软件测试从这里开始

版本更新记录

版本号	说明	作者	修改日期	备注
V0.1	创建	晏斌	2006-5-10	
V1.0.0.0	发布	晏斌	2006-8-31	

目录

序

从事测试工作以来，从前人那里学到了很多测试知识,也有一些切身感受。本文是对目前掌握的测试知识的整理和归纳。本文诣在点到需要掌握的一些知识点,如果想对某个知识点了解更多,请查阅针对该知识点的专门资料.

请读者注意：本文只列举出了一些作者认为经常用到的知识点，并不能覆盖所有的测试知识点。文中可能会有一些错误和纰漏，请读者及时反馈给我，不胜感激！

另外，文中很多知识都没有做详细的阐述，应该说只要文中提到的知识点，都有文档可寻，如果读者想获得更详细的知识，可以在网络中下载相关文档来阅读，或者可以发送邮件到我的邮箱，我会及时予以答复。邮箱地址在文档的页脚中可以见到。

作者从事测试工作时间不长，写此文档的目的是把自己学到的一些东西能与大家共享，希望大家知识共享，共同进步。

第 1 章 开题篇

作为该文档的开始，本章将介绍对测试以及测试行业做一些简单的阐述。包括测试行业的前景，测试人员的基本素质和注意事项等等。

第 1 节个人职业发展方向

- 测试技术型
- 测试管理型
- 转型

第 2 节测试行业简介

软件测试在软件生命周期中占据重要的地位，在传统的瀑布模型中软件测试学仅处于运行维护阶段之前，是软件产品交付用户使用之前保证软件质量的重要手段。近来，软件工程界趋向于一种新的观点，即认为软件生命周期每一阶段中都应包含测试，从而检验本阶段的成果是否接近预期的目标，尽可能早的发现错误并加以修正如果不在早期阶段进行测试，错误的延时扩散常常会导致最后成品测试的巨大困难 。由于测试工作的重要性和复杂度，测试慢慢的独立发展成为一个行业，并且在迅猛发展。

在典型的软件开发项目中，软件测试工作量往往占软件开发总工作量的 40％以上。而在软件开发的总成本中，用在测试上的开销要占 30％到 50％。

1.2.1如何认识测试

- 测试的春天到了；
- 测试是一个方法论而不是一个技术；
- 软件测试工程学或者质量工程学也应该诞生了，管理和技术并重。

第 3 节软件测试的误区

- × 软件开发完成后进行软件测试；
- × 软件质量问题是测试人员的错误，软件发布后如果发现问题，那是软件测试人员的错。
- × 测试技术要求不高，比编程容易，随便找一个人就可以了；
  有编程经验对测试 BUG 的敏感性；
  需要编写自动化测试脚本的能力；
  除了技术还有管理，谁都可以做，但是结果不一样。
- × 测试跟着开发动，有时间就多测，没时间就少测；
  必须有计划有组织。
- × 测试是测试人员的事，与开发人员无关；
  开发人员需要自测，还需要沟通协作。

- 软件测试是没有前途的工作，只有程序员才是软件高手；
- 测试是软件开发的后期活动；软件测试=程序测试；
  “软件缺陷具有生育能力” ；
  需求测试和设计测试也是软件测试的一种；
  软件测试应该涵盖整个软件生命周期；
  同时，软件测试本身也应被测试。
- 测试要执行所有可能的输入；
  在实际测试中，穷举测试工作量太大，实践上行不通；
  一般采用等价类和边界值分析等措施来进行实际的软件测试；
  寻找最小最重要的用例集合成为我们精简测试复杂性的一条必经之道。
- 好的测试一定要使用很多的测试工具。

工具所能发挥的作用依赖于使用工具的人。因此，对工具的过分依赖将降低人的能动性，并最终使测试本身受到损害。适当的使用测试工具能够减轻测试人员的机械性工作，提高工作效率，而滥用工具会降低测试的质量。并不是任何工作都适合自动化的，如何合理的自动化测试，合理的选择适当的测试工具已经是研究人员感兴趣的一个课题。

第 4 节测试工程师素质

1.4.1基本素质

- 沟通能力、自信心、幽默感、记忆力<挖掘以往错误>、耐心、怀疑精神、自我督促、洞察力<发现重点>；
- 广泛的经验；
- 表达能力、问题描述能力；
- 会提问，会寻求 Help；
- 逻辑思维能力；
- 团队协作能力；
- 处理日常事务的能力和处理突发事件的能力

1.4.2专业素质

- 对于系统测试，把握需求是第一位的。对产品熟练，能够快速熟悉新的产品需求， 很强的需求理解能力显得很重要；
- 测试基础：明确测试流程中各个阶段的工作，对测试的认知程度，决定了测试流程管理的规范性，测试工作的质量；
- 测试方案的分析设计能力、测试案例的设计能力(测试案例的覆盖率、优先级等)；
- 测试工具的使用(包括测试管理和测试执行工具，也包括开发工具的能力)；
- 编程能力，数据库知识，网络知识，操作系统知识；
- 团队协作能力，与各个小组之间的沟通能力；
- 测试管理，管理决定了工作质量。尤其是测试经理，需要管理团队测试的能力。
  一般的说，技术上的问题都不是问题，目前的软件更缺乏行之有效的管理。

第 5 节测试工程师分类

- 测试工程师一般分为两类：测试工具软件开发工程师和软件测试工程师。

测试工具软件开发工程师主要负责编写测试工具代码，并利用测试工具对软件进行测试；或者开发测试工具为软件测试工程师服务。

软件测试工程师主要负责理解产品的功能要求，然后对其进行测试，检查软件有没有错误，决定软件是否具有稳定性，并写出相应的测试规范和测试案例。

- 按职位分类：测试部门经理、测试技术总监、测试主管、测试项目经理、测试设计人员、测试执行人员、测试协助员、技术支持；
- 按测试类型分类：功能测试工程师、自动化测试工程师、性能测试工程师等；
- 按测试对象分类：web 测试工程师、数据库测试工程师、C/S 测试工程师、个人软件测试工程师等；

第 6 节测试工作的未来

//此文为引述

◆ **测试工作未来预见**

更好的方法对测试人员更好的培训、更好的欣赏将改革软件产业。具体地说，诸如可执行的说明书、基于模型的测试产生、BUG 预防、系统模拟这些技术，将在这场演变过程中扮演重要的角色。

◆ **BUG 预防和早期检测**

因为现在把重点放在产品交付的质量上来了(而不是在于找到了多少 BUG)， 预防实践和静态分析仪这样的检测工具将成为主流。

◆ **仿真测试**

仿真工具变得很普遍，使得仿造计算机环境变得容易起来。在开发过程的早期就可以进行意外和错误流程的测试。代码稳定后，再用真实环境验证仿真是否准确无误。

◆ **及时的测试用例**

庞大的测试用例管理系统将成为昔日的东西，大量的测试用例生成了却没有被使用。测试用例将不再像腐烂的存货一样被收藏起来，因此，让测试用例保持最新变得容易起来。

◆ **积极的方法**

误导人的方法，比如计算 BUG 的数量、计算测试用例的数量，将不复存在。有用的方法，比如需求覆盖、模型覆盖、代码覆盖将驱动项目开发。

◆ **更少更精的测试人员**

机器将代替测试人员做大部分他们以往创建测试所做的繁琐工作，测试小组需要比以往更少的测试人员，留下来的测试人员将是经过更多高度培训过的。他们所做的工作将更加有趣，因为在测试中他们将致力于更大的问题，而不是在抱怨中艰难地开展工作。

◆ **更多更好的测试**

测试人员将可以在一天中进行成千上万的测试，所以，如何首先运行最有用的测试将成为一大挑战。相关的工具将允许测试人员为他们的测试区分优先级，以及将测试目标放在那些最易出现重大 BUG 的地方。

◆ **测试人员的角色更换**

测试中界限模糊 ，在测试领域工作使得专职测试的人员和专职创建测试工具的人员界限模糊，一个既是“通过程序破坏事物的测试员”又是“创建程序用于破坏事物的程序员”

的专业出现了，——关于如何称呼这个新的专业，新闻圈内的人们还在进行着无休止的争论。

测试与开发界限模糊， 测试人员与开发人员一前一后，共同创造可测试的、高质量的代码。测试人员帮助开发人员消除需求中的问题，使得开发人员的工作更易完成，同时，开发人员写出更清晰、可测性更高的代码，使得测试人员的工作更易完成。

顾客反馈与测试合为一体 ，交付的产品质量更高。测试人员进行根本原因的分析，我们会问比如“我们怎么会遗漏了这个 BUG 呢？”或者“我们将来如何防止这类 BUG？”这些问题，我们的工作就是使顾客满意。

◆ **新的挑战出现**

复杂和相互关联的计算机世界使得了测试安全这一类的新问题让测试人员不断努力工作，但这没关系——因为这些挑战使测试人员精力充沛。

◆ **测试人员获得尊重**

测试人员将不再是在最后时刻才被叫来“对产品狂轰烂炸”，他们将在整个软件开发过程中提供一个可见的、重要的、增值的服务。人们意识到，测试是有益的、有趣的甚至富有乐趣。

