软件测试

授课教师：张剑波

授课班级：111161-2班

2019年3月

Chapter2 软件测试技术

Q & A

- 软件测试的目的是 ____。

A.评价软件的质量　　B.发现软件的错误

C.找出软件中的所有错误　　D.证明软件是正确的

内容提要

- 静态测试
- 动态测试
  - 经典测试方法
  - 基于质量特征的测试
  - 基于经验的测试

案例引发的思考

- 某年某月某日，产品环境的2000多封自动发出的Email让我们项目组许多人的邮箱爆了。追查下来根源是一个很不起眼的缺陷。我们的程序对一个布尔值做了if(XXX = true)的判断，可来自上游系统的这个值不光是有true和false，还有空。也就是说上游系统中使用的是一个大布尔，是有true, false,null三态的， 而我们程序使用的是小布尔，只有true和false两态。
- 掉在大小布尔这个坑里也不是头一回了。记忆中前两年我们也掉进来过。有执著的QA一枚，翻箱倒柜地找，终于找到了当时的email，上面赫然写着“待改进事项：在PMD（我们采用的代码静态检查工具之一）中加入对大小布尔的提醒”。显然，这个事情后来没有被执行下去。。。既然代码静态检查是能够帮助我们解决这个问题的。但为什么后来没有做呢？执著的QA开始找人了解情况，积极反思。。。

原因分析

- 有的程序员说：“静态测试是有利于遵循编程规范啊，可我手上的代码大部分是别人留下来的，惨不忍睹啊！你是要我先把功能做出来，还是先把前面留下来的不规范代码都重新整理一遍？等我有空再说吧！”
- 有的程序员说：“等我有空了我也不想做。我觉得还不如研究一下新技术呢，代码能跑就行了，代码规范那体力活我可没有兴趣。”
- 有的团队能够意识到静态测试的价值，但尝试了一段时间发现难以坚持。主要原因是他们的静态测试太依赖于人来保障其执行。
- 实际工作中最难处理的就是遗留系统中的代码不符合规范的问题。
- 无论是人还是工具进行静态测试，都需要及时经常地进行。

静态测试的优势

- 由于静态测试发现的往往不是那种非常严重的缺陷，而只是特定情况下潜在的缺陷，或者只是维护时代码的可读性等问题，所以比起动态测试报告的对用户产生影响的缺陷，静态测试往往成为在面临压力下开发团队和个人比较容易先放弃的一个活动。
- 但大家可曾意识到大量动态测试暴露的缺陷，常常会归结于代码的可读性、可维护性、可测试性差、复杂度高、重复代码多等等。因此更改代码时容易引起一些副作用，所以越改越错、越错越多。
- 根据有关统计，在实际使用中代码检查比动态测试更有效率，能快速找到缺陷，发现30%～70%的逻辑设计和编码缺陷。与动态测试相比，静态测试具有发现缺陷早、降低返工成本、覆盖重点和发现缺陷的概率高的优点。

静态测试

- 评审
  - 评审过程
  - 角色和职责
  - 评审类型
- 静态分析
  - 控制流分析
  - 数据流分析
  - 编码标准一致性检查
- 静态测试工具介绍

（一）评审

- 评审的作用
  - 提高质量：尽早发现软件工作产品中的缺陷，通过修复缺陷直接提高软件产品的质量；同时尽早发现和修复缺陷，也可以减少将缺陷带入到下个阶段的机会，间接地提高质量。
  - 降低成本：缺陷发现和修复的成本随着开发阶段的演进而上升，因此尽早发现和修复缺陷可以直接降低成本；同时减少缺陷的雪崩效应也可以间接地降低成本；
  - 加快进度：在开发阶段的后期发现缺陷，不仅发现缺陷的难度增加，其发现的效率也将降低；
  - 提升能力：团队培训

