outline

- 编码
- 配置管理
- 质量管理
- 软件度量

概述

- 场景：Who is to blame?
  - 用户责怪开发人员，草率的开发过程导致低质量的软件
  - 开发人员责怪用户，不合理的交付期限、持续的需求变更、在未完成全部确认前要求交付软件
  - Who is right?
  - Both

概述

- 1960s，大型软件开发出现了质量问题
  - 交付的软件运行速度慢、不可靠、难于维护和复用
  - “Let’s Stop Wasting $78 Billion a Year”
  - “American businesses spend billions for software that doesn’t do what it’s supposed to do”
- 吸收制造业的质量管理技术，加上新型软件开发技术以及软件测试技术，提高软件质量
- 软件质量管理

质量管理原则

- 在组织层上，质量管理关注于建立标准组织过程
  - 质量管理小组负责定义该过程
- 在项目层上，质量管理关注于标准过程的实施
- 质量管理关注于建立项目的质量计划
  - 质量计划给出项目的质量目标
  - 定义项目使用的过程和标准

质量管理过程

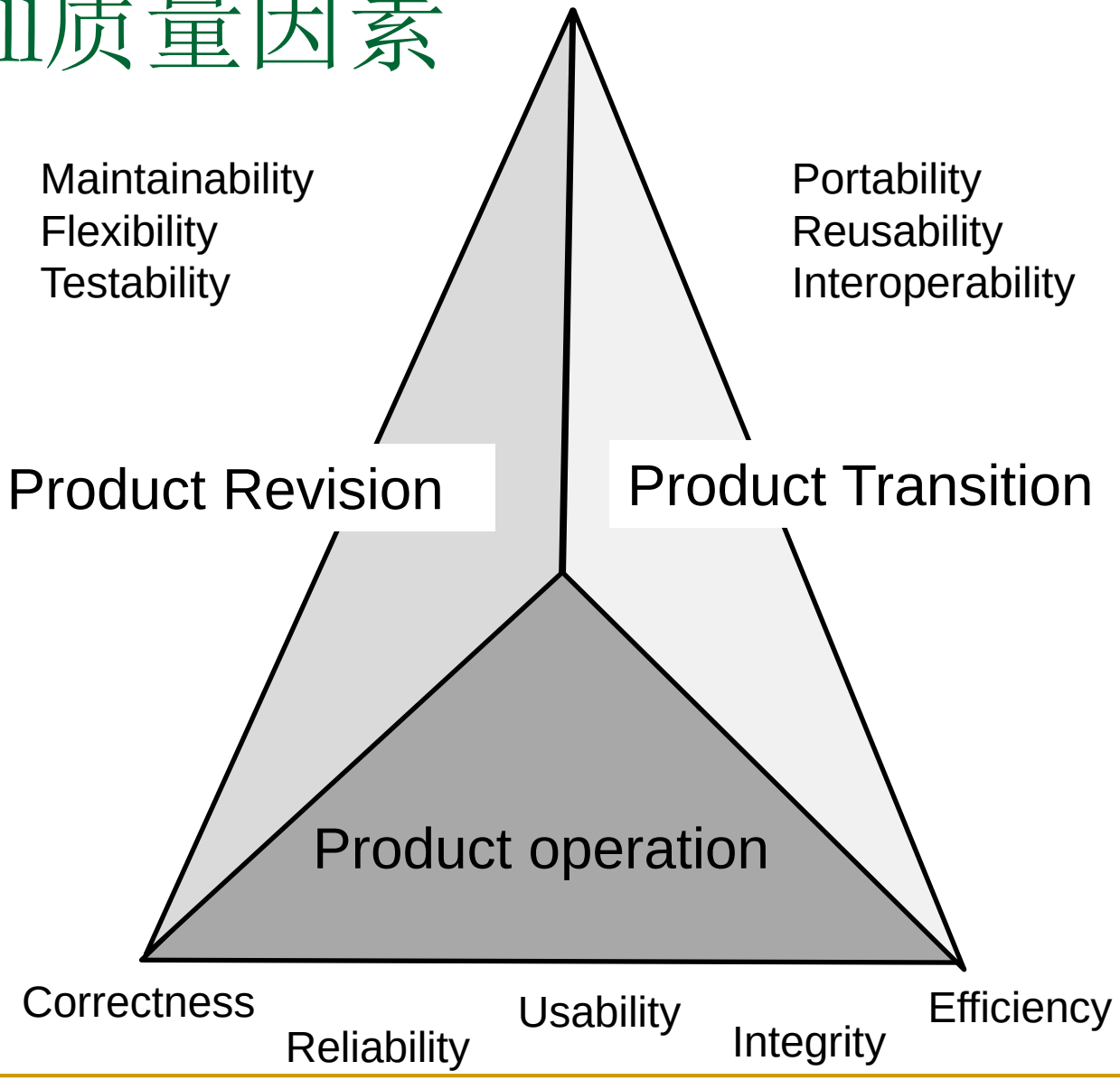
- 质量管理过程
  - 检测软件开发过程，确保工作产品符合组织的标准和目标
  - 质量管理小组独立于开发团队，直接向高层汇报

