爱开发 重创新 更智慧

Innovate2011

IBM Rational 软件创新论坛

Software. Everyware.

软件开发中心解决方案之软件开发过程质量管理平台

刘鹤辉 hehuiliu@cn.ibm.com 高级技术专家
李燕生 liyans@cn.ibm.com 资深咨询顾问
陈大金 chendj@cn.ibm.com 高级客户技术专家

议程

- 开发中心质量管理面临挑战总结
  - 当前开发中心质量管理面临的主要挑战
  - 提高开发人员代码质量的最佳实践
  - 最佳实践实施面临的主要挑战
- 质量管理平台解决方案
  - 质量管理平台整体方案
  - 基于最佳实践经验的方法论
  - 质量管理平台实现
- 实施案例分享
  - 在国内某开发中心实施本方案的路线图
  - 实施过程体会

当前开发中心质量管理面临的主要挑战

- 业务需求的紧迫性
  - 实际客户对产品质量的苛刻要求已成为大家的普遍共识
  - 软件应用的普遍性和实际应用环境的复杂性更加重了对软件产品质量要求的严苛性
- 大量的存量代码（遗产代码）加重了代码质量管理的复杂性
  - 很少有一个产品（应用）是从头全新开发的
    - 更多的情况是在原有产品（应用）/平台的基础上进行定制/改进开发的
  - 新的开发人员对原有产品（应用）的架构、设计不熟悉
    - 每进行一点修改都会存在潜在的不可预知的影响
    - 同时新开发人员也不具备对改动影响进行透彻分析的技能
  - 已有产品（应用）中存在的大量技术债务使这种状况进一步恶化
    - 臃肿的架构和代码结构
      - 新开发人员要读懂原有代码尚且困难，更何况还要进行修改

当前开发中心质量管理面临的主要挑战

- 开发中心的快速发展则更进一步加重了代码质量管理的难度
  - 大量的新员工加入，技能在短时间内无法很快提高
  - 对已有代码更加难以控制
  - 熟悉存量代码的员工已经不再编程序了
- 开发进度非常紧张，不可能有充分的时间进行回归测试

提高开发人员代码质量的最佳实践

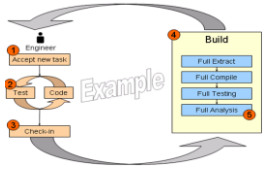
- 软件工程经过多年的发展积累，特别是新的质量管理理念的应用和敏捷开发方法的实际应用，业界已经形成了提高开发人员代码质量的一系列最佳实践

自动化静态代码分析

通过自动化的静态代码分析自动发现程序中存在的代码问题，从而可以尽早修复问题；通过自动化分析，获得代码的度量信息，及时了解代码质量状况，为后续改进指明方向。

自动化单元测试

通过单元测试一方面可以尽早发现代码缺陷，另一方面则可为后续的代码改动构建一个强大的安全网；自动化单元测试中的覆盖率分析则为单元测试的持续改进指明方向。



持续集成

通过持续集成平台持续不断的编译构建、静态代码分析、自动化单元测试等，为程序质量提供持续不断的健康状况反馈，及时发现代码质量问题。

最佳实践实施面临的主要挑战

- 如何结合完整的方法论的支持
  - 具体实施时需要按照哪些步骤进行
  - 需要注意哪些事项
- 缺乏一个完整的端到端的支撑平台
  - 很多开发中心已经拥有了部分或者分离的最佳实践实施技术平台/框架
    - 单元测试框架
    - 静态分析工具
    - 持续集成工具
  - 但缺乏一个可以将这些分离实践整合的完整平台
  - 或者在某些具体点的实施上碰到了困难而停顿不前
- 缺乏深入的度量监控机制
  - 究竟实施效果如何？存在哪些问题？应该如何改进？
  - 代码中还存在哪些方面的质量风险？后续应该从哪些方面进行改进？

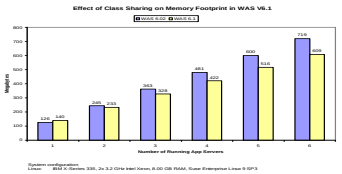
议程

- 开发中心质量管理面临挑战总结
  - 当前开发中心质量管理面临的主要挑战
  - 提高开发人员代码质量的最佳实践
  - 最佳实践实施面临的主要挑战
- 质量管理平台解决方案
  - 质量管理平台整体方案
  - 基于最佳实践经验的方法论
  - 质量管理平台实现
- 实施案例分享
  - 在国内某开发中心实施本方案的路线图
  - 实施过程体会

