IBM Software Group

IBM Rational Test RealTime

组件测试（单元测试）—C 和Ada

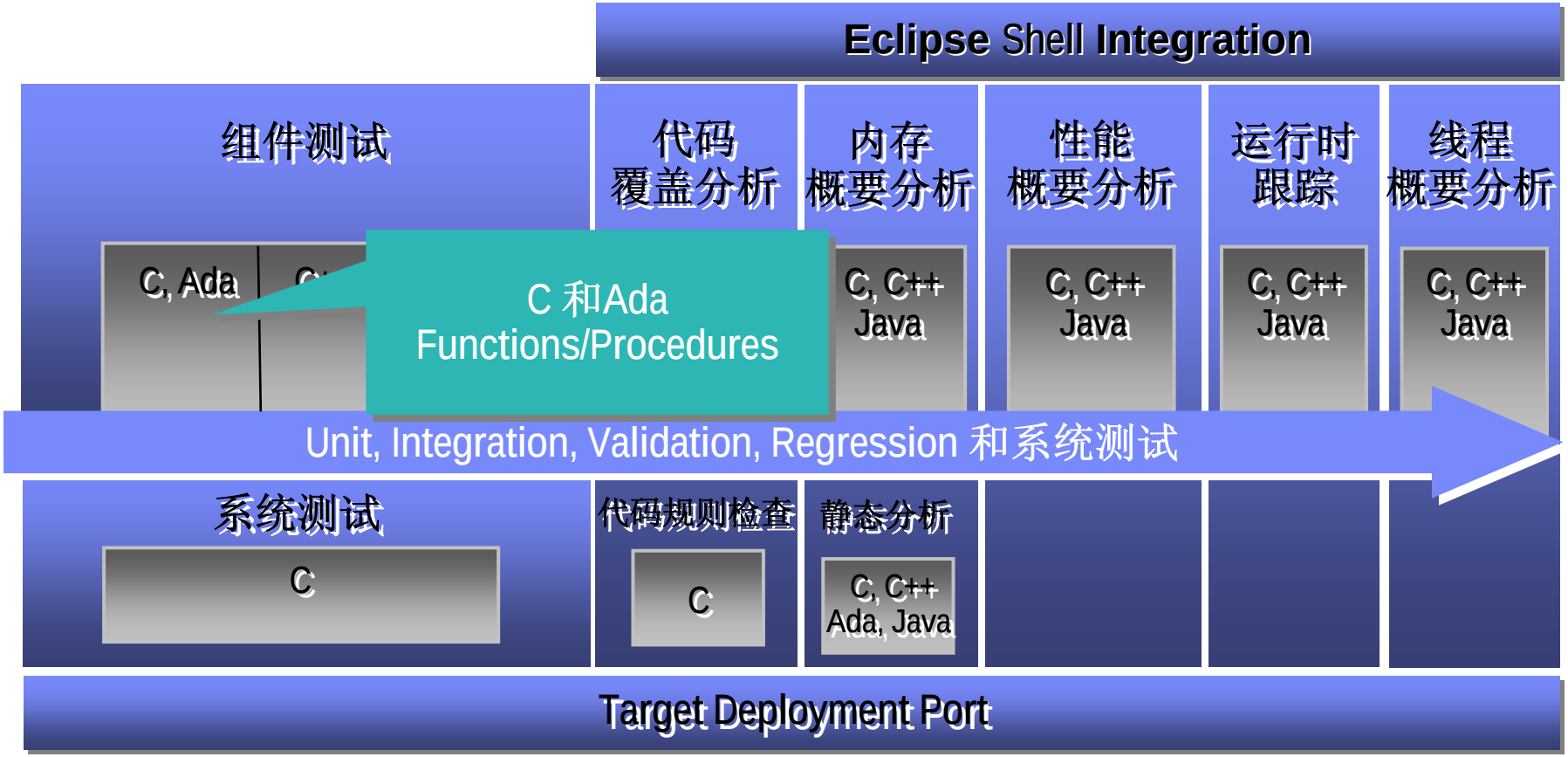
2007年制造业研讨会

IBM 软件部　靳超

chaojin@cn.ibm.com

e business on demand.

Test RealTime: *单元测试－C 和Ada*

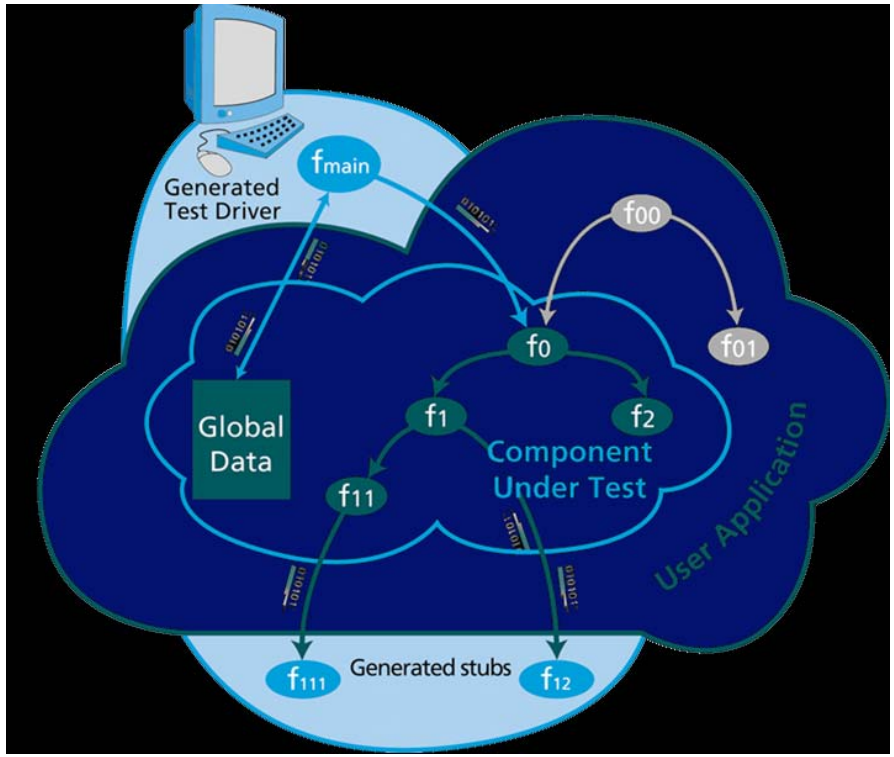


Test RealTime: *单元测试*

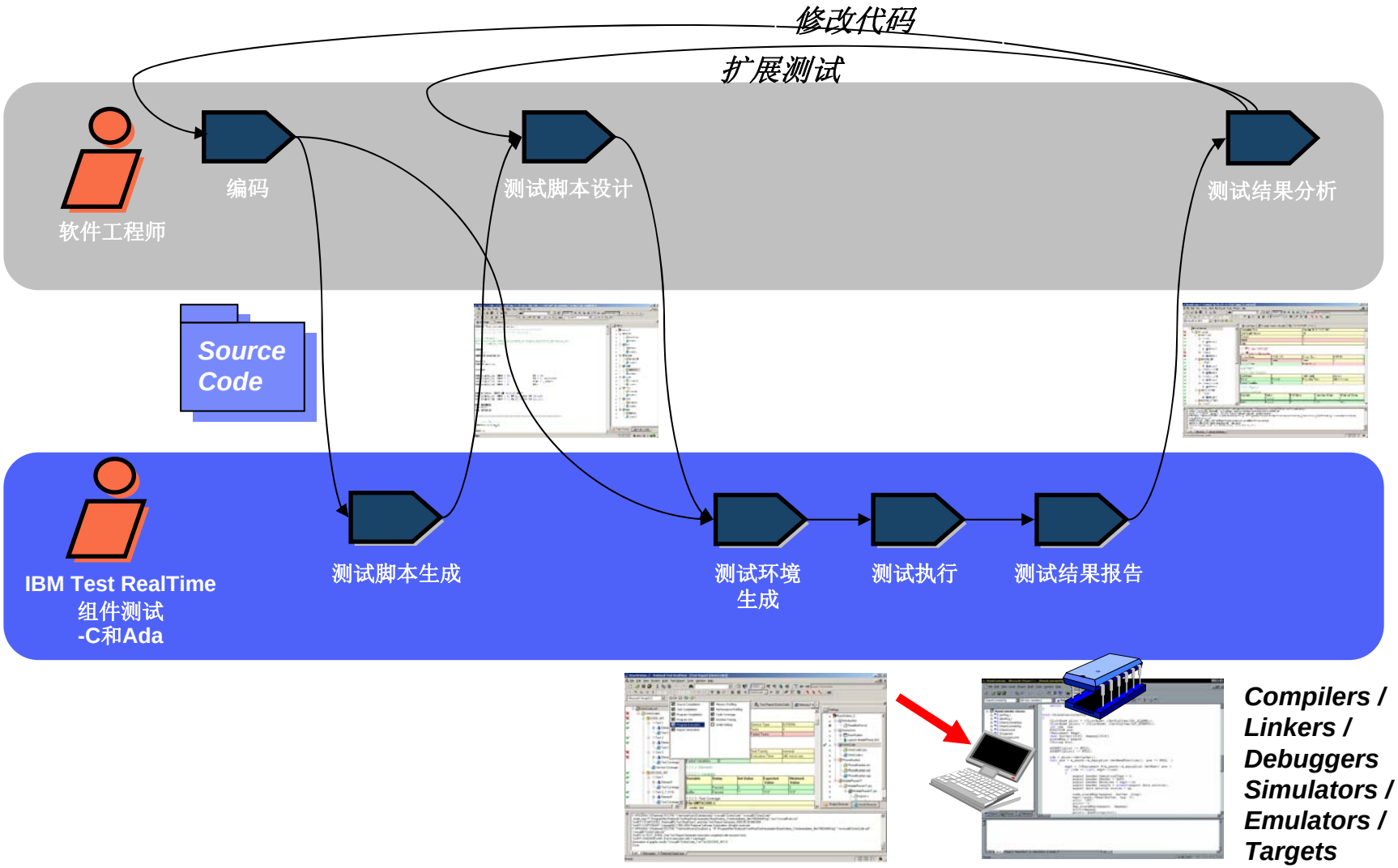
- 通过函数调用，实现基于源码的、数据驱动的功能和结构测试
- 自动化
  - 生成桩模块和驱动模块
  - 生成测试模板
  - 生成测试用例组
  - 运行测试
  - 代码查桩，用于运行分析
  - 生成测试报告
  - 回归测试
- 详尽的报告
- 用静态度量确定测试的优先级
  - 软件复杂度水平
- 和Test RealTime运行时分析集成
  - 内存和性能分析
  - 覆盖率和运行时跟踪

