第十四章 质量概念

王美红

主要内容

- 什么是软件质量
- 产品度量框架

14.1 什么是质量

- 现如今软件质量仍然是一个问题，每年耗费了大量的资金
- 客户责备开发人员，认为粗心的实践导致低质量的软件。开发人员责备客户，认为不合理的交工日期以及连续不断的变更使开发人员在还没有完全验证时就交付了软件。这都是问题所在。

14.1 什么是质量

- 质量是什么，不好清楚的定义
- 质量涵盖很多观点
- 在最一般的意义上，软件质量可以这样定义：
  - 在一定程度上应用有效的软件过程，创造有用的产品，为生产者和使用者提供明显的价值。

14.1 什么是质量

- 教材强调三个重要的方面
  - 有效的软件过程为生产高质量的软件产品奠定了基础
  - 有用的产品是指交付最终用户要求的内容、功能和特性，但最重要的是，以可靠的、无误的方式交付这些东西
  - 通过为软件的生产者和使用者增值：软件生产者在维护、改错及客户支持方面工作量降低；用户加快业务流程。

14.1 什么是质量

- 软件质量是对明确*陈述的功能和性能需求、明确记录的开发标准以及对所有专业化软件开发应具备的隐含特征的符合度。*

- Gavin的质量维度
  - Gavin建议采用多维的观点考虑质量：
    - 性能质量—内容、功能和特性是否交付
    - 特性质量—用户惊喜特性是否提供
    - 可靠性—所有的特性和能力是否提供
    - 符合性—标准、编码惯例
    - 耐久性—是否能对软件进行维护和改正
    - 适用性—可否短时间内完成维护和改正
    - 审美—难量化，但不可缺
    - 感知—偏见或好的声誉

14.1 什么是质量

McCall 的质量因素

- 影响软件质量的因素可以分为两大类：
  - 可以直接测量的因素（如：测试期间发现的错误）
  - 只能间接测量的因素（如：易用性和可维护性）
- 所有情况下，度量都必须发生。