开发过程质量管理平台方案

全自动化的完整的支撑平台

持续集成平台



持续集成报告

获取源代码及配置文件

自动编译构建

静态代码分析

单元测试/覆盖率分析

代码质量度量

搜寻合适的构建服务器

自动构建部署

手工/自动冒烟，系统测试

触发

统一版本管理

代码提交

深度的代码质量度量

质量度量模型

虚拟测试服务器池

基于最佳实践经验的方法轮支撑

应用

单元测试/覆盖率分析

静态代码分析

客户端

开发团队

持续集成最佳实践方法论

单元测试方法论

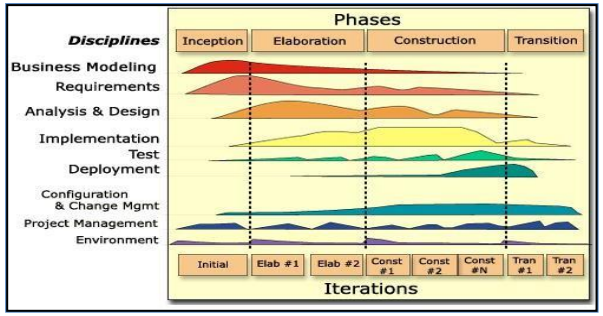
质量度量分析方法论

议程

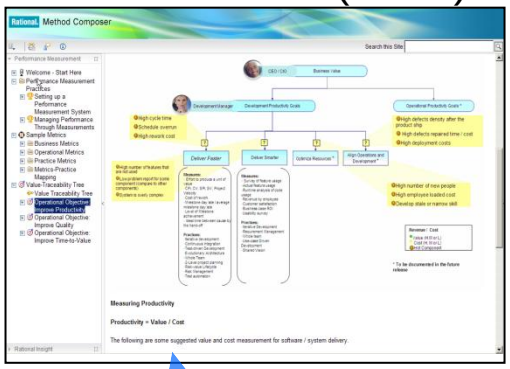
- 开发中心质量管理面临挑战总结
  - 当前开发中心质量管理面临的主要挑战
  - 提高开发人员代码质量的最佳实践
  - 最佳实践实施面临的主要挑战
- 质量管理平台解决方案
  - 质量管理平台整体方案
  - 基于最佳实践经验的方法论
  - 质量管理平台实现
- 实施案例分享
  - 在国内某开发中心实施本方案的路线图
  - 实施过程体会

基于最佳实践经验的方法论

Rational Unified Process (RUP)



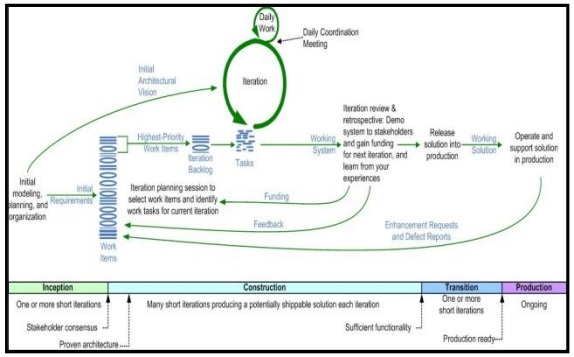
Measured Capability Improv-ement Framework (MCIF)



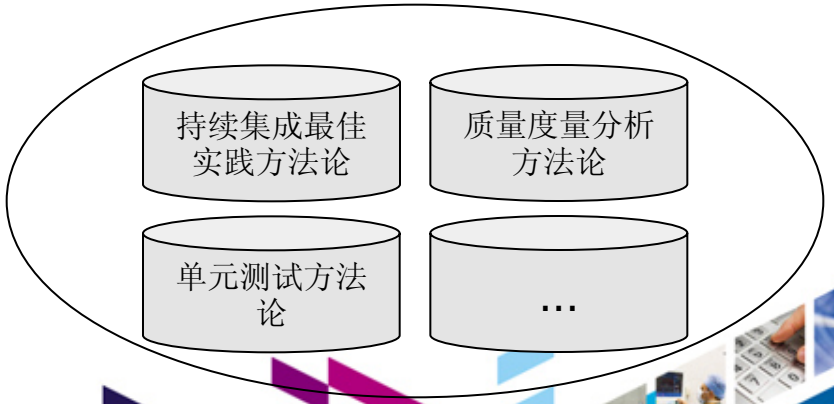
获取

获取

Disciplined Agile Delivery(DAD)