测试环境构建和功能

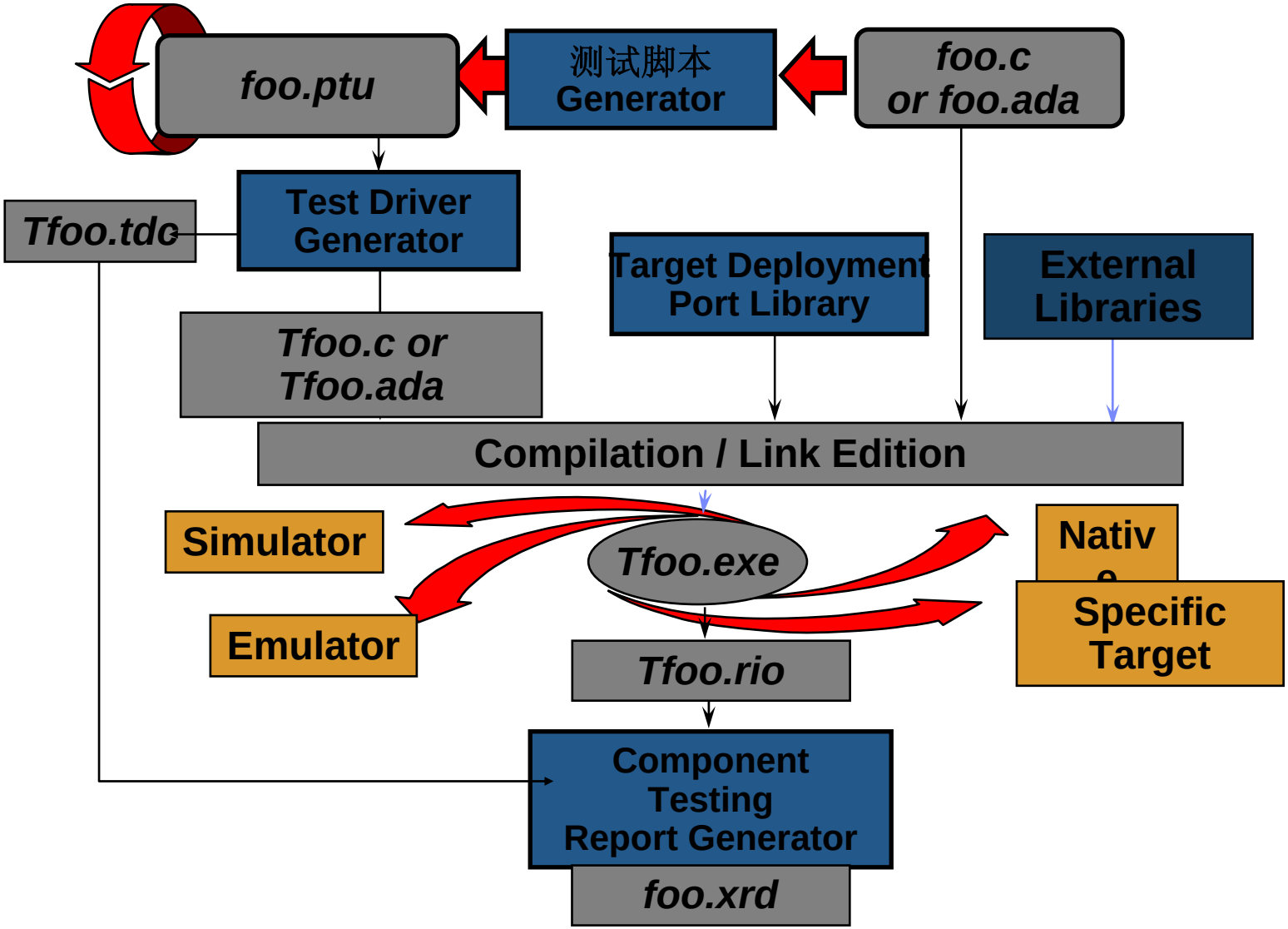
1. 驱动模块
   - 用特定的参数调用待测函数
   - 检查返回值
   - 访问全局变量
2. 桩模块
   - 检查输入值
   - 返回特定参数