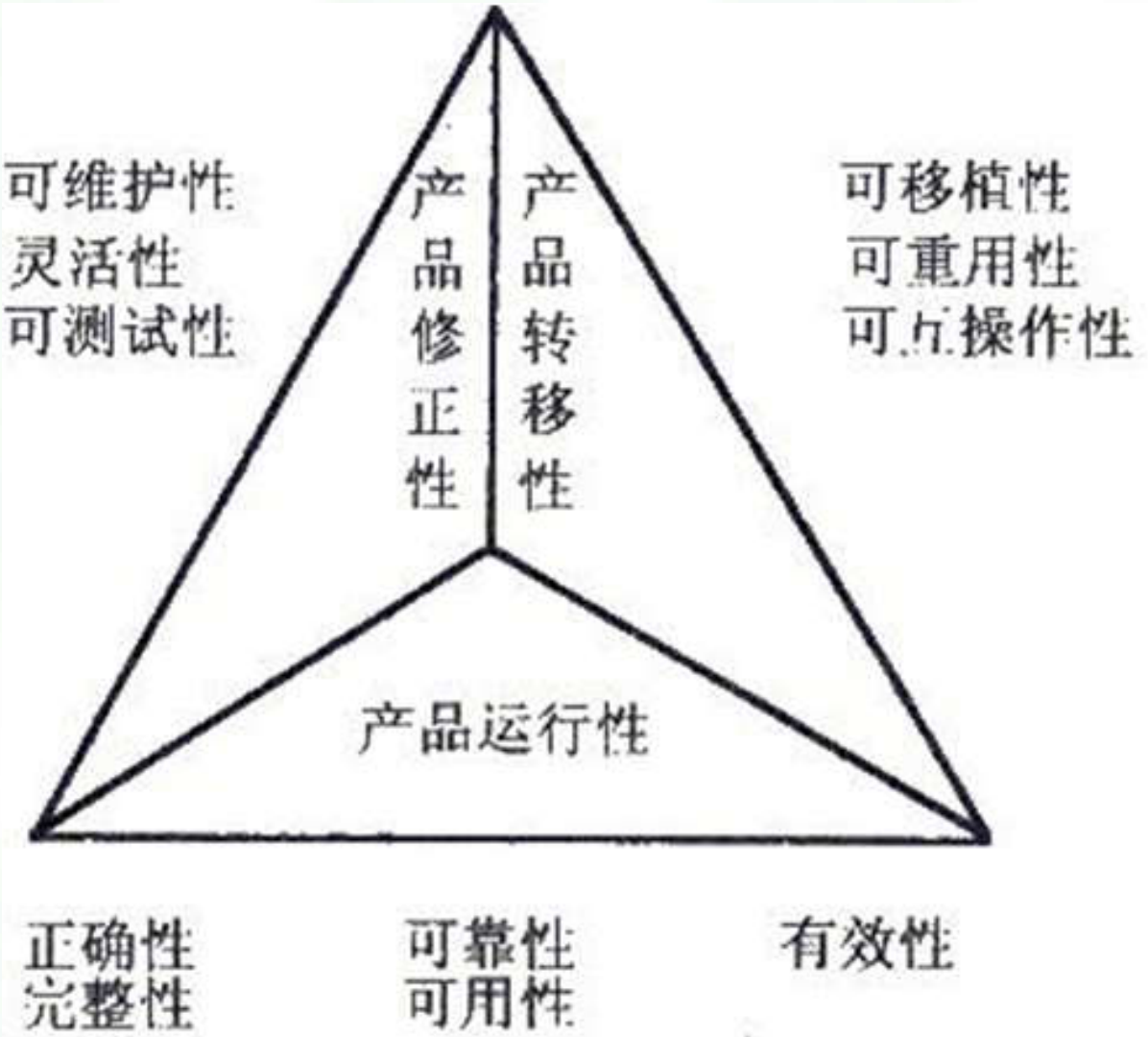


图　　McCall的软件质量要素

McCall、Richards 和 Walters对软件质量的分类：

- 正确性：程序满足其需求规格说明和完成用户目标的程度。
- 可靠性：期望程序以所要求的精度完成其预期功能的程度。
- 效率：程序完成其功能所需计算资源和代码的数量
- 完整性：对为授权的人员访问软件或数据的可控程度。
- 易用性：对程序学习、操作、准备输入和解释输出所需要的工作量。
- 可维护性：定位和修复程序中的一个错误所需要的工作量。
- 灵活性：修改一个运行的程序所需的工作量。
- 可测试性：测试程序以确保它能完成预期功能所需要的工作量。
- 可移植性：将程序从一个硬件和软件系统环境移动到另一个所需要的工作量。
- 可复用性：程序（或程序的一部分）可以在另一个程序中使用的程度。
- 可操作性：将一个系统连接到另一个系统所需要的工作量。

14.1 什么是质量

- ISO质量因素：
  - ISO 9126标准标识了六个关键的质量属性：
    - 功能性，子属性：适应性、准确性、互操作性、依从性和安全性
    - 可靠性，子属性：成熟性、容错性和可恢复性
    - 易用性，子属性：可理解性、易学习性和可操作性
    - 效率，子属性：时间表现和资源表现
    - 可维护性，子属性：可分析性、可修改性、稳定性和可测试性
    - 可移植性，子属性：适应性、可安装性和可替代性。

14.1 什么是质量

- 定向质量因素
  - 软件团队可以提出一套质量特征和相关的问题以调查满足每个质量因素的程度。
  - 为了进行评价，需要说明白具体的、可测量的属性——细化。

- 直觉
  - 界面布局易于理解吗？
  - 界面操作容易找到和上手吗？
  - 界面使用了可识别的隐喻吗？
  - 输入安排地节约敲击键盘和点击鼠标吗？
  - 界面符合 3 个重要原则吗？
  - 美学的运用有助于理解和使用吗？

