation

[RQ-10-10] The integration and verification activities of [RQ-10-09] shall be specified considering:

NOTE 2 Methods for verification can include:

— requirements-based test;

— interface test;

— resource usage evaluation;

— verification of the control flow and data flow;

— dynamic analysis; and/or

— static analysis.

[RC-10-12] Testing should be performed in order to confirm that unidentified weaknesses and vulnerabilities remaining in the component are minimized.

NOTE 6 Unnecessary functionalities can contain a weakness.

NOTE 7 Testing methods can include:

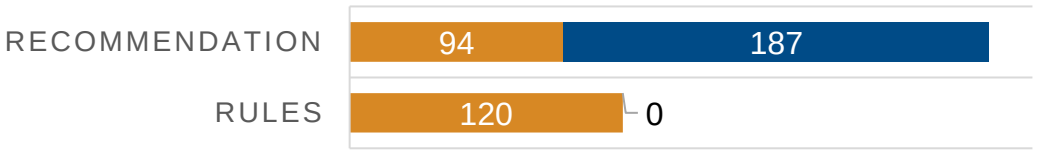
— functional testing;

— vulnerability scanning;

Polyspace 支持信息安全编码标准

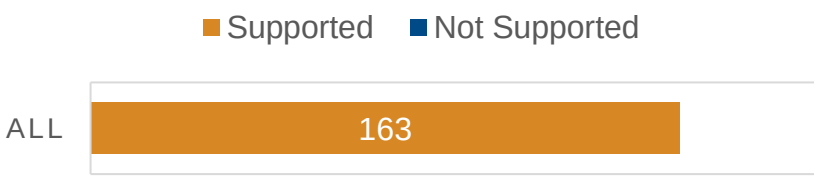
Update to 2024b

SEI CERT C 100%

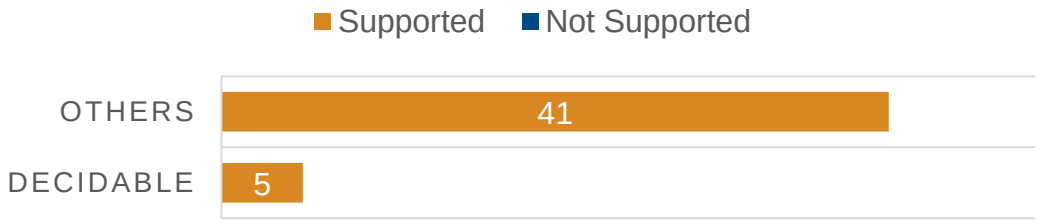


MISRA C 2012/23 AMD1 100%

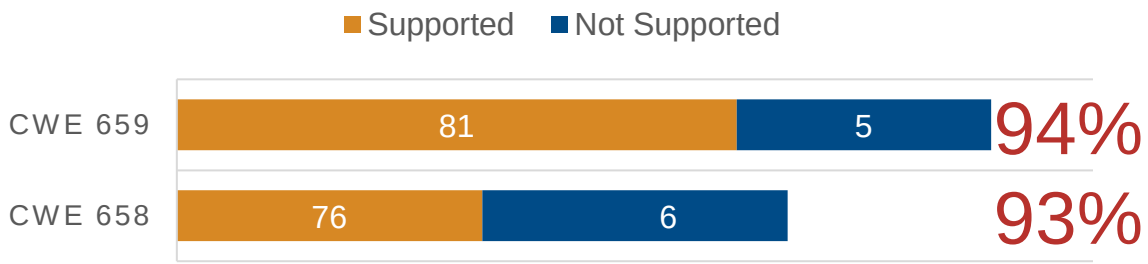
SEI CERT C++ 100%



ISO 17961 100%



CWE V4.12



CWE updates yearly and have weakness for other language.Only 658/659 are rules for C/C++.