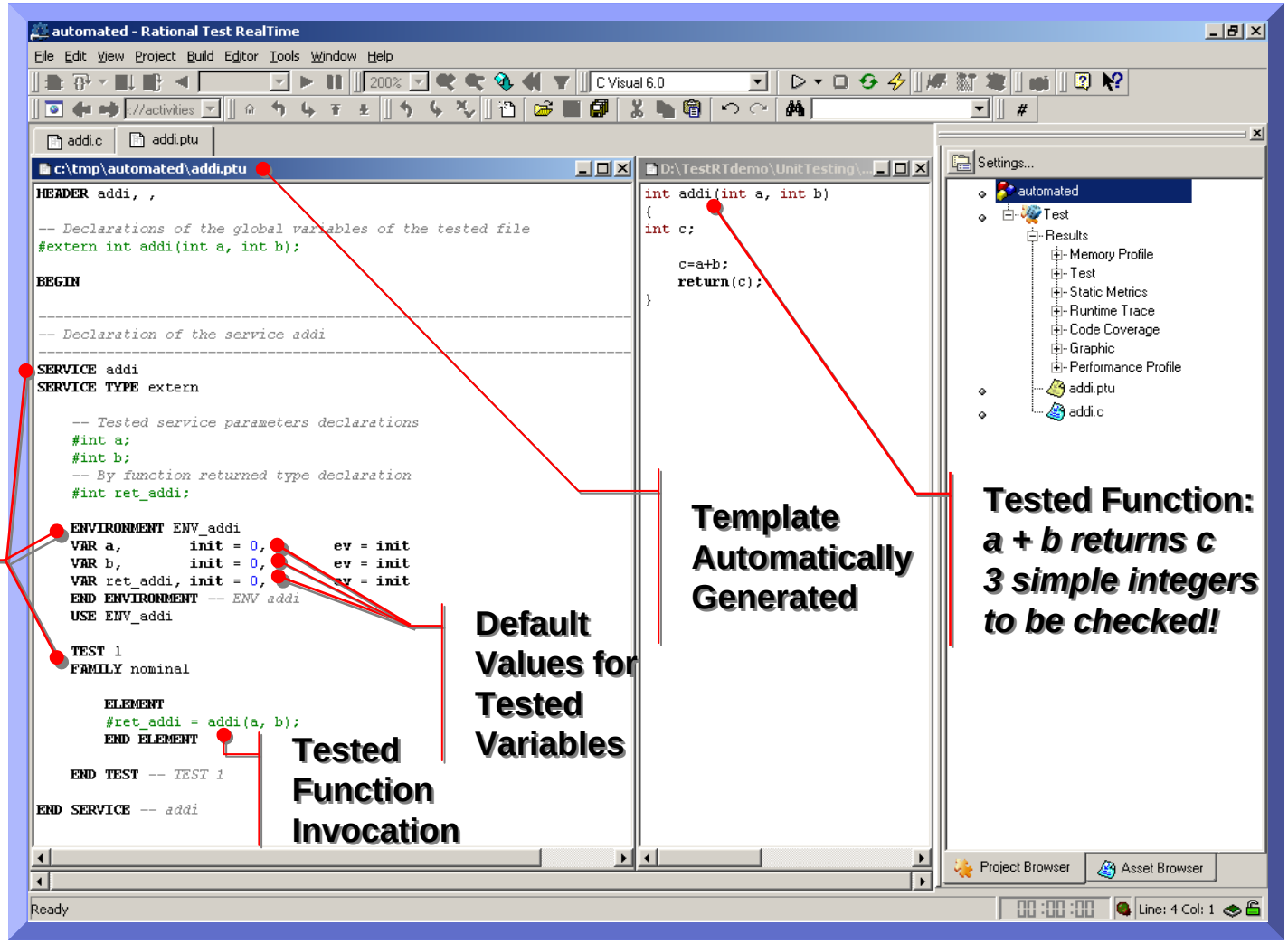
单元测试－C和Ada：*过程*



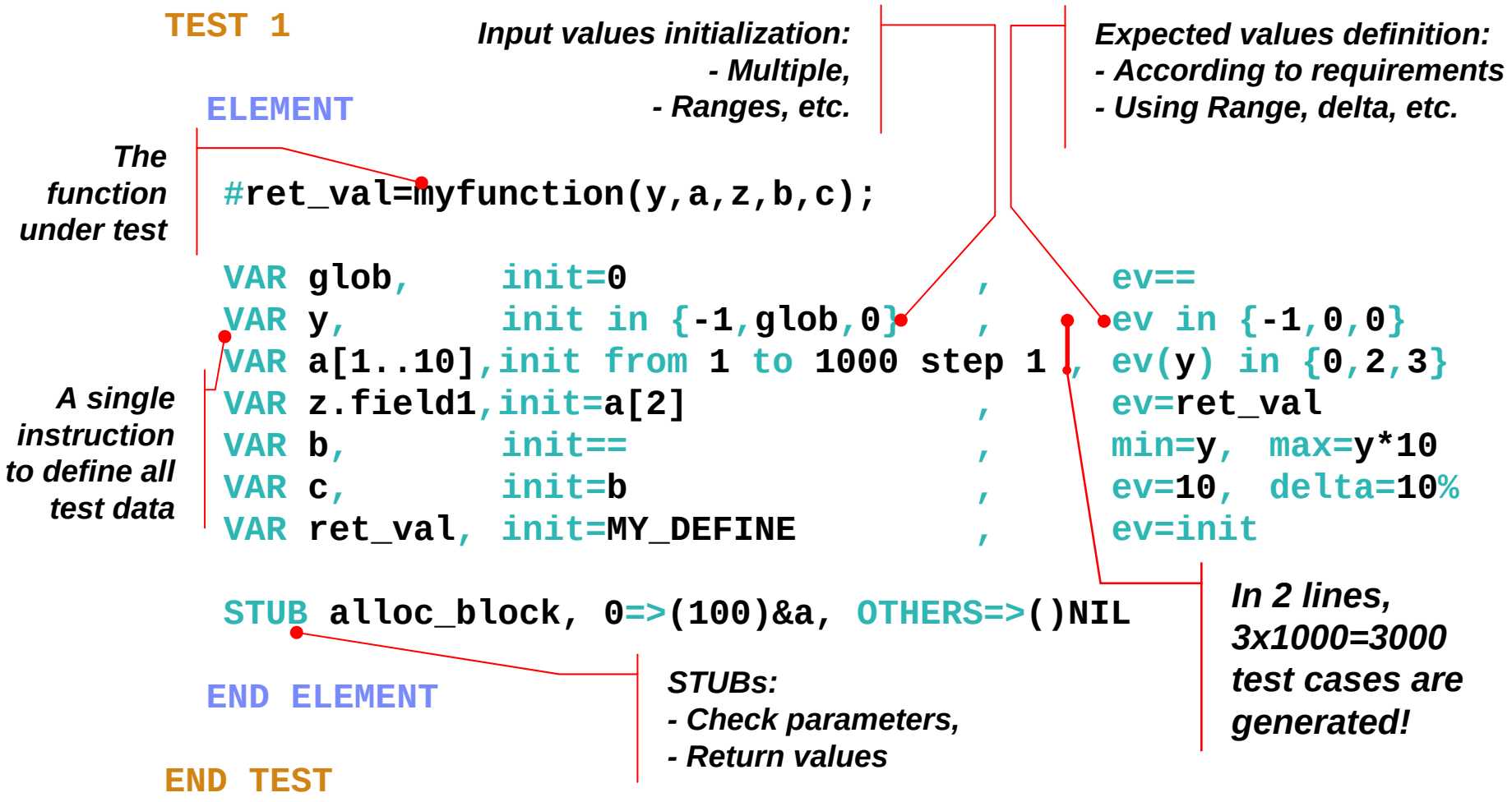
单元测试-C和Ada: 相关文件



单元测试-C和Ada: *测试脚本模板*

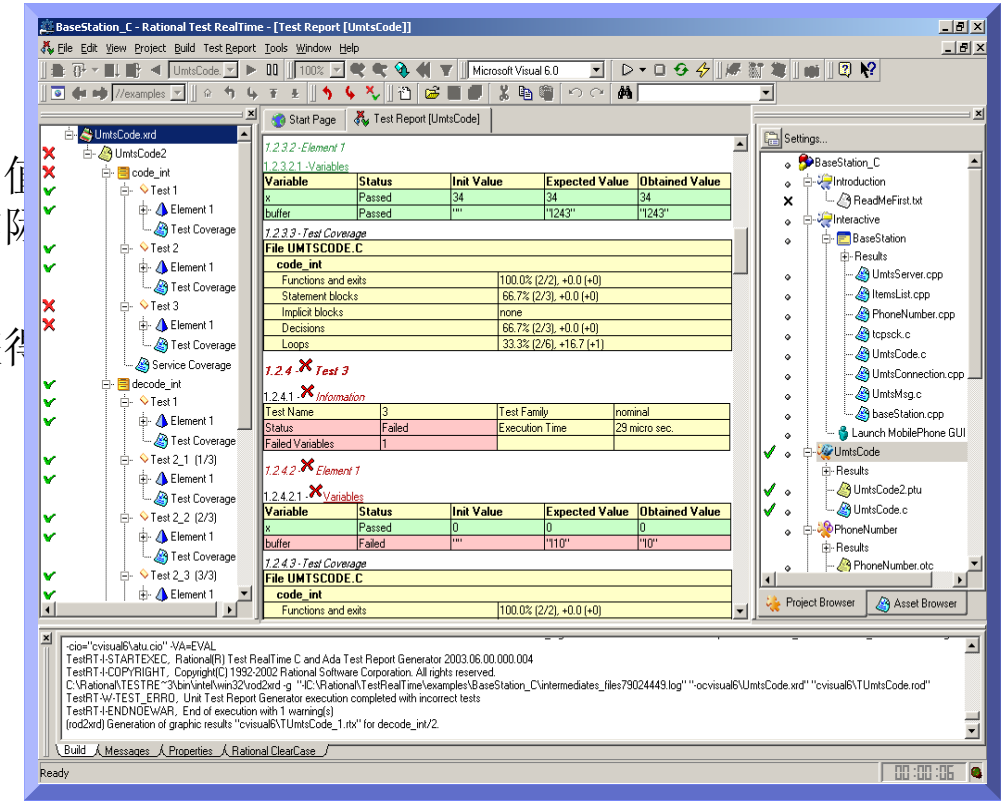


单元测试-C和Ada: *测试脚本语言*



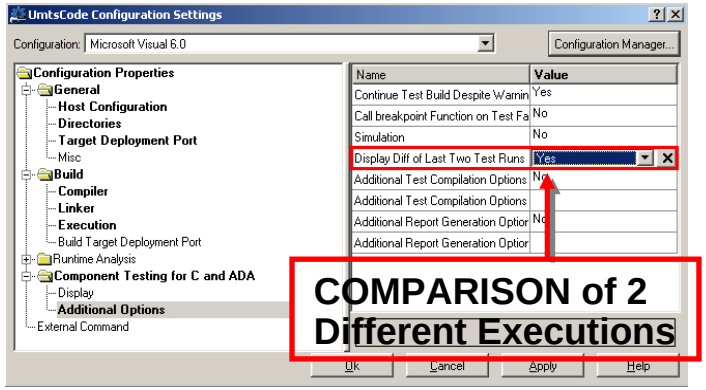
单元测试-C和Ada: *测试报告*

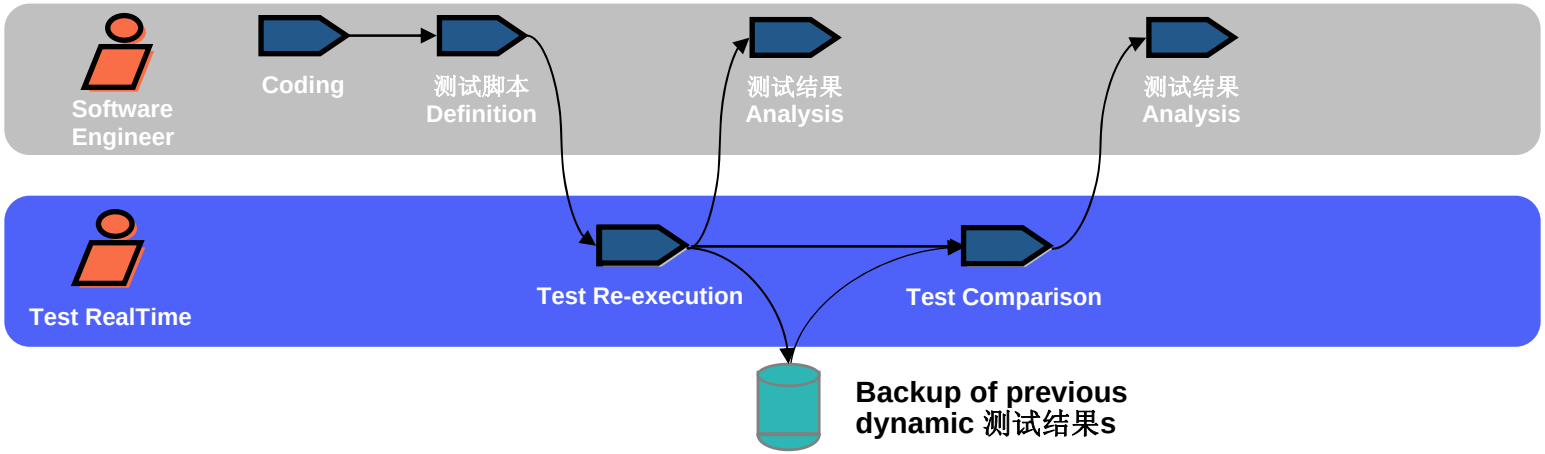
- 易于阅读和汇报
  - 易于检查通过和失败的测试
  - 易于对所有相关的变量和桩的值初始化、设置预期值、获取实际值
  - 从运行时分析的覆盖率功能获得待测代码的覆盖率信息
- 输出为 **HTML** 格式以便于
  - 分布式开发
  - 测试转包



单元测试-C和Ada: *测试比较*

- 对于:
  - 相同待测模块的两次迭代
  - 两种不同的开发环境
  - 查桩和没查桩的代码
  - 自动生成的代码和手动的代码





Variable	Status	Init Value	Expected Value	Obtained Value	Obtained value Comparison
x1	Failed	9	9	10	9
y1	Failed	9	9	10	9

IBM Software Group

Test RealTime 单元测试-C和Ada

DEMO

e business on demand.

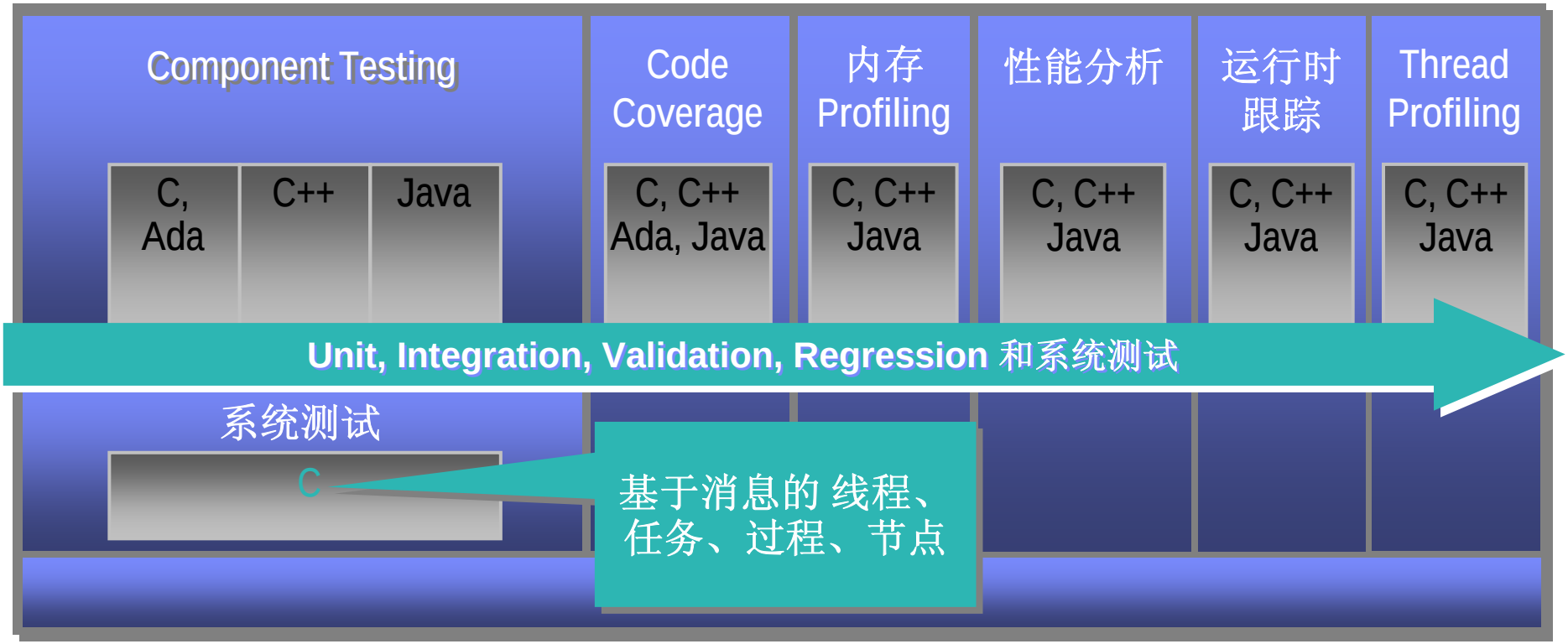
IBM Software Group

IBM Rational Test RealTime 系统测试-C

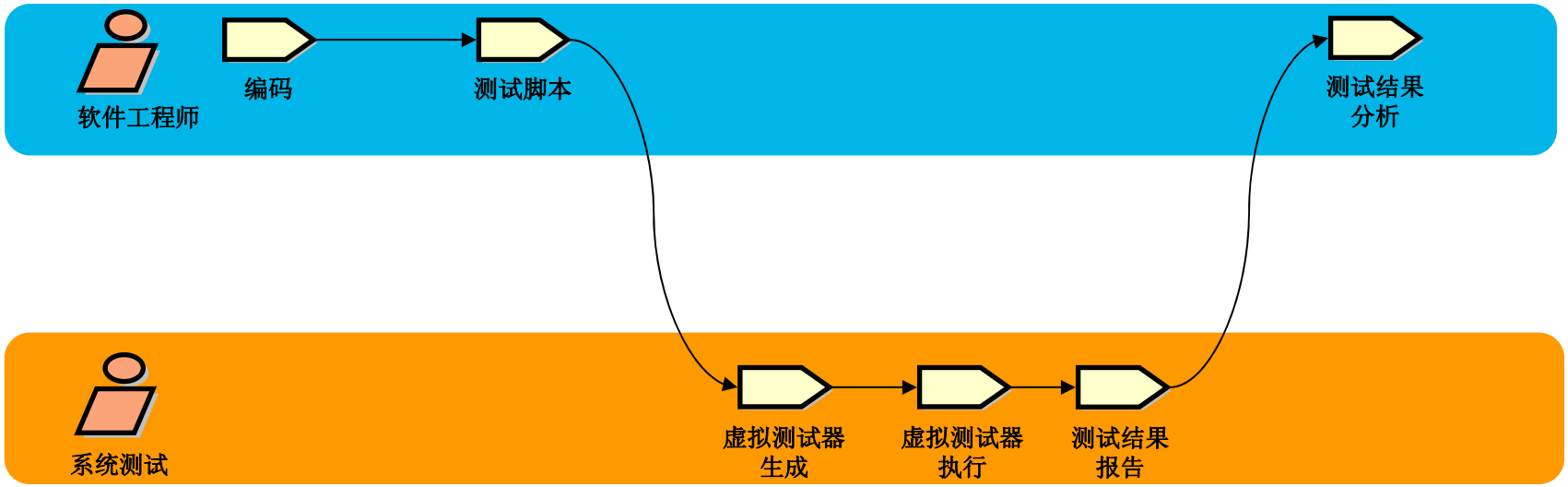
2007年制造业研讨会

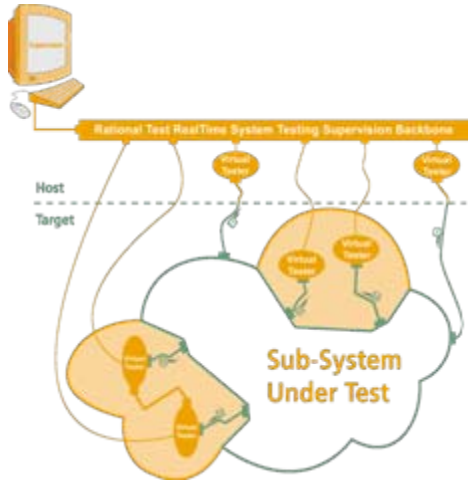
e business on demand.