- 效率：
  - 界面的布局和风格可以使用户有效地找到操作和信息吗？
  - 一连串的操作（或数据输入）可以用简单动作达到吗？
  - 输出的数据和显示的内容能被立即理解吗？
  - 分层操作是否组织得能使用户完成某项工作所需导航的深度更小？

- 健壮性：
  - 如果输入了规定边界上的数据或恰好在规定边界外的数据，软件能识别出错误吗？更为重要的是，软件还能继续运行而不出错或性能不下降吗？
  - 界面能识别出常见的可识别的错误或操作操作，并能清晰地指导用户回到正确的轨道上来吗？
  - 当发现了错误的情况（与软件功能有关），界面是否提供有用的诊断或指导？

- 丰富性
  - 界面是否能按照用户的特定要求进行客户化？
  - 界面是否提供宏操作以使用户将单个的行为或命令当做一连串的常用操作？

14.1 什么是质量

- 过渡到量化观点
  - 我们可以提出一组应用于软件质量评估的软件度量。在所有的情况下，这些度量表间接的测度（从不真正测量质量，而是测量质量的一些表现）

14.2 软件质量困境

- “足够好”的软件？
  - 成本高，可能错失市场机会
  - “足够好”可以起作用，但只是对于少数几个公司，而且只是在有限的几个应用领域。

14.2 软件质量困境

- 质量成本
  - 表面上看，质量是有成本的，但是缺乏质量也要成本。
  - 既要了解实现质量的成本，也要了解低质量软件的成本。

    为了了解这些费用，一个组织必须收集度量数据，为目前的质量成本提供一个基准

14.2 软件质量困境

- 质量成本可以分为：
  - 预防成本—管理、技术活动、测试、培训
  - 评估成本—技术评审、收集、估算、测试和调试
  - 失效成本—内部失效成本和外部失效成本