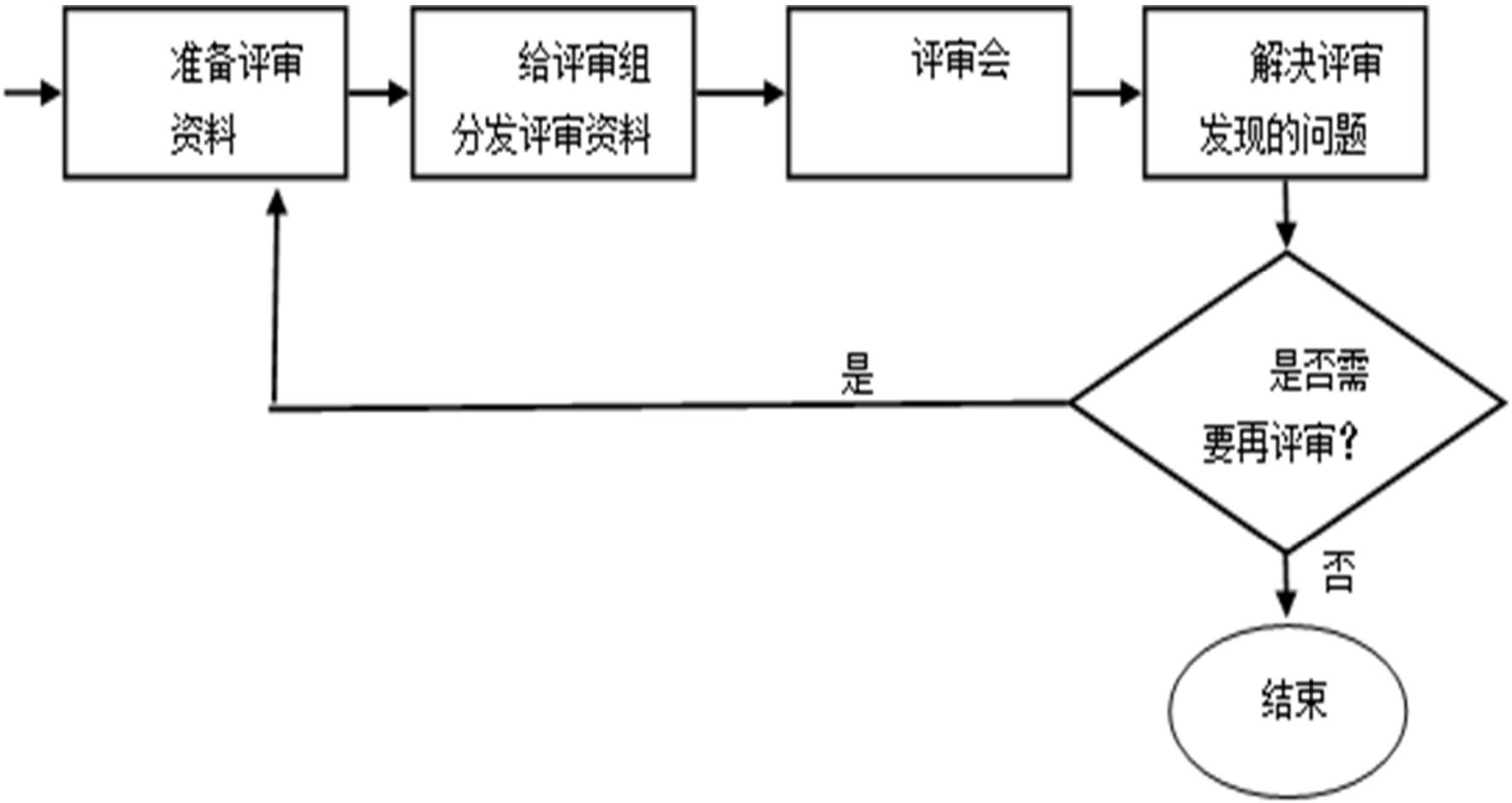
评审的基本原则

- 评审的基本原则
  - 评审对象是产品，讨论只针对事，不针对人。评审会只考虑问题是否是什么现象，不涉及负责人；
  - 注重评审效率；
  - 某阶段未通过阶段评审不得进入下一个软件研制阶段；
  - 评审就要挑刺，找问题、缺陷和隐患；
  - 评审组人员面越广越好；
  - 评审组不做无休止的争论和辩驳，将争论点记录下来，工艺后甄别；
  - 评审只是提出问题，没有解决问题的任务；
  - 使用“评审检查单”以提高评审的效果。

1. 评审过程

- 组建评审组
- 评审组长负责主持和控制全部评审活动
- 评审计划
- 评审准备
- 评审会
- 提交评审报告
- 建立评审过程

评审过程图



2. 参与评审的角色

- 参与评审会的角色
  - 主审员：协调本次审查并主持讨论。
  - 责任人：负责被审查的产品。
  - 讲解员：在审查会上讲解被审查的产品。
  - 审查员：审查产品。
  - 记录员：记录在审查会议上讨论的问题。
  - 产品经理：责任人的管理者。

评审角色的职责

- 主审员：具有专业技能和管理技能，一般由产品经理和责任人选择
  - 确保审查组成员用于审查的时间
  - 确保产品查经理了解审查工作
  - 制定审查会计划，安排资料、后勤的准备
  - 审查会前验收审查准备情况
  - 确保审查会高效、有序进行
  - 确保审查会上确定的问题文档化
  - 问题追踪直到解决
  - 审查会后完成会议记录和审查报告

评审角色的职责（续1）

- 责任人：准备要审查的信息或工作产品
  - 确保要审查的工作产品已就绪
  - 按时提供审查所需要的信息
  - 帮助主审员做好会议安排、资料准备、问题改正进度安排
  - 及时解决审查组确定的所有问题
  - 坚持客观性，避免辩解
  - 在审查会上阐明审查员不清楚的问题

评审角色的职责（续2）

- 讲解员：负责对被审的工作产品进行释义
  - 完全熟悉正在审查的工作产品
  - 确定信息的逻辑块并能解释每一个信息
- 审查员：寻找工作产品与所依据的文档、标准之间的差异，确定存在的问题
  - 完全熟悉要审查的工作产品
  - 完全熟悉审查一句的文档和标准
  - 鉴别工作产中存在的问题
  - 保持客观性
  - 对产品而不是责任人提出批评

评审角色的职责（续3）

- 记录员：在审查会上记录审查组确定的问题及其说明
  - 完全熟悉要审查的工作产品
  - 记录审查组提出的所有问题
  - 提供主审员要求的其他补充信息
- 产品经理
  - 帮助决定审查的内容
  - 将审查工作纳入项目计划
  - 分配审查资源
  - 保障审查培训工作
  - 参与主审员的选定工作
  - 支持主审员完成所要求的任何修改工作