Test RealTime: *系统测试-C*



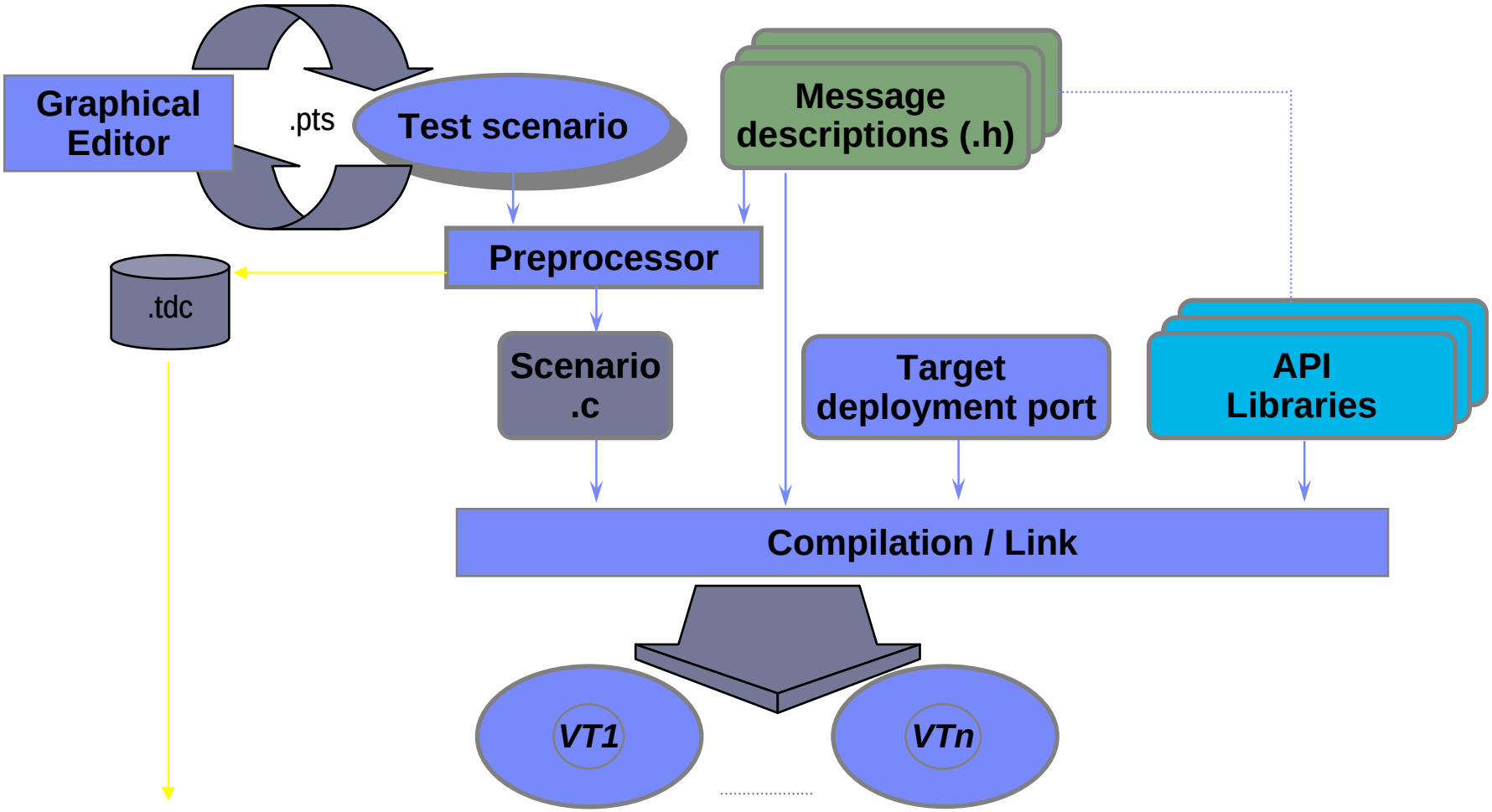
系统测试过程



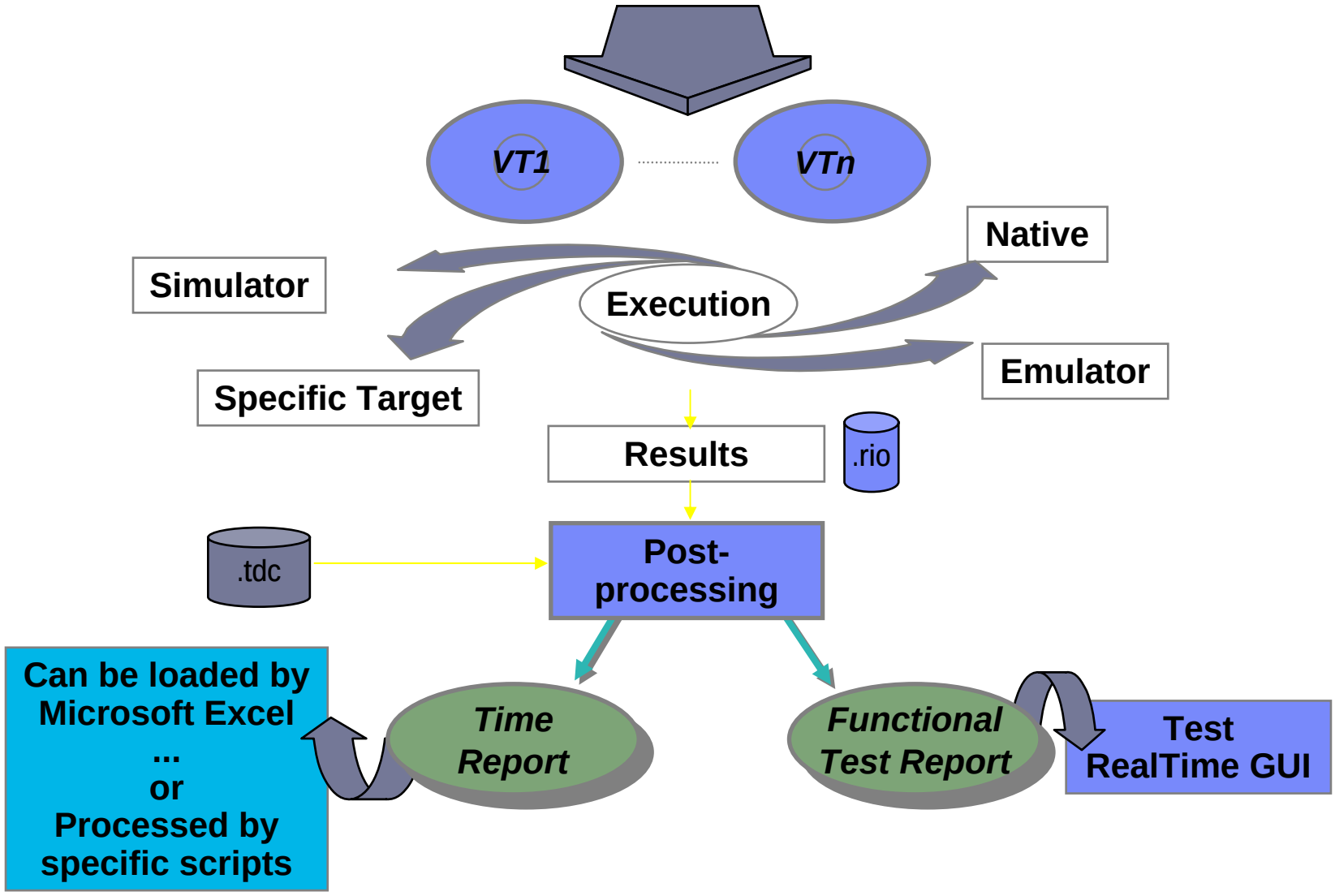


RTOS /
Standard APIs /
Distributed environments /
Simulators /
Emulators /
Targets

构建虚拟测试器


Graphical Editor
.pts
Test scenario
Message descriptions (.h)
Preprocessor
.tdc
Scenario .c
Target deployment port
API Libraries
Compilation / Link
VT1
VTn

虚拟测试器执行

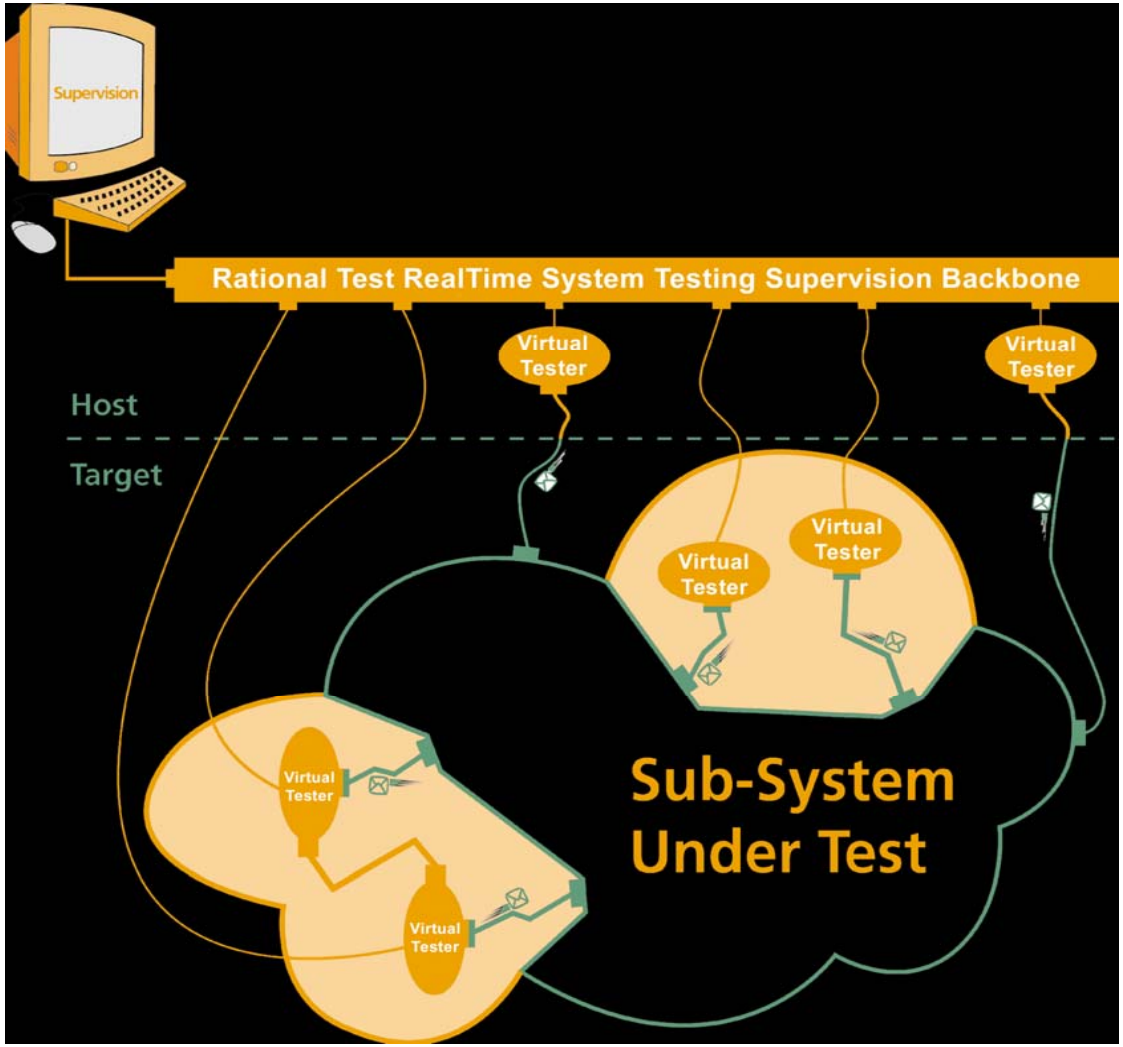

VT1
VTn
Native
Simulator
Execution
Emulator
Specific Target
Results
.rio
.tdc
Post-processing
Can be loaded by Microsoft Excel ... or Processed by specific scripts
Time Report
Functional Test Report
Test RealTime GUI

