软件测试与维护

一、简介

Bug

Bug有多种：defect、fault、problem、error...

problem的来源：需求定义、设计、实现、支持系统、软件测试不足、草率的重构

bug来源百分比：规格说明书(55%)、设计阶段(25%)、代码阶段(15%)、其他(5%)

软件测试的定义

对软件产品进行测试，尽可能早地发现bug，并督促相关人员尽早解决它们

测试模型

V模型：近似瀑布模型，先开发再测试

W模型：测试从需求阶段便开始，紧跟软件开发过程

H模型：包括测试准备和测试执行，是一个独立流程，贯穿产品整个周期

X模型：边开发边集成测试

二、软件测试的现实

软件测试的9大公理

1.将程序测试完全很重要；2.测试是基于风险的实践；3.测试无法显示缺省的bug

4.bug越找越多；5.不是所有bug被找到后都会被修复；6.很难说bug什么时候是真的bug

7.规格说明书永远不是最终版；8.测试工程师不是项目中最受欢迎的；9.软件测试是需要训练和技术的一个职业

软件测试的补充原则

必须基于“质量第一”

事先定义好产品的质量标准

第三方测试更客观、更有效

重视文档，妥善保存测试过程一切文档

软件测试的分类

按方法：黑盒测试、白盒测试

按目标/特性：功能测试、健壮性测试、性能测试、适用性测试、安全性测试、可靠性测试

静态测试：需求评审、概要设计评审、详细设计评审、代码检查

动态测试：单元测试、集成测试、系统测试、验收测试

测试与调试的区别

测试发展的初期，测试就是调试；而现在测试是一个系统化工程化的概念，有自己的生命周期，调试的范畴更小一些

调试不属于测试，是编码阶段的工作，由程序员完成；而测试由测试员或程序员完成

测试管理体系

测试规划（确定目标和策略）->测试设计（确定测试方案及用例）->测试实施（执行用例）->配置管理（测试配置管理）->资源管理（资源综合调配与管理）->测试管理（对以上过程综合管理）

SQA（Software Quality Assurance）

软件质量保证是通过对软件产品和活动有计划的进行评审和审计来验证软件是否合乎标准的系统工程活动

SQA与软件测试的关系

SQA 是管理工作，审查对象是流程，强调以预防为主

测试是技术实施工作，测试对象是产品，主要是以事后检查（文档、程序）为主

SQA指导测试、监控测试，测试为SQA提供依据，是SQA的一个环节、一个手段

组建测试队伍

任务与责任

任务：测试计划、测试用例设计、执行测试、评估测试结果、递交测试报告

责任：尽早发现问题，督促尽早解决，帮助制定合理的开发计划，分析、总结、跟踪问题，提高程序、文档的规范性、易读性、可维护性等

基本构成

测试组长：负责项目的管理、测试计划、任务安排等

环境配置人员：设置、配置和维护实验室的测试环境

测试设计人员/资深测试工程师：规格说明书、设计的审查，测试用例的设计，技术难题的解决，培训和指导，实际测试任务的执行

一般（初级）测试工程师：执行测试用例和相关的测试任务

三、测试计划

测试策略

测试策略通常是描述测试工程的总体方法和目标。描述目前在进行哪一阶段的测试（如单元测试、集成测试、系统测试）以及每个阶段内进行的测试种类（如功能测试、性能测试、压力测试等）和方法，以确定合理的测试方案使得测试更有效