14.2 软件质量困境

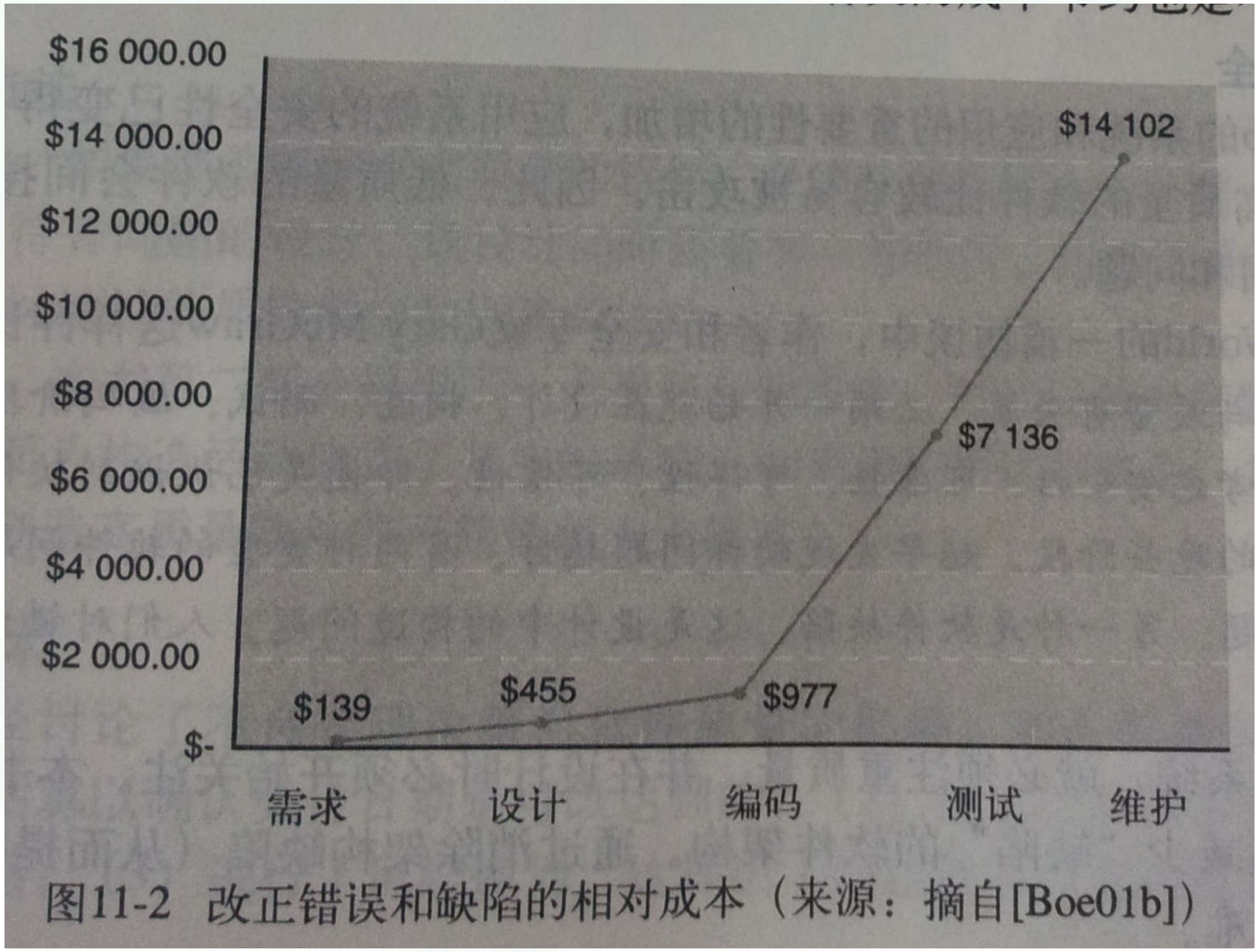


图11-2 改正错误和缺陷的相对成本（来源：摘自[Boe01b]）

- 风险--质量低劣导致风险，其中一些非常严重
- 疏忽和责任—到了交付时，互相抱怨
- 质量和安全--没有表现出高质量的软件比较容易被攻击
- 管理活动的影响—决策对软件质量有重大影响
- 估算决策—如果交付日期不合理，坚持立场很重要
- 进度安排决策—构件之间有先后依赖关系，造成有些缺陷发现晚，影响质量
- 面向风险的决策—当风险变成现实，质量水平必然下降

14.3 实现软件质量

1. 软件工程方法：采用适当的分析和设计方法
2. 项目管理技术：进度管理、风险规划、变更管理等
3. 机器学习和缺陷预测：基于统计、机器学习优化分类算法
4. 质量控制：如检查代码、一系列的测试步骤、测量等
5. 质量保证：如质量保证还包含审核和报告功能，用以评估质量控制过程的有效性和完整性

14.4 微软软件质量测试常用度量

1. 产品设计规范(Spec或设计文档)质量状态
2. 缺陷（bug）数据有关度量
3. 测试案例度量
4. 测试规范度量
5. 测试过的系统数量
6. 自动化测试度量
7. CodeCoverage（代码覆盖）
8. 单一功能测试验收质量度量

1.产品设计规范质量状态分类

- 常用的五种状态
  - 一页 (One page)– 用于安排时间和分配人员
  - 草稿 (Draft)– 用于提出疑问和初步设想以供讨论
  - 审阅 (Review)–有所有的设计技术细节，可以供审阅
  - 提交审核会 (Inspection)–所有的设计技术细节到位、没有明显遗留疑问、漏洞等
  - 开始编码 (Coding)– 开发人员可以开始编写代码来实现该设计功能规范