系统测试-C: *基于消息的分布式系统测试*

- 对以下待测对象进行集成测试和确认性测试:
  - 单个线程 …
  - 单个或多个任务 …
  - 单个或多个物理节点…
  - 直至大型网络系统
- 通过消息传递函数，进行功能测试、压力测试和性能测试
- 强大的脚本语言
- 详细的测试报告
- 回归测试
- 可以和运行时分析功能结合使用
  - 内存 和性能分析
  - 覆盖率和运行时跟踪

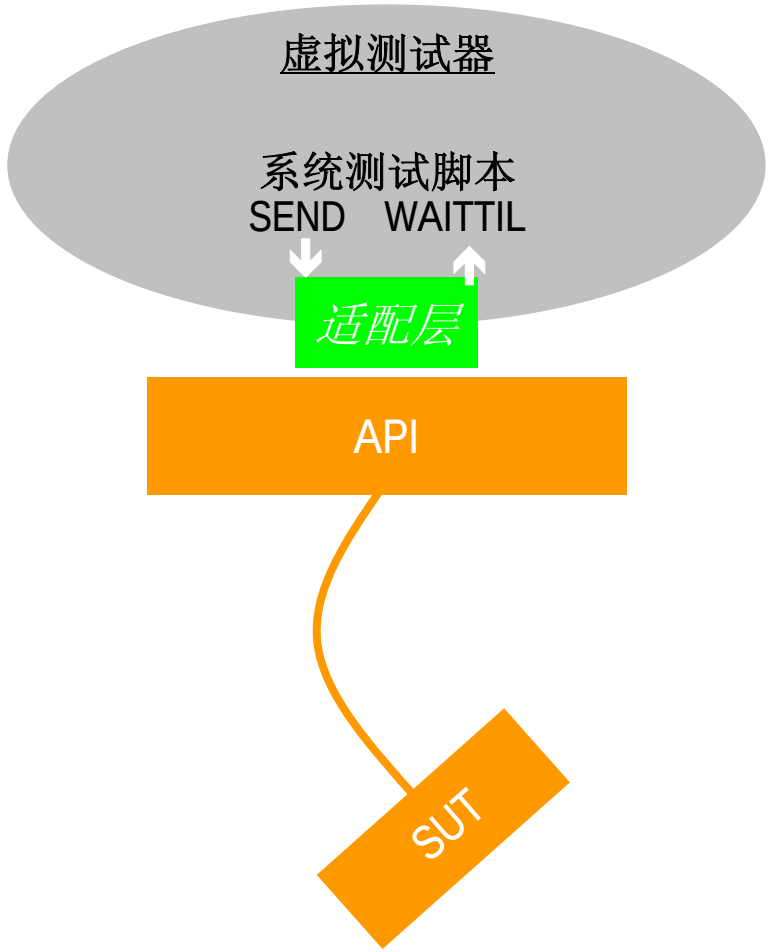
测试环境体系架构和功能

1. **虚拟测试器**
   - 模拟外部系统
   - 虚拟内部的桩模块
2. **每个虚拟测试器**
   - 发送事件到待测系统
   - 控制事件流
   - 验证事件数据和时间
   - 可再现的压力测试
3. **系统测试监控器**
   - 监控服务程序，用于虚拟测试器事件分发、通讯、同步



系统测试-C: *消息适配层*

- 支持各类通讯界面
  1. 适配层
     - 用C程序创建发送和接受事件
     - 事件的符号管理
     - 使测试脚本独立于消息API
  2. 消息API
     - 待测系统的一部分
     - 由一个通讯模块实现
     - 定义如何发送和接受消息



对不同的通讯方法进行系统测试

系统测试
works with
TCP/IP
UDP
X.25
MQSeries
RS 232/422
RS 485
GPIB
CAN/VAN
ARINC 429
…
and
your interfaces

系统测试-C: *测试脚本的结构*

```
INITIALIZATION init_proc()
TERMINATION end_proc()
EXCEPTION recover_proc()

SCENARIO main

   SCENARIO test_case1

   END SCENARIO -- test_case1

   SCENARIO test_case2

      INSTANCE Virtual_Tester1

      END INSTANCE – Virtual_Tester1

      INSTANCE Virtual_Tester2

      END INSTANCE – Virtual_Tester2

   END SCENARIO -- test_case2

END SCENARIO -- main
```

前导处理块，后续处理块，
错误处理块

测试脚本由
SCENARIOs 和sub-SCENARIOs组成 …

一个**scenario**可以分成几个**INSTANCE**块，来定义一些异步事件 **(虚拟测试器)**

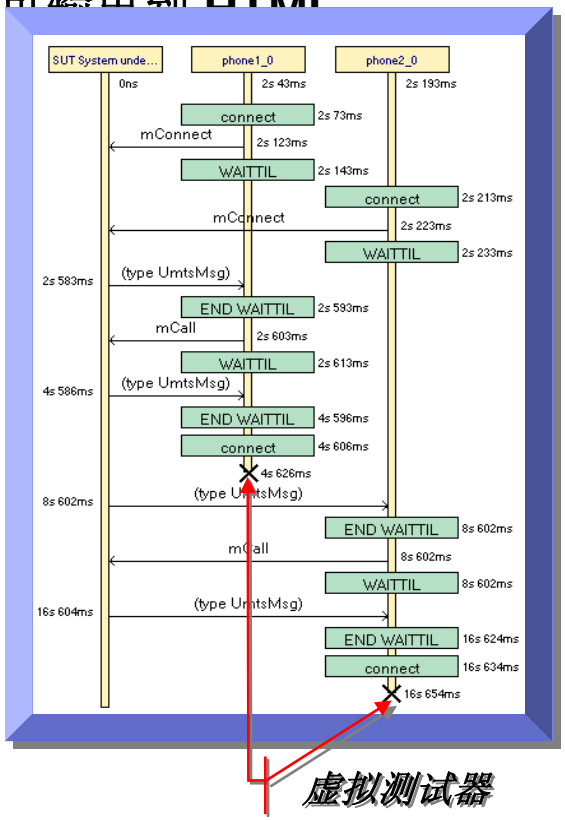
系统测试-C: *测试脚本语言*

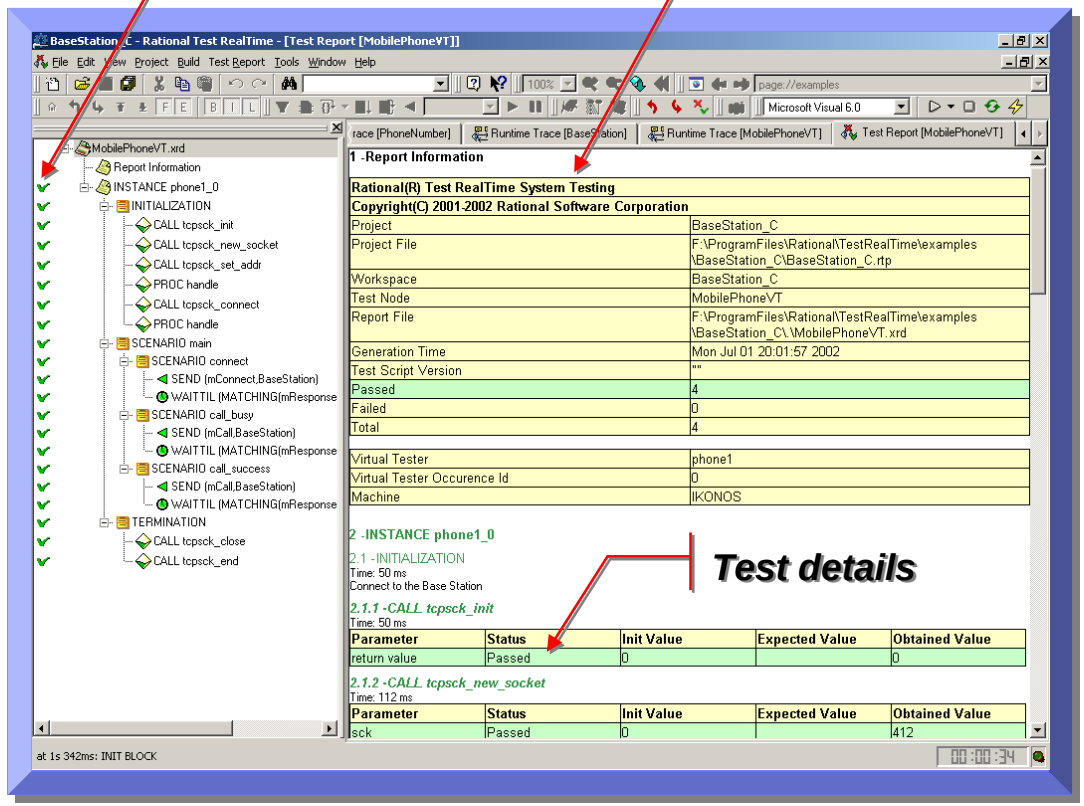
- ***Loops***
- ***Synchronizations between 虚拟测试器***
- ***IF statements***
- ***Init of outgoing Events***
  Send an Event via a defined communication channel
- ***Set up a timer***
- ***Definition of constraints on incoming Events***
  Wait for complex conditions
- ***Call to external C code***

```
CHANNEL MY_COMMTYPE:mylink
SCENARIO example
  WHILE (TIME(mytimer) < 100)
      RENDEZVOUS start_example
      IF (sync == 0) THEN
       VAR creq, INIT={send=>…,neg=>{opt=> …, }}
       SEND (mylink, creq)
      END IF
      TIMER mytimer
      DEF_messagge cresp, EV= { … }
      WAITTIL (MATCHED(cresp)||MATCHING(cack),WTIME>15)
      CALL myexternal_func()
  END WHILE
END SCENARIO -- example
```

系统测试-C: *报告*

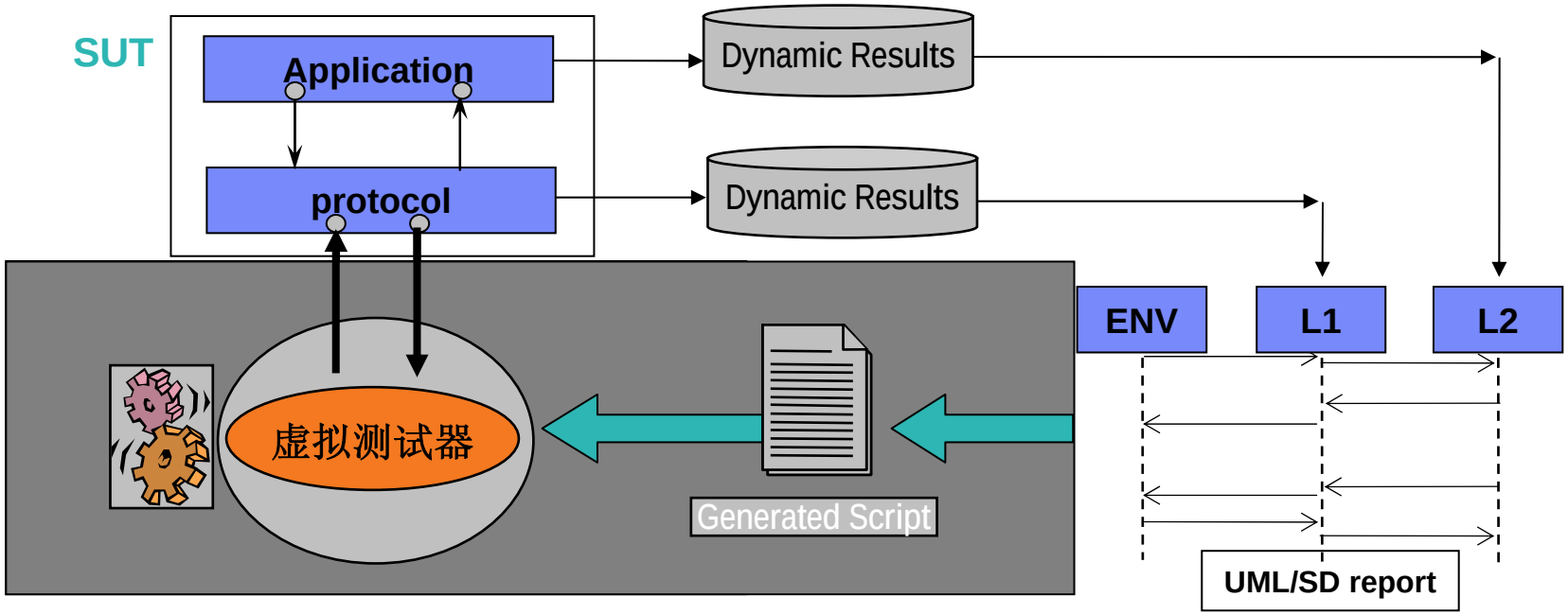
- 在 顺序图、测试报告 和源码间实现超连接来方便的浏览
- 可输出到 HTML