Polyspace覆盖常见缺陷的检测

- 数值
  - 整数除以零 (影响: High)
  - 浮点数除以零 (影响: High)
  - 整数转换溢出 (影响: High)
  - 无符号整数转换溢出 (影响: Low)
  - 整数常量溢出 (影响: Medium)
  - 无符号整数常量溢出 (影响: Low)
  - 符号变化整数转换溢出 (影响: Medium)
  - 浮点数转换溢出 (影响: High)
  - 整数溢出 (影响: Medium)
  - 无符号整数溢出 (影响: Low)
  - 浮点数溢出 (影响: Low)
  - 浮点操作数被吸收 (影响: High)
  - 无效的标准库整数例程使用 (影响: High)
  - 无效的标准库浮点例程使用 (影响: High)
  - 负值移位 (影响: Low)
  - 移位运算的右操作数越界 (影响: Low)
  - 对数值使用纯字符类型 (影响: Medium)
  - 对负值执行按位运算 (影响: Medium)
  - 超出整数精度 (影响: Low)
  - 可能无效的布尔操作数运算 (影响: Low)
  - 整数到浮点数的转换存在精度损失 (影响: Low)
- 静态内存
  - 数组访问越界 (影响: High)
  - 对空指针执行解引用 (影响: High)
  - 指针访问越界 (影响: High)
  - 不可靠的函数指针转换 (影响: Medium)
  - 不可靠的指针转换 (影响: Medium)
  - 堆栈变量的指针或引用超出作用域 (影响: High)
  - 无效的标准库内存例程使用 (影响: High)
  - 无效的标准库字符串例程使用 (影响: High)
  - 对空指针执行算术运算 (影响: Low)
  - 错误分配转换对象的大小 (影响: High)
  - 在未检查最大缓冲区的情况下使用路径操作函数 (影响: High)
  - 字符串格式设定符不正确导致缓冲区溢出 (影响: High)
  - 字符串操作中目标缓冲区上溢 (影响: High)
  - 字符串操作中目标缓冲区下溢 (影响: High)
  - 使用自动变量作为 putenv 族函数参数 (影响: High)
  - 在指向不同数组的指针之间做减法或进行比较 (影响: High)
- 动态内存
  - 使用之前已释放的指针 (影响: High)
  - 无保护动态内存分配 (影响: Low)
  - 释放之前已释放的指针 (影响: High)
  - 无效的指针释放 (影响: High)
  - 内存泄漏 (影响: Medium)
  - Windows 上的分配/取消分配函数不匹配 (影响: Low)
  - 内存重新分配后对齐发生改变 (影响: Low)
- 数据流
  - 写入后未被读取 (影响: Low)
  - 未初始化的变量 (影响: High)
  - 未初始化的指针 (影响: High)
  - 变量遮蔽 (影响: Low)
  - 缺失 return 语句 (影响: Low)
  - 不可达代码 (影响: Medium)
  - 死代码 (影响: Low)
  - 无用的 if 条件 (影响: Medium)
  - 无限循环 (影响: High)
  - 未完整访问的数组 (影响: Low)
  - 静态未调用函数 (影响: Low)
  - 指向未初始化值的指针转换为常量指针
  - 因始终为 false 的条件停用的代码 (影响
  - 无用预处理器条件句指令 (影响: Low)
- 编程
  - 断言 (影响: High)
  - 无效的 = 运算符使用 (影响: Medium)
  - 无效的 == 运算符使用 (影响: High)
  - 声明不匹配 (影响: High)
  - Typedef 不匹配 (影响: High)
  - 使用相等运算符进行浮点比较 (影响: Medium)
  - 字符串数组中缺失空字符 (影响: Low)
  - 在转换中删除限定符 (影响: Low)
  - 在 sizeof 中使用错误的类型 (影响: High)
  - 运算符优先级规则导致可能的非预期表达式计算 (影响: High)
  - 用不正确的参数调用标准函数 (影响: Medium)
  - 无效的内存组织假设 (影响: Medium)
  - 不正确的指针缩放 (影响: Medium)
  - 写入 const 限定对象 (影响: High)
  - 不正确的数组初始化 (影响: Medium)
  - 使用大小参数为零的 memset (影响: Medium)
  - 无效的 va_list 参数 (影响: High)
  - 大小不为正数的可变长度数组 (影响: High)
  - 重叠赋值 (影响: Low)
  - 内存重叠的复制 (影响: Medium)
  - 可能的 sizeof 误用 (影响: High)
  - 错误的文件访问模式或状态 (影响: Medium)
  - 修改由不可重入标准函数返回的内部缓冲区 (影响: Low)
  - 使用非预期值调用 memset 族函数 (影响: Low)
  - 格式字符串设定符和参数不匹配 (影响: Low)
  - 无效的标准库例程使用 (影响: High)
  - 不符合 AUTOSAR 规范 (影响: High)
  - 指针和整数之间的不安全转换 (影响: Medium)
  - 从字符串到数值的不安全转换 (影响: Low)
  - 退出处理程序异常终止 (影响: Medium)
  - 比较浮点值内存 (影响: Low)
  - 比较字符串内存 (影响: Medium)
  - 比较填充数据内存 (影响: Medium)
  - 误用不可重入标准函数的返回值 (影响: High)
  - 环境指针因前面的运算而失效 (影响: Medium)
  - 误用 errno (影响: High)
  - errno 未重置 (影响: High)
  - 误用符号扩展字符值 (影响: Medium)
  - 字符值被吸收转化为 EOF (影响: High)
  - 从文件流中交替执行输入和输出运算间未执行刷新和定位调用 (影响: Low)
  - 从计算异常信号处理程序返回结果 (影响: Low)
  - 从信号处理程序内部调用 signal (影响: Medium)
  - 从信号处理程序调用的函数不是异步安全的 (影响: Medium)
  - 从信号处理程序调用的函数不是异步安全的(严格 ISO C) (影响: Medium)
  - 通过非原型函数指针进行调用 (影响: Medium)
  - 误用 FILE 对象 (影响: Low)
  - 误用具有灵活数组成员的结构体 (影响: Low)
  - 在信号处理程序内访问共享数据 (影响: Medium)