◆ **测试变得流行**

软件测试人员开始扬眉吐气，而且，由于破坏事物至少可以带来创建事物一样的乐趣，人们开始在开发和测试角色之间转换，所有的人将学到更多关于如何得到良好代码的知识。

◆ **激情“吸毒者”继续存在**

新的过程运行得如此良好，使得需求撰写者，开发人员以及测试人员不再具有生命力，这就使得那些在激情掌控的世界被提升的人惶惶不可终日，那样的世界意味着工作到深夜、最后一刻测试才参与，以及如同交战开火般的会议。而这些人对于那些还没有受新的运行过程控制的公司来说还具有吸引力。

◆ **测试人员该怎么做**

不管我的预测是否成为现实，未来也会按照它自己的方式到来，下面就是如何准备面临未来的五个意见：

- 不要接受测试的现状，四处看看，并且思考“我们在做些什么毫无意义的事情？”
- 领悟如何更好的测试，并且分享这些知识。只有每一个人都试图使他所写的代码达到最佳状态时，整体质量才会改进。
- 行业受软件测试的创新思维激发。用参加会议，加入邮件列表，网上冲浪，这些方式来解在测试前沿发生的一切。
- 参加一个编程学习班，即使你不打算编写大量的代码。将学习班当作是在BUG 领土上的一次侦察飞行。
- PC 先驱 Alan Kay 所言：“预测未来的最好方式就是开创未来”。

第 2 章 软件质量体系

第 1 节软件能力成熟度模型：CMM

CMM 为企业的软件过程能力提供了一个阶梯式的进化框架，阶梯共有五级。第一级只是一个起点，任何准备按 CMM 体系进化的企业都自然处于这个起点上，并通过它向第二级迈进。除第一级外，每一级都设定了一组目标，如果达到了这组目标，则表明达到了这个成熟级别，可以向下一级别迈进。

从纯粹的个人行为发展到有计划有步骤的组织行为…

第一级：初始级(Initial)；

第二级：可重复级(Repeatable)；

第三级：已定义级(Defined)；

第四级：受管理级(Managed)；

第五级：优化级(Optimizing)。

2.1.1初始级

初始级的软件过程是未加定义的随意过程，项目的执行是随意甚至是混乱的。也许有些企业制定了一些软件工程规范，但若这些规范未能覆盖基本的关键过程要求，且执行没有政策、资源等方面的保证时，那么它仍然被视为初始级。

- **关注点：**

  工作方式处于救火状态，不断的应对突如其来的危机；

  工作组：软件开发组、工程组；

- **提高：**

  需要建立项目过程管理，建立各种计划，开展 QA 活动。

2.1.2可重复级

根据多年的经验和教训，人们总结出软件开发的首要问题不是技术问题而是管理问题。因此，第二级的焦点集中在软件管理过程上。一个可管理的过程则是一个可重复的过程，可重复的过程才能逐渐改进和成熟。可重复级的管理过程包括了需求管理、项目管理、质量管理、配置管理和子合同管理五个方面；其中项目管理过程又分为计划过程和跟踪与监控过程。通过实施这些过程，从管理角度可以看到一个按计划执行的且阶段可控的软件开发过程。

- **关注点：**

规则化

引入需求管理、项目管理、质量管理、配置管理、子合同管理等；

引入工作组：测试组、评估组、质量保证组、配置管理组、合同组、文档支持组、培训组；

- **提高：**

  SEPG、建立软件过程库和文档库。

2.1.3已定义级

在可重复级定义了管理的基本过程，而没有定义执行的步骤标准。在第三级则要求制定企业范围的工程化标准，并将这些标准集成到企业软件开发标准过程中去。所有开发的项目需根据这个标准过程，裁剪出与项目适宜的过程，并且按照过程执行。过程的裁剪不是随意的，在使用前必须经过企业有关人员的批准。

- **关注点：**

  文档化，标准的一致的；

  软件过程标准化文档化，质量可以得到控制；

  工作组： SEPG、软件评估组。

- **提高：**

  对软件过程定量分析，加强质量管理。

2.1.4已管理级

第四级的管理是量化的管理。所有过程需建立相应的度量方式，所有产品的质量（包括工作产品和提交给用户的最终产品）需要有明确的度量指标。这些度量应是详尽的，且可用于理解和控制软件过程和产品。量化控制将使软件开发真正成为一种工业生产活动。

- **关注点：**

  量化，可预测的；（自此，软件开发变成一种工业生产活动。）

  软件过程具有精确的评测方法，量化的控制软件过程的产品和质量，可根据’意外情况”确定出错的原因；

  工作组： 定量过程管理组；

- **提高：**

  防止和规避缺陷的能力，技术革新的能力，过程改进。

2.1.5优化级

优化级的目标是达到一个持续改善的境界。所谓持续改善是指可以根据过程执行的反馈信息来改善下一步的执行过程，即优化执行步骤。如果企业达到了第五级，就表明该企业能够根据实际的项目性质、技术等因素，不断调整软件生产过程以求达到最佳。

- **关注点：**

  持续改善；

  工作组：缺陷防范活动协调组、技术改革管理活动组、软件过程改进组；

- **改进：**

  软件过程优化。

第 3 章 软件生命周期

第 1 节软件生命周期

3.1.1需求管理（软件需求）Requirements Management

3.1.1.1需求的内容规范

需求应当具有一下特点：

- ✧ 完整性：每一项需求都必须将所要实现的功能描述清楚，以使开发人员获得设计和实现这些功能所需的所有必要信息。
- ✧ 正确性：每一项需求都必须准确地陈述其要开发的功能。
- ✧ 一致性：一致性是指与其它软件需求或高层（系统，业务）需求不相矛盾。
- ✧ 可行性：每一项需求都必须是在已知系统和环境的权能和限制范围内可以实施的。
- ✧ 无二义性：对所有需求说明的读者都只能有一个明确统一的解释，由于自然语言极易导致二义性，所以尽量把每项需求用简洁明了的用户性的语言表达出来。
- ✧ 健壮性：需求的说明中是否对可能出现的异常进行了分析，并且对这些异常进行了容错处理。
- ✧ 必要性：“必要性”可以理解为每项需求都是用来授权你编写文档的“ 根源 ”。要使每项需求都能回溯至某项客户的输入，如 Use Case 或别的来源。
- ✧ 可测试性：每项需求都能通过设计测试用例或其它的验证方法来进行测试。
- ✧ 可修改性：每项需求只应在 S R S 中出现一次。这样更改时易于保持一致性。另外，使用目录表、索引和相互参照列表方法将使软件需求规格说明书更容易修改。
- ✧ 可跟踪性：应能在每项软件需求与它的根源和设计元素、源代码、测试用例之间建立起链接链，这种可跟踪性要求每项需求以一种结构化的，粒度好（ f i n e - g r a i n e d ）的方式编写并单独标明，而不是大段大段的叙述。
- ✧ 优先级。

3.1.1.2需求建模

需求的建模包括把需求转换成图形模型或形式化语言模型。需求的图形化分析模型包括数据流图（ Data Flow Diagram ， DFD ）、实体关系图（ Entity-Relationship Diagram ， ERD ）、状态转化图（ State-Transition Diagram ， STD ）、对话图（ Dialog Map ）和类图（ Class Diagram ）。这些图形化模型一般都需要借助一定的工具。选择好的分析工具应该有助于获得需求的质量特性，包括：有效性、一致性、可靠性、可存活性、可用性、正确性、可维护性、可测试性、可扩展性、可交互性、可重用性、可携带性等。

3.1.1.3审查

需求必须经过产品经理、软件经理、系统测试组、软件工程组、系统工程组、质量保证组、软件配置管理组、文档支持组等小组的审查。

通过静态手工方法进行需求测试中最常使用的手段是同行评审。

3.1.1.4执行

- 建议需求文档，分配需求(分配需求软件工程组制定软件计划的基础)，修改需求。
- 需求的管理需要在应用领域和软件工程方面经验比较丰富的人来担任。
- 建议使用配套的需求管理工具
- 除了建立相关文档，还需要对所有软件工程组人员进行项目应用领域的培训。

3.1.1.5需求变更

变更审查：

- 变更对现有约定的影响；
- 提出需求变更引起的后续软件活动变更，评估风险，建立文档，全程跟踪。

3.1.1.6交付工件

- 程序陈述和需求说明说(包括用户需求和技术需求)；
- 需求文档的分类：用户需求 CR，技术需求 TR，项目需求 PR；
- 需求的状态：已批准，已分配，已实现…

3.1.2软件项目计划 Software Project Planning

为软件工程的运作和软件项目活动的管理提供合理的基础和可行的工作计划。

3.1.3设计阶段

包括功能的描述和设计
交付工件：设计说明说和功能说明书

3.1.4编码阶段

包括实时源代码，队目标代码进行单元测试
交付工件：软件，文档和产品信息

3.1.5核实阶段

包括各种测试行为

3.1.6系统维护阶段

交付工件：基于设计和技术或用户需求的测试说明计划和结果等

第 2 节软件开发过程中常见的问题

- 需求说明差
- 不切实际的时间表
- 测试不充分
- 不断增加功能
- 交流问题

第 3 节流程中的组及工作

3.3.1流程中的组

- **系统分析人员**
  提出测试需求并跟踪，确定测试的对象/方法和范围。
- **软件开发人员**
  提供开发计划 SDP，参与制定和评审测试计划；
  提供软件功能需求规格/需求分析/测试建议等文档，参与制定和评审测试方案和案例；
  响应测试需求，跟踪解决缺陷。
- **测试人员**
  略
- **配置管理人员**
  对测试文档测试代码及相关配置项进行配置管理。
- **质量保证人员**
  质量保证，参与相关评审。由于质量保证和测试关系比较密切，多写一点。