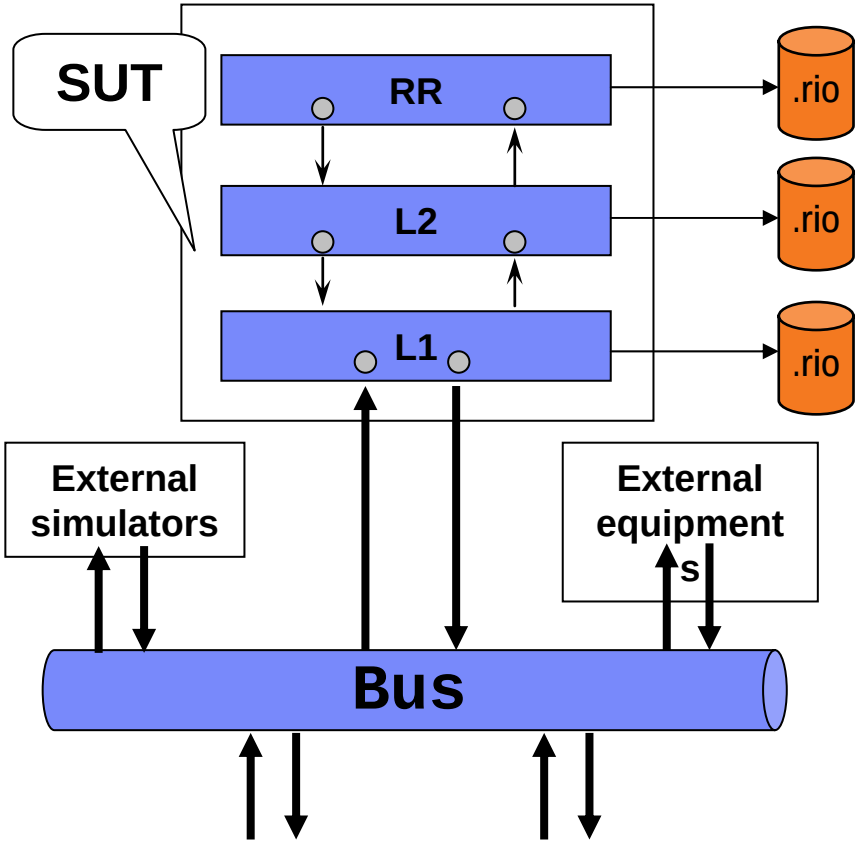
Probe : *显示消息交互并回放*

- 部署过程：
  - 在应用程序或模拟器中插入probes以记录发送和接受的消息
  - 使用模拟器、真实设备或GUI界面激励系统运行，生成行的测试测试脚本模板
  - 运行后，以顺序图显示消息或事件的交互UML/SD，并生成脚本用于回放



Probes查桩

- 在源码中插入probe
  - 能在源码中如何位置插入probe，包括应用代码或消息API



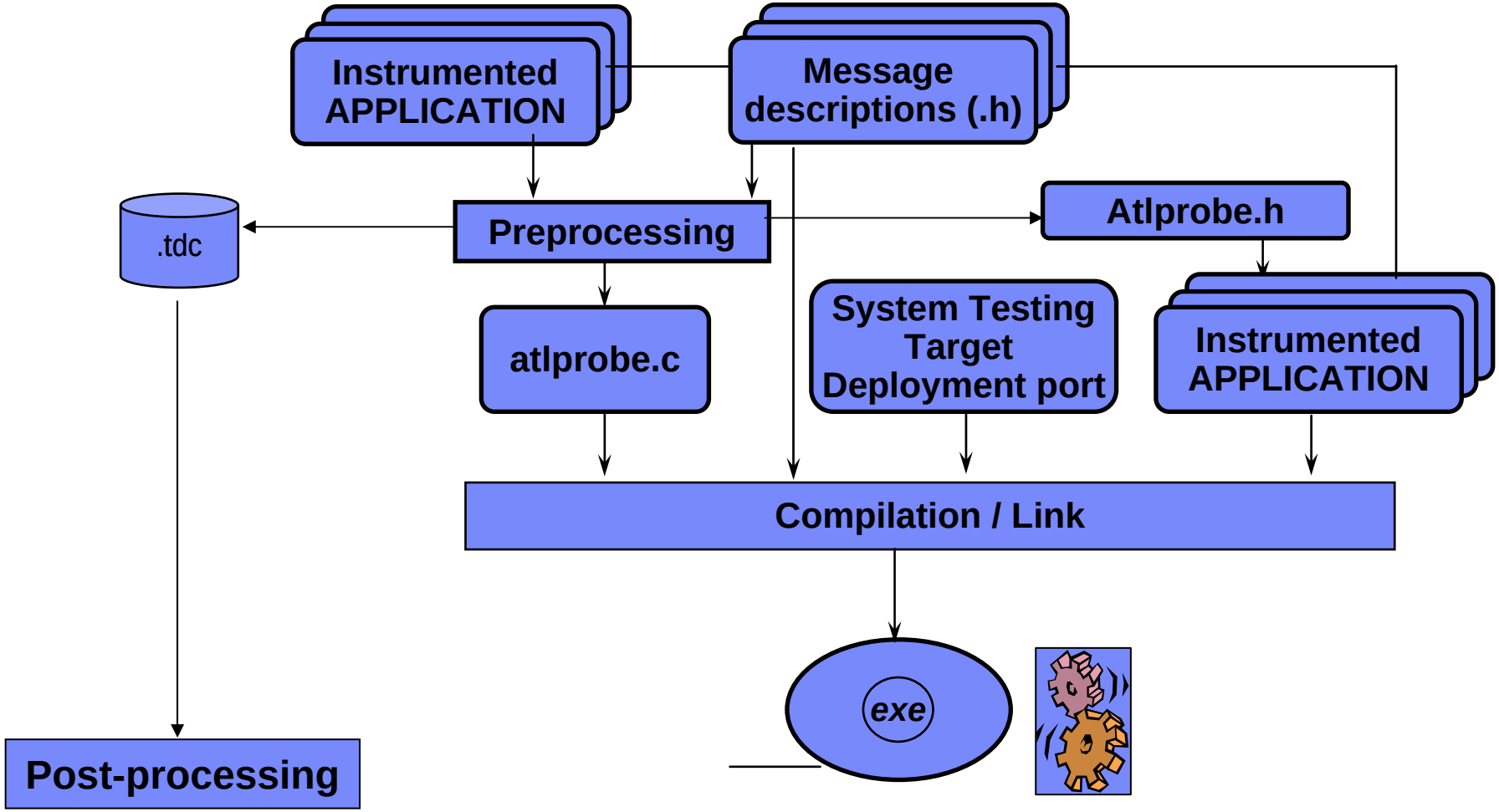
- 每个任务中可以使用不至一个probe.
- 每个 taskId 将生成一个.rio 文件和在脚本中对应的INSTANCE

Part of the code where the instrumentation is added

atl_send_trace(*taskId,dest,msg,type,name***);**

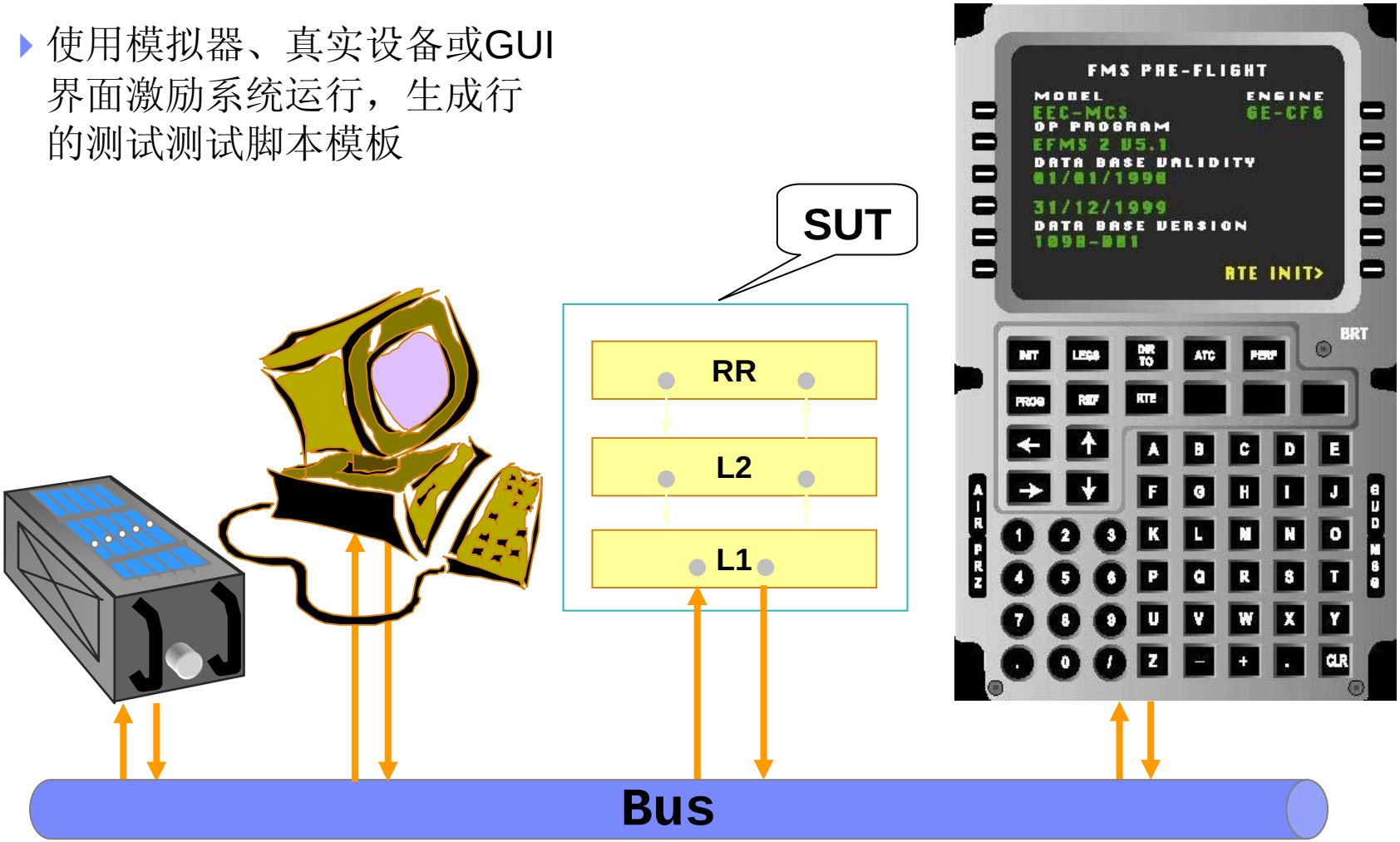
atl_recv_trace(*taskId,src,msg,type,name***);**