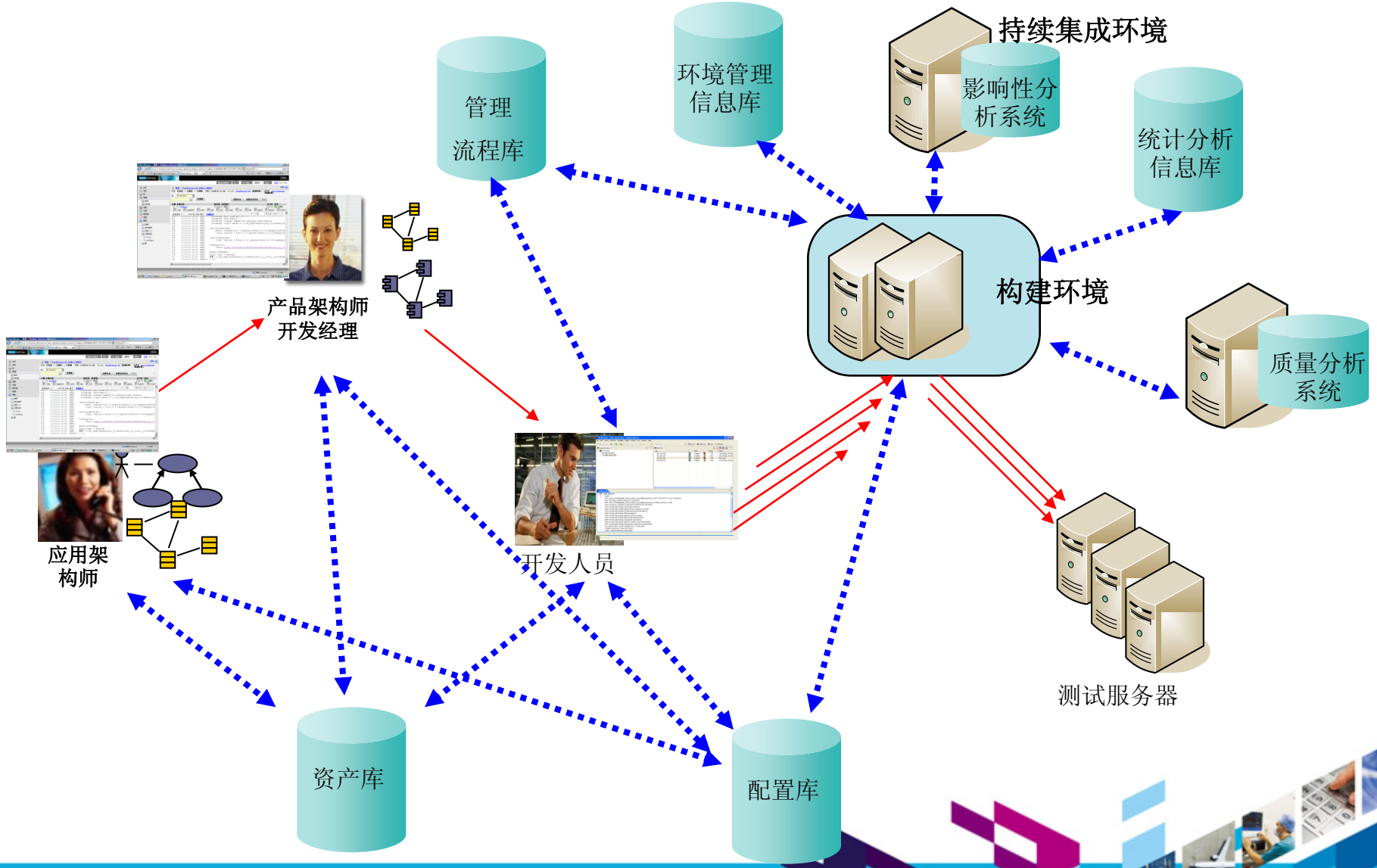
获取



持续集成最佳实践方法论

- 持续集成应该是多层次，多角度的
- 项目级持续集成
  - 偏重项目开发过程中的计量的不断验证。
  - 将单元测试，静态代码分析，覆盖率分析，功能测试等融入持续集成。
  - 项目工件的质量由项目团队自己负责，并提供企业作为分析依据。
- 企业级持续集成
  - 偏重基于软件流水线的自动编译，构建，组装，发布，信息分析。
  - 将端到端的自动化处理，和冒烟测试，信息统计分析和发布融入持续集成过程。
  - 产品的质量有专门的质量团队进行评审。

企业级持续集成样例


持续集成环境
管理
流程库
环境管理
信息库
影响性分
析系统
统计分析
信息库
构建环境
质量分析
系统
产品架构师
开发经理
应用架
构师
开发人员
测试服务器
资产库
配置库

单元测试方法论

对单元测试实际执行过程中开发人员可能存在主要困惑的理解

- 单元测试在发现缺陷的能力上似乎并不像理论上描绘的一样有效
- 代码覆盖率指标和现实中紧迫进度压力之间的矛盾
- 理想的自动回归测试场景与现实中疲于修改测试用例代码之间的矛盾

对单元测试实际执行过程中所可能存在的困惑的原因分析

- 开发者测试的有效性不仅仅依赖于代码覆盖率，实际上更取决于开发者测试用例的有效性
- 开发者测试用例编写的效率取决于程序设计技能和优秀的测试支持框架
- 自动回归测试需要建立在具有良好设计的测试用例代码和持续代码（包括测试用例代码）重构基础上