SQA 的职责 ：

保障软件组织流程体系得到遵守、促使软件组织过程改进、指导项目实施流程、增加开发活动透明度、评审项目活动、审核工作产品、协助工作产品问题解决、度量数据采集、分析、提供决策参考、进行缺陷预防、实现质量目标。

- 组织和协调产品开发组内部的软件技术和开发标准、流程的培训和教育
- 部门的和特定产品的软件开发过程度量（ Metrics ）以及软件产品质量的度量（ Metrics ）
- 指出产品开发过程中应该遵循的有关软件开发的标准和流程，并监督开发过程对标

准和流程的符合度；

- 软件质量管理，采用 Inspection、Review 和 Audit 技术
- 通过软件开发流程及标准的推行以及对软件开发过程的不断总结和优化，使软件开发过程得到持续不断的优化和提高

◆ **其他人员**

如： PM/项目组/测试组/SQA/SEPG/SCM/售前/售后/市场/等等都要参与到项目中。

3.3.2测试的组间协作

测试人员，需求撰写人员和开发人员，都是软件流程中的一份子。

◆ **测试人员帮助需求撰写人员**

测试人员与需求撰写人员共同工作，在需求完成以后，审查以及理解需求。早期的审查以及建模可以暴露很多关于一致性、完整性和模糊性的BUG，这个时候修补这些 BUG 付出的代价还十分小。

◆ **需求撰写人员帮助测试人员**

测试小组建造模型，用于产生对其产品行为的测试。需求撰写人员审查模型，以确保他们充分覆盖了产品特征集。这样产生的测试模块将成为一个”可执行需求”。

◆ **测试人员帮助开发人员**

因为需求清楚，毫不含糊，开发人员更好的理解了他们的代码将要完成什么。在正式的将代码提交做测试之前，测试人员提供给开发人员一些模型，以便开发人员可以在自己的代码中实现它们。

◆ **开发人员帮助测试人员**

基于”特征对特征”这样的方式(防止以往的”后期才介入开发，一股脑找出产品问题”的方式)，开发人员和测试人员共同保证代码易于实施自动测试。开发人员的代码中处处都是易测试性的开关，使得错误检测更加容易。

◆ **测试人员帮助测试人员**

测试用一种高级语言来模拟，因此别的特征的测试小组(甚至别的产品的测试小组)可以复查和改进测试模型。这就形成了一个测试专家的共同体。

第 4 章 软件测试基础

第 1 节测试理论

4.1.1测试定义

IEEE 中对测试的定义：使用人工或自动手段来运行或测定某个系统的过程，其目的在于检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差别。

4.1.2测试前提

- ✧ 软件可测试性：是一个计算机程序能够被测试的容易程度。
- ✧ 软件可测试性检查表：
- ✧ 可操作性－运行地越好，被测试的效率越高。
- ✧ 可观察性－所看见的，就是所测试的。
- ✧ 可控制性－对软件的控制越好，测试越能够被自动执行与优化。
- ✧ 可分解性－通过控制测试范围，能够更好地分解问题，执行更灵巧的再测试。
- ✧ 简单性－需要测试的内容越少，测试的速度越快。
- ✧ 稳定性－改变越少，对测试的破坏越小。
- ✧ 易理解性－得到的信息越多，进行的测试越灵巧。

4.1.3测试目的

目的：发现程序中的错误，提高产品可靠性。

4.1.4测试规律

规律：木桶原理/八二原则。

4.1.4.1木桶原理

产品质量的关键因素是分析、设计和实现，测试应该是融于其中的补充检查手段，其他管理、支持、甚至文化因素也会影响最终产品的质量。应该说，测试是提高产品质量的必要条件，也是提高产品质量最直接、最快捷的手段，但决不是一种根本手段。反过来说，如果将提高产品质量的砝码全部押在测试上，那将是一个恐怖而漫长的灾难。

4.1.4.2八二原则

说法一：在分析、设计、实现阶段的复审和测试工作能够发现和避免80%的 Bug，而系统测试又能找出其余 Bug 中的 80%，最后的 5%的 Bug 可能只有在用户的大范围、长时间使用后才会曝露出来。因为测试只能够保证尽可能多地发现错误，无法保证能够发现所有的错误。

说法二：80% 的程序缺陷常常生存在软件 20% 的程序空间里。

4.1.5测试原则

- ✧ 测试的工作是有计划的；
- ✧ zero-bug 是一种理想，good-enough 是我们的原则，应该在软件测试成本和软件质量效益两者间找到一个平衡点；
- ✧ 不应测试自己开发的程序；
- ✧ 尽早的进行，前期就要介入；<美国国防部(千行代码出错率小于 0。01) >

时期	缺陷百分比(单位%)
需求分析	9。0
概要设计	17。2
详细设计	21。6
编码和单元测试	16。4
集成测试	19。4
系统联调和系统测试	16。4
总计	100

要尽早的发现缺陷和修正缺陷主要有以下原因：

- ➢ 缺陷的修改成本随着阶段的推移将急剧上升，在产品发布之后修正一个缺陷的成本将是需求阶段的 100 倍，甚至更高；
- ➢ 缺陷具有放大的特点，缺陷修改延迟几个星期甚至几个月将使得系统更容易出错(”软件缺陷具有生长和生育的能力。”)；
- ➢ 设计判定和一些小的代码限制及条件很容易被忘掉；
- ➢ 尽早修正缺陷可以节省重新分析设计的时间；
- ➢ 早期的问题反馈有助于防止类似错误的产生；
- ➢ 大量的缺陷和问题跟踪工作将被减轻；
- ➢ 如果必要的话，可以重新设计和编码，而这个工作越往后期越不可能；
- ➢ 需求和设计时出现的缺陷占很大的比例。

- ✧ 测试都应追溯到用户需求。正如我们所知：软件测试的目标在于揭示错误。而最严重的错误（从用户角度来看）是那些导致程序无法满足需求的错误；
- ✧ 测试设计和测试操作进行分离；
- ✧ 软件缺陷具有免疫性，必须更换不同的测试方式和测试数据。在软件测试中采用单一的方法不能高效和完全的针对所有软件缺陷，因此软件测试应该尽可能的多采用多种途径进行测试；
- ✧ 测试本身应该被测试；
- ✧ 测试和质量关系密切，质量活动需要有规划和监控及明确的输出。

4.1.6主要内容

- ✧ 测试要考虑到所有的出错可能性。同时要做一些不是按常规做的、非常奇怪的事。
- ✧ 除了漏洞之外，测试还应考虑性能问题，保证软件运行良好，非常快，没有内存泄露，不会出现软件运行越来越慢的情形。
- ✧ 测试要考虑软件的兼容性。

4.1.7不利因素

测试可能存在的不利因素：

- × 没有得到足够的培训
- × 心里准备不足
- × 缺乏测试工具
- × 缺乏管理的标准和支持
- × 缺乏客户和最终使用者的参与
- × 没有足够的时间进行测试
- × 对于独立的测试人员过度信任
- × 版本改变的太快
- × 测试人员处于不受重视的情况中
- × 不能说不

第 2 节测试生命周期

4.2.1测试生命周期

- ➢ 对测试人员进行业务培训
- ➢ 测试需求分析
- ➢ 编写测试计划
- ➢ 编写测试案例
- ➢ 测试执行(包括缺陷跟踪)
- ➢ 编写测试报告

备注：需要重视各步骤结束前的评审工作。

4.2.2流程中的文档

4.2.2.1测试计划

测试计划和产品开发紧密相关，由多个部分组成。所有大型的商业软件都需要完整的测试计划，需要具体到每一个步骤，并且每一个部分都要符合规范要求。

测试计划包括内容：1) 概述；2) 测试目标和发布标准；3) 计划将测试的领域；4) 测试方法描述；5) 测试进度表；6) 测试资源；7) 配置范围和测试工具 。

4.2.2.2测试规范

测试规范是指微每一个在测试计划中确定的产品领域所写的文档，用来描述该领域的测试需求。编写测试规范，需要参照项目经理写的产品规范，开发人员写的开发计划。每个领域都应该有一份详细的测试规范，所以还需要参照测试计划。

测试规范包括的内容：1) 背景信息；2) 被测试的特性；3) 功能考虑；4) 测试考虑； 5) 测试想定 。

4.2.2.3测试案例

测试案例是指描述如何测试某一个领域的文档，这些文档符合测试规范中的需求说明。根据测试规范的测试想定(scenario)开发，根据测试反馈信息，对于没有考虑到的新问题，不断添加测试案例。 测试案例没有固定格式，只要清楚表明了测试步骤和需要验证的事实，使得任何一位测试人员都可以根据测试案例的描述完成测试。