测试计划的标准格式（IEEE）

四、测试用例

定义

满足特定目的的测试数据、测试代码、测试规程的集合

是发现软件缺陷的最小测试执行单元

有特殊的书写标准和基本原则

生成的基本准测

代表性：能够代表并覆盖各种输入数据、操作和环境设置等

可判定性：测试结果的正确性是可判定的

可再现性：对同样的测试用例，系统的执行结果应当是相同的

设计步骤

为测试需求确定测试用例->为测试用例确定输入输出->编写测试用例->评审测试用例->跟踪测试用例

作用

避免盲目测试；估算测试工作量；减少回归测试的复杂程度；方便书写软件测试缺陷报告；实施不同级别的测试

五、测试用例设计（黑盒）

测试数据和测试用例的区别

测试数据：被用来测试系统的输入数据（输入）

测试用例：被测试系统的输入和根据规格说明书预测的该系统的输出（输入&输出）

测试用例设计技术

等价类划分法

分为有效等价类和无效等价类，每个等价类中取一个数据作为测试的输入

输入条件规定了取值范围或值的个数：一个有效等价类和两个无效等价类

输入条件规定了输入值的集合或者“必须如何”：一个有效等价类和一个无效等价类

输入条件是一个布尔量：一个有效等价类和一个无效等价类

规定输入数据的一组值且对每一个输入值分别处理：N个有效等价类和一个无效等价类

规定输入数据必须遵守的规则：一个有效等价类和N个无效等价类（从不同角度违反规则）

已划分的等价类中各个元素在程序中处理方式不同：划分为更小的等价类

边界值分析法

使等价类的每个边界都要作为测试条件，是等价类划分法的一个补充

错误推测法

通过经验和直觉推测出程序的错误所在

判定表驱动分析法

建立步骤

1.列出所有的条件桩和动作桩

2.确定规则的个数。假如有n个条件桩，就有2^n种规则

3.填入条件项

4.填入动作项。得到原始判定表。

5.根据原始判定表的规则，设计测试用例，要求覆盖所有原始判定表的规则

因果图法

着重分析输入条件的各种组合

1.分析软件规格说明描述中，哪些是原因、结果，并给每个原因和结果赋予一个标识符

2.找出原因与结果之间、原因与原因之间对应的关系，画出因果图

3.把因果图转换为判定表

4.把判定表的每一列作为依据，设计测试用例

场景法

三步曲

1.设计场景：根据用例的主事件流和备选事件流的组合给出不同场景

2.设计测试用例标准覆盖场景

3.根据测试用例标准给出具体的测试数据

六、白盒测试

控制流测试

逻辑分支覆盖法

语句覆盖

每一个可执行语句至少执行一次

判定覆盖

程序中每个判断的取真分支和取假分支至少经历 一次

条件覆盖

程序中每个判断的每个条件的可能取值至少执行一次

判定-条件覆盖

判定中每个条件的所有可能（真/假）至少出现一次，并且每个判定本身的判定结果（真/假）也至少出现一 次

条件组合覆盖

每个判断的所有可能的条件取值组合至少执行一次

路径法

路径覆盖

保证程序中每条可能的路径都至少执行一次

基本（独立）路径测试法

定义：从入口到出口的路径，至少经历一个从未走过的边，这样形成的路径叫独立路径

步骤：画程序框图->画控制流图->计算环复杂度->确定基本路径集->设计测试用例

基本路径数=环复杂度=区域数=边数量-节点数量+2=判断节点数量+1

数据流测试

使用路径（du-Path）定义：是PATHS(P) 中的路径，使得对某个v∈V,存在定义和使用节点DEF(v,m)和USE(v,n),使得m和n是该路径的最初和最终节点

数据流测试的策略：遍历所有使用路径

变异测试

变异算子：用来模拟典型的用户输入错误(如运算符或变量名使用错误等)以及强制使某些表达式满足一定的条件(如使表达式的值为0)