3. 评审类型

- 需求评审
- 概要设计评审
- 详细设计评审
- 数据库设计评审
- 测试评审

1. 需求评审

- 是降低需求风险的一个重要手段
- 所有评审活动中最难且最容易忽视的一个
- 经常存在的问题：
  - 需求报告很长，短时间内评审者根本不能把需求报告读懂，想清楚。
  - 没有作好前期准备工作，需求评审的效率很低。
  - 需求评审的节奏无法控制。
  - 找不到合格的评审员，与会的评审员无法提出深入的问题。

对需求评审的建议

- 分层次评审：目标性、功能性、操作性
- 正式评审与非正式评审结合
- 分阶段评审：拆分为小规模的评审
- 精心挑选评审员：需方、供方
- 对评审员进行培训：领域专家、管理人员
- 充分利用需求评审检查单：QA、评审员
- 建立标准的评审流程：进入、材料、步骤等
- 做好评审后的跟踪工作：需求变更、复审、跟踪
- 充分准备评审：材料分发、进入条件；检查单

“软件需求规格说明”评审细则-1

- 是否清晰地定义了引用文档?
- 是否以软件配置项为单位，分别对各个软件配置项进行了需求分析?
- 是否对每个软件配置项的外部接口进行了清晰、完整的说明?
- 是否对每个软件配置项所包含的功能及其性能进行了适当的了解?
- 是否对每个软件功能的输入、输出进行了细致的说明?
- 是否对每个软件功能所对应的处理模型或处理流程(还包括容错处理模型、异常处理模型等)进行了详实的说明?

“软件需求规格说明”评审细则-2

- 对于安全关键软件，是否清晰地表示了软件必须处理的安全关键事件或危险事件?
- 软件配置项的功能是否满是软件研制任务书的要求或系统设计文档的要求?
- 软件需求分析的方法、使用的工具是否得当?
- 是否明确了对软件的非功能性要求?
- 是否明确提出了软件的安全性、可靠性要求?
- 是否明确提出了软件的易用性需求?
- 是否明确提出了软件的维护性需求?
- 是否明确提出了软件的可移植性需求?
- 是否明确了对软件的数据保密性、完整性要求?

“软件需求规格说明”评审细则-3

- 软件需求是否是可测试、可度量、可验证的?
- 是否具有需求追踪表，向上可追溯到“软件研制任务书”或“系统/子系统设计文档”?
- 所有需求是否都进行了明确定义，用例图覆盖了主要的用户功能需求?
- 用例图是否清楚说明了功能、角色、主事件流、异常事件流、前置条件、后置条件和非功能需求等内容?
- 对于业务流程，是否有详细的描述，包括处理机制、算法等?
- 文档编写是否规范?
- 文档描述是否正确、完整、一致?

2. 概要设计评审

- 一般应考察以下几个方面：
  - 概要设计说明书是否与软件需求说明书的要求一致?
  - 概要设计说明书是否正确、完整、一致?
  - 系统的模块划分是否合理?
  - 接口定义是否明确?
  - 文档是否符合有关标准规定?

概要设计评审细则-1

- 是否做到了以软件部件为基础进行软件体系结构的设计，即体系结构中的组成部分必须为实体部件?
- 软件体系结构是否优化、合理、稳健?
- 是否将软件需求规格说明中定义的功能、性能等全部都分配给了具体的软件部件?
- 为各个软件部件分配的功能、性能是否合理、和谐?
- 是否清晰、合理地定义了各个软件部件的接口?
- 软件部件的扇入、扇出是否符合要求(一般应控制在7以下)?
- 对于安全关键功能，是否采用了必要的设计策略?
- 是否说明了软件可靠性设计的具体措施?

概要设计评审细则-2

- 是否清晰、合理地设计了软件部件之间的协同关系，如同步、互斥、顺序等?
- 是否清晰、完整地列出了所有要求监督的软件工作过程，如软件需求分析、软件设计、配置项测试、配置管理等?
- 是否具体地策划了软件质量监督的监督节点?
- 是否建立了软件设计与软件需求的追踪表，审查分配给每个CSC(计算机软件部件)的功能或任务是否可追溯到“软件需求规格说明”或“接口需求和设计文档”?
- 逻辑视图对CSCI(计算机软件配置项)的层次分解是否合理?分解到最低层的包或类的粒度是否合适，即该包/类的后续实现者是否不必了解全系统的情况，中途换人是否不影响工作进度。

概要设计评审细则-3

- 分解的包/类是否涵盖了所有的功能需求?
- 主要用例的主流程是否用分解的设计元素进行描述?
- 是否为完成需求的功能增加了必要的包/类，使得层次分解的结果是一个完整的设计?
- 进程视图是否描述了所有主要的进程/线程，进程的生命期、功能、同步、通信等机制描述是否完备?
- 实现视图是否描述了CSCI的实现组成，每个组件是否分配了合适的需求功能，组件的表现形式(exe、d11或ocx等)是否合理?

概要设计评审细则-4

- 部署视图是否描述了CSCI的安装运行情况，能否对未来的运行景象形成明确概念?
- 文档编写是否规范?
- 文档描述是否正确、一致、完整?

3. 详细设计评审

- 一般应考察以下几个方面：
  - 详细设计说明书是否与概要设计说明书的要求一致?
  - 模块内部逻辑结构是否合理，模块之间的接口是否清晰?
  - 数据库设计说明书是否完全，是否正确反映详细设计说明书的要求?
  - 测试是否全面、合理?
  - 文档是否符合有关标准规定?