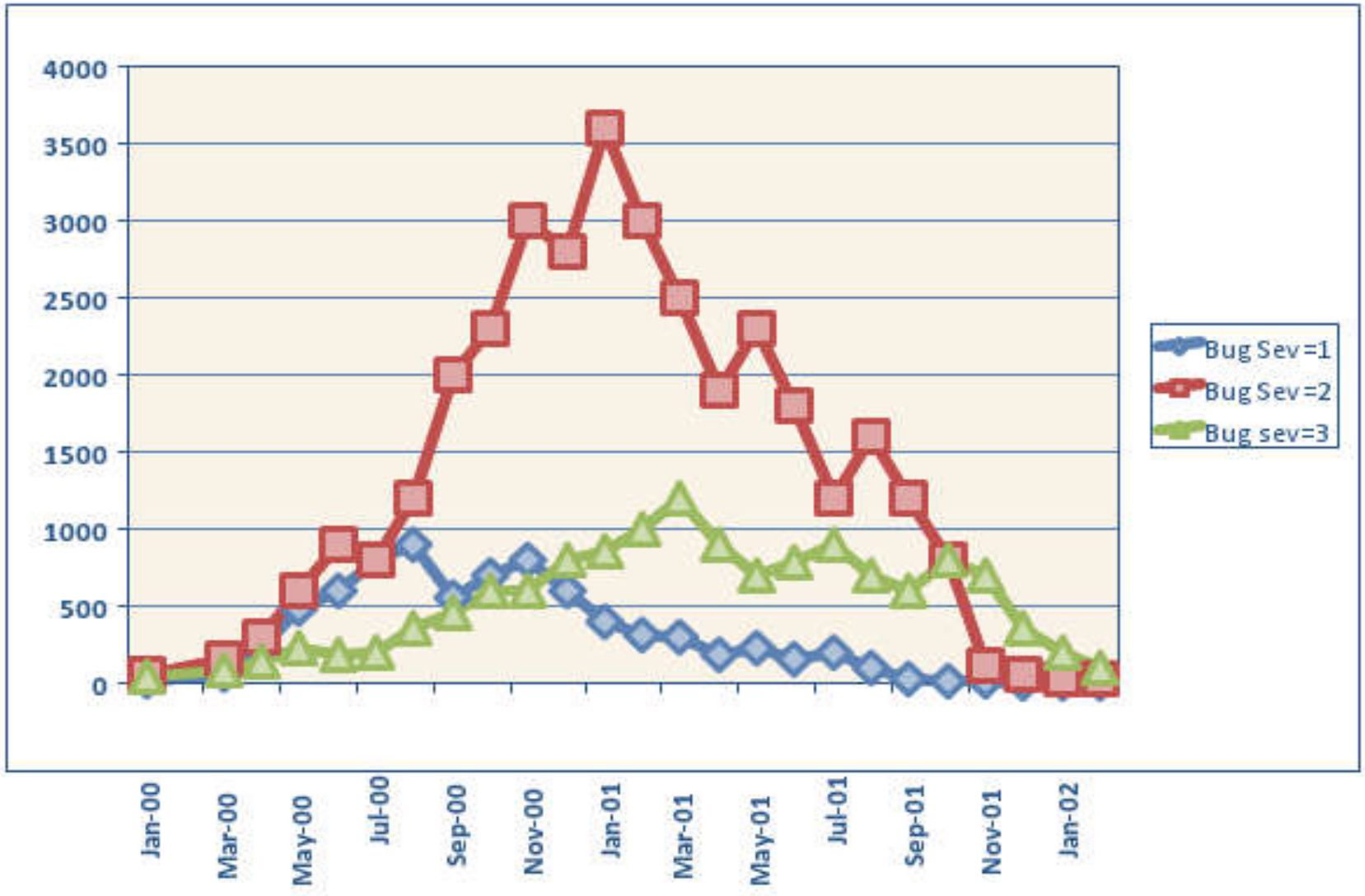
产品设计规范质量到位状况

- 五种状态中各占的%是多少?
- 按事先计划日程完成的比例是多少?
- 多少%开发人员至少有一个指派给他的功能可以进行编码?