以及信息安全类缺陷

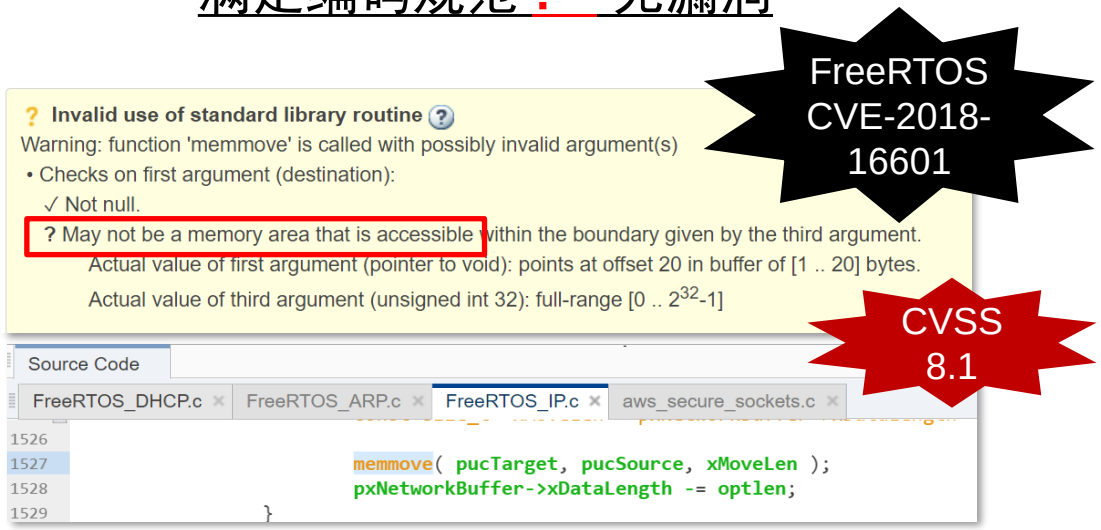
- 安全
  - 易受攻击的路径操作 (影响: Low)
  - 使用危险标准函数 (影响: Low)
  - chroot() 之后未调用 chdir("/") 即进行文件操作 (影响: Medium)
  - Umask 与 chmod-style 参数一起使用 (影响: Low)
  - 易受攻击的权限分配 (影响: Medium)
  - 从常量种子得出的确定性随机输出 (影响: Medium)
  - 从可预测种子得出的可预测随机输出 (影响: Medium)
  - 易受攻击的伪随机数生成器 (影响: Medium)
  - 分配有绝对地址的函数指针 (影响: Low)
  - 在检查时间和使用时间之间(TOCTOU)进行文件访问 (影响: Medium)
  - 从相对路径加载库可以被外部执行者所控制 (影响: Medium)
  - 从相对路径执行二进制文件可以被外部执行者所控制 (影响: Medium)
  - 使用不安全的临时文件 (影响: High)
  - 堆栈中存在未清除的敏感数据 (影响: Medium)
  - 释放前未清除敏感堆内存 (影响: Medium)
  - 敏感数据被打印输出 (影响: Medium)
  - 使用过时的标准函数 (影响: Low)
  - 网络连接操作顺序不正确 (影响: Medium)
  - 数据长度和大小不匹配 (影响: Medium)
  - 不安全的标准函数 (影响: Medium)
  - 不安全的标准加密函数 (影响: Medium)
  - 切换条件缺少 case (影响: Low)
  - 丢弃特权的顺序错误 (影响: High)
  - 特权丢弃未经验证 (影响: High)
  - 未检查敏感函数的返回值 (影响: High)
  - 误用 readlink() (影响: Medium)
  - 未检查 errno (影响: Medium)
  - 向子进程暴露文件描述符 (影响: Medium)
  - 不安全的系统函数调用 (影响: High)
  - 结构体填充可能导致信息泄漏 (影响: Low)

- 加密
  - 加密运算不一致 (影响: Medium)
  - 缺失加密算法 (影响: Medium)
  - 缺失加密密钥 (影响: Medium)