软件质量

An **effective software process** applied in a manner that creates a **useful product** that provides **measurable value** for those who produce it and those who use it.

—*The Business Value of Quality*
Bessin

McCall质量因素


Maintainability
Flexibility
Testability
Portability
Reusability
Interoperability
Product Revision
Product Transition
Product operation
Correctness
Reliability
Usability
Integrity
Efficiency

实现高质量软件的活动

- 软件工程方法
- 项目管理技术
- 质量控制
  - 评审模型，确保它们是完整、一致的
  - 检查代码，揭示并修复错误
  - 测试活动揭示处理逻辑、数据操作、接口通信的错误
  - 度量和反馈，调整软件过程，使得工作产品满足质量目标
- 质量保证
  - 质量管理

软件质量保证

Software quality assurance(SQA) is an umbrella activity that is applied throughout the software process.

SQA基本要素

- 标准
  - IEEE　ISO
  - SQA确保软件组织采纳和遵循这些标准
- 评审和审计
  - 技术评审是为了发现错误
  - 审计是SQA执行的一种评审方法
- 测试
  - SQA确保制定正确的测试计划，高效地执行测试
- 错误收集和分析
  - SQA收集和分析错误数据，以便更好地理解错误是如何引入的，以及如何消除这些错误

SQA基本要素

- 变更管理
- 教育
  - SQA牵头软件过程改进
- 安全管理
  - SQA确保使用正确的过程和技术，保证软件的安全性
- 风险管理
  - SQA确保正确执行风险管理