2.缺陷统计数据的度量

- 所有缺陷数量的时间走势或趋势统计 (Bug Trends By Time)
- 未被处理的缺陷按照严重程度的统计 (Active Bugs By Severity)
- 未被处理的缺陷按照优先程度的统计 (Active Bugs By Priority)
- 未被处理的缺陷数量的时间走势或趋势统计 （Active Bugs Over Time）
- 所有的缺陷按照严重程度的统计(All Bugs By Severity)
- 新被发现的缺陷按严重程度的统计 (Opened Bugs By Severity)
- 已处理的缺陷按照严重程度的统计 (Resolved Bugs By Severity)
- 被修复的缺陷按照严重程度的统计 (Fixed By Severity)

所有的缺陷按照严重程度的统计(All Bugs By Severity)



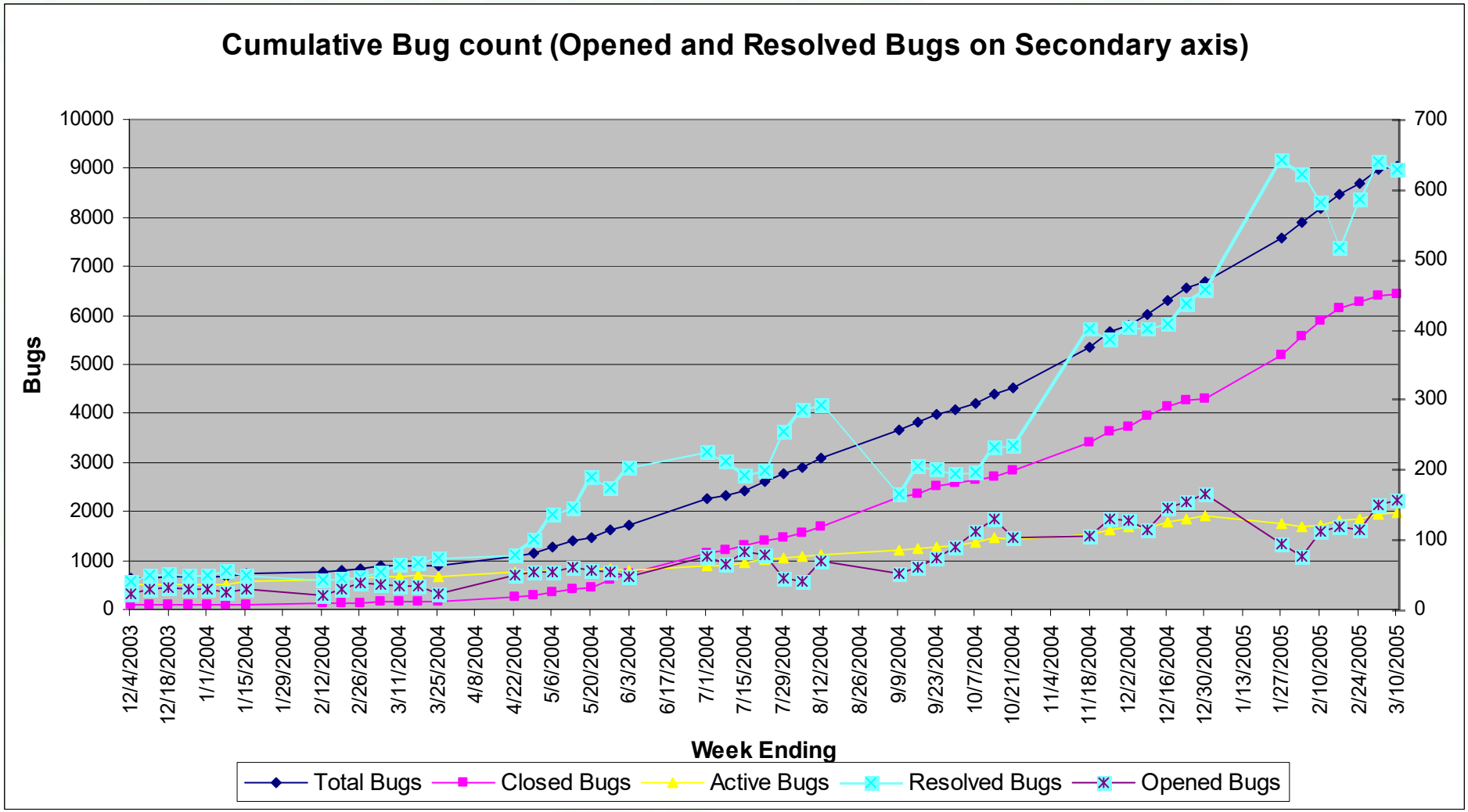
2.缺陷统计数据的度量(续)