4.2.2.4测试报告

测试管理人员以测试报告的形式向整个产品开发部门报告测试结果及发现的缺陷或错误。撰写测试报告的目的是为了让整个产品开发部门了解产品开发的进展情况，以使缺陷或错误能够迅速得到修复。 测试报告的格式并无定式，要求能够完整、清楚地反映当前的测试进展情况，要易懂，不要使人迷惑或产生误解。

4.2.2.5缺陷或错误报告

测试人员以缺陷或错误报告的形式向开发人员报告所发现的缺陷或错误。撰写缺陷或错误报告的目的是为了使缺陷或错误能够得到修复，测试人员的缺陷或错误报告撰写的好坏会直接影响到开发人员对缺陷或错误的修复。

一份缺陷或错误报告应该包括的几个要点：1) 缺陷或错误名称；2) 被测试软件的版本；3) 优先度与严重性；4) 报告测试的步骤；5) 缺陷或错误造成的后果；6) 预计的操作结果；7) 其他信息。

4.2.3软件开发测试模型

常用 V 模型，对于 X、W 等模型没有做深入研究。

第 3 节测试人员的责任

测试人员的任务就是站在使用者的角度上，通过不断地使用和攻击刚开发出来的软件产品，尽量多地找出产品中存在的问题。

4.3.1测试启动需确定的工作

4.3.1.1需求阶段

- 确定测试策略；
- 确定收集了足够的需求；
- 产生功能性的测试用例；
- 需要接收的资料
- 需求规格说明书；
- 产品文档等；

4.3.1.2设计阶段

- 确定设计和需求之间的联系；
- 确定进行了足够的设计；
- 产生结构和功能的测试用例；

需要接收的资料

- 输入说明；
- 过程说明；
- 文件说明；
- 输出说明；
- 控制说明；
- 系统流程图；
- 硬件和软件的需求；
- 操作手册说明书；
- 数据保留的策略；

4.3.1.3编码阶段

- 确定和设计之间的联系；
- 确定拥有执行的足够条件；
- 产生结构和功能的测试用例；

需要接收的资料

- 编码说明书；
- 程序文档；
- 计算机程序列表；
- 可执行的程序；
- 程序流程图；
- 操作介绍；
- 单元测试结果；
- 测试阶段
- 确定设计了足够的测试用例；
- 测试应用系统已经完成，并且可测；
- 关键资源已经到位；

4.3.1.4维护阶段

- 缺陷的跟踪；
- 新的版本测试；

4.3.2测试需完成的工作

4.3.2.1需求评审

确定需求是足够的；

4.3.2.2制定测试计划

- -确定测试需求
- -评估风险
- -制定测试策略
- -确定测试资源
- -创建时间表
- -生成测试计划

4.3.2.3设计测试

- -准备工作量分析文档
- -确定并说明测试用例
- -确定测试过程，并建立测试过程的结构

- -复审和评估测试覆盖

4.3.2.4实施测试

- -记录或通过编程创建测试脚本
- -确定设计与实施模型中的测试专用功能
- -建立外部数据集
- 执行测试
- -执行测试过程
- -评估测试的执行情况
- -恢复暂停的测试
- -核实结果
- -调查意外结果
- -记录缺陷

4.3.2.5对测试进行评估

- -评估测试用例覆盖
- -评估代码覆盖
- -分析缺陷
- -确定是否达到了测试完成标准与成功标准

第 4 节测试方法和分类

本节只做一个概括指出，下一章将详细讲解。

4.4.1测试分类

- 白盒测试；
- 黑盒测试。

4.4.2黑盒测试的测试用例设计方法：

- 等价类划分方法；
- 边界值分析方法；
- 错误推测方法；
- 因果图方法；
- 判定表驱动分析方法；
- 正交实验设计方法；
- 功能图分析方法 <流程场景(重要执行通路/出错处理通路) >；

4.4.3系统测试类型

- 恢复测试；
- 完整>安全>性测试；
- 强度测试<使资源出现短缺的情况，检查系统的应对>；
- 容量测试；
- 结构测试；
- 性能测试<疲劳测试/压力测试/响应时间>；
- 配置测试；
- 安装、卸载测试；
- 用户界面测试<界面友好性>；
- 功能测试<正确性测试/容错性（健壮性）测试>；
- 比较测试；
- 可移植性<兼容性测试>；
- 接口间测试；
- 数据库测试。

第 5 节软件产生错误的原因

4.5.1程序开发产生缺陷的原因

- 需求不清晰；
- 软件复杂性；
- 程序编码错误；
- 需求变化；
- 时间压力；
- 代码文档贫乏；
- 开发工具自身错误；

4.5.2测试导致缺陷的原因

- 测试目标定义错误；
- 在开发生命周期中，错误的选择了测试介入时期；
- 选择了低效的测试技术；
- 测试人员专业知识培训不够，工作低效；
- 计划不够详细，测试的随意性很大；
- 测试人员同开发人员沟通困难。

4.5.3程序缺陷包含的因素

生产软件的最终目的是为了满足客户需求，我们以客户需求作为评判软件质量的标准，软件缺陷（ Software Bug ）的具体含义包括下面几个因素：

- 软件未达到客户需求的功能和性能；
- 软件超出客户需求的范围；
- 软件出现客户需求不能容忍的错误；
- 软件的使用未能符合客户的习惯和工作环境。

4.5.4软件缺陷包含的因素

软件缺陷除了包括 程序运行时出现问题上来问题，还包括考虑到设计等方面的因素。我们还可以认为软件缺陷还可以包括软件设计不符合规范，未能在特定的条件（资金、范围等）达到最佳等。

所以说，认为软件测试仅限于程序提交之后是错误的。

第 6 节关于缺陷

4.6.1常规测试点

- **输入/输出的测试要点**
  - 文件属性是否正确？
  - 打开文件语句是否正确？
  - 格式说明书与输入/输出语句是否一致？
  - 缓冲区大小与记录长度是否匹配？
  - 使用文件之前先打开文件了吗？
  - 文件结束条件处理了吗？
  - 输入/输出错误检查并处理了吗？
  - 输出信息中由文字书写错误吗？
- **局部数据结构的测试要点**
  - 错误的或不相容的说明
  - 使用尚未赋值或尚未初始化的变量
  - 错误的初始值或不正确的缺省值
  - 错误的变量名字（拼写错或截短了）
  - 数据类型不相容
  - 上溢、下溢或地址异常
- **计算中的常见错误**
  - 计算次序不对或误解了运算符的优先次序
  - 混合运算（运算对象的类型彼此不相容）
  - 变量初始值不正确
  - 精度不够

- ✧ 表达式的符号表示错误
- ◆ **测试方案中的错误**
- ✧ 比较数据类型不同的量
- ✧ 逻辑运算符不正确或优先次序的错误
- ✧ 当由于精度问题两个量不会相等时，程序中却期待着相等条件的出现
- ✧ “差 1”错（即，多循环一次或少循环一次）
- ✧ 错误的或不存在的循环终止条件
- ✧ 当遇到发散的迭代时不能终止循环
- ✧ 错误地修改循环变量
- ◆ **评价出错处理时的常见错误**
- ✧ 对错误的描述是难于理解的
- ✧ 记下的错误与实际遇到的错误不同
- ✧ 在对错误进行处理之前，错误条件已经引起系统干预。
- ✧ 对错误的处理不正确

描述错误的信息不足以帮助确定造成错误的位置。

4.6.2为什么缺陷很难找

- × 需求解释有错误；
- × 用户定义错了需求；
- × 需求记录错误；
- × 设计说明有误；
- × 编码说明有误；
- × 程序代码有误；
- × 数据输入有误；
- × 测试错误；
- × 问题修改不正确；
- × 正确的结果是由于其它的缺陷产生的。

4.6.3缺陷的提交艺术

Bug report 的核心是对错误的描述。编写好的 bug report 是一种好的艺术形式。采用以下的 10 条技巧可以帮助你的小组提高编写 bug report 的质量：

- ✧ 组织 Structure：测试人员应该采用深思熟虑的，小心谨慎的方法执行测试，并且做详尽的记录。这样可以促使他们对测试下的系统有很好的认识。当错误发生的时候，一个有组织的测试人员能够知道最早出现问题的地方。
- ✧ 重现 Reproduce：测试人员在编写 bug report 之前必须在检查问题是否可重现。如果错误不可再重现，仍然应该写下来，但是必须说明问题的偶然性。一个好的处理原则就是在编写 bug report 之前反复尝试 3 次。
- ✧ 隔离 Isolate：在尝试编写 bug report 之前，必须试着隔离错误。可以采用改变一些变量的方法，如系统的配置，它可能可以改变错误的症状。这些信息可以为开发人员着手调试提供思路。
- ✧ 归纳 Generalize：在测试人员发现了一个已隔离的，可重现的问题后，应该对问题进行