变异测试是一种对测试集的充分性进行评估的技术，以创建更有效的的测试集，不同于路径测试和数据流测试，没有测试数据的选择规则。

七、单元测试

单元测试定义

单元测试是对软件基本组成单元进行的测试

时机：一般在代码完成后由开发人员完成，QA人员辅助

单元的概念：有明确的功能、性能定义、接口定义的软件设计最小单位--模块

单元测试内容

局部数据结构、模块接口、出错处理、独立路径、边界条件

单元测试作用

在集成测试前保证最小代码单元的质量

单元测试策略

黑盒测试：等价类划分、边界值分析

动态白盒测试：逻辑覆盖、路径覆盖

静态白盒测试：在代码执行前仔细和方法性地检查软件设计、架构、代码

非正式代码审查

同级审查：程序员和测试人员组成的非正式小组扮演审查者角色

走查：更正式点，代码作者向程序员和测试人员组成的小组展示每一行代码的作用，审查者倾听，询问问题。

正式代码审查

一对一，严格审查，有文档，有规范

代码演示者不是代码的作者

其他参与者是检察员

有一个主持人确保规则遵循和会议顺利进行

会后要形成审查报告

极限编程（XP开发）

测试驱动开发（TDD）

先编写测试代码，再进行开发

先编写产品的框架，是指先编写类、方法空的实现

编译通过然后编写测试类，针对产品类的功能编写测试用例

然后编写产品类的代码，每写一个功能点都运行测试，随时补充测试用例

编写的测试代码以后需要修改的可能性比较小

代码审查清单

逻辑错误？

计算错误

比较错误

控制流错误

子程序的参数错误

输入/输出错误

其他检查

是否适合其他操作系统平台

是否国际化

单元测试的过程和文档管理

1，在详细设计阶段完成单元测试计划。---《单元测试计划》

2，建立单元测试环境，完成测试设计和开发。---《单元测试用例》

3，执行单元测试用例，并且详细记录测试结果。---《缺陷跟踪报告》

4，判定测试用例是否通过。

5，提交《单元测试报告》

单元测试工具

JUnit

Parasoft Jtest

Parasoft C++ Test

八、集成测试

集成的概念

把小的单元模块拼接、组合起来

集成测试的概念

将单元组装起来再进行测试，以检查这些单元之间的接口是否存在问题。如：数据丢失、模块间相互影响、组合后不能实现主功能等

软件集成测试前的准备：人员安排、测试计划、测试内容、集成策略、测试方法

集成测试的目的

保证系统设计的完整性

保证组件之间适当的交互

运行简单的系统级别的测试

集成测试的策略

Top-down

跟随自顶向下开发流程，从顶级模块开始，对低一级的模块编写桩模块

桩模块：用以模拟被测模块工作过程中的所调用的模块。桩模块由被测模块调用

优点：早期验证主要功能；总是有一个顶层系统；模块可以按照接口规范编写

缺点：性能问题延迟出现，桩模块编写成本高

Bottom-up

从底层模块开始，对上一级的模块编写驱动模块

驱动模块：用以模拟被测模块的上级模块。启动被测模块，并打印出相应的结果

优点：原子函数得到更多测试，底层模块错误发现早，成本低

缺点：完整的系统到后期才能完全呈现出来

混合策略

对软件结构中较上层使用自顶向下，较下层使用自底向上

Big Bang

等到所有组件都开发完成后，一次性集成测试

省去写驱动模块和桩模块的成本

不易发现错误，一旦出错很难确定错误出在哪

Critical-First

优先集成最重要的组件，其他部分后期慢慢加上

