IBM Software Group

保证嵌入式系统的代码质量

—— MISRA-C编程规范与C语言自动化代码复审

Rational. software

IBM 软件部 张剑平

ON DEMAND BUSINESS

内容提要

- 代码复审的重要性
- **C**代码复审的依据：**MISRA-C**
- 自动化的**C**代码复审

缺陷的代价

- 安全关键系统 **(Safety Critical Systems)**
  - 过程、汽车和航空控制
  - 关键任务型航空电子设备
- 缺陷成本高昂
  - Pentium处理器的缺陷
  - 火星大气轨道探测器的坠毁
  - Ariane 5的爆炸

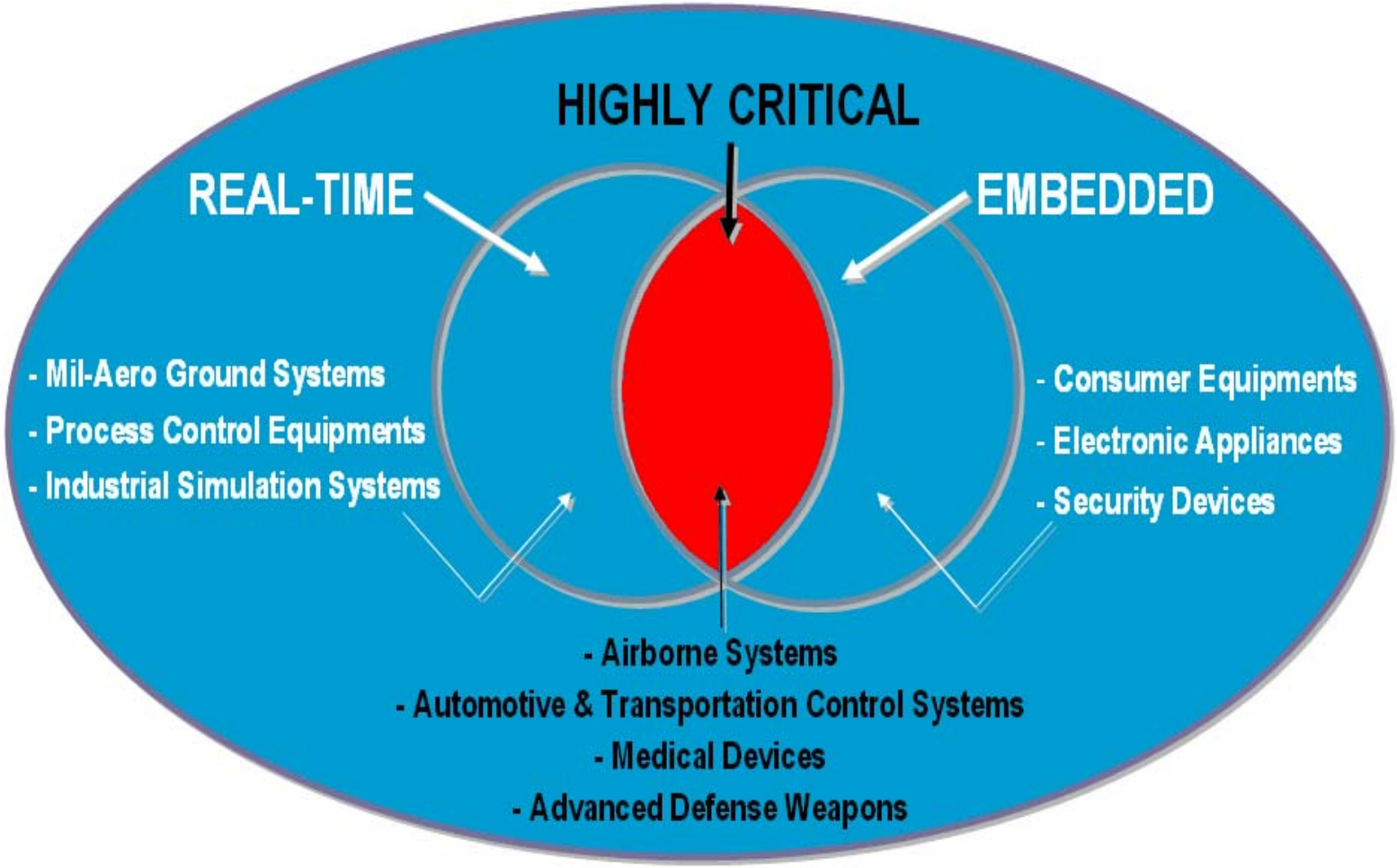
Shooting Down of Airbus 320

US Vincennes shot down Airbus 320
Mistook airbus 320 for a F-14
290 people dead
Software bug - cryptic and misleading output displayed by the controlling software