归纳。同一个问题是否出现在其他的模块或其他的地方？同一个故障是否有更加严重的问题？

- 对比 Compare：如果测试人员以前曾经验证过现在出错的测试用例，那么他就应该检查以前的测试结果以检查相同的条件是否通过以前的测试。如果是的话，那么这个问题就象是一个回归的错误。注意由于同一测试条件有可能出现在多个测试用例中，这个步骤就不仅仅只是检查一个测试用例在以前的多个结果。
- 总结 Summarize：在 bug report 的第一行写上错误的总结是非常关键的。测试人员要花些时间思考已发现的错误对客户有何影响。这不仅仅要求测试人员编写的报告要能够吸引读者，使和管理层的沟通清晰，还要能够帮助设置错误修复的优先级别。
- 精简 Condense：在 bug report 的初稿完成后，测试人员应该反复阅读它，集中剔除那些没有关系的步骤或词语。隐含的或模糊的说明和那些由于对没有任何关系的细节或者那些在重现错误过程中不需要的步骤而消磨报告欢迎程度的无穷唠叨都不是bug report 的目标。
- 消除歧义 Disambiguate：测试人员在精简空话的同时或其之后随即应该再仔细检查报告是否有会产生误解的地方。测试人员应该尽量避免使用模糊的，会产生歧义的和主观的词语。目标是使用能够表述事实，清楚的，不会产生争执的词语。
- 中立 Neutralize：如文中所述，作为坏消息的传递人，和善地提交消息是一个挑战。如同所有的错误总结一样，独立的 bug report 在措辞方面应该保持公正。攻击开发人员，指责潜在的错误，企图诙谐或使用挖苦将引起开发人员的憎恶，并且使注意力从"提高产品质量"这个大的目标上转移开了。谨慎的测试人员只用Bug report 来描述事实。
- 检查 Review：一旦测试人员感觉 bug report 是他能够编写的最好版本，他应该将报告再给一个或多个同行进行检查。他的同事们也应该给出一些建议，为了澄清问题不断地提问，如果适当的话，甚至可以挑战"错误成灾"的结论。在允许的时间里，测试小组应该尽可能提交最好的 bug report。

4.6.4衡量优秀的 bug report 的质量指标

- 对管理层来说，是清晰明了的，特别是在概要这一级；
- 对于开发部门是有用的，主要是给出能够让开发人员高效地调试问题的相关信息。可以很快的将 bug 从"Opened"状态转变成"Closed"状态，减少为得到更多的信息从开发人员打回的差的 bug report 并导致测试人员返工的时间。

4.6.5缺陷的生命周期

4.6.5.1BUG 生命周期（TestDirector）：

bug 生命周期：new-→open-→fixed-→close。

- 由测试员发现缺陷（defect），并加入缺陷，些缺陷状态为 new；
- 由项目管理把缺陷 new 状态置为 open，把缺陷公布出来；
- 开发者修复缺陷后，把缺陷由 open 置为 fixed，如果拒绝修改可以置为 rejected；
- 上缺陷的发现人员对缺陷进行回归测试，如果修改正确，把缺陷状态由fixed 置为 closed，如果缺还是存在，则置为 Reopen。

4.6.5.2缺陷生命周期(ClearQuest)：

- 测试人员
  - 增加一个新的 Defect 记录到 ClearQuest 中，并指定提交给项目经理，此时 Defect 的状态为 submitted.
- 项目经理
  - 查询该 Defect 记录，通过 assign 动作分发，把 Defect 的状态改为 assigned；
  - 如果拒绝修改可以置为 rejected；
  - 如果需要延期修改把问题置为 postponed 状态。
- 开发人员
  - 开发人员应在第一时间查看自己的Defect，通过open动作确认该Defect，此时 Defect 的状态由 assigned 改为 opened；如果不是属于自己的 Defect 或不做修改,则通过 rejected 动作将 Defect 的状态由 assigned 改为 rejected,并把 owner 修改为项目经理，项目经理将该 Defect 通过 reassign 重新发往新的 owner。
  - 开发人员如果认为需要延期修改，则通过 rejected 动作将 Defect 的状态由 assigned 改为 rejected,并把 owner 修改为项目经理。由项目经理把问题置为 postponed 状态。
  - 开发人员在 open 该 Defect 后，应根据缺陷的严重程度以及解决的优先级，1 类的致命错误和 2 类的严重错误和优先级为紧急的，应立即着手解决，并且不得将这类 Defect 带入下一次新版本的迭代，特殊情况的需报告项目经理；其他类的错误也应按优先级的要求逐一进行解决。
  - 开发人员修改程序，解决 Defect 后，同时在 ClearQuest 上通过 modify 把导致 Defect 的原因和解决方法记录下来,同时将 Defect 的状态由 opened 改为 resolved。
- 测试人员
  - 进行确认测试，若成功，把 Defect 的状态改为 closed，该条测试记录的生命周期完成；若错误继续存在（或拒绝复审），则测试人员修改测试记录，通过 reassign 再次发布该 Defect 给相应的开发人员。重复第 3 步至第 4 步直到该 Defect 真正解决完毕。 若需要延期复审,则测试人员修改测试记录,问题状态不变。

4.6.5.3缺陷生命周期内，注意事项：

- 测试人员只需要接收 fixed 和 rejected 状态的缺陷,其他缺陷无论任何原因流转到测试人员,都可以直接驳回(并描述驳回原因)。
- 每个状态发生变化时,相关处理人员必须填写缺陷修改的相关说明.如果发现上一状态没有完整填写说明或描述不清,可以直接驳回(并描述驳回原因)。
- 开发人员要配合测试人员的工作，遵守测试管理规范的流程，尽快解决自己的软件 Defect,对于不能按要求解决的，应以一定的形式通过项目经理及测试人员知照。
- QA（质量保证）人员应定期的检查开发人员和测试人员是否遵守测试管理规范的流程,出现违反规定的情况应告知测试组长或项目经理。

4.6.6缺陷类型定义

4.6.6.1缺陷按严重性分类

本规范定义以下五类缺陷：

A 类——致命错误，不能完全满足系统要求，基本业务功能未实现，系统崩溃、不稳定或挂起等导致系统不能继续运行。

- 系统崩溃，死机，非法退出，无法继续操作，或引起其他软件系统出错。
- (如：操作系统崩溃，其他软件崩溃，执行主流程时，数据库发生死锁)
- 业务流程或重要功能错误(如：主要流程某对象状态发生错误，严重的数值计算错误等)。
- 数据通讯完全错误； 与数据库连接错误。
- B 类——严重错误，严重地影响系统要求或基本功能的实现，且没有办法更正（重新安装或重新启动不属于更正办法）。使系统不稳定、破坏数据、产生错误结果，部分功能无法执行。
- 一般性功能不符，业务流程不正确，需求没有实现。
- 重要流程和场景下，导致数据错误，操作无效，操作结果错误。
- 程序接口错误。
- 造成数据库不稳定的错误。

C 类——一般性错误。

- 界面错误(严重的界面提示错误或不友好表现)。
- 非重要功能无法正确执行， 实现不正确， 实现不完整，但不影响一起功能(如删除时没有考虑数据关联，对其它模块造成影响； 系统界面上，一些可接受输入的控件点击后无作用；对数据库的操作不能正确实现)。
- 非严重性产生错误结果，但不影响一起功能。
- 正确性不受影响，但系统性能和响应时间受到影响。

D 类——轻微错误，使操作者不方便或遇到麻烦，但它不影响执行工作功能或重要功能，或对最终结果影响有限的问题。

- 系统处理需要优化。
- 输入限制未放在前台进行控制，或控制错误。
- 增删改等功能，在次要界面不能实现，但在主要界面可以实现。
- 界面定义不一致，界面定义不规范， 显示格式不规范。
- 提示文字，没有，不明确，不简明， 不清楚，不正确，未采用标准术语。(如： 重要删除操作未给出提示。
- 可编辑区和不可编辑区不明显； 必填项与非必填项应加以区别。
- 键盘支持不好。(如在可输入多行的字段中，不支持回车换行)。
- 界面不能及时刷新，影响视觉效果。
- 滚动条无效。
- 光标跳转设置不好，鼠标（光标）定位错误。

E 类——测试建议（非缺陷）。

- 系统各个位置初始值的建议。
- 流程优化建议等等。

4.6.6.2缺陷按技术种类分类

类　别	描　　述
功能性错误	列在说明中的需求没有在最终系统中达到
系统错误	存在或产生于所开发的系统之外的软硬件错误
逻辑错误	程序运行起来不像要求的样子
用户界面错误	字段和控件标号不一致，功能提供的不一致等
数据错误	访问数据库时出错
编码错误	源代码中存在的语法错误
测试错误	测试者误操作却认为发现了问题

4.6.6.3测试标准

待测试系统的质量要求

软件系统测试时，发现一级错误（大于等于1）、二级错误（大于等于2）暂停测试返回开发。

软件测试合格须符合以下标准

单位： 千行源代码（KLOC）

A 类错误	B 类错误	C 类错误	D 类错误	E 类建议
无	无	2	4	暂不作要求

对于随机出现的 BUG 的数量也必须考虑，软件产品未经测试合格，不允许投运。

4.6.6.4国标中有关缺陷数量的描述

程序中不存在未改的 A、B 级 BUG；C 级 BUG 的数量每千行源代码（KLOC）中不超过 1 个；D、E 级 BUG 的数量每千行源代码（KLOC）中不超过 2 个；

在交付给用户的文档资料中，允许存在的 BUG 数量按以下方法计算：用程序的千行源代码（KLOC）数量除以 25，所得数加上 3 即为文档中允许存在的最大 BUG 数量。例如，如果程序的千行源代码（KLOC）的数量是 1000，即该程序有 1 000 000 行源程序，则与该程序相关的文字资料中允许的最大 BUG 数就是（1000/25+3=）43 个。