优点：保证最重要的组件优先运行

集成测试的模式

渐增式

把下一个要测试的模块同已经测试好的模块结合起来进行测试，测试完以后再把下一个应该测试的模块结合进来测试

优点：测试彻底，较早发现错误

缺点：工作量大，耗时间

非渐增式

先分别测试每个模块，再把每个模块按设计要求放在一起结合成所要的程序，如大棒模式

优点：工作量少，可并行，大的工作量在底层

缺点：错误难诊断，发现错误晚

测试环境的建立

测试环境是测试的基础

设计环境

实施环境

管理环境

五要素

软件

硬件

网络环境

数据准备

测试工具

测试部门建设方案

独立运作

与第三方测试机构合作

与相关企业合作

测试外包

项目经理室式外包

软件逻辑架构

系统物理架构

TPC事务处理性能委员会

九、系统测试

目的

获得一个作为整体的系统的完整性的信心

功能测试

概念

在集成测试结束之后，依据系统的需求规格说明书和产品功能说明书对系统的整体功能进行的全面俄车市，称为功能测试

目的和内容

程序安装、启动正常，有相应的提示框、错误提示等

系统的界面清晰、美观

菜单、按钮操作正常、灵活，能处理一些异常操作

能接受正确的数据输入，对异常数据的输入可以进行提示、容错处理等

数据的输出结果准确，格式清晰，可以保存和读取

功能逻辑清楚，符合使用者习惯

系统的各种状态按照业务流程而变化，并保持稳定

能配合多种硬件周边设备

软件升级后，能继续支持旧版本的数据

与外部应用系统的接口有效

工具：QTP

回归测试

定义

对修正缺陷后的软件进行再次的测试，不仅测试被修复的软件缺陷是否已经解决，还要测试软件旧有的功能与非功能是否满足要求

目的

方法

再测试全部用例

基于风险选择测试：测试关键的、重要的、可疑的，跳过非关键的、稳定的

基于操作剖面选择测试

再测试修改部分

冒烟测试

定义

每日Build建立后，对系统的基本功能进行简单的测试。这种测试强调功能的覆盖率

目的

先通过最基本的测试，如果最基本的测试都有问题，就直接打回开发部了，减少测试部门时间的浪费

非功能测试

常用名词解释

响应时间

并发

分类

压力测试

定义

在一种需要反常数量、频率或资源的方式下，执行可重复的负载测试或强度测试，以检查程序对异常情况的抵抗能力，找出性能瓶颈。

压力估算

给出合理的压力估算值（结合峰值），给压力测试提供数据依据

根据峰值结合TPCC、TPCE、TPCH等指标，结合服务器的性能给出系统的硬件配置估算、给出硬件配置建议

稳定性压力测试（可靠性压力测试）

以合理的估算压力连续运行被测系统，检查系统运行时的稳定程度

采用24*7(24小时*7天)的方式让系统不间断运行

为了给出系统一个实际使用的指标和范围、使用边界条件

破坏性压力测试

持续不断地给被测系统增加压力，直到将被测系统压垮为止（让问题和薄弱环节快速暴露出来，找出瓶颈）。用来测试系统所能承受的最大压力。

采用破坏的手段，目的是为了找到系统的瓶颈

注意事项

问题的分析：日志文件、进程监控、查看运行参数、分解问题

累积效应；不要重启、不要归零、考虑累积情况

容量测试

通过测试预先分析出反映软件系统应用特征的某项指标的极限值（如最大并发用户数、数据库记录数等），系统在其极限值状态下还能保持主要功能正常运行

还将确定测试对象在给定时间内能够持续处理的最大负载或工作量

通过压力测试的稳定性测试，可以得到容量值