单元测试实施整体方案

测试用例设计方法

如何设计高质量的单元测试用例？如何达到较高的代码覆盖率？等用例设计的方法论指导

测试用例代码框架生成引擎

基于特定测试支持框架和执行框架，根据具体程序语言、应用类型特征，应用模型驱动开发思路所开发的用于生成测试用例代码框架的生成引擎，该引擎为可选。

单元测试支持框架

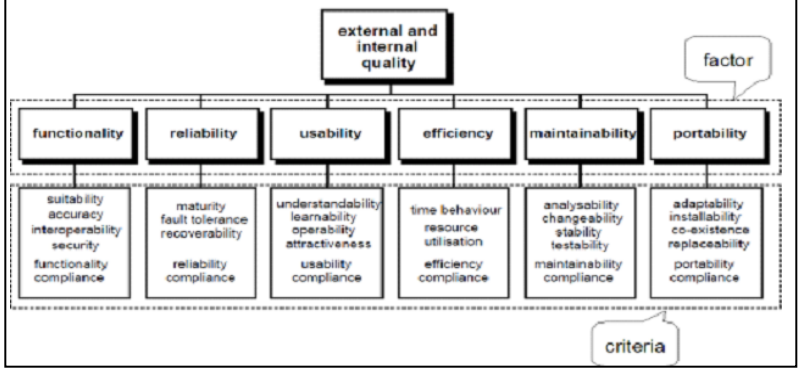
根据具体应用业务特征，基于测试用例执行框架所开发的用于优化用例代码结构，提高测试用例开发效率的测试支持框架

测试用例执行框架

基于特定平台/语言的测试用例执行框架，如JUnit等

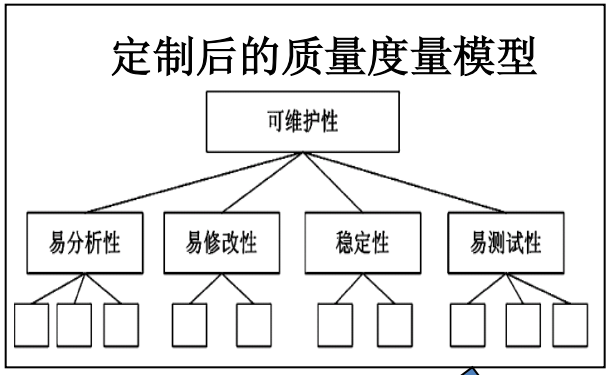
质量度量分析方法论

一般质量度量模型



定制

定制后的质量度量模型



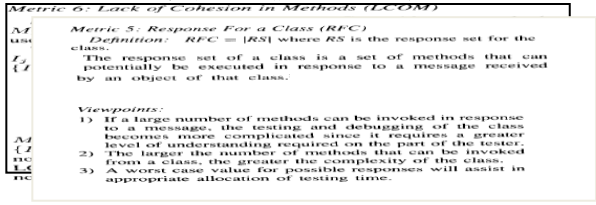
抽取

MCIF质量度量指标

Capability Improvement Metrics (Sample)

Value	Description
DEFECT_DENSITY	Total number of identified defects divided by the size of the system (number of use cases, SLOC, function points)
DEFECT_DENSITY_TARGET	Maximum Defect Density acceptable for the project
TEST_COVERAGE_OF_REQUIREMENTS	Percentage of requirements targeted for a given iteration that have associated test cases
MIN_TEST_COVERAGE_OF_REQUIREMENTS	Minimum Test Coverage of Requirements percentage acceptable for a given iteration
TESTS_PLANNED	Number of tests scheduled to be executed in the iteration.
TESTS_IMPLEMENTED	Number of tests built and ready to be executed (both manually and automatically) in the iteration
TESTS_ATTEMPTED	Number of tests that have been executed. The sum of all passed, failed and blocked tests
TESTS_FAILED_AND_BLOCKED	Number of tests that have a most recent result of failed or blocked
TESTS_FAILED_AND_BLOCKED_TOLERANCE	Acceptable number of tests that have a most recent result of failed or blocked

业界质量度量指标



Metric 6: Lack of Cohesion in Methods (LCOM)

Metric 5: Response For a Class (RFC)

Definition: $RFC = |RS|$ where RS is the response set for the class.

The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class.

Viewpoints:

1) If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding required on the part of the tester.
2) The larger the number of methods that can be invoked from a class, the greater the complexity of the class.
3) A worst case value for possible responses will assist in appropriate allocation of testing time.

质量度量体系

议程

- 开发中心质量管理面临挑战总结
  - 当前开发中心质量管理面临的主要挑战
  - 提高开发人员代码质量的最佳实践
  - 最佳实践实施面临的主要挑战
- 质量管理平台解决方案
  - 质量管理平台整体方案
  - 基于最佳实践经验的方法论
  - 质量管理平台实现
- 实施案例分享
  - 在国内某开发中心实施本方案的路线图
  - 实施过程体会

全自动化质量管理平台架构

IBM产品 定制开发 基于工具开发/配置 工具内置

Rational
代码分析工具集
(代码分析，单元测试，安全检查)