4.6.7常用词汇

待整理…

第 5 章 测试的方法和分类

//欲知更详细的内容，请联系页脚中的邮箱。

第 1 节测试方法

白盒测试和黑盒测试

5.1.1白盒测试

白盒测试，也称为结构化测试、基于代码的测试，是一种测试用例设计方法，已知产品的内部工作过程，通过测试证明每种内部操作是否符合设计规格要求。

它基于程序的控制结构；是基于一个应用代码的内部逻辑知识；基于覆盖全部代码、分支、路径、条件，导出测试用例。

- **白盒测试产生的测试用例检查点：**
  - 保证一个模块中的所有独立路径至少被使用一次；
  - 对所有逻辑值均需测试 true 和 false ；
  - 在上下边界及可操作范围内运行所有循环；
  - 检查内部数据结构以确保其有效性。
- **逻辑细节测试的原因：**
- 逻辑错误和不正确假设与一条程序路径被运行的可能性成反比。当我们设计和实现主流之外的功能、条件或控制时，错误往往开始出现在我们工作中。日常处理往往被很好地了解，而“ 特殊情况“ 的处理则难于发现。
- 我们经常相信某逻辑路径不可能被执行，而事实上，它可能在正常的基础上被执行。程序的逻辑流有时是违反直觉的，这意味着我们关于控制流和数据流的一些无意识的假设可能导致设计错误，只有路径测试才能发现这些错误。
- 笔误是随机的。当一个程序被翻译为程序设计语言源代码时，有可能产生某些笔误，很多将被语法检查机制发现，但是，其他的会在测试开始时才会被发现。笔误出现在主流上和不明显的逻辑路径上的机率是一样的。

5.1.1.1逻辑覆盖

语句覆盖、判断覆盖、条件覆盖

5.1.1.2语句覆盖

语句覆盖就是设计若干个测试用例，运行被测程序，使得每一条可执行语句至少执行一次。

5.1.1.3判断覆盖

判定覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次。判定覆盖又称为分支覆盖。

判定路径覆盖（ Decision-to-Decision Paths Coverage ， DDP Coverage ）是判定覆盖的一个变体。这里的判定指的是一个序列语句，其起始位置是函数入口或一个判定（如 If，while， switch 等）的开始，结束位置是下一个判定的开始。

通过计算哪些判定路径已经走过，哪些没有走过，我们就可以得到 DDP 覆盖率了。其公式如下：

DDP 覆盖＝（至少被执行到一次的判定路径数量）/（系统中判定路径总数）

5.1.1.4条件覆盖

条件覆盖就是设计若干个测试用例，运行被测程序，使得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－条件覆盖就是设计足够的测试用例，使得判断中每个条件的所有可能取值至少执行一次，同时每个判断中的每个条件的可能取值至少执行一次。

◆ **条件组合覆盖**

条件组合覆盖就是设计足够的测试用例，运行被测程序，使得每个判断的所有可能的条件取值组合至少执行一次。

◆ **更改条件判定覆盖（ Modified Conditions/Decisions Coverage ，MC/DC Coverage ）**

是判定条件覆盖的一个变体。更改条件判定覆盖主要为多条件测试提供了方便，通过分析条件判定的覆盖来增加测试用例，防止测试的指数上升趋势。

MC/DC 标准满足下列需求：

- 被测试程序模块的每个入口点和出口点都必须至少被走一次。并且每一个程序判定的结果至少被覆盖一次。
- 通过分解逻辑操作，程序的判定被分解为基本的布尔条件表达式，每个条件独立的作用于判定的结果，覆盖所有条件的可能结果。

◆ **分支条件组合覆盖（ Branch Condition Combination Coverage ）**

是一种比判定条件覆盖更强的覆盖。它的含义是，设计一定的测试用例，使得每个分支的各操作数值的组合都遍历一次。其计算公式如下：

分支条件组合覆盖＝（被评价到的分支条件组合数）/（分支条件组合总数）

5.1.1.5函数覆盖

有很多测试工具，例如 TrueCoverage ， Logiscope 等，都提供了函数覆盖的概念，函数覆盖是针对系统或一个子系统的测试的，它表示在该测试中，有哪些函数被测试到了，其被测试到的频率有多大，这些函数在系统所有函数中占的比例有多大。函数覆盖是一个比较容易自动化的技术，同时也易于理解。其公式如下：

函数覆盖 ＝ （至少被执行一次的函数数量） / （系统中函数的总数）

由于函数覆盖也是基于代码的，因此你可以把其归入到白盒的范畴内。

5.1.1.6基本路径测试

在程序控制流程图的基础上，导出可以执行的路径集合，设计测试用例。

Z 路径覆盖是路径覆盖的一个变体。路径覆盖是白盒测试最为典型的问题。着眼于路径分析的测试可称为路径测试。完成路径测试的理想情况是做到路径覆盖。对于比较简单的小程序实现路径覆盖是可能做到的。但是如果程序中出现多个判断和多个循环，可能的路径数目将会急剧增长，达到天文数字，以至实现路径覆盖不可能做到。

为了解决这一问题，我们必须舍掉一些次要因素，对循环机制进行简化，从而极大地减少路径的数量，使得覆盖这些有限的路径成为可能。我们称简化循环意义下的路径覆盖为 Z 路径覆盖。

这里所说的对循环化简是指，限制循环的次数。无论循环的形式和实际执行循环体的次数多少，我们只考虑循环一次和零次两种情况。也即只考虑执行时进入循环体一次和跳过循环体这两种情况。

对于程序中的所有路径可以用路径树来表示。当得到某一程序的路径树后，从其根结点开始，一次遍历，再回到根结点时，把所经历的叶结点名排列起来，就得到一个路径。如果我们设法遍历了所有的叶结点，那就得到了所有的路径。

当得到所有的路径后，生成每个路径的测试用例，就可以做到 Z 路径覆盖测试。

◆ **条件测试路径选择**

当程序中判定多于一个时，形成的分支结构可以分为两类：嵌套型分支结构和连锁型分支结构。

对于嵌套型分支结构，若有 n 个判定语句，需要 n+1 个测试用例；

对于连锁型分支结构， 若有 n 个判定语句，需要有 2n 个测试用例，覆盖它的 2n 条路径。当 n 较大时将无法测试。

循环测试路径选择

循环分为 4 种不同类型：简单循环、连锁循环、嵌套循环和非结构循环。

- 简单循环

  零次循环：从循环入口到出口

  一次循环：检查循环初始值

  二次循环：检查多次循环

  m 次循环： 检查在多次循环

  最大次数循环、比最大次数多一次、少一次的循环。

- 嵌套循环

  对最内层循环做简单循环的全部测试。所有其它层的循环变量置为最小值；

  逐步外推，对其外面一层循环进行测试。测试时保持所有外层循环的循环变量取最小值，所有其它嵌套内层循环的循环变量取”典型”值。

  反复进行，直到所有各层循环测试完毕。

  对全部各层循环同时取最小循环次数，或者同时取最大循环次数。

- 连锁循环

如果各个循环互相独立，则可以用与简单循环相同的方法进行测试。但如果几个循环不是互相独立的，则需要使用测试嵌套循环的办法来处理。

- 非结构循环

  这一类循环应该使用结构化程序设计方法重新设计测试用例。

5.1.2黑盒测试

不基于内部设计和代码的任何知识，而是基于需求和功能性。

通过测试证明每个实现了的功能是否符合功能设计规格要求。

黑盒测试注重于测试软件的功能性需求，也即黑盒测试使软件工程师派生出执行程序所有功能需求的输入条件。黑盒测试并不是白盒测试的替代品，而是用于辅助白盒测试发现其他类型的错误。

◆ **黑盒测试试图发现以下类型的错误：**

1 ）功能错误或遗漏；

2 ）界面错误；

3 ）数据结构或外部数据库访问错误；

4 ）性能错误；

5 ）初始化和终止错误。

◆ **黑盒测试用于回答以下问题：**

1 ）如何测试功能的有效性？

2 ）何种类型的输入会产生好的测试用例？

3 ）系统是否对特定的输入值尤其敏感？

4 ）如何分隔数据类的边界？

5 ）系统能够承受何种数据率和数据量？

6 ）特定类型的数据组合会对系统产生何种影响？

◆ **黑盒测试可以导出标准的测试用例集：**

1 ）所设计的测试用例能够减少达到合理测试所需的附加测试用例数；

2 ）所设计的测试用例能够告知某些类型错误的存在或不存在，而不是仅仅与特定测试相关的错误。

5.1.2.1等价类划分方法

等价类，就是指某个输入域的集合，集合中的每个输入对揭露程序错误来说是等效的。

等价类划分是一种典型的黑盒测试方法，使用这一方法时，完全不考虑程序的内部结构，只依据程序的规格说明来设计测试用例。等价类划分方法把所有可能的输入数据，即把程序的输入域划分成若干部分，然后从每个部分中选取少数代表性数据作为测试用例，这就是等价类划分方法。它是功能测试的基本方法。