评审

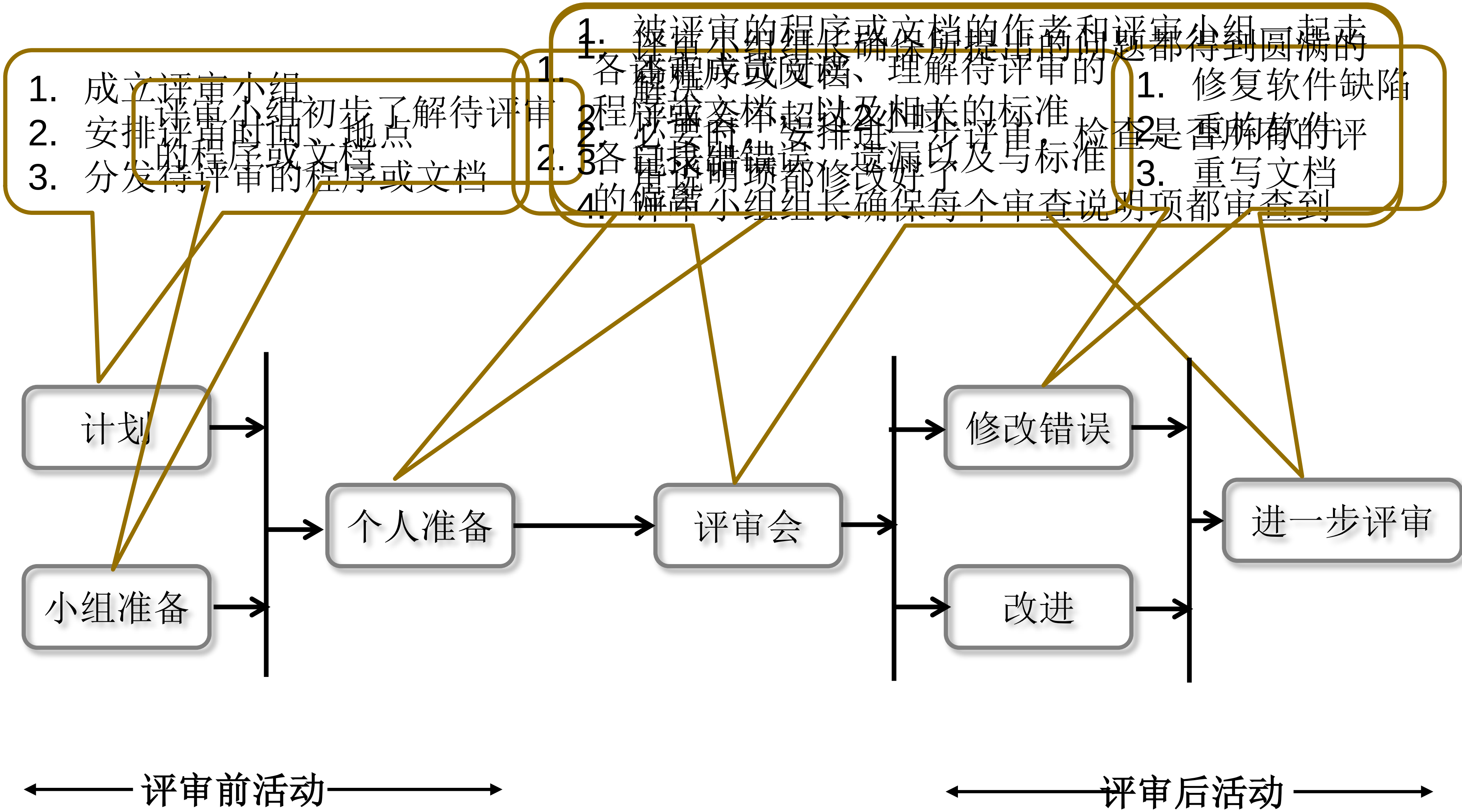
不管你有没有发现他们，缺陷总是存在，问题只是你最终发现它们时，需要多少纠正成本。评审的投入把质量成本从昂贵的、后期返工转变为早期的缺陷发现。

——卡尔·威格

技术评审（TR）

- 软件过程早期查错最有效的机制
  - 如果在早期发现错误，修改的成本就较少
- 非正式评审（informal review）
  - 桌面检查（desk check）
    - 与同事一起检查
    - 没有预先计划和准备、不开会
    - 检查清单
    - 少于1~2小时
    - 结对编程
- 正式技术评审（formal technical review, FTR）
  - 走查（walkthroughs）
  - 审查（inspections）

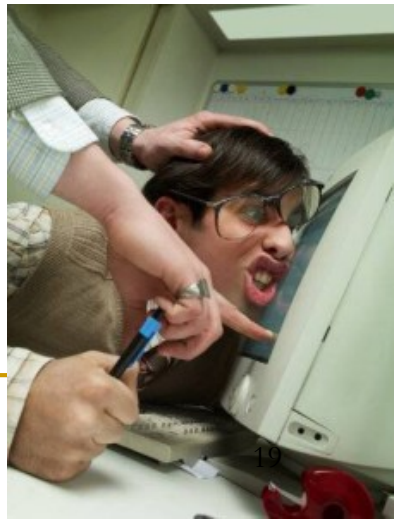
评审过程



1. 成立评审小组
2. 安排评审时间、地点
3. 分发待评审的程序或文档
评审小组初步了解待评审的程序或文档
1. 修复软件缺陷
2. 重构软件
3. 重写文档
计划
小组准备
个人准备
评审会
修改错误
改进
进一步评审
评审前活动
评审后活动

代码审查（code inspection）

- Capers Jones分析了超过12,000个软件开发项目，其中使用正式代码审查的项目，发现潜在缺陷率约在60-65%之间，若是非正式的代码审查，发现潜在缺陷率不到50%。大部分的测试，发现的潜在缺陷率会在30%左右。
- 一般的代码审查速度约是每小时200~400行代码