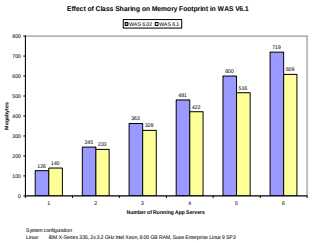
获取静态分析结果、单元测试结果、覆盖率信息

接口调用，获得抽象语法树

代码度量数据

度量数据

持续集成报告



Command调用

代码质量度量分析

数据获取

报告生成

持续集成平台 ***Build Forge***

Clear Case

工具集调用

源代码及配置文件获取

自动编译构建

搜寻测试服务器

自动部署

自动冒烟/系统测试

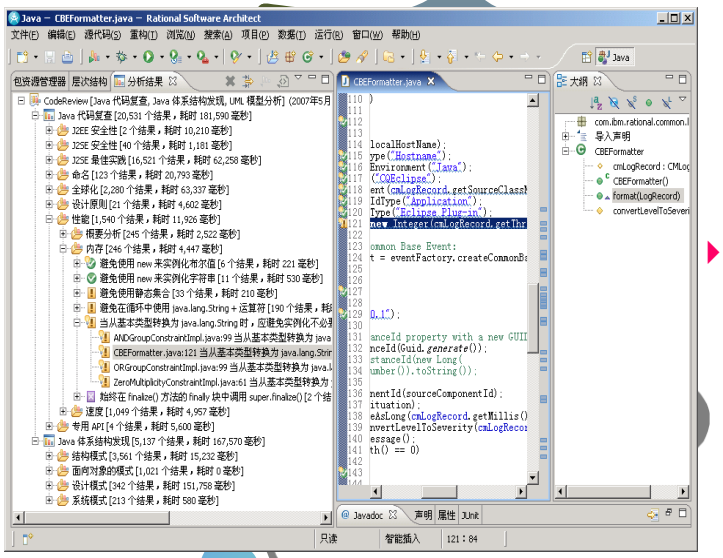
度量数据集成/报表生成

Rational代码质量分析工具集

- 静态代码检查——Rational Software Analyzer
  - 代码质量分析
    - 代码评审
    - 数据流分析
  - 代码复杂度分析
  - 代码结构分析
- 动态代码检查 for Java/J2EE ——Rational Application Developer
- 动态代码检查 for C/C++ ——Rational PurifyPlus
  - 内存分析
  - 覆盖率分析
  - 代码性能分析
- 代码安全检查——Rational AppScan Source

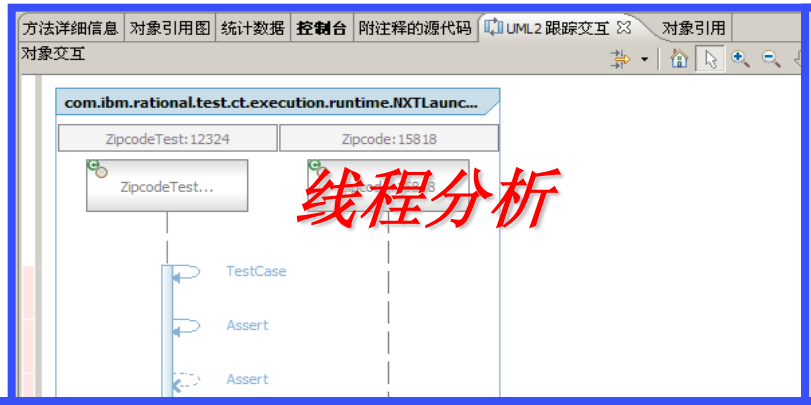
静态代码分析工具
Rational Software Analyzer

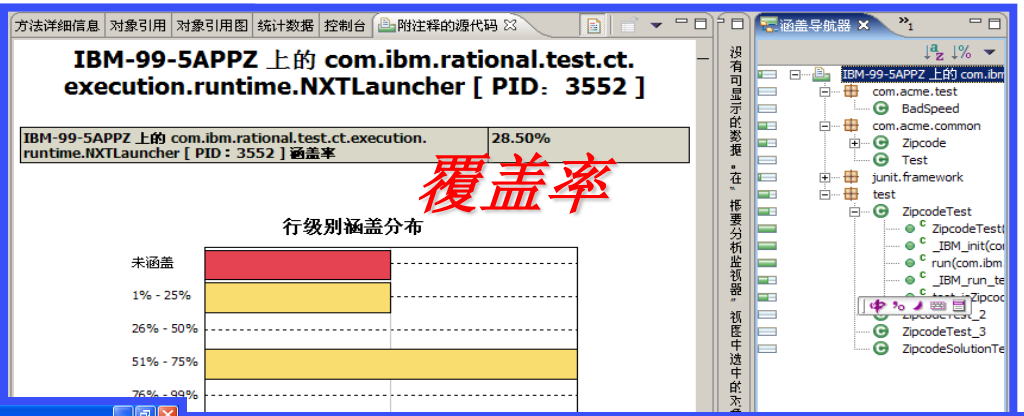
- **Code Quality**
  - **Basic code review**
    - **550+ Java rules, ~160 C/C++ rules**
    - **Globalization**
    - **Performance**
    - **Best Practices**
    - **Design Principles**
  - **Data flow analysis**
    - **~20 Java rules, 83 C/C++ rules**
    - **Resource leaks**
    - **Memory leaks**
    - **User Interface object abuse**