性能测试

相关性能指标

客户端指标：响应时间、并发数

服务器端指标

processor time：服务器CPU占用率。一般平均达到70%时，服务

memory available Mbyte：

throughput（吞吐量）：

physics disk time：物理磁盘读写时间情况

定义

通过测试确定系统运行期间的性能表现与性能数据，得到如：CPU使用效率、运行速度、响应时间、占用系统资源等方面的系统数据，给出合理的系统配置

作用

确定在什么样的压力下系统达到最佳状态（稳定性压力测试），什么压力是系统的极限（破坏性压力测试）

根据性能指标确定实际的软硬件运行环境：如配置合理的CPU数、CPU的处理能力、内存量等等

目的

通过性能测试找出系统的性能瓶颈，解决问题，提升性能，并且给出性能指标数据，给出合理的环境配置。最大限度满足用户需求，提升产品质量

性能合理的范围

服务器CPU的占有率：一般在85%以内

内存占有率：80%以内

工具：loadrunner、QALoad

LoadRunner

（1）Virtual User Generator 创建脚本

创建脚本，选择协议

录制脚本

编辑脚本

检查修改脚本是否有误

（2）中央控制器（Controller）来调度虚拟用户

创建Scenario（一个测试用例），选择脚本

入股模拟多级测试，设置IP Spoofer

（3）运行脚本

分析Scenario

（4）分析测试结果

安全测试

检查系统对非法侵入的防范能力。安全测试期间，测试人员假扮非法入侵者，采用各种办法试图突破防线。

理论上讲，只要有足够的时间和资源，没有不可进入的系统。因此系统安全设计的准则是，使非法侵入的代价超过被保护信息的价值，测试非法侵入者已无利可图

容错测试

检查软件在异常条件下自身是否具有防护性的措施或者某种灾难性恢复的手段

性能测试与瓶颈分析关键步骤

步骤一：性能测试与数据收集

步骤二：性能瓶颈分析

步骤三：性能调优解决方案

验收测试

定义

在软件产品完成了功能测试和系统测试之后、产品发布之前所进行的软件测试活动。它是技术测试的最后一个阶段,也称为交付测试

过程和主要内容

前提

系统或软件产品已通过了系统测试的软件系统

测试内容

验证系统是否达到了用户需求规格说明书（可能包 括项目或产品验收准则）中的要求，测试试图尽可能地发现软件中存留的缺陷，从而为软件进一步改善提供帮助，并保证系统或软件产品最终被用户接受

还需进行易用性测试、兼容性测试、安装测试、文档（如用户手册、操作手册等）测试等几个方面的内容

标准和注意事项

完成标准

完全执行了验收测试计划中的每个测试用例

在验收测试中发现的错误已经得到修改并且通过了测试或者经过评估留待下一版本中修改

完成软件验收测试报告

注意事项

必须编写正式的、单独的验收测试报告

验收测试必须在实际用户运行环境中进行

由用户和测试部门共同执行。如公司自开发产品，应由测试人员，产品设计部门，市场部门等共同进行

其他方法

α测试

α测试是指软件开发公司组织内部人员模拟各类用户对即将面市软件产品（称为α版本）进行测试，试图发现错误并修正

β测试

经过α测试调整的软件产品称为β版本。紧随其后的β测试是指软件开发公司组织各方面的典型用户在日常工作中实际使用β版本，并要求用户报告异常情况、提出批评意见。然后软件开发公司再对β版本进行改错和完善