等价类划分方法设计测试用例

首先，是把所有可能的输入数据，即程序的输入域划分成若干部分（子集），然后从每一个子集中选取少数具有代表性的数据作为测试用例。

确定测试用例：为每个等价类规定一个唯一的编号；设计一个测试用例，使其尽可能多地覆盖尚未覆盖的有效等价类；设计一个新的测试用例，使其只覆盖一个无效等价类。

该方法是一种重要的，常用的黑盒测试用例设计方法。使用该方法设计测试用例要经历划分等价类（列出等价类表）和选取测试用例两步。

◆ **划分等价类**

划分等价类： 等价类是指某个输入域的子集合。在该子集合中，各个输入数据对于揭露程序中的错误都是等效的。并合理地假定：测试某等价类的代表值就等于对这一类其它值

的测试。因此，可以把全部输入数据合理划分为若干等价类，在每一个等价类中取一个数据作为测试的输入条件，就可以用少量代表性的测试数据。取得较好的测试结果。等价类划分可有两种不同的情况：有效等价类和无效等价类。

有效等价类：是指对于程序的规格说明来说是合理的，有意义的输入数据构成的集合。利用有效等价类可检验程序是否实现了规格说明中所规定的功能和性能。

无效等价类：与有效等价类的定义恰巧相反。

等价类组合：一些有代表性的有效等价类和无效等价类的组合。

由于等价划分和边界值都只孤立地考虑各个输入数据的测试功效，而没有考虑多个输入数据的组效应，可能会遗露了输入数据易于出错的组合情况，选择输入组合的一个有效的途径是利用判定表或判定树，列出输入数据各种组合与程序应作的动作（及相应的输出结果）之间的对应关系，然后为判定表的每一列至少设计一个测试用例。

设计测试用例时，要同时考虑这几种等价类。因为，软件不仅要能接收合理的数据，也要能经受意外的考验。这样的测试才能确保软件具有更高的可靠性。

◆ **划分等价类的方法**

下面给出六条确定等价类的原则：

- 在输入条件规定了取值范围或值的个数的情况下，则可以确立一个有效等价类和两个无效等价类。（注意：一般的输入要考虑：值的范围、值的个数）

  例如，在程序的规格说明中，对输入条件有一句话：“项数可以从 1 到 999”。

  则：有效等价类：“1≤项数≤999”；两个无效等价类：“项数＜1”或“项数＞999”。
- 在输入条件规定了输入值的集合或者规定了”必须如何”的条件的情况下，可确立一个有效等价类和一个无效等价类。

  例如，在对变量标识符规定为”以字母打头的……串”。

  则：有效等价类：所有以字母打头的串；

  无效等价类：不在此集合内（不以字母打头）的串。
- 在输入条件是一个布尔量的情况下，可确定一个有效等价类和一个无效等价类。
- 在规定了输入数据的一组值（假定 n 个），并且程序要对每一个输入值分别处理的情况下，可确立 n 个有效等价类和一个无效等价类。
- 在规定了输入数据必须遵守的规则的情况下，可确立一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）。

  例如，规定：一个语句必须以“;”结束。

  则：有效等价类：以“;”结束的语句；

  无效等价类：以“:”结束、以“,”结束、以“ ”结束、以 LF 结束等等的语句。
- 在明确已划分的等价类中各元素在程序处理中的方式不同的情况下，则应再将该等价类进一步的划分为更小的等价类。

◆ **设计测试用例**

设计测试用例：在确立了等价类后，可建立等价类表，列出所有划分出的等价类。然后从划分出的等价类中按以下原则设计测试用例：

- 为每一个等价类规定一个唯一的编号。
- 设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖地有效等价类，重复这一步。直到所有的有效等价类都被覆盖为止。
- 设计一个新的测试用例，使其仅覆盖一个尚未被覆盖的无效等价类，重复这一步。直到所有的无效等价类都被覆盖为止。
- 划分等价类等价类的原则。

5.1.2.2边界值分析方法

边界值分析是考虑边界条件而选取测试用例的一种黑盒测试方法，是对等价类划分方法的补充。实践证明，软件在输入、输出域的边界附近容易出现差错，而不是在输入范围的内部。因此针对各种边界情况设计测试用例，可以查出更多的错误。

使用边界值分析方法设计测试方案首先应该确定边界情况，通常输入等价类和输出等价类的边界，就是应该注重测试的程序边界情况。选取的测试数据应该正好等于、刚刚小于和刚刚大于边界值，也就是说，按照边界值分析法，应该选取刚好等于、稍小于和稍大于等价类边界值作为测试数据，而不是选取每个等价类内的典型值或任意值作为测试数据。

◆ **基于边界值分析方法选择测试用例的原则**

✧ 如果输入条件规定了值的范围，则应取刚达到这个范围的边界的值，以及刚刚超越这个范围边界的值作为测试输入数据。

✧ 如果输入条件规定了值的个数，则用最大个数，最小个数，比最小个数少一，比最大个数多一的数作为测试数据。

✧ 根据规格说明的每个输出条件，考虑值的范围情况。

✧ 根据规格说明的每个输出条件，考虑值的个数情况。

✧ 如果程序的规格说明给出的输入域或输出域是有序集合，则应选取集合的第一个元素和最后一个元素作为测试用例。

✧ 如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构的边界上的值作为测试用例。

✧ 分析规格说明，找出其它可能的边界条件。

5.1.2.3错误推测方法

错误推测法： 基于经验和直觉推测程序中所有可能存在的各种错误， 从而有针对性的设计测试用例的方法。

错误推测方法的基本思想：列举出程序中所有可能有的错误和容易发生错误的特殊情况，根据他们选择测试用例。

◆ **常见依据**

✧ 在单元测试时曾列出的许多在模块中常见的错误。

✧ 以前产品测试中曾经发现的错误等。

✧ 已发现缺陷的测试方法的推广。

✧ 容易发生错误的情况。如：输入或输出为 0 的情况。输入为空格或输入表格只有一行，共享参数同时使用在几个模块中等等。

✧ 补充等价类和边界值法遗漏的一些等价类组合。

✧ 一些位置使用了共享变量，设计测试用例，修改一个共享变量，看其他位置有没有同时做修改。

5.1.2.4因果图方法

因果图方法是对等价类的扩展，可以理解为“等价类组合判定表”。因果图即输入等价类与输出等价类的关系图。

分析程序规格说明的描述中哪些是原因，哪些是结果。原因是输入条件或是输入条件的等价类。结果是输出条件。因果图是一种形式语言，由自然语言写成的规范转换而成，这种形式语言实际上是一种使用简化记号表示数字逻辑图。因果图法是帮助人们系统地选择一组高效测试用例的方法，此外，它还能指出程序规范中的不完全性和二义性。

因果图方法最终生成的就是判定表。它适合于检查程序输入条件的各种组合情况。如果有 N 个条件，每个条件有 2 个取值（0，1），产生 2 的 N 次方个规则。

◆ **因果图的适用范围**

如果在测试时必须考虑输入条件的各种组合，可使用一种适合于描述对于多种条件的组合，相应产生多个动作的形式来设计测试用例，这就需要利用因果图。因果图方法最终生成的就是判定表。它适合于检查程序输入条件的各种组合情况。

前面介绍的等价类划分方法和边界值分析方法，都是着重考虑输入条件，但未考虑输入条件之间的联系，相互组合等。考虑输入条件之间的相互组合，可能会产生一些新的情况。但要检查输入条件的组合不是一件容易的事情，即使把所有输入条件划分成等价类，他们之间的组合情况也相当多。因此必须考虑采用一种适合于描述对于多种条件的组合，相应产生多个动作的形式来考虑设计测试用例。这就需要利用因果图（逻辑模型）。

◆ **因果图生成测试用例的基本步骤**

✧ 分析软件规格说明描述中， 那些是原因(即输入条件或输入条件的等价类)，那些是结果(即输出条件)， 并给每个原因和结果赋予一个标识符。

✧ 分析软件规格说明描述中的语义。找出原因与结果之间， 原因与原因之间对应的关系。根据这些关系，画出因果图。

✧ 由于语法或环境限制， 有些原因与原因之间，原因与结果之间的组合情况不不可能出现。 为表明这些特殊情况， 在因果图上用一些记号表明约束或限制条件。

✧ 把因果图转换为判定表。

✧ 把判定表的每一列拿出来作为依据，设计测试用例。

◆ **判定表驱动分析方法**

前面因果图方法中已经用到了判定表。判定表（Decision Table）是分析和表达多逻辑条件下执行不同操作的情况下的工具。在程序设计发展的初期，判定表就已被当作编写程序的辅助工具了。由于它可以把复杂的逻辑关系和多种条件组合的情况表达得既具体又明确。

判定表四个组成部分

✧ 条件桩（Condition Stub）：列出了问题得所有条件。通常认为列出得条件的次序无关紧要。

✧ 动作桩（Action Stub）：列出了问题规定可能采取的操作。这些操作的排列顺序没有约束。

✧ 条件项（Condition Entry）：列出针对它左列条件的取值。在所有可能情况下的真假值。

✧ 动作项（Action Entry）：列出在条件项的各种取值情况下应该采取的动作。

判定表规则

规则：任何一个条件组合的特定取值及其相应要执行的操作。在判定表中贯穿条件项和动作项的一列就是一条规则。显然，判定表中列出多少组条件取值，也就有多少条规则，既条件项和动作项有多少列。