Probe 代码生成和待测系统重建

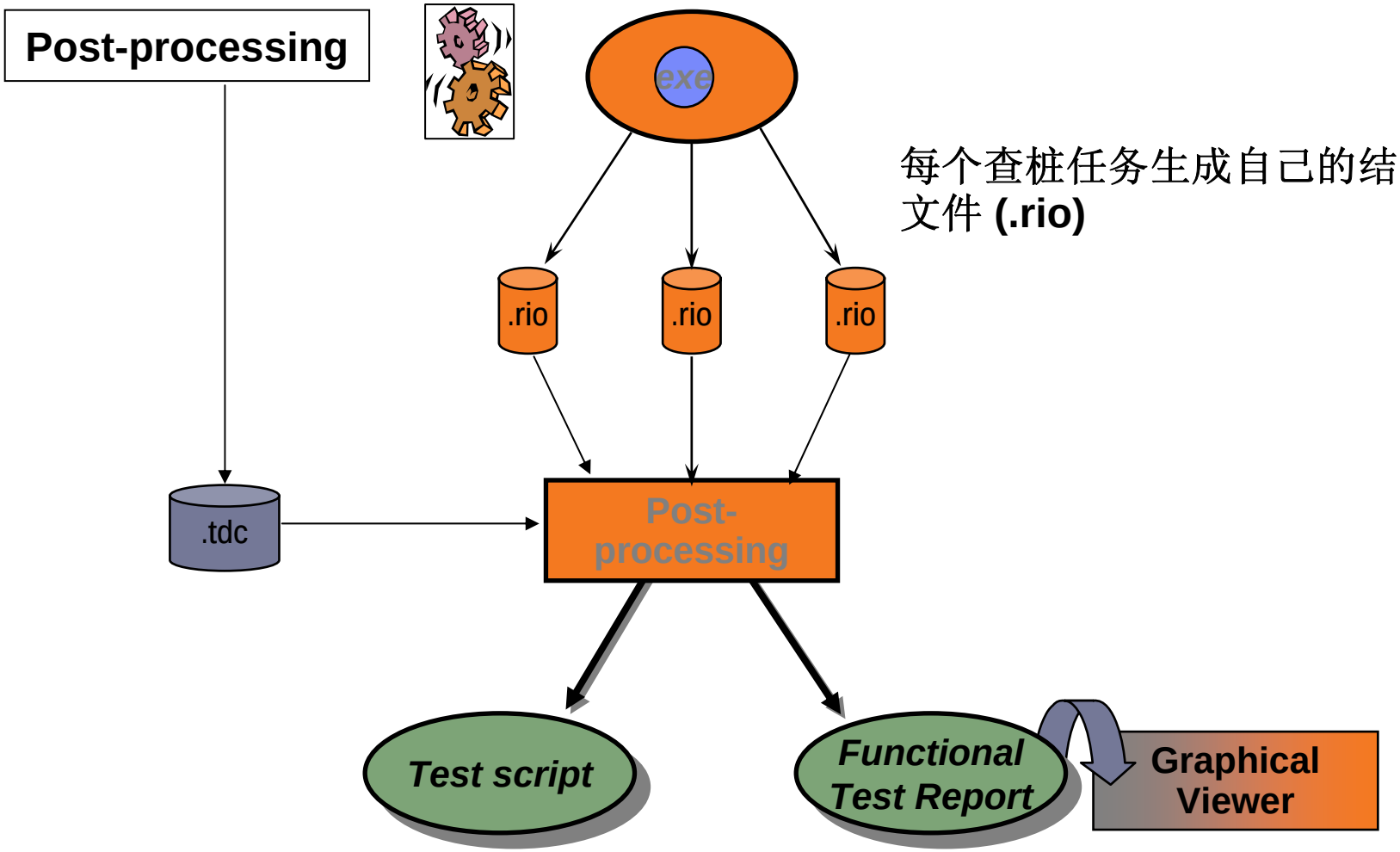


执行并生成脚本

- 使用模拟器、真实设备或GUI界面激励系统运行，生成行的测试测试脚本模板



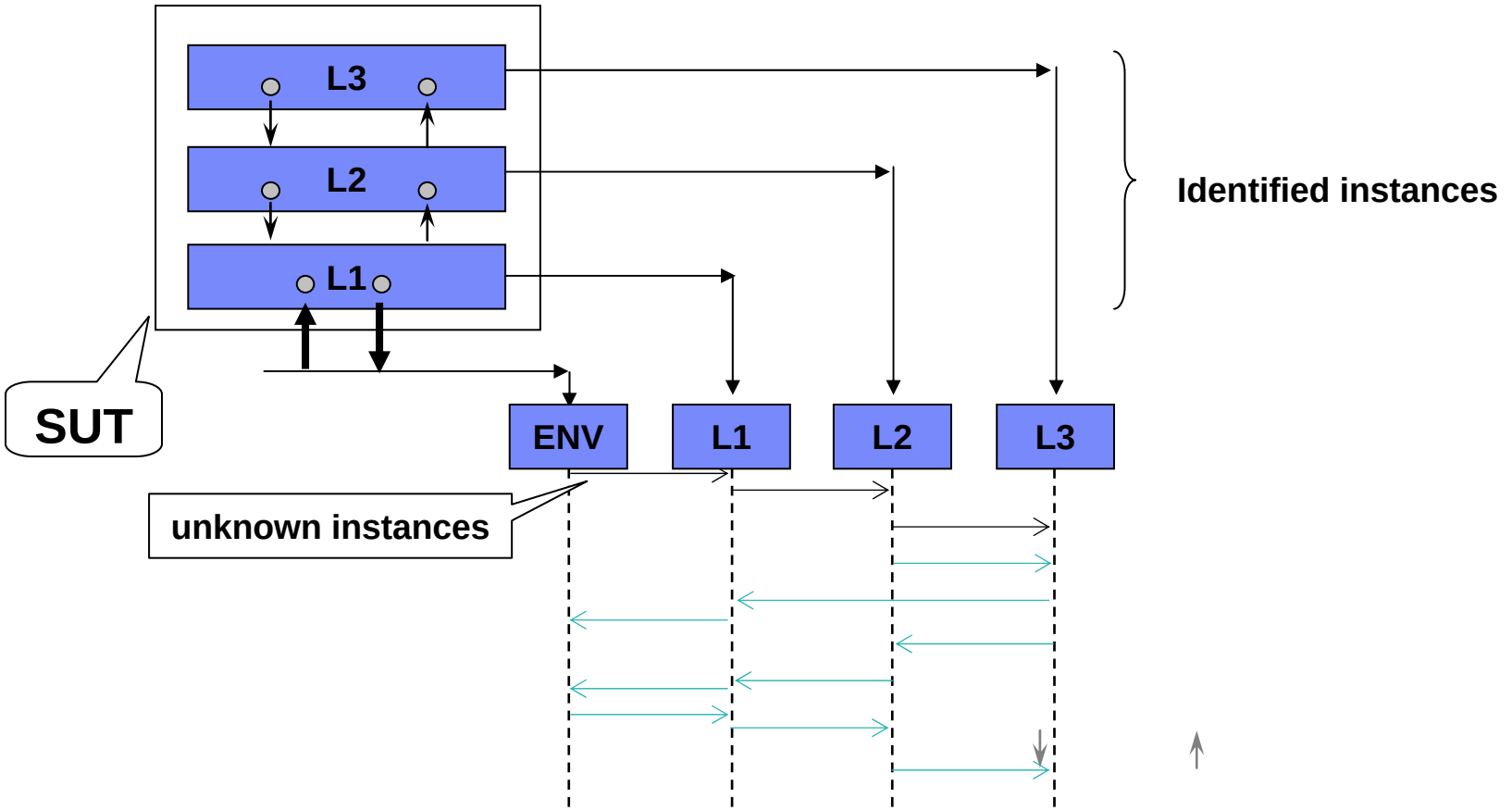
运行后生成报告



每个查桩任务生成自己的结果文件 **(.rio)**

用顺序图显示系统消息交互

- 运行后，以顺序图显示消息或事件的交互UML/SD，并生成脚本用于回放



Probe的用途

- 用Probe生成测试脚本用于测试应用
- 用Probe替代旧的模拟器
- 用Probe用于回归测试
- 用Probe 功能校验
- 用Probe 用于调试
- 用Probe 用于规格验证
- 用Probe基于规格的开发－规格错误的注入
- 用Probe 替代人机操作中的问题－24小时工作、随机错误、交互较慢

总结：Probes特性

- Probes 实现出入应用代码以记录手工测试中的消息交互
- 从记录的消息中生成测试脚本
- 自动回放手工测试
- 几乎不依赖复杂的环境，更易于及早实现自动化测试
- 在源码的特定位置放置probe，决定需要记录和回放的部分
- 生成测试脚本，通过拷贝和修改即可适应不同的测试场景
- Probe功能的多种使用方式，以支持测试的不同需求。

IBM Software Group

Test RealTime 系统测试-C

DEMO

e business on demand.

IBM Software Group

IBM Rational Test RealTime 代码编程规则检查- MISRA-C编程规范

2007年制造业研讨会

e business on demand.

缺陷的代价

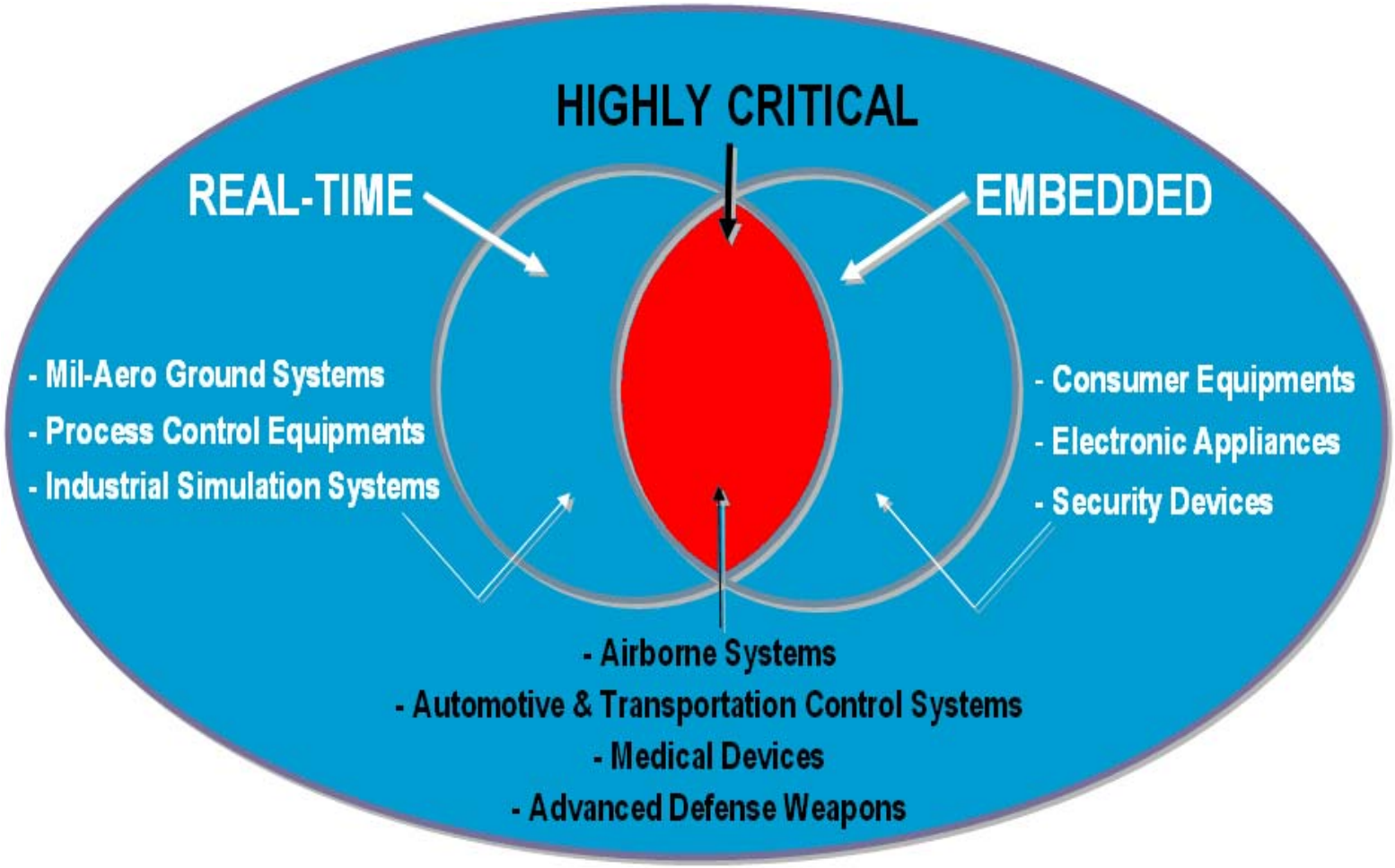
- 安全关键系统 (Safety Critical Systems)
  - 过程、汽车和航空控制
  - 关键任务型航空电子设备
- 缺陷成本高昂
  - Pentium处理器的缺陷
  - Ariane 5的爆炸
  - 火星大气轨道探测器的坠毁