  - 缺失块加密初始化向量 (影响: Medium)
  - 缺失要处理的加密数据 (影响: Medium)
  - 缺失加密最终步骤 (影响: Medium)
  - 弱加密算法 (影响: Medium)
  - 弱加密模式 (影响: Medium)
  - 常量加密密钥 (影响: Medium)
  - 常量块加密初始化向量 (影响: Medium)
  - 可预测的加密密钥 (影响: Medium)
  - 可预测的块加密初始化向量 (影响: Medium)
  - 缺失公钥 (影响: Medium)
  - 缺失私钥 (影响: Medium)
  - 缺失对等密钥 (影响: Medium)
  - 缺失密钥生成参数 (影响: Medium)
  - 缺失用于加密、解密或签名运算的数据 (影响: Medium)
  - 不安全的密钥生成参数 (影响: Medium)
  - 不安全的 RSA 公钥指数 (影响: Medium)
  - 对 RSA 算法使用弱填充 (影响: Medium)
  - 对 RSA 算法运算使用不兼容的填充 (影响: Medium)
  - RSA 算法缺失盲化 (影响: Medium)
  - 加密算法的密钥不正确 (影响: Medium)
  - 加密运算的上下文初始化不正确 (影响: Medium)
  - RSA 算法缺失填充 (影响: Medium)
  - 不安全的哈希算法 (影响: Medium)
  - 未正确初始化摘要运算的上下文 (影响: Medium)
  - 哈希运算缺失加密盐 (影响: Medium)
  - 不安全的 SSL/TLS 协议 (影响: Medium)
  - 缺失 X.509 证书 (影响: Medium)
  - 缺失证书认证中心列表 (影响: Medium)
  - 缺失哈希算法 (影响: Medium)
  - 未向上下文添加数据 (影响: Medium)
  - 哈希更新运算后缺失最终步骤 (影响: Medium)
  - 未设置 TLS/SSL 连接方法 (影响: Medium)
  - TLS/SSL 连接方法设置不正确 (影响: Medium)
  - 缺失 X.509 证书的私钥 (影响: Medium)
  - 未检查 X.509 对等证书 (影响: Medium)
  - 未检查服务器证书的通用名 (影响: Medium)