450万美元
1.6亿美元
10年，10亿欧元

关键任务型软件在哪里运行呢？

- 绝大多数都运行在嵌入式和实时系统中



嵌入式软件编程的常见问题

- 没有任何一种编程语言可以确保它的最终执行过程会和程序员的预期完全相同：
  - 程序员的失误：比如敲错了字母或理解错了算法；很多初学者会把逻辑比较“==”写成赋值“=”，这种错误简单的编译器是无法察觉的。
  - 程序员对编程语言的错误理解：C语言的编译预处理过程其实非常复杂，但是很多程序员并没有意识到这一点。
  - 编译器没有按照程序员的意图工作：不同的编译器的工作方式是不同的，这一点常常被经验不足的程序员忽视。
  - 编译器的错误：编译器是一种软件工具，同样也会有Bug；另外C语言的复杂性使得有些编译器的编写者对其理解错误。
  - 操作平台的差异：嵌入式操作系统有上千种，这些平台之间的差异肯定是存在的。
  - 运行时错误：程序运行中出现的错误更是数不胜数，一些诸如溢出、指针等错误只有在运行过程中才能被发现。

如何消除代码中的缺陷呢？

- 静态测试
  - 代码审查 Code Inspection
  - 代码走查 Code Walk-through
  - 办公桌检查 Desk Checking
- 动态测试
  - 黑盒测试
  - 白盒测试
  - 穷举和选择测试

静态测试

- 定义
  - 不需要程序处于运行状态的测试。
- 目的
  - 检查程序静态结构，找出编译不能发现的错误和人的主观认识上的偏差。
- 范围
  - 需求定义、设计文档、源代码（着重分析）
- 特点
  - 研究表明，对于某些类型的错误，静态测试更有效 。
  - 经验表明，组织良好的代码复审可以发现程序中30%到70%的编码和逻辑设计错误。
  - 不存在错误定位问题。

代码复审的益处

- 主要收益：降低修复缺陷的成本，提高生产率
  - 根据Glen Russell (Nortel Technologies)在1991年的调查，大约65%到90%的运行时缺陷可以通过代码审查发现，而成本只需测试的1/4到2/3 。
- 其他益处：
  - 更好的控制开发流程
  - 更高的质量
  - 降低缺陷密度
  - 降低查找和定位缺陷的成本

- Walkthroughs found between 30 and 70 percent of errors in a program. (McConnell, 73)
- Code reading found twice as many defects per hour of effort as testing. (McConnell, 74)
- Inspections found 60-90% of defects in a program. (McConnell, 75)
- Inspections produced a net schedule savings of 10-30%. (McConnell, 75)
- An hour of inspection avoids 33 hours of maintenance; inspections are 20 times more efficient than testing. (McConnell, 75)
- The introduction of inspections in software maintenance reduced production crashes by 77%. (Humphrey, 186)
- Inspections increased productivity 14%-25%. (Humphrey, 186-187)
- Inspections assist programmers to recognize and fix their own errors early in the process. (Humphrey, 171) Finding problems which could not be caught in testing. (Humphrey, 186)
- A study of 2019 user-found problems that resulted in code changes, found 57.7% of the problems could have been detected by design inspections, 62.7% with code inspections. (Humphrey, 187)
- When programmers know their work will be critically examined, inspections motivate better work -- that is, developers take more pride in quality and avoid sloppy mistakes. (Humphrey, 171)
- Reviews provide visibility into the state of the project. It provides an opportunity for discussion, intermediate milestones, and an assessment of the adequacy of some aspect of the project. (Christensen and Thayer, 291)