- 已发现缺陷的数量和已修复的缺陷的数量的比率(Fixed/Found)。也被称为修改率或纠错率(Fix Rate)
- 未处理的缺陷数量和已处理的的缺陷数量的比率(active/resolved)
- 已处理的被修复的缺陷数量和已处理的缺陷数量的比率（Resolved as Fixed/resolved）
- 重新被激活的已修复的缺陷数量(Bug re-activation rate)
- 通过测试找到的缺陷的统计(Bugs opened by testing activity)

2.缺陷统计数据的度量(续)

- 不同语言版本缺陷数量的统计(Bugs opened by Language version)
- 被报告存在缺陷的各功能统计(Where your bugs were found)
- 处理缺陷的平均时间的统计(Average Time to Resolve)
- 关闭缺陷的平均时间的统计(Average Time to Close)
- 被处理缺陷的不同结论统计(Resolved Bugs By Resolution)

里程碑编程阶段缺陷变化趋势



3.测试案例度量

- 运行测试案例数量和通过测试的案例数量之比
- 不同产品开发阶段该比率变化
- 测试案例包括的范围
- 运行测试案例的频率
- 有测试案例的功能数量

4.测试规范度量

测试规范：微软把针对怎样测试某功能的，有细分功能后的具体测试条例等细节的测试文档叫做测试规范（**Test Design Specification** 或简称 **TDS**）。

- 测试规范数量和所有功能数量之比
- 满足撰写要求的测试规范数量和所有测试规范数量之比
- 必要的内容遗漏的比率

5.测试过的系统数量

- 所支持的不同语言系统的总数与测试过的语言系统数量
- 所支持系统的总数与测试过的系统数量
  - Windows 2000 (SPx)
  - Windows XP (SPx)
  - Windows 2003 Server (SPx)
  - Tablet PC
  - 新的系统平台

6.自动化测试度量

- 测试的可自动化程度
- 能自动化的和实现自动化的比率
- 运行通过的自动化脚本比率
- 不同产品开发阶段该比率变化

7.Code Coverage（代码覆盖）

- 代码覆盖度量定义和目的
- 代码覆盖种类
- 代码覆盖的有效使用
  - 开发人员：单元测试 (unit testing)
  - 测试人员：系统测试(system testing)和自动化测试

代码覆盖是什么？

- 动态白盒测试评价技术
  - 已经执行（测试）了什么（what *has* been executed）
  - 没有执行的（测试）有什么 what has *not* been executed and still remains to be tested.