Shooting Down of Airbus 320

- 1988
- US Vicennes shot down Airbus 320
- Mistook airbus 320 for a F-14
- 290 people dead
- Why: Software bug - cryptic and misleading output displayed by the tracking software

关键软件在哪里运行呢？

- 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。
  - 运行错误：程序运行种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
  - 人工方式进行的代码复审。
- 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

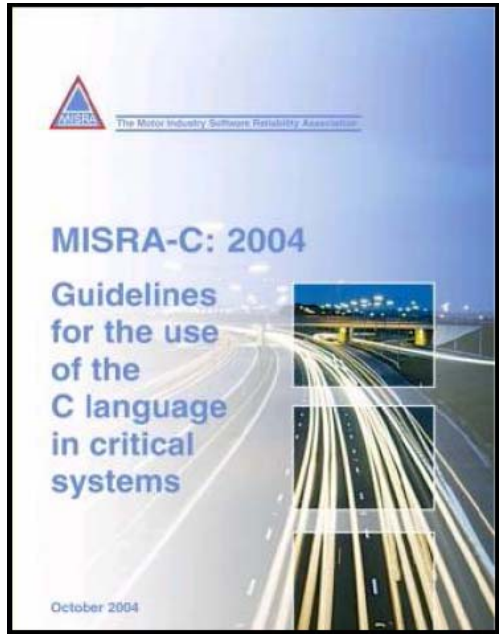
- 主要收益：降低修复缺陷的成本，提高生产率
  - 根据Glen Russell (Nortel Technologies)在1991年的调查，大约65%到90%的运行时缺陷可以通过代码审查发现，而成本只需测试的1/4到2/3 。
- 其他益处：
  - 更好的控制开发流程
  - 更高的质量
  - 降低缺陷密度
  - 降低查找和定位缺陷的成本

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



<表达式>类规则

例1：

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

不同的编译器给出的结果不一样，是b[1]还是b[2]？

应：x=b[i];

i++;

例2：

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

表达式并没有执行，只是得到表达式类型int的size

<Switch语句>

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

如果default 标签被错误拼写为defalt ， 编译程序将不会把它当作一个错误。如果程序设计员没有注意到这个错误，预期的默认操作将永远不会执行。

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
- 代码复审发现的问题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

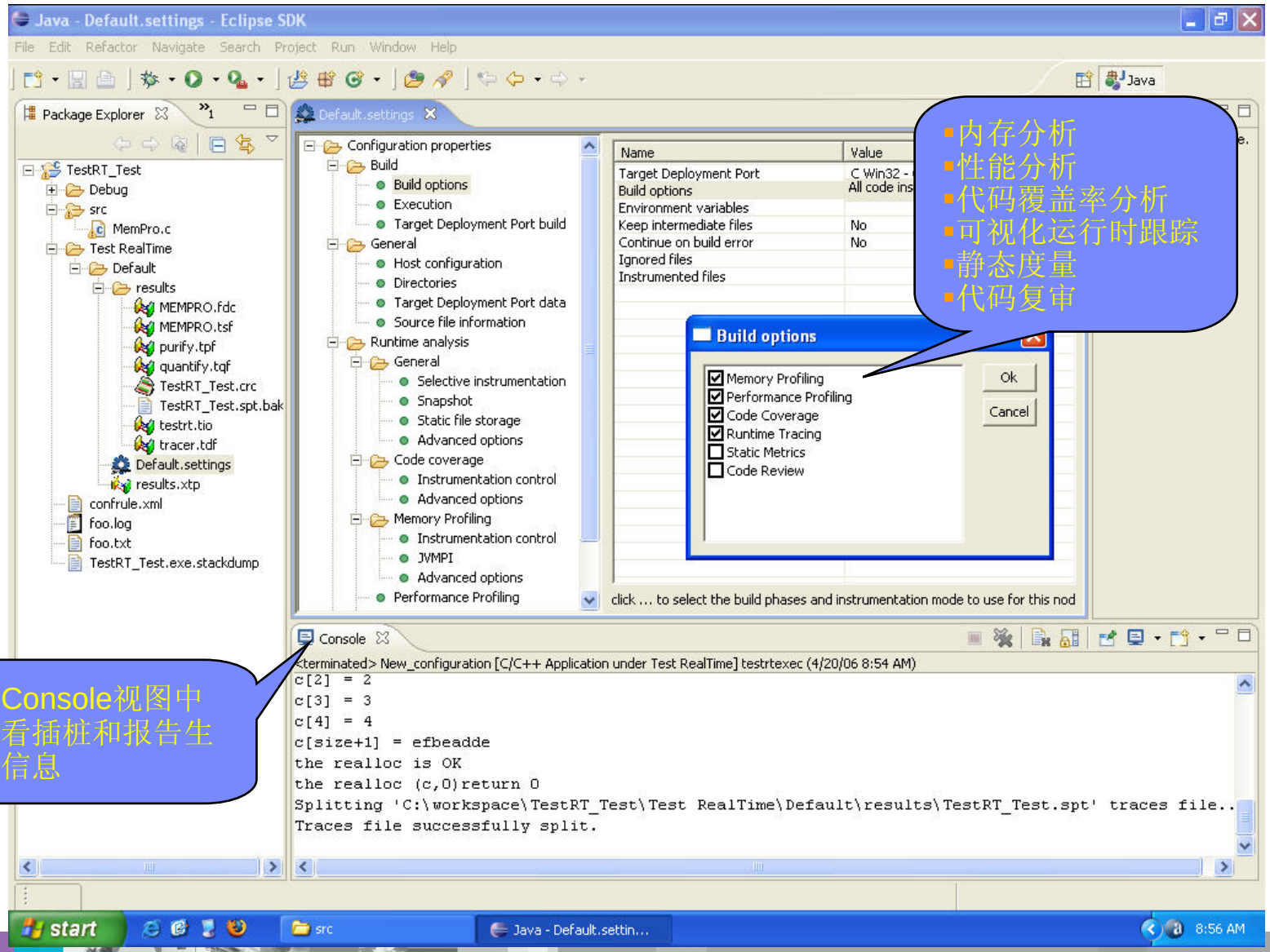
多才多艺的Test RealTime

- 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

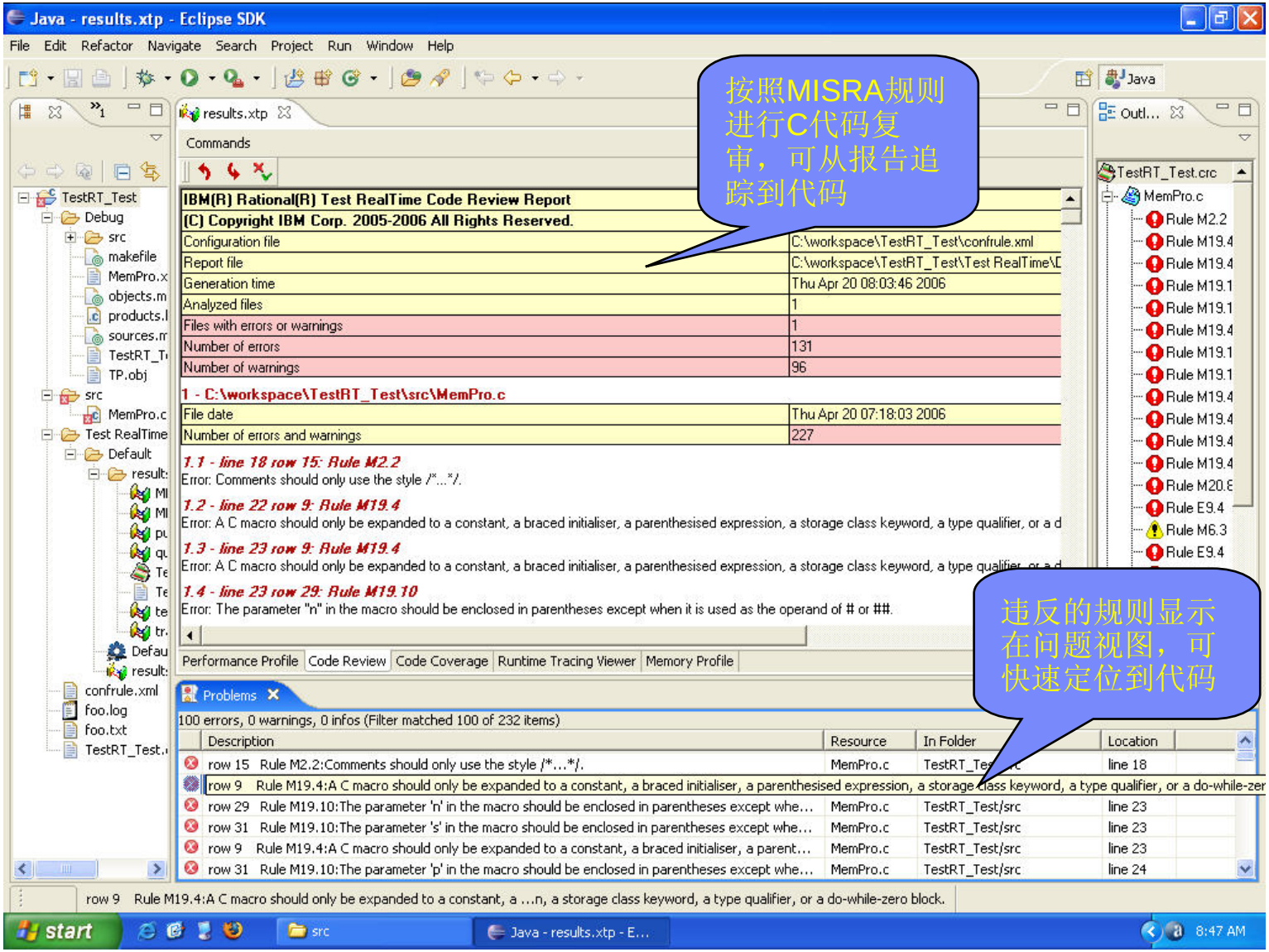
与Eclipse 3.1和 CDT 3.0的集成



- 内存分析
- 性能分析
- 代码覆盖率分析
- 可视化运行时跟踪
- 静态度量
- 代码复审

在Console视图中查看插桩和报告生成信息

符合MISRA-C 2004标准的C语音代码复审



IBM Software Group

IBM Rational Test RealTime-代码编程规则检查

DEMO

e business on demand.