详细设计评审细则

- 是否将软件部件分解为软件单元?
- 是否对每个软件单元规定了程序设计语言所对应的处理流程?
- 对每个单元的入口、出口是否给予了清晰完整的设计?
- 软件单元之间的关系是否清晰完整?
- 文档编写是否规范?
- 文档描述是否正确、一致、完整?

4. 数据库设计评审

- 一般应考察以下几个方面：
  - 概念结构设计
  - 逻辑结构设计
  - 物理结构设计
  - 数据字典设计
  - 安全保密设计

数据库设计评审细则-1

- 是否进行了数据库系统模式设计、子模式设计以及物理设计?
- 数据的逻辑结构是否满足完备性要求?
- 数据的逻辑结构是否满足一致性要求?
- 数据的冗余度是否合理?
- 数据库的后备、恢复设计是否合理、有效?
- 对数据的存取控制是否满足数据的安全保密性要求?
- 对数据的存取控制是否满足实时性要求?
- 网络、通信设计是否合理、有效?

数据库设计评审细则-2

- 审计、控制设计是否合理?
- 屏幕设计、报表设计是否满是要求?
- 文件的组织方式和存取方法是否合理、有效?
- 数据安排是否合理、有效?
- 数据在存储介质上的分配是否合理、有效?
- 数据的压缩、分块是否合理、有效?
- 缓冲区的大小和管理是否满足要求?
- 文档编写是否规范?
- 文档描述是否正确、一致、完整?

5. 测试评审

- 评审内容
  - 软件测试需求规格说明
  - 软件测试计划
  - 软件测试说明
  - 软件测试报告
  - 软件测试记录

“软件测试需求规格说明”评审细则

- 测试依据是否完整、有效?
- 是否标识了所有被测对象?
- 是否提出了对被测对象的评价方法?
- 是否对被测对象的测试内容进行了适当的分类，即标识了测试类型?
- 对于每个测试项，是否提出了测试的充分性要求?
- 对于每个测试项，是否清晰地给出了追踪关系?
- 是否明确提出了测试项目的终止条件?
- 是否对软件单元提出了圈复杂度的测试要求?

“软件测试需求规格说明”评审细则

- 测试内容(测试项)是否覆盖了每个部件的所有外部接口?
- 测试内容(测试项)是否覆盖了配置项的所有功能和性能?
- 测试内容(测试项)是否覆盖了配置项的所有外部接口?
- 测试内容(测试项)是否涵盖了系统的所有功能和性能?
- 测试内容(测试项)是否涵盖了系统的所有外部接口?
- 文档描述是否正确、一致、完整?
- 文档编写是否规范?

“软件测试计划”评审细则-1

- 是否明确了测试组织与成员，并为每个成员合理地分配了责任?
- 测试人员是否具有相对的独立性?
- 测试人员的资质是否符合测试项目的要求?
- 测试依据是否完整、有效?
- 是否标识了所有被测对象?
- 是否提出了对被测对象的评价方法?
- 是否对被测对象的测试内容进行了适当的分类，即标识了测试类型?
- 对于每个测试项，是否提出了测试的充分性要求?

“软件测试计划”评审细则-2

- 对于每个测试项，是否提出了测试的终止条件?
- 对于每个测试项，是否清晰地给出了追踪关系(单元测试计划应追踪到详细设计文档，部件测试计划应追踪到设计文档或概要设计文档，配置项测试计划应追踪到软件需求规格说明，系统测试计划应追踪到系统设计说明)?
- 是否明确提出了测试环境要求，包括软件环境、硬件环境、测试工具等?
- 是否明确提出了测试项目的终止条件?
- 确定了测试进度，测试进度是否合理、可行?

“软件测试报告”评审细则

- 是否对测试过程进行了描述?
- 是否对测试用例的执行情况进行了描述?
- 是否对未执行的测试用例说明了未执行的原因?
- 测试报告结论是否客观?
- 文档编写是否规范?
- 文档描述是否正确、一致、完整?

“软件测试记录”评审细则

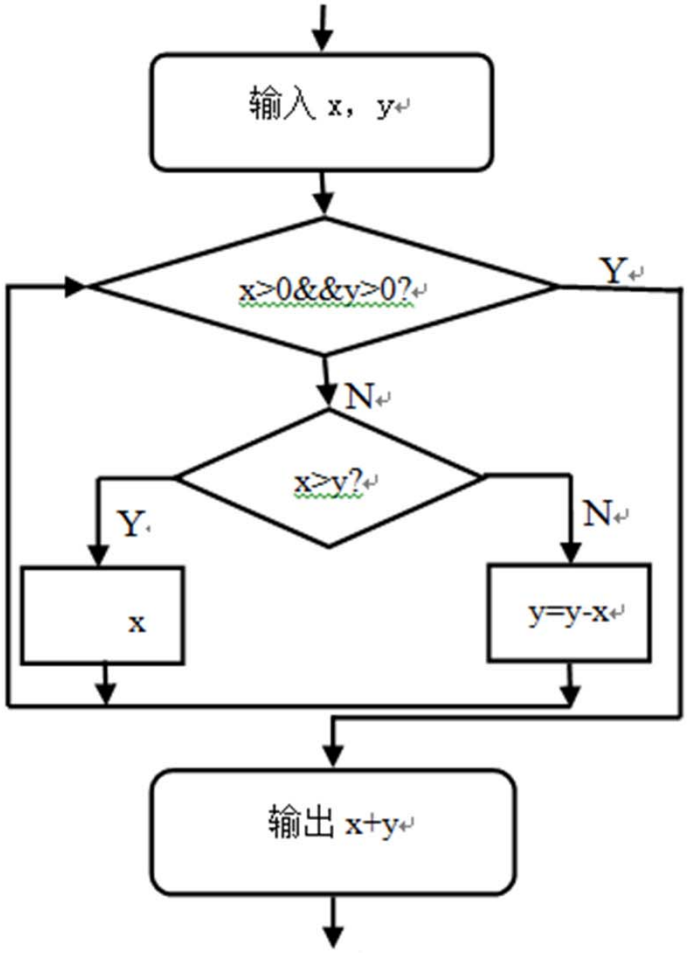
- 是否有测试人员签名?
- 测试用例执行结果描述是否充分、明确?
- 测试用例执行过程描述是否充分、明确?
- 文档编写是否规范?
- 文档描述是否正确、一致、完整?