需要有源代码

内部辅助工具

使用代码覆盖度量的目地

- 经验总结：大约的20%代码囊括缺陷总数的80%
- 目的不是要达到某个神奇的数字，而是要发现测试中的漏洞
- 达到比较广泛的覆盖率相对容易，但要达到100%覆盖常需要多得多的成本
  - 平均目标 65%
  - 理想目标 75%

代码覆盖度量种类

1. 代码函数覆盖数量
2. 代码运行使用到的功能覆盖数量
3. 代码数据种类覆盖数量
4. 代码函数条件覆盖数量
5. 代码通路(path)覆盖数量

代码覆盖结果分析

E43 fx 330

	A	B	C	D	E	F	G
1	TeamNa	FunctionName	SourceFile	CountBytes	BytesHit	PercentByte	PercentBlocksHit
2	Excel	EndHHConvert	拼写检查	52	0	0	0
3	Excel	FAllocSpudr	拼写检查	247	205	83	73.33
4	Excel	FCopySpudr	拼写检查	71	41	57.75	63.64
5	Excel	FEnsureSpeller	拼写检查	258	200	77.52	84.21
6	Excel	FindListLcidMacro	拼写检查	47	47	100	100
7	Excel	FGetUserDict	拼写检查	184	175	95.11	87.5
8	Excel	FSetUserDict	拼写检查	200	168	84	81.25
40	Excel	SetSpellDlgCaption	拼写检查	105	105	100	100
41	Excel	SetSpellingOptions	拼写检查	183	112	61.2	82.35
42	Excel	SgnCmpLangSz	拼写检查	89	89	100	100
43	Excel	SpellAL	拼写检查	406	330	81.28	75
44	Excel	SpellBuffer	拼写检查	3624	3053	84.24	80.37
57	Excel	SpellGetIntVar	拼写检查	26	22	84.62	75
58	Excel	SpellGetStrVar	拼写检查	57	49	85.96	83.33
59	Excel	SpellGetString	拼写检查	20	20	100	100
60	Excel	SpellHeaderFooter	拼写检查	664	565	85.09	80.49
61	Excel	SpellInitVars	拼写检查	77	77	100	100
62	Excel	SpellLoadLibrary	拼写检查	247	198	80.16	66.67
63	Excel	SpellMain	拼写检查	1184	887	74.92	74.38
64	Excel	SpellTextSelection	拼写检查	879	624	70.99	70.27
65	Excel	Spelling	拼写检查	1027	782	76.14	75
66	Excel	SpellingDialog	拼写检查	1250	990	79.2	80.18
67	Excel	UEnableSpell	拼写检查	131	126	96.18	95.24
68	Excel	UFindListLcid	拼写检查	67	42	62.69	54.55
69							
70							
71				代码覆盖	71.64%	74.02%	

使用代码覆盖度量改进测试

- 代码覆盖度量只能揭示测试的漏洞，并不能直接改进测试
- 为什么有些代码没有执行到？
- 脚本运行时执行到了代码不意味着测试的深度和全面性
- 先查功能代码覆盖率，再计划写自动化脚本的优先顺序
- 撰写测试用例已覆盖所有要测试的功能行为，然后编写自动化脚本加以验证
- 添加新自动化脚本覆盖找到的漏洞

使用代码覆盖度量结果分析

- 没有覆盖代码的可能原因和改进措施：
  - 遗漏的功能行为：追加测试
  - 程序中有‘死角’代码，没有功能行为可以执行该代码：删除？
  - 很难模拟的出错条件：可否有其他方法？
  - 过时的功能规范？-- 更新功能规范

8.单一功能测试验收质量度量

- 预先计划的详细测试：100%完成？
- 自动化测试覆盖率：≥65%？
- 自动化测试运行结果：0%失败率？
- 发现缺陷的难易程度：4小时发现缺陷≤2
- 缺陷严重度和数量变化趋势：近期无高严重度缺陷
- 功能稳定程度：近期代码无需改变、自动化运行一直保持100%通过

使用软件质量度量的注意事项和建议

- 应当加以分析后挑选适当度量
- 不应作为唯一的测试质量衡量标准
- 考虑人为因素和不定性因素的影响
- 同一产品不同功能也应使用统一衡量标准
- 分析度量结果以指导测试和开发过程
- 研发适合自己产品使用的质量度量