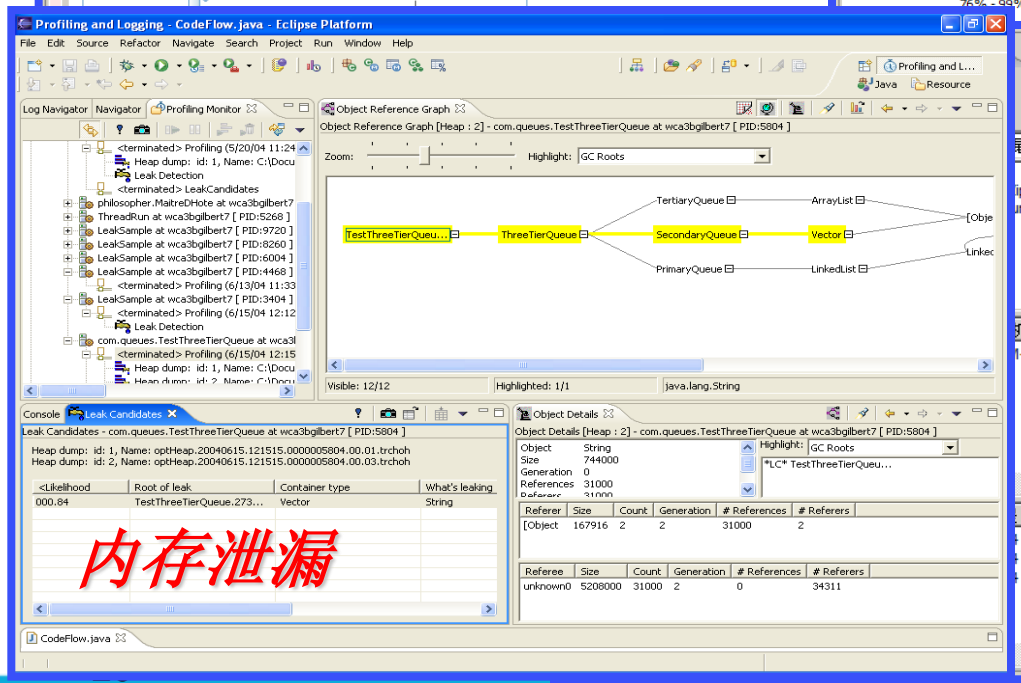


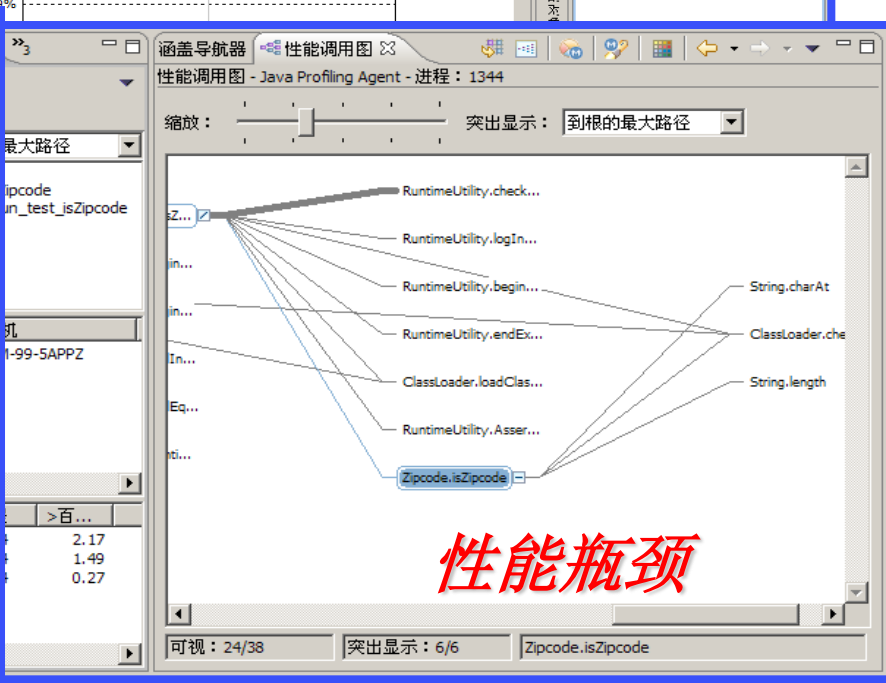
- **Code Complexity**
  - **40+ Java rules**
    - **Basic line counting metrics**
    - **Complexity/Maintainability metrics**
    - **Halstead metrics**
  - **16 C/C++ rules**
    - **Basic Metrics**
- **Code Structure - Architecture**
  - **~25 Java rules, ~25 C/C++ rules**
  - **Pattern Recognition Tool**
  - **Identify good coding patterns**
  - **Highlight bad coding anti-patterns**

动态代码分析工具 for Java/J2EE(RAD)