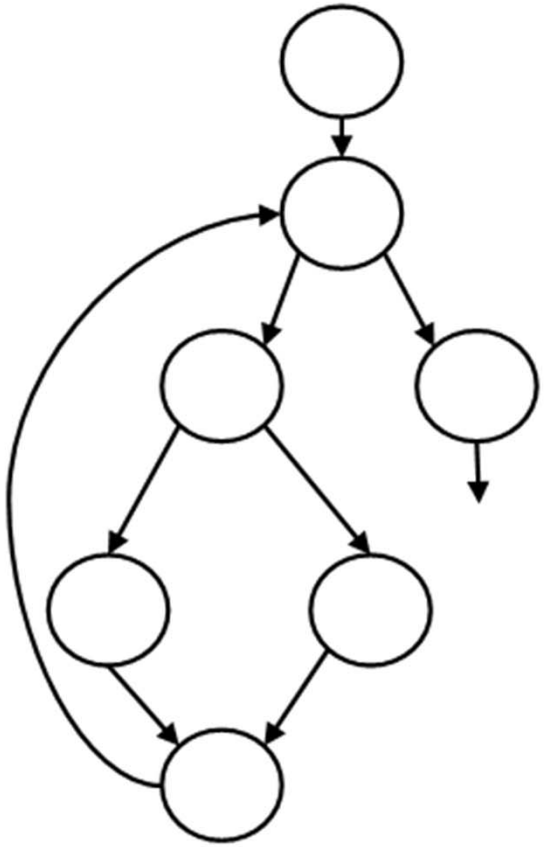
（二）静态分析

- 控制流分析
- 数据流分析
- 代码检查：编码标准一致性检查，主要检查方式
- 代码质量度量

1. 控制流分析

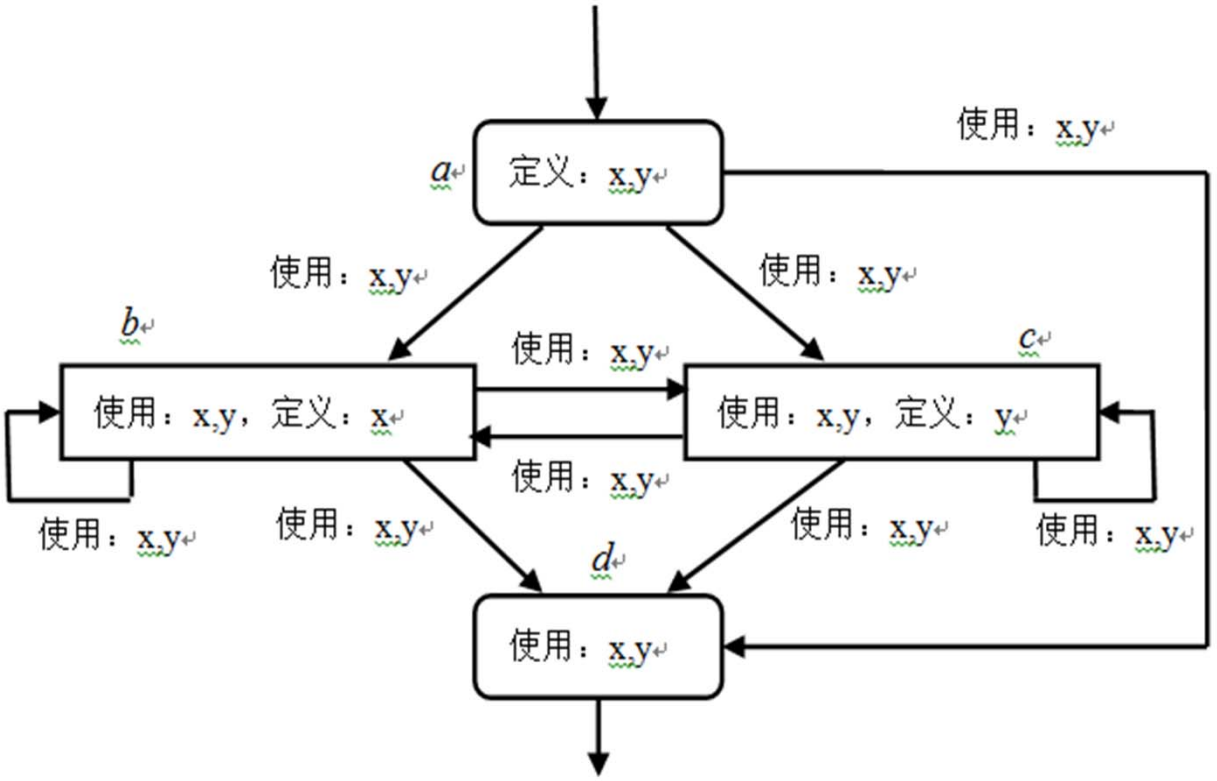
- 对用结构化程序语言书写的程序，可以通过使用“控制流覆盖规则”从程序推导出其对应的控制流图