用户界面和易用性测试

用户界面七要素：符合标准和规范；直观性；一致性；灵活性；舒适性；正确性；实用性

易用性测试没有具体量化的指标，主观性较强

兼容性测试

定义

软件兼容性测试是指验证软件之间是否正确地交互和共享信息

包括

硬件兼容

软件之间兼容

数据之间兼容

向前和向后兼容

向后兼容是指可以使用软件的以前版本

向前兼容指的是可以使用软件的未来版本

可安装性和可恢复性测试

系统软件安装

应用软件安装

服务器的安装

客户端的安装

产品升级安装

文档测试

软件文档已成为软件的一个重要组成部分，而且种类繁多，对文档的测试也变得必不可少

十一、软件缺陷与测试报告

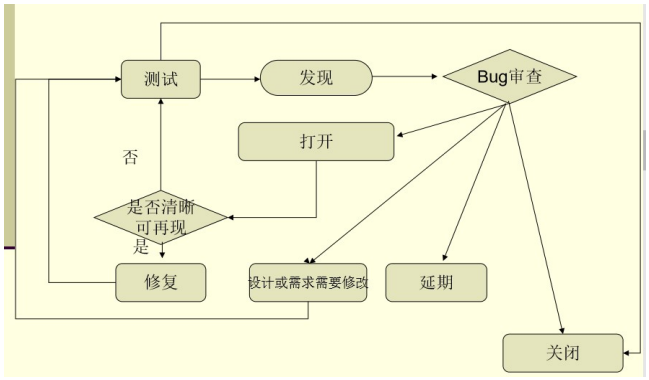
缺陷描述

软件缺陷的描述是软件缺陷报告中测试人员对问题的陈述的一部分并且是软件缺陷报告的基础部分

缺陷属性

缺陷标识；缺陷类型；缺陷严重程度；缺陷优先级；缺陷产生可能性；缺陷状态；缺陷起源；缺陷来源；缺陷原因

缺陷生命周期



软件缺陷跟踪系统

主要包含缺陷管理功能和统计功能

常见免费缺陷管理工具：Mantis，Bugzilla

付费缺陷管理工具：Testdirector，JIRA

测试报告与缺陷报告的区别

缺陷报告：描述清楚缺陷，是测试过程的结果

测试报告：是分析与总结，是阶段性的总结

二者关系：缺陷报告为测试报告提供数据与依据，测试报告是缺陷报告的总结

十三、配置管理与软件维护

配置管理

定义

软件配置管理，是指通过执行版本控制、变更控制等规程，以及使用合适的配置管理软件，来保证所有配置项的完整性和可跟踪性

配置管理是对工作成果的一种有效保护

配置项

定义

软件配置管理的对象是软件配置项，它们是在软件工程过程中产生的信息项

包括

与合同、过程、计划和产品有关的文档和数据

源代码、目标代码和可执行代码

相关产品，包括软件工具、库内的可利用软件、外购软件及用户提供的软件

命名/编号

软件的每个组件/部件的标识必须唯一，以便于用该标识符来跟踪和报告软件配置项的状态

版本

定义

版本是某一配置项的已标识了的实例

也可以说，不可变的源对象经质量检查合格后所形成的新的相对稳定的格局（配置）称为软件版本

通常软件对象的版本组呈树形结构

版本控制

版本控制就是管理在整个软件生存周期中建立起来的某一配置项的不同版本

版本管理

要求

能够根据用户的不同需求，提供不同的版本的软件配置项以配置不同的系统

保存软件配置项的老版本，以便能够对以后出现的问题作调查

能够根据各种版本的软件配置来配置生成开发项目版本的一个新版本

能够支持多个软件开发人员同时对一个软件配置项进行操作与处理

将各软件配置项的各种版本进行高效存储

基线

基线指一个配置项在其生存周期的某一特定时间，被正式标明、固定并经正式批准的阶段性版本

只有由正式技术评审而得到的软件配置项协议和软件配置的正式文本才能成为基线

配置控制组/委员会

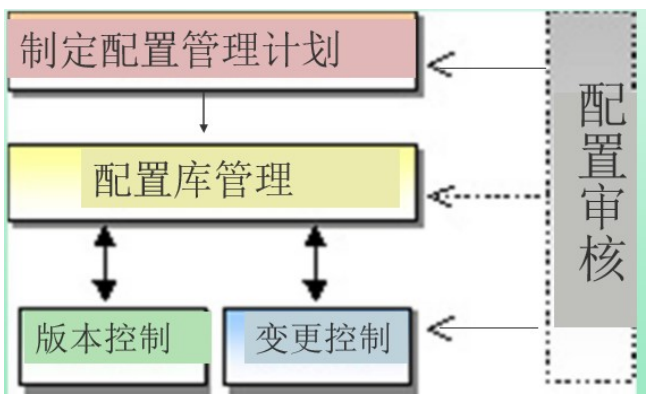
配置控制组/委员会是指一组负责评估和审批配置项变更的人员，以确保所有的变更都是经过审核的