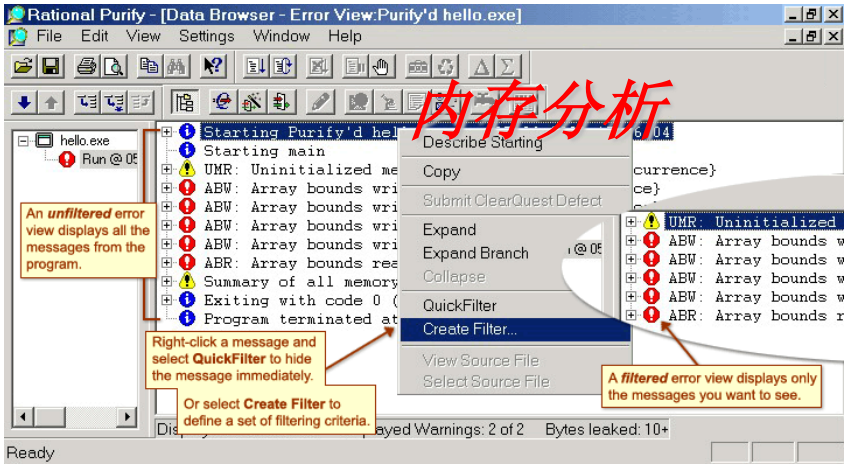


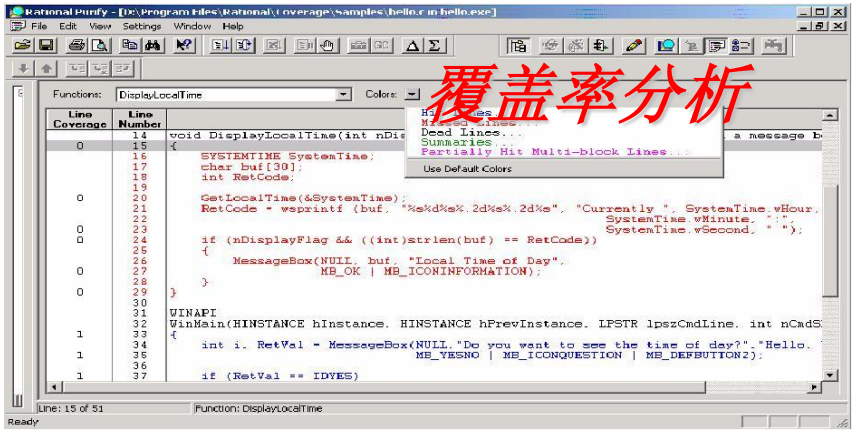


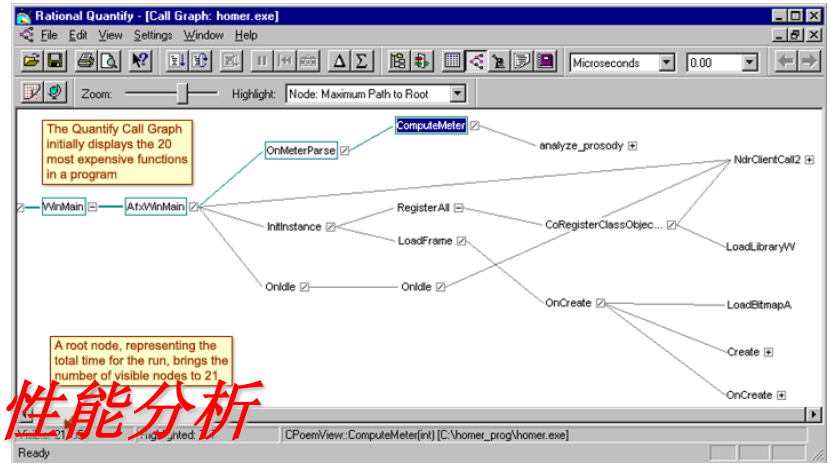


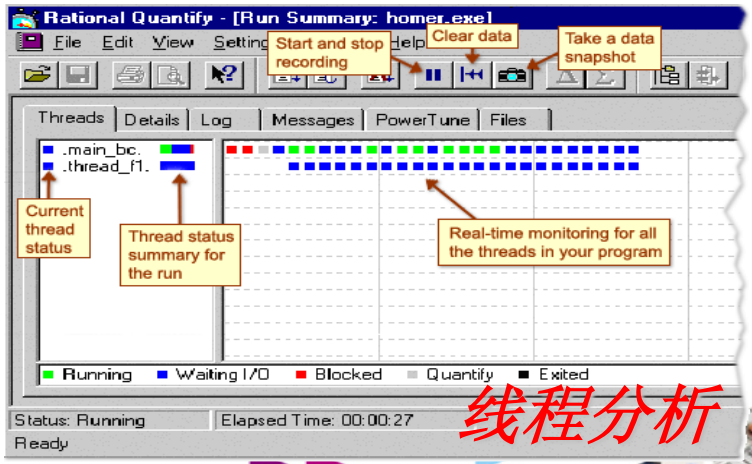
动态代码分析工具 for C/C++

Rational PurifyPlus



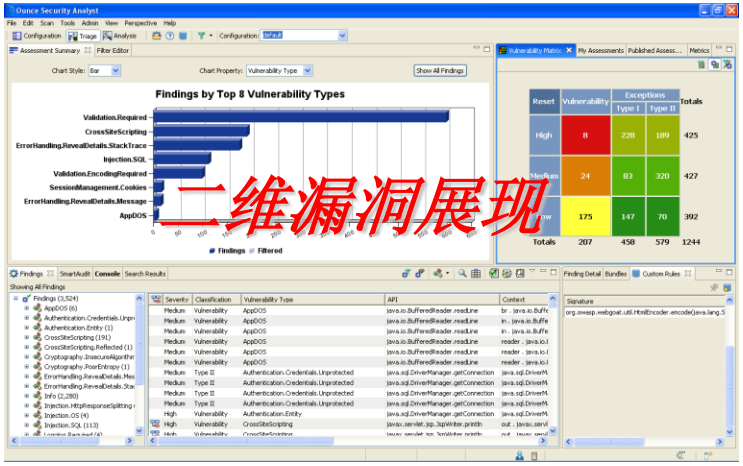




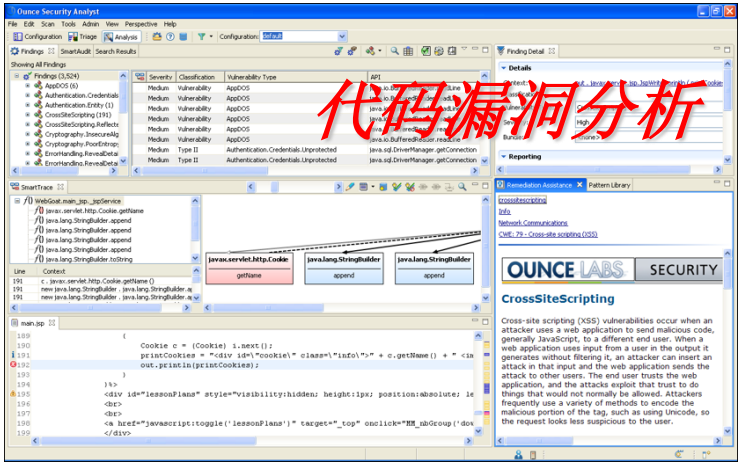


代码安全分析工具

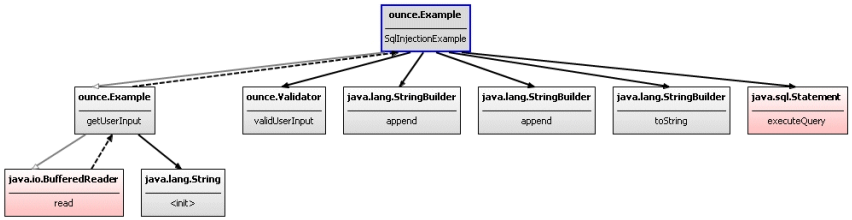
Rational AppScan Source



二维漏洞展现



代码漏洞分析



漏洞链路分析

代码级的修复建议

议程

- 开发中心质量管理面临挑战总结
  - 当前开发中心质量管理面临的主要挑战
  - 提高开发人员代码质量的最佳实践
  - 最佳实践实施面临的主要挑战
- 质量管理平台解决方案
  - 质量管理平台整体方案
  - 基于最佳实践经验的方法论
  - 质量管理平台实现
- 实施案例分享
  - 在国内某开发中心实施本方案的路线图
  - 实施过程体会

在国内某开发中心实施本方案的路线图

基于RAD和Junit的**单元测试框架**，该框架可根据配置信息自动生成桩模块和部分单元测试用例代码

单元测试框架研发，推广

好的单元测试框架可以很快地减少开发人员单元测试用例编写的工作量，收到立杆见影的效果。同时也可以培养开发人员单元测试的习惯。