- 被污染的数据
  - 使用外部控制元素进行主机更改 (影响: Medium)
  - 使用外部控制的环境变量 (影响: Medium)
  - 字符串格式被污染 (影响: Low)
  - 符号变化转换被污染 (影响: Medium)
  - 循环界限值被污染 (影响: Medium)
  - 内存分配大小被污染 (影响: Medium)
  - 从外部控制路径加载的库 (影响: Medium)
  - 从外部控制路径执行的命令 (影响: Medium)
  - 执行外部控制命令 (影响: Medium)
  - 可变长度数组大小被污染 (影响: Medium)
  - 模操作数被污染 (影响: Low)
  - 除法操作数被污染 (影响: Low)
  - 使用被污染的索引进行数组访问 (影响: Medium)
  - 使用被污染的偏移量进行指针解引用 (影响: Low)
  - 使用被污染的指针 (影响: Low)
  - 被污染的 NULL 或不以 null 结尾的字符串 (影响: Low)

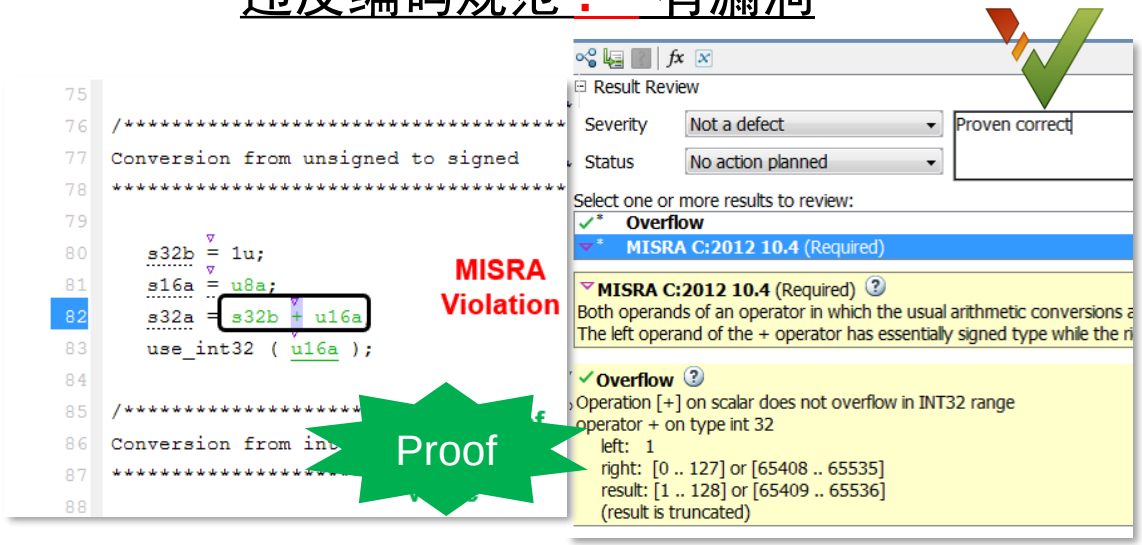
编码安全规范、漏洞检查十分必要，但不够

满足编码规范 != 无漏洞



Inconsistent arguments to `memmove` → DoS!
Not checked by CERT/MISRA/…

违反编码规范 != 有漏洞



Valid mixing of different data types → No vulnerability
Okay to violate CERT/MISRA/…

Certainty only
through formal proof
(„sound formal verification“)