内容提要

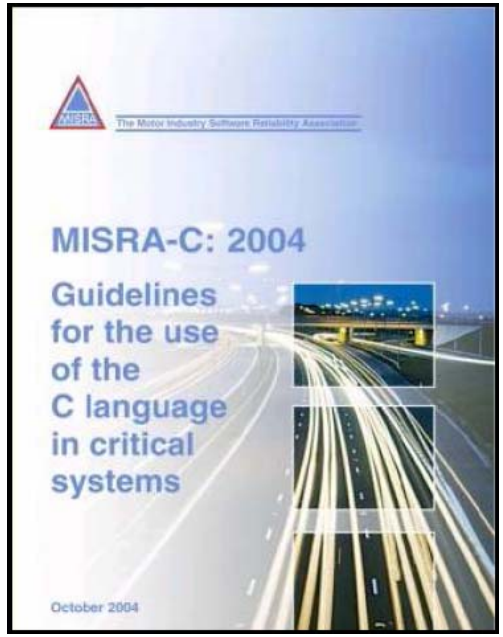
- 代码复审的重要性
- **C代码复审的依据：MISRA-C**
- 自动化的C代码复审

MISRA简介

- MISRA (The Motor Industry Software Reliability Association)——汽车工业软件可靠性协会
  - 位于英国的跨国汽车工业协会
  - 成员包括了大部分欧美汽车生产商
  - 核心使命——帮助厂方开发安全的、高可靠性的嵌入式软件
  - 最出名的成果——MISRA-C

MISRA-C简介

- 包含的一系列规则定义了C语言的一个安全性子集
- MISRA-C 1998：Guidelines for the use of the C language in vehicle based software
  - 69页，17类，127条规则(93条强制遵守+ 34条建议遵守)
- MISRA-C 2004：Guidelines for the use of C language in critical systems
  - 109页，21类，141条规则(121条强制遵守，20条建议遵守)
  - 细化复杂的规则——“One rule, one issue”
  - 明确模糊的规则
  - 引入一些数学运算相关的规则
  - 在整体框架上和98版保持兼容，废除了98版的15条规则
  - 考虑了非英语环境
- 如能完全遵守这些标准，则C代码是：
  - 易读的、可靠的、可移植的、易于维护的



MISRA-C 2004 的21个规则类

1. Environment (4:1)
2. Language Extensions (3:1)
3. Documentation (5:1)
4. Character Sets (2:0)
5. Identifiers (4:3)
6. Types (4:1)
7. Constants (1:0)
8. Declarations and Definitions (12:0)
9. Initialisation (3:0)
10. Arithmetic Type Conversion (6:0)
11. Pointer Type Conversion (3:2)
12. Expressions (9:4)
13. Control Statement Expressions(6:1)
14. Control Flow (10:0)
15. Switch Statements (5:0)
16. Functions (9:1)
17. Pointers and Arrays (5:1)
18. Structures and Unions (4:0)
19. Preprocessing Directives (13:4)
20. Standard Libraries (12:0)
21. Run-Time Failures (1:0)

(#Required : #Advisory)

<标识符>类规则示例

例：

```
int16_t   i;
void func()
{
   int16_t I;
   i=3;
}
```

Rule6.5 (强制): 在内部范围的标识符不能和外部的标识符用同样的名字，因为会隐藏那个标识符

<表达式>类规则

例1：

```
int i=1;
X=b[i] + i++;
```

Rule12.2(强制)：表达式的值应和标准允许的评估顺序一致

例2：

```
int32_t=i;
int32_t=j;
j=sizeof(i=1234);
```

Rule12.3(强制):sizeof操作符不能用在包含边界作用的表达式上

<控制流>类规则示例1

例：

```
Switch (event)
{
   Case www;
            do_wakeup();
            break;
            do_more();
   …
}
```

Rule 14.1(强制):不要有执行不到的代码

<控制流>类规则示例2

```
If (test){
   x = 1;
 }
 else
   x = 3;
   y = 2;
```

Rule 14.9.1(强制): If语句块必须使用{ }括起

<Switch语句>

```
Switch (x) {
        :
        :
   defalt:
        err=1;
        break;
}
```