2. 数据流分析

- 控制流测试是面向程序的结构，控制流图和测试覆盖准则一旦给定，即可产生测试用例
- 数据流测试是面向程序中的变量



程序插装

- 借助被测程序中插入操作来实现测试目的的方法。
  - 测试覆盖率
  - 测试用例的有效性度量
  - 断言检测：**Assertion**

3. 代码检查

- 代码检查是静态测试的主要方法， 主要包括：
  - 桌面检查
  - 代码走查
  - 代码审查
  - 静态结构分析：流程图审查
- 代码检查：主要检查代码和流图设计的一致性、代码结构的合理性、代码编写的标准性、可读性、代码的逻辑表达的正确性等方面。包括变量检查、命名和类型审查、程序逻辑审查、程序语法检查和程序结构检查等内容。

代码检查的目的

- 检查程序是不是按照某种标准或规范编写的。
- 发现程序缺陷。
- 发现程序产生的错误。
- 检查代码是不是流程图要求的。
- 检查有没有遗漏的项目。
- 使代码易于移植，因为代码经常需要在不同的硬件中运行，或者使用不同的编译器编译。
- 使代码易于阅读、理解和维护。

代码检查需要的文档

- 需求文档
- 程序设计文档
- 程序的源代码清单
- 代码编码标准
- 代码缺陷检查表
- 流程图等

（1）桌面检查

- 桌面检查是程序员对源程序代码进行分析、检验，并补充相关的文档，发现程序中的错误的过程。
- 由于程序员熟悉自己的程序，可以由程序员自己检查，这样可以节省很多时间，但要注意避免自己的主观判断。

桌面检查注意问题

- 检查代码和设计的一致性。
- 代码对标准的遵循、可读性。
- 代码逻辑表达的正确性。
- 代码结构的合理性。
- 程序编写与编写标准的符合性。
- 程序中不安全、不明确和模糊的部分。
- 编程风格问题等。

（2）代码走查

- 由程序员和测试员组成的审查小组，通过逻辑运行程序，发现问题。小组成员要提前阅读设计规格书、程序文本等相关文档，利用测试用例，使程序逻辑运行。
- 可分为以下两个步骤：
  - 小组负责人把材料发给每个组员，然后由小组成员提出发现的问题。
  - 通过记录，小组成员对程序逻辑及功能提出自己的疑问，开会探讨发现的问题和解决方法。

代码走查缺陷表

- 格式问题
- 入口和出口的连接
- 存储器问题：初始化
- 判断及转移
- 性能
- 可维护性
- 逻辑：设计是否实现
- 可靠性
- 内存设计
- 类的高级特性：继承、组合

（3）代码审查

- 由程序员和测试员组成的审查小组，通过阅读、讨论、分析技术对程序进行静态分析的过程。
- 代码审查可分为以下两步：
  - 第一步：小组负责人把程序文本、规范、相关要求、流程图及设计说明书发给每个成员。
  - 第二步：每个成员依据所发材料作为审查依据，但是由程序员讲解程序的结构、逻辑和源程序。在此过程中，小组成员可以提出自己的疑问；程序员在讲解自己的程序时，也能发现自己原来没有注意到的问题。

代码检查项目

- 主要有以下几点：
- 目录文件组织
- 检查函数
- 数据类型及变量
- 检查条件判断语句
- 检查循环体制
- 检查代码注释
- 其它检查

1）目录文件组织

- 目录文件组织要遵循以下原则：
  - 所有的文件名简单明了，见名知意。
  - 文件和模块分组清晰。
  - 每行代码在80个字符以内。
  - 每个文件只包含一个完整模块的代码。

2）检查函数

- 检查函数要遵循以下原则：
  - 函数头清晰地描述了函数的功能。
  - 函数的名字清晰地定义了它所要做的事情。
  - 各个参数的定义和排序遵循特定的顺序。
  - 所有的参数都要是有用的。
  - 函数参数接口关系清晰明了。
  - 函数所使用的算法要有说明。

3）数据类型及变量

- 数据类型及变量要遵循以下原则：
  - 每个数据类型都有其解释。
  - 每个数据类型都有正确的取值。
  - 数据结构尽量简单，降低复杂性。
  - 每一个变量的命名都明确地表示了其代表什么。
  - 所有的变量都要被使用。
  - 全部变量的描述要清晰。

4）检查条件判断语句

- 检查条件判断语句要遵循以下原则：
  - 条件检查和代码在程序中清晰表露。
  - **if/e1se**的使用正确。
  - 数字、字符和指针判断明确。
  - 最常见的情况优先判断。

5）检查循环体制

- 检查循环体制要遵循以下原则：
  - 任何循环不得为空。
  - 循环体系清晰易懂。
  - 当有明确的多次循环操作时使用如r循环。
  - 循环命名要有意义。
  - 循环终止条件清晰。