代码审查清单

Fault class	Inspection check
数据缺陷	• 是否所有变量在使用前都初始化了？ • 是否所有的常量都命名了？ • 数组的上界是否等于数组的长度或长度-1？ • 如果使用了字符串，分隔符是否已确定是哪种形式？ • 是否有可能出现缓存溢出？
控制缺陷	• 每个条件选择语句中的条件判断是否正确？ • 每个循环是否设置了长度和正确的终止条件？ • 复合语句是否正确的加上了括号？ • 在switch case语句中，是否所有可能的条件都考虑了？ • 在switch case语句中，是否每个case中都使用了break？
输入/输出缺陷	• 是否所有的输入变量都使用了？ • 是否所有的输出变量在输出前都赋值了？ • 所有输入是否都进行了检查（检测正确的类型，长度，格式和范围）？
异常管理缺陷	• 是否所有可能的错误情形都考虑到了？

代码审查清单

Fault class	Inspection check
接口缺陷	• 是否所有的函数和方法的调用，都传入了正确数量的参数？ • 是否所有的形参和实参的类型都匹配？ • 所有参数的顺序正确吗？
常规项	• 代码能够工作么？它有没有实现预期的功能，逻辑是否正确等。 • 所有的代码是否简单易懂？ • 代码符合你所遵循的编程规范么？通常包括大括号的位置，变量名和函数名，行的长度，缩进，格式和注释。 • 是否存在多余的或是重复的代码？ • 代码是否尽可能的模块化了？ • 是否有被注释掉的代码？ • 是否有可以被库函数替代的代码？ • 是否有可以删除的日志或调试代码？

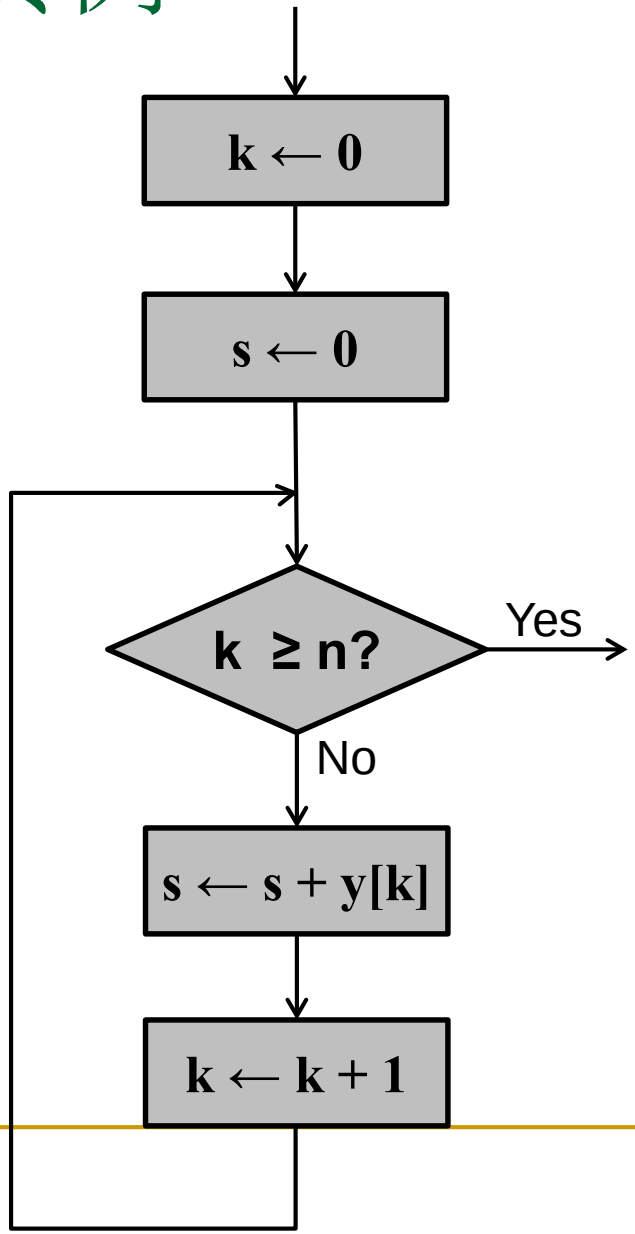
代码审查清单

Fault class	Inspection check
文档	• 是否有注释，并且描述了代码的意图？ • 所有的函数都有注释吗？ • 对非常规行为和边界情况处理是否有描述？ • 第三方库的使用和函数是否有文档？ • 数据结构和计量单位是否进行了解释？ • 是否有未完成的代码？如果是的话，是不是应该移除，或者用合适的标记（TODO）进行标记？