基于RAD的**静态代码**分析，加入该开发中心自定义的静态代码规则，实现在客户端和持续集成平台上的全自动检查

基于RAD的静态代码分析实施

全自动化静态代码分析则可以极大的提高代码检查的效率

基于单元测试工具和静态代码分析工具，部署一个项目级的**持续集成平台**

项目级持续集成平台建设

基于静态分析工具，研发一个**代码质量度量平台**，并集成到项目级持续集成平台中

代码质量度量建设

根据前期的实施效果，做进一步的完善并实施企业级持续集成平台

…（单元测试框架、静态代码分析规则的完善、企业级持续集成平台等）

实施过程体会

- 开发人员质量管理平台建设不仅仅是个技术的问题，很大程度上还是个文化建设的问题
- 从文化建设角度
  - 在强大的进度压力下，开发人员都是非常现实的，只有真正让开发人员尝到好处，才能顺利推动单元测试、静态分析等的实施
    - 否则，任何绚丽的理论和工具终都会被开发人员所抛弃
  - 质量管理平台的真正落地有赖于整个组织树立正确的质量意识
    - 软件产品的质量不是测试部门测试出来的
    - 软件产品的质量归根到底还是依赖于开发人员自身
  - 方法论的支撑往往是至关重要的

实施过程体会

- 从技术的角度
  - 工欲善其事，必先利其器
    - 选择好的产品并按照正确的方式进行使用（如根据企业本身情况进行的定制等）往往是很关键的
    - 否则开发人员就会因为看不到好处或者反而降低了效率而不愿意使用
  - 一定要站在整个中心从长远的角度考虑问题，而不能仅从当前出发
    - 对于开源软件和商业软件的选择
    - 对于研发方式的选择（自研或者借力已有成熟产品、服务）

谢谢大家

欢迎提问

Innovate2011

IBM Rational 软件创新论坛

Software. Everyware.