6）检查代码注释

- 检查代码注释时要遵循以下原则：
  - 有一个简单的关于代码结构的说明。
  - 每个文件和模块都要有相应的解释。
  - 源代码能够自我解释，并且易懂。
  - 每个代码的解释说明要明确地表达出代码的意义
  - 所有注释要具体、清晰。
  - 所有无用的代码及注释要删除。

7）其他检查

- 包括如下内容：
  - 软件的扩展字符、编码、兼容性、警告/提示信息。
  - 标准检查：检查程序中是否有违反标准的问题。
  - 风格检查：检查程序的设计风格。
  - 比较控制流：比较设计控制流图和实际程序生成的控制流图的差异。
  - 检查必须遵守规定代码的语法格式和规则。
  - …

（4）静态结构分析

- 静态结构分析主要是以图形的方式表现程序的内部结构，例如：函数调用关系图、函数内部控制流图。
- 通过应用程序各函数之间的调用关系展示了系统的结构。列出所有函数，用连线表示调用关系和作用。
- 静态结构主要分析：
- 1. 可以检查函数的调用关系是否正确；
- 2. 是否存在孤立的函数而没有被调用；
- 3. 明确函数被调用的频繁度，对调用频繁的函数可以重点检查。

（三）静态测试工具介绍

- 静态测试工具
  - **TrueTime：支持Visual C++、Visual Basic及Java程序语言**
  - **BoundsChecker: Visual C++开发环境所开发的程序代码的自动捕捉错误及调试工具**
  - **TrueCoverage**
  - **VectorCast**
  - **Rational SQA ROBOT**
  - **QARUN**

5个开源的代码审查工具

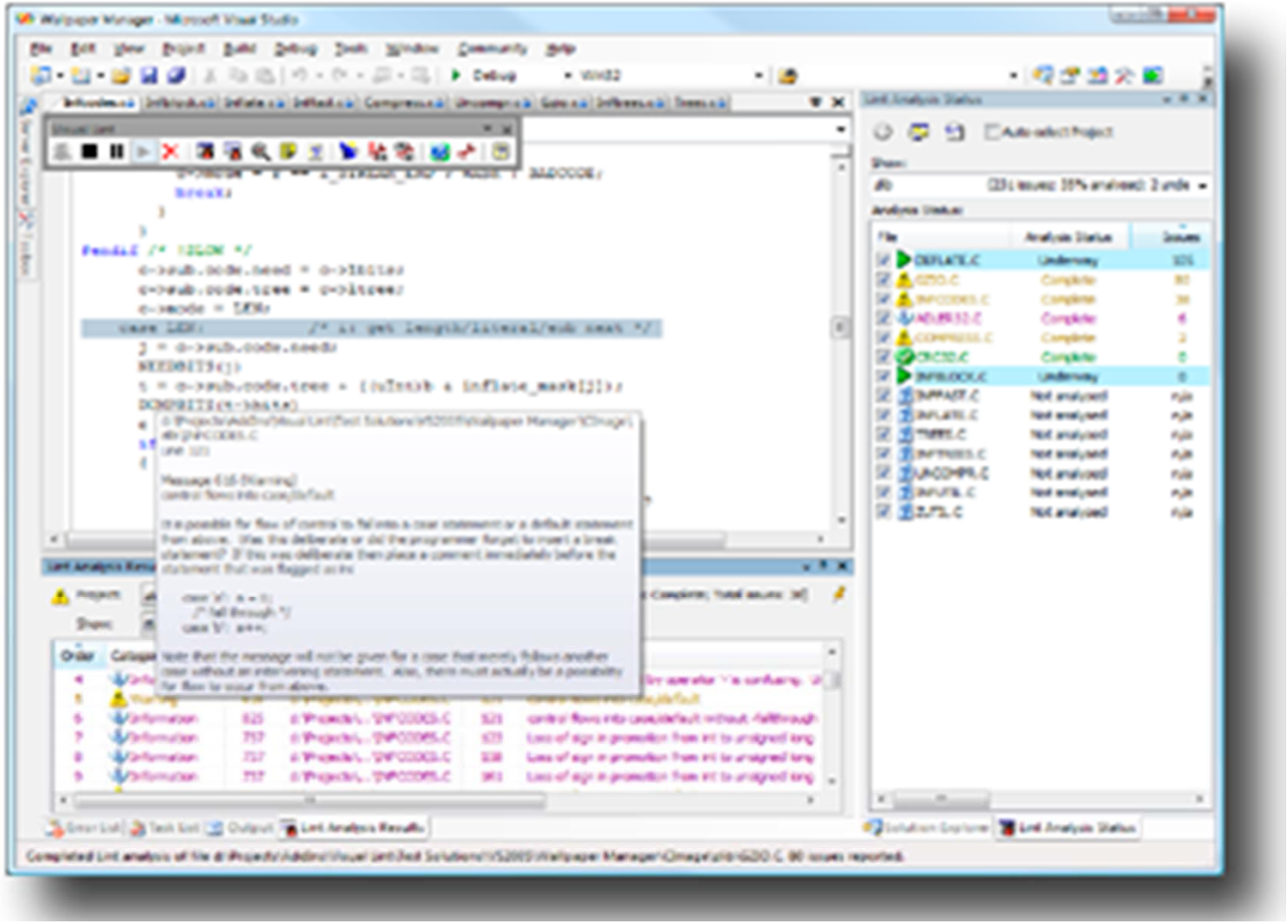
- **Jupiter:** 是集成在 **Eclipse** 下执行代码审查工作的插件。
- **Review board:** 是一个 基于**web** 的工具，主要设计给**diango** 和**python**的用户。
- **Codestriker :** 是一个基于**Web**的应用，其主要使用 **GCI-Perl** 脚本支持在线的代码审查。
- **Groogle:** 是一个基于**WEB**的代码评审工具。
- **JCR: Java Code Reviewer**
- **OneDev: https://onedev.io** 开源**Git**管理