SQA措施

- 评审
- 软件测试
- **程序正确性证明（correctness proof）**
  - 证明程序能够完成预定的功能（满足规格说明）
  - Floyd不变式断言法
  - Hoare规则公理法
  - Dijkstra弱谓词变换方法
  - ……

实例

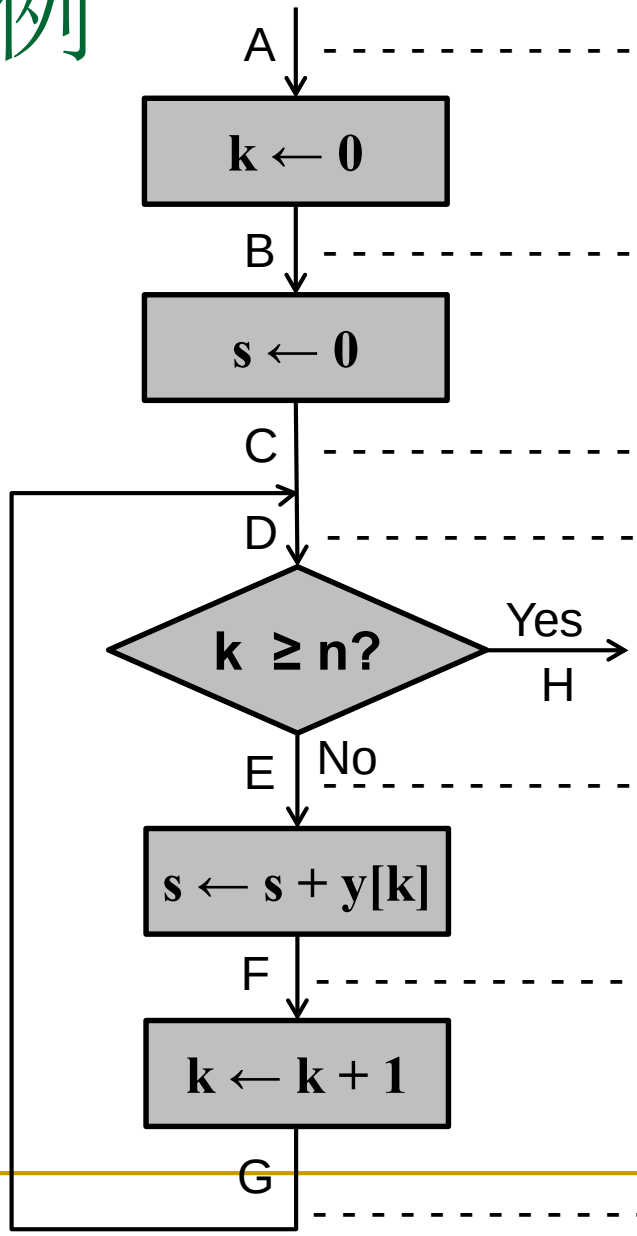



```
int k, s;
int y[n];
k = 0;
s = 0;
While (k < n)
{
  s = s + y[k];
  k = k + 1;
}
```

- 目标：证明这段代码的正确性
- 当代码执行后，变量*s*将被赋值为数组*y*中所有*n*个元素之和
  - 输入断言
  - 循环不变式
  - 输出断言

实例

（（输入断言为真→输出断言为真）→程序正确）



A - - - - - - - - - - - - - - - - - - - **n ϵ {1, 2, 3, …}**

（输入断言）

B - - - - - - - - - - - - - - - - - - - k = 0

C - - - - - - - - - - - - - - - - - - - k = 0, s = 0

（循环不变式）

D - - - - - - - - - - - - - - - - - - - **k ≤ n, and s = y[0]+y[1]+⋯+y[k-1]**

Yes → **k = n, and s = y[0]+y[1]+⋯+y[n-1]**（输出断言）

E - - - - - - - - - - - - - - - - - - - $k_0 < n$, and s = y[0]+y[1]+⋯+y[k_0-1]

F - - - - - - - - - - - - - - - - - - - $k_0 < n$, and s = y[0]+y[1]+⋯+y[k_0-1]+y[k_0]

G - - - - - - - - - - - - - - - - - - - $k_0 \leq n$, and s = y[0]+y[1]+⋯+y[k_0-1]

证明过程小结

- 给定输入断言
- 证明无论循环执行多少次，循环不定式都成立
- 进一步证明n次迭代后，循环终止，并且k和s的值都满足输出断言
- 因此，程序正确