变更管理

变更管理是软件配置管理的一个要素，由评估、协调、批准或不批准以及对正式构造配置标识的配置项实施变更等活动组成。

变更管理更多的是指变更的管理流程、人员按照一定的流程和制度去执行变更的过程控制

过程



1.制定配置管理计划

配置管理员制定《配置管理计划》，主要内容包括配置管理软硬件资源、配置项计划、基线计划、交付计划、备份计划等

2.配置库管理

配置管理员为项目创建配置库，并给每个项目成员分配权限。各项目成员根据自己的权限操作配置库。配置管理员定期维护配置库，例如清除垃圾文件、备份配置库等

3.版本控制

版本控制的目的是按照一定的规则保存配置项的所有版本，避免发生版本丢失或混淆等现象，并且可以快速准确地查找到配置项的任何版本

4.变更控制

在项目开发过程中，配置项发生变更几乎是不可避免的。变更控制的目的就是为了防止配置项被随意修改而导致混乱

5.配置审核

为了保证所有人员（包括项目成员、配置管理员和CCB）都遵守配置管理规范，质量保证人员要定期审核配置管理工作

相关培训

管理员培训

开发人员培训

管理流程培训

常用工具

SVN

GIT

软件维护

定义

软件维护是指软件系统交付使用以后，为了改正错误或满足新的需要而修改软件的过程

原因

改正程序中的错误和缺陷

改进设计以适应新的软、硬件环境

增加新的应用范围

类型

改正性维护

在软件交付使用后，因开发阶段的问题以及测试得不彻底、不完全，必然会有部分隐藏的错误遗留到运行阶段

为了识别和纠正软件错误、改正软件功能、非功能（性能）上的缺陷、排除实施中的误使用，应当进行的诊断和改正错误的过程就叫做改正性维护

适应性维护

在使用过程中，外部环境（新的硬、软件配置）、数据环境（数据库、数据格式、数据输入/输出方式、数据存储介质）可能发生变化

为使软件适应这种变化，而去修改软件的过程就叫做适应性维护

完善性维护

在软件的使用过程中，用户往往会对软件提出新的功能与性能要求

为了满足这些要求，需要修改或再开发软件，以扩充软件功能、增强软件性能、改进加工效率、提高软件的可维护性

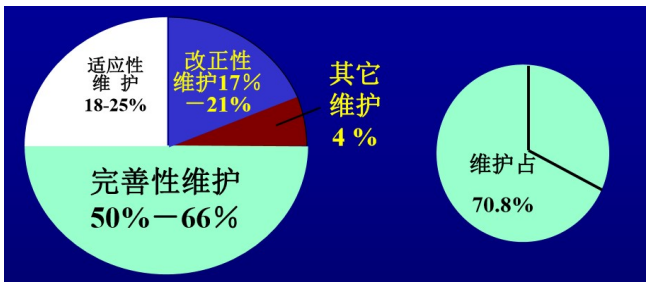
这种情况下进行的维护活动叫做完善性维护

预防性维护

预防性维护即软件再工程，是为了提高软件的可维护性、可靠性等，为以后进一步改进软件打下良好基础

采用先进的软件工程方法对需要维护的软件或软件中的某一部分（重新）进行设计、编制和测试，称为预防性维护

各种维护类型和维护工作量的比例



结构化维护VS非结构化维护

如果采用软件工程的方法进行软件开发，保证每 阶段都有完整且详细的文档，这样维护会相对容易，被称为结构化的维护

如果不采用软件工程方法开发软件，软件只有程序而欠缺文档，则维护工作变得十分困难，称为非结构化的维护

影响软件维护工作量的因素

系统的大小

程序设计语言

系统年龄

先进的软件开发技术

其他一些因素

软件维护的过程

维护申请->维护修改报告->维护记录保存->维护后评审->维护后测试->维护后验收

可维护性

定义

软件可维护性 是指纠正软件系统出现的错误和缺陷，以及为满足新的要求进行修改、扩充或压缩的难易程度

衡量特性

1.可理解性

可理解性表明人们通过阅读源代码和相关文档，了解程序功能及其如何运行的容易程度

2.可靠性

可靠性表明一个程序按照用户的要求和设计目标，在给定的一段时间内正确执行的概率

3.可测试性

可测试性表明论证程序正确性的容易程度。程序越简单，证明其正确性就越容易。而且设计合用的测试用例，取决于对程序的全面理解

一个可测试的程序应当是可理解的，可靠的，简单的

4.可修改性

可修改性表明程序容易修改的程度

一个可修改的程序应当是可理解的、通用的、灵活的、简单的

5.可移植性

可移植性表明程序转移到一个新的计算机环境的可能性的大小

它表明程序可以容易地、有效地在各种各样的计算机环境中运行的容易程度

6.效率

效率表明一个程序能执行预定功能而又不浪费机器资源的程度

这些机器资源包括内存容量、外存容量、通道容量和执行时间

7.可使用性

从用户观点出发，可使用性定义为程序方便、实用、及易于使用的程度

文档管理

维护合同、维护方案、维护手册、维护申请表、维护记录、维护报告