PC-Lint

- http://www.gimpel.com/html/index.htm
- PC-Lint 是GIMPEL SOFTWARE 公司研发的C/C++软件代码静态分析工具，他的全称是 PC-Lint/FlexeLint for C/C++。
- PC-Lint 能够在Windows、MS-DOS 和OS/2 平台上使用，以二 进制可执行文档的形式发布，而FlexeLint 运行于其他平台，以源代码的形式发布。
- 大型的软件研发组织都把PC-Lint 检查作为代码走查的第一 道工序
- Visual Lint - Interactive Code Analysis
  - http://www.riverblade.co.uk/products/visual_lint/index.html

Visual Lint

- http://www.riverblade.co.uk/products/visual_lint/index.html

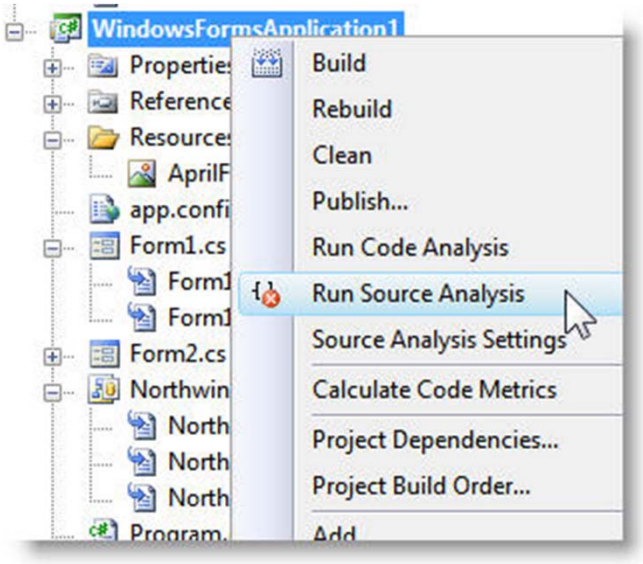


Microsoft Source Analysis for C#

- IDE

Source Analysis - 8

!	Description	File	Line
⊗	SA1633: The file has no header, the header Xml is invalid, or the header is not located at the top of the file.	Program.cs	1
⊗	SA1200: All using directives must be placed inside of the namespace.	Program.cs	1
⊗	SA1200: All using directives must be placed inside of the namespace.	Program.cs	2
⊗	SA1200: All using directives must be placed inside of the namespace.	Program.cs	3
⊗	SA1200: All using directives must be placed inside of the namespace.	Program.cs	4
⊗	SA1600: The class must have a documentation header.	Program.cs	8
⊗	SA1400: The class must have an access modifier.	Program.cs	8
⊗	SA1400: The method must have an access modifier.	Program.cs	14



PMD：Java程序代码检查工具

- looks for potential problems like
  - Possible bugs - empty try/catch/finally/switch statements
  - Dead code - unused local variables, parameters and private methods
  - Suboptimal code - wasteful String/StringBuffer usage
  - Overcomplicated expressions - unnecessary if statements, for loops that could be while loops
  - Duplicate code - copied/pasted code means copied/pasted bugs
- Eclipse
  - http://pmd.sf.net/eclipse

More Java Tools

- Checkstyle
  - http://checkstyle.sourceforge.net/
  - Eclise plugin
    - http://eclipse-cs.sourceforge.net/
    - http://www.mvmsoft.de/content/plugins/checkclipse/checkclipse.htm
- Jalopy
  - source code beautifier originally built by Marco Hunsicker and enhanced for Java 5 by Steve Heyns
  - http://jalopy.sourceforge.net/

静态测试方法小结

- 综合使用各种测试方法：
- （1）先用自动化工具来进行静态结构分析；
- （2）先从静态测试开始，如：静态结构分析、代码走查和静态质量度量，然后进行动态测试，如：覆盖率测试；
- （3）利用静态分析的结果作为依据，再使用代码检查和动态测试的方式对静态分析结果进行进一步确认，提高测试效率及准确性；
- （4）覆盖率测试是后期白盒测试中的重要手段，在测试报告中可以作为量化指标的依据，对于软件的重点模块，应使用多种覆盖率标准衡量代码的覆盖率；
- （5）在不同的测试阶段，测试的侧重点是不同的。

课堂练习

- 静态测试中的桌面检查是一种_____的检查方法。

A.程序员自己检查自己编写的程序

B.由同行帮忙检查自己编写的程序

C.几个同行自行组成小组，以小组为单位检查编写的程序

D.由程序员和测试员组成的审查小组，通过逻辑运行程序发现问题

本节小结

- 评审
  - 评审过程
  - 角色和职责
  - 评审类型
- 静态分析
  - 控制流分析
  - 数据流分析
  - 编码标准一致性检查