Rule 15.3(强制)：switch的最后应是default

<预处理指令>类规则

- Rule19.1(建议)：#include语句的前面只能有其他预处理指令和注释
- Rule19.2(建议)：#include指令中的头文件名称不能包含非标准的字符
- Rule19.5(强制)：宏不能在函数体内定义
- Rule19.8(强制)：类函数宏调用时不能没有它的参数

<数学类型转换(隐式)>类规则

- Rule 10.1(强制):整型表达式不要隐式转换为其他类型：
  - a）转换到更大的整型
  - b）表达式太复杂
  - c）表达式不是常数是一个函数
  - d）表达式不是一个常数是一个返回表达式

- Rule 10.2(强制)：浮点数表达式不要隐式转换为其他类型：
  - a）转换到更大的浮点数
  - b）表达式太复杂
  - c）表达式是一个函数
  - d）表达式是一个返回表达式

<运行时故障>类规则

- Rule 21.1(强制)：通过使用以下手段确保把运行时故障最小化：
  - 静态分析工具/技术
  - 动态分析工具/技术
  - 编写明确的代码避免运行时错误

内容提要

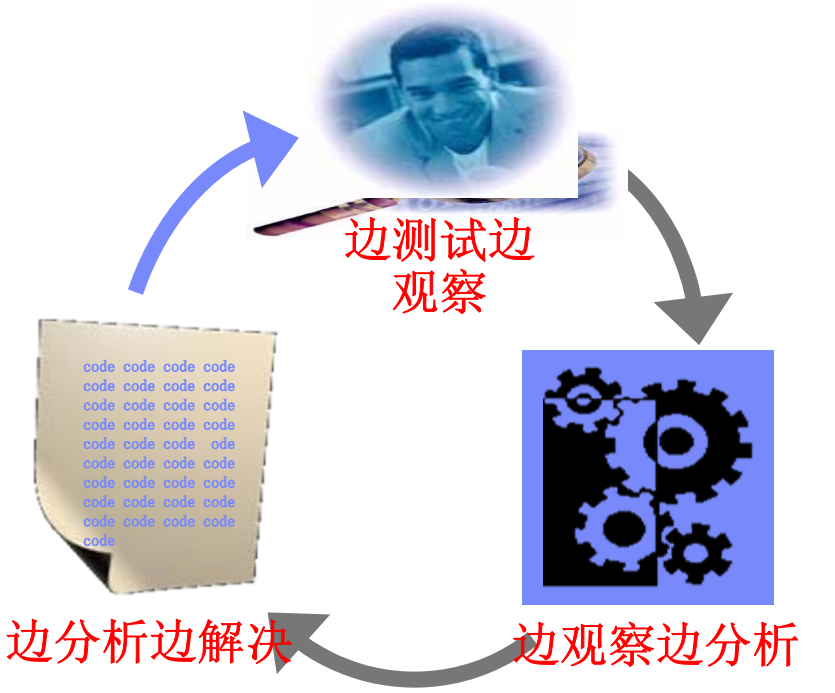
- 代码复审的重要性
- C代码复审的依据：MISRA-C
- 自动化的C代码复审

自动化的代码复审

- 自动化的过程执行快速，可以经常进行代码复审，从而提高代码质量，降低程序出错的风险
- 代码复审发现的错误可以让你更早的发现和解决问题，越早发现越容易变更，且变更的成本也越低廉
- 将日常的、机械化的检查自动化，可以让开发人员关注于更复杂的和更有创造性的设计决策上
- 自动的代码复审仍然需要与人工代码检查相结合