Confidentiality
Integrity
Availability

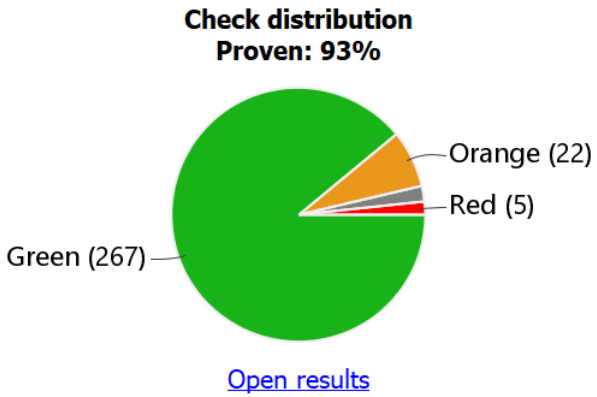
高级SAST软件常见应用示例

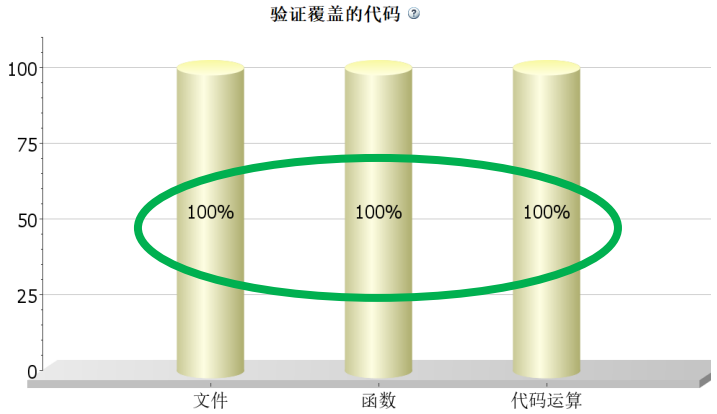
鲁棒性分析

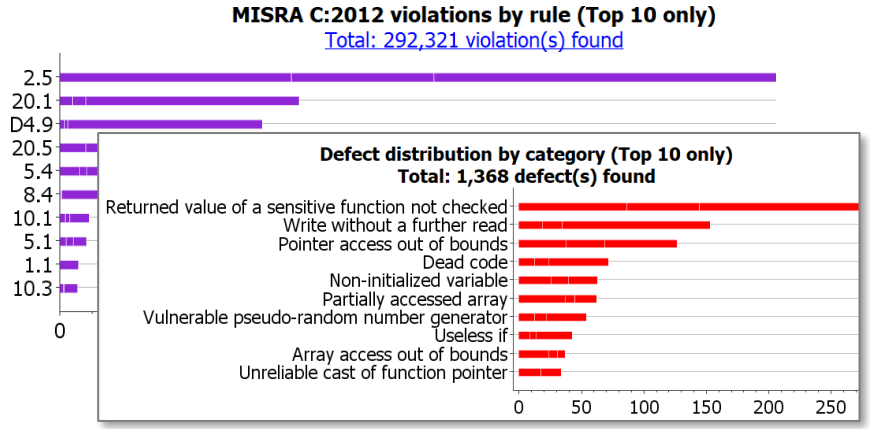
- Works with *all* inputs? *Always*?
- Works on target processor?
  - Floating point errors? Soft float?
  - Interrupts & race conditions?
  - Stack size? Memory leaks?

编码规范和最佳实践

- CERT C(++) – Secure coding standard
- CWE – Common Weaknesses
- AUTOSAR-C++ & MISRA guidelines
- Taint Analysis, Persistence Analysis
- Common Vulnerability , defects