多才多艺的Test RealTime

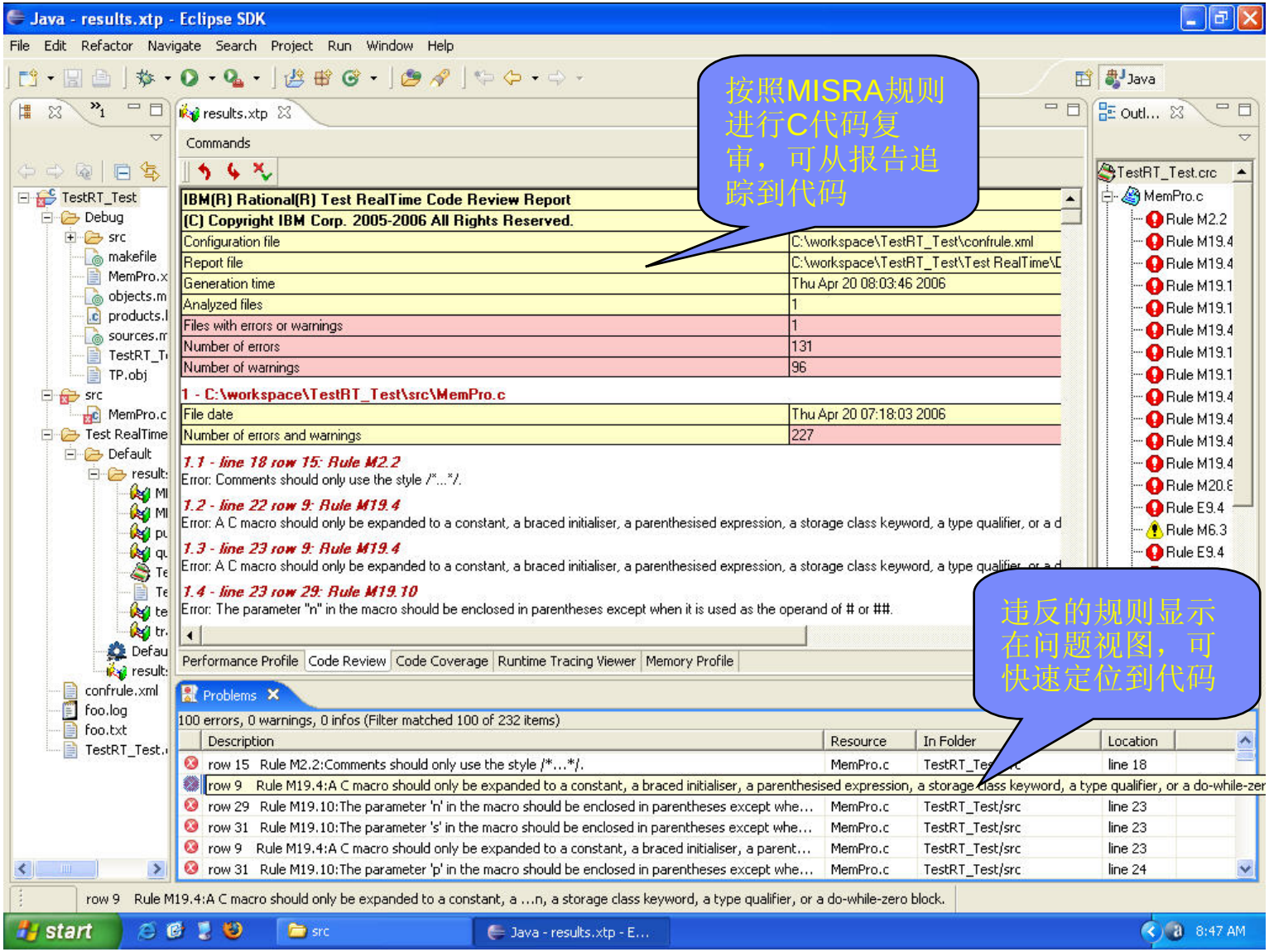
- 对所有组件进行白盒和黑盒测试
  - 对C进行代码复审(MISRA-C 2004)
  - 对C/C++、Java进行组件测试
  - 对C线程、任务、进程进行基于消息的单元测试和集成测试
  - 与Eclipse 3.1 和 CDT 3.0集成
- 全面的运行时分析功能
  - 内存分析
  - 性能分析
  - 代码覆盖率分析
  - 基于UML的运行时跟踪和线程分析



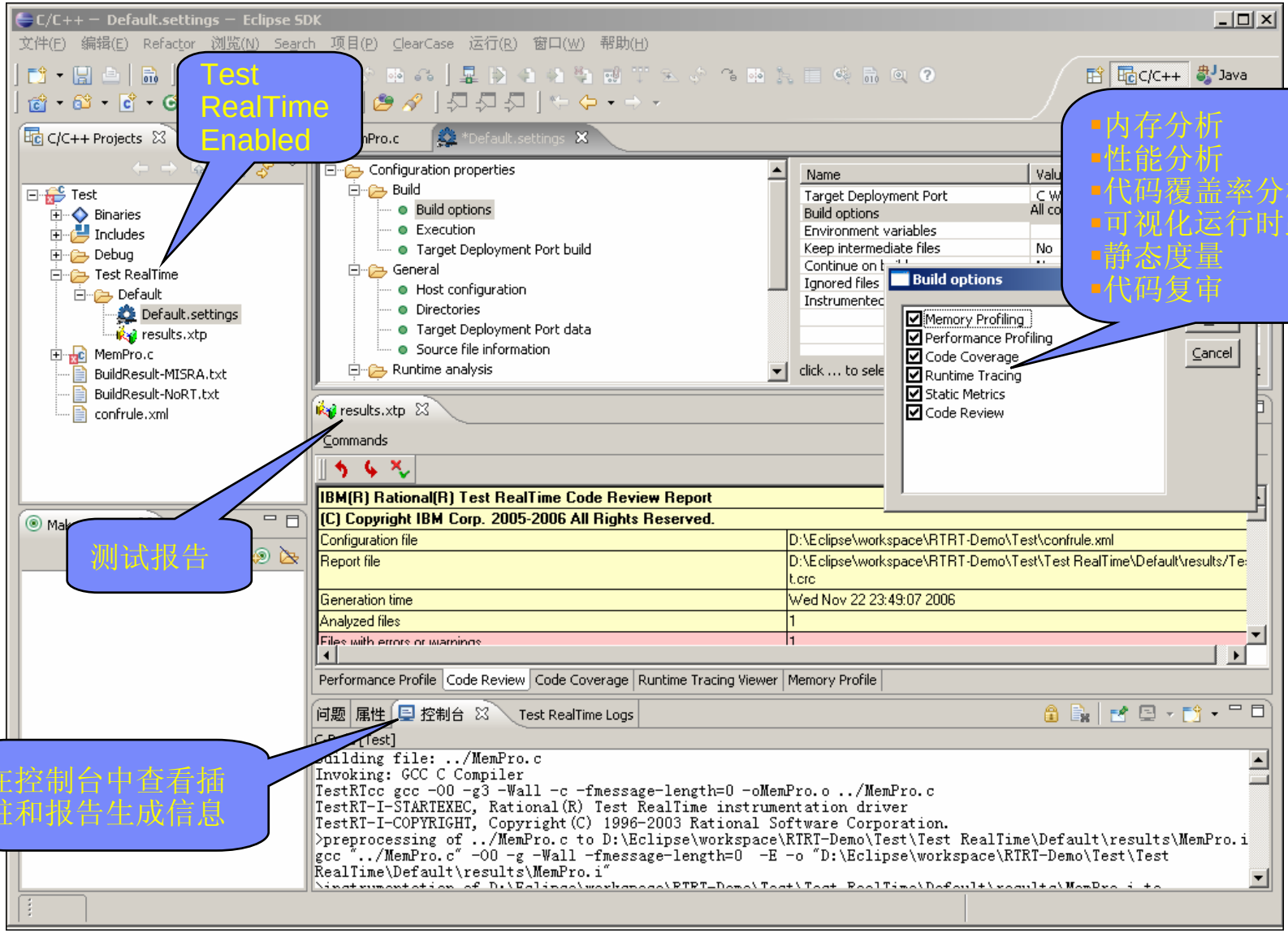
Test RealTime自动化代码复审的作用

- 将代码审查的工作自动化，探测和报告不符合MISRA规则的代码，使开发和质量保证工作更专业、更有效率。
- 自动检查与公司或工业编程标准的一致性，同时强制执行C语言的安全子集。
- 提高C代码质量，提高可靠性、可移植性、可维护性，减少将来的维护费用。
- 教育开发人员如何避免C语言开发中的各种问题。
- 允许用户定制或增加符合本公司标准或习惯的检查项目

符合MISRA-C 2004标准的C语音代码复审



与Eclipse 3.1和 CDT 3.0的集成




DEMO

Questions

Thank You