考虑所有输入和所有程序状态

我们的一部分客户

Used since 1999

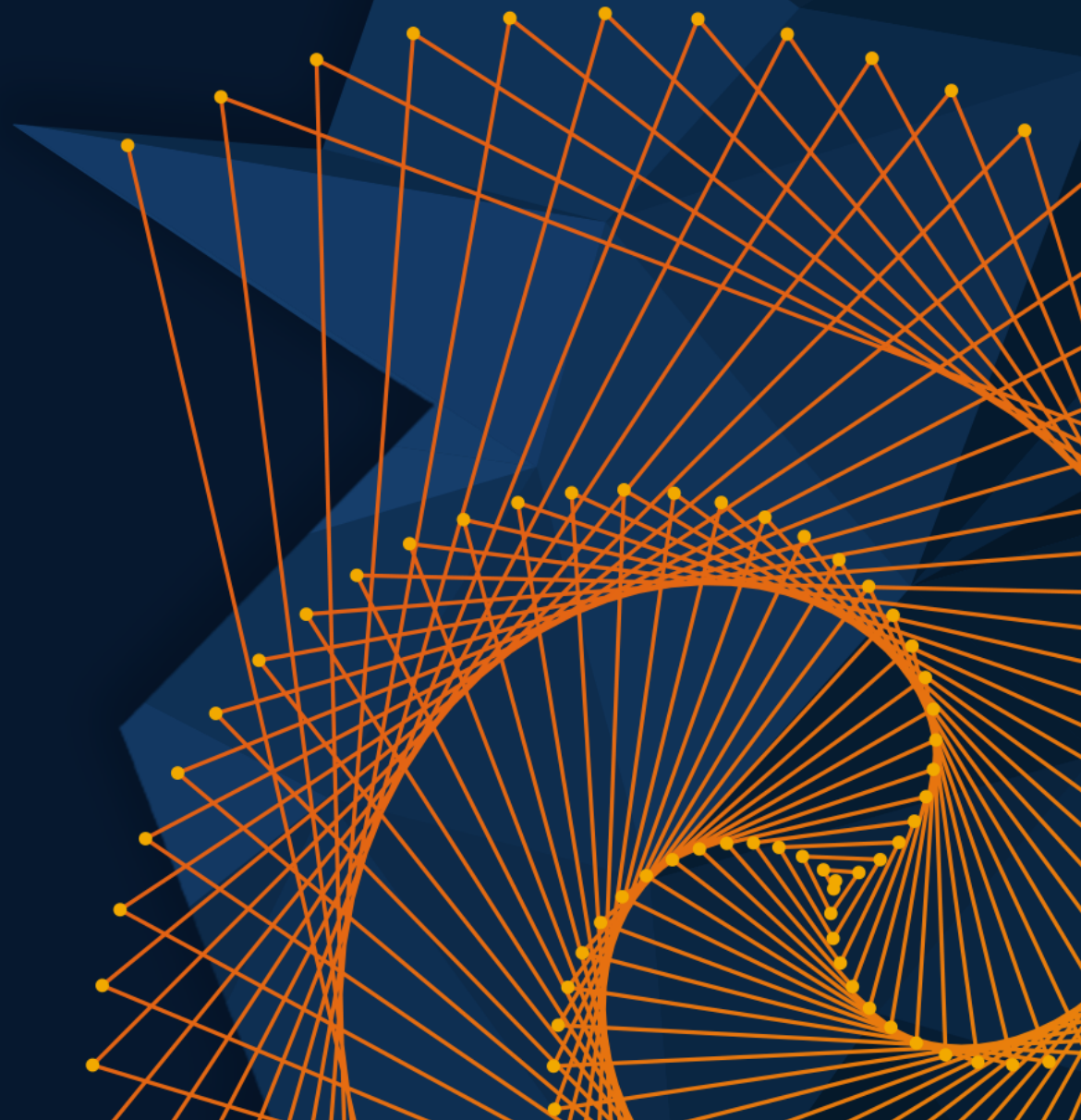
- Aerospace and defense
- Automotive
- Industrial automation and machinery
- Railway transportation
- Consumer electronics
- Medical devices

总结

- Polyspace具备完整的静态分析能力
- 形式化方法搞定深层次的运行时错误
- 集成到开发、评审和持续集成中，高效低成本的提高软件质量
- Polyspace支持任何等级的功能安全和信息安全认证

MATLAB EXPO
中国

感谢聆听！



MathWorks®