判定表的建立步骤

➢ 确定规则的个数。假如有 n 个条件。每个条件有两个取值（0，1），故有 种规则。

➢ 列出所有的条件桩和动作桩。

➢ 填入条件项。

➢ 填入动作项。等到初始判定表。

- 简化。合并相似规则（相同动作）。

适合使用判定表设计测试用例的条件

- ✧ 规格说明以判定表形式给出，或很容易转换成判定表。
- ✧ 条件的排列顺序不会也不影响执行哪些操作。
- ✧ 规则的排列顺序不会也不影响执行哪些操作。
- ✧ 每当某一规则的条件已经满足，并确定要执行的操作后，不必检验别的规则。
- ✧ 如果某一规则得到满足要执行多个操作，这些操作的执行顺序无关紧要。

5.1.2.5正交试验设计法

从大量数据中选择适量的/有代表性的测试数据。

5.1.2.6功能图分析方法

又可以称作流程测试或状态迁移测试。(注意：重要执行通路/出错处理通路)
类似于白盒测试中的逻辑覆盖和路径法。
需要懂得控制语句（循环，顺序，选择，重复）
生成局部测试用例-生成测试路径-局部测试用例合成

第 2 节测试过程

- ⮝ 单元测试、集成测试、确认测试、系统测试、验收测试；
- ⮝ α 测试、β 测试 ；

5.2.1单元测试

单元测试是对软件中的基本组成单位进行的测试，如一个模块、一个过程等等。它是软件动态测试的最基本的部分，也是最重要的部分之一，其目的是检验软件基本组成单位的正确性。一个软件单元的正确性是相对于该单元的规约而言的。因此，单元测试以被测试单位的规约为基准。单元测试的主要方法有控制流测试、数据流测试、排错测试、分域测试等等。

单元测试，测试内容包括模块程序结构检查、代码测试和模块内功能测试。典型地由程序员而非测试员来做，因为它需要知道内部程序设计和编码的细节知识。这个工作不容易作好，除非应用系统有一个设计很好的体系结构， 还可能需要开发测试驱动器模块或测试工具。

5.2.2集成测试

集成测试是在软件系统集成过程中所进行的测试，其主要目的是检查软件单位之间的接口是否正确。它根据集成测试计划，一边将模块或其他软件单位组合成越来越大的系统，一边运行该系统，以分析所组成的系统是否正确，各组成部分是否合拍。集成测试的策略主要

有自顶向下和自底向上两种。

也可以理解为，在软件设计单元、功能模块组装、集成为系统时，对应用系统的各个部件（软件单元、功能模块的接口、连接等）进行的联合测试，以决定他们能否在一起共同工作。部件可以是代码块、独立的应用、网络上的客户端或服务器端程序。

5.2.3系统测试

系统测试是基于系统需求说明书的黑盒类测试，是对已经集成好的软件系统进行彻底的测试，以验证软件系统的正确性和性能等满足其规约所指定的要求，检查软件的行为和输出是否正确并非一项简单的任务，它被称为测试的“先知者问题”。因此，系统测试应该按照测试计划进行，其输入、输出和其他动态运行行为应该与软件规约进行对比。软件系统测试方法很多，主要有功能测试、性能测试、随机测试等等。

5.2.4验收测试

验收测试旨在向软件的购买者展示该软件系统满足其用户的需求。它的测试数据通常是系统测试的测试数据的子集。所不同的是，验收测试常常有软件系统的购买者代表在现场，甚至是在软件安装使用的现场。这是软件在投入使用之前的最后测试。
简言之，为用户方组织的有效性和系统测试。

5.2.5回归测试

回归测试是在软件维护阶段，在软件设计错误修正、设计修改以及软件升级后，对软件进行修改之后进行的测试（主要针对软件修改、影响部分）。其目的是检验对软件进行的修改的正确性。修改的正确性有两重含义：一是所作的修改达到了预定目的；二是不影响软件的其他功能的正确性、不产生新的缺陷。

5.2.5.1回归测试概述

回归测试可以通过重新执行所有的测试用例的一个子集人工地进行，也可以使用自动化的捕获回放工具来进行。捕获回放工具使得软件工程师能够捕获到测试用例，然后就可以进行回放和比较。

回归测试集包括三种不同类型的测试用例：

- 能够测试软件的所有功能的代表性测试用例。
- 专门针对可能会被修改影响的软件功能的附加测试。
- 针对修改过的软件成分的测试。

回归测试是测试实施过程中最为重要的环节之一，回归测试做得好与坏，直接影响到测试工作的效果。而且，回归测试的工作量占测试实施的工作量的5 成以上。

5.2.5.2回归测试类型

一般回归测试

一般回归测试：适用于各个系统测试阶段中，验证前一次系统测试中失败的用例，和产品的有效性。

一般的回归测试可以在一段时间就使用一个更新了的版本(基于测试用例的测试)，但是一般说来在一个回归测试中尽量保证被测版本不变是有益的。

- **最终回归测试**

最终回归测试：适用于产品正式发布前的创建版本，验证发布版本的有效性。

在执行最终回归测试的时候要求被测试的版本在一段时间内是固定不变的，也就是在产品发布中经常说到的“cook－time”。在这段时间里面，产品会被反复的测试。某些测试用例可能要被执行几次，以确认在发布版本中不会存在用户在使用过程中会遇到Bug。所有的基于发布版本的 Bug 修改工作应当在最终回归测试前完成。由于它测试的对象是将要发布给用户的版本，所以最终回归测试是相当重要的。因为通过最终回归测试的版本就是用户将要使用的版本。

5.2.5.3回归测试策略

对测试用例库(在给定的预算和进度下，尽可能有效率和有效力)进行维护并依据一定的策略选择相应的回归测试包。

为什么？

在软件生命周期中，即使一个得到良好维护的测试用例库也可能变得相当大，这使每次回归测试都重新运行完整的测试包变得不切实际。测试组不得不选择一个缩减的回归测试包来完成回归测试。

回归测试的价值在于它是一个能够检测到回归错误的受控实验。当测试组选择缩减的回归测试时，有可能删除了将揭示回归错误的测试用例，消除了发现回归错误的机会。然而，如果采用了代码相依性分析等安全的缩减技术，就可以决定哪些测试用例可以被删除而不会让回归测试的意图遭到破坏。

有些回归测试中，往往维持一个“不变化的测试用例集合”，换句话说就是每次回归测试使用的测试用例都是固定不变的，回归测试用例的集合应该按照修改了的 bug 和 这些 bug 的类型来确定。如果维持一个不变的测试用例集合，那么有可能在测试中发现不到修改一个 bug 给整个系统带来的副作用，因为测试人员的精力被分散了。如果我们可以分析测试用例和修改 bug 之间的关联的话，则可以显著的提高回归测试的效率。

测试用例库的维护

测试用例库的维护主要包括以下几个方面：

- **删除过时的测试用例**

因为需求的改变等原因可能会使一个基线测试用例不再适合被测试系统，这些测试用例

就会过时。例如，某个变量的界限发生了改变，原来针对边界值的测试就无法完成对新边界测试。所以，在软件的每次修改后都应进行相应的过时测试用例的删除。

◆ **改进不受控制的测试用例**

随着软件项目的进展，测试用例库中的用例会不断增加，其中会出现一些对输入或运行状态十分敏感的测试用例。这些测试不容易重复且结果难以控制，会影响回归测试的效率，需要进行改进，使其达到可重复和可控制的要求。

◆ **删除冗余的测试用例**

如果存在两个或者更多个测试用例针对一组相同的输入和输出进行测试，那么这些测试用例是冗余的。冗余测试用例的存在降低了回归测试的效率。所以需要定期的整理测试用例库，并将冗余的用例删除掉。

◆ **增添新的测试用例**

如果某个程序段、构件或关键的接口在现有的测试中没有被测试，那么应该开发新测试用例重新对其进行测试。并将新开发的测试用例合并到基线测试包中。

通过对测试用例库的维护不仅改善了测试用例的可用性，而且也提高了测试库的可信性，同时还可以将一个基线测试用例库的效率和效用保持在一个较高的级别上。

5.2.5.4回归测试包的选择

测试包的选择原则

在选择回归测试用例的时候更多的是要考虑修改bug 可能产生问题的严重性，而不是单纯的考虑一个 bug 本身的严重性。在修改一个 minor 等级的 bug 时，修改工作产生的副作用可以会导致一个 major 的 bug 产生。而在修改一个严重等级的 bug 时候，修改工作可能对其他代码一点影响也没有。所以测试人员在选择回归测试用例的时候需要综合的考虑上面 2 方面的因素。

在选择测试用例的时候需要避免的是选取那些必定Fail 的用例和那里同 bug fixes 没有多少关系的用例。在最终回归测试用例的选择上“positive”的用例应该多于“negative”用例，因为过多的“negative”干扰测试的重点和开发人员的精力。在一般性的回归测试中则应该选取“positive”和“negative”混合的测试用例。前面中说道的“negative”用例可以解释为那些以系统出错为目的的用例。

测试包的基本属性

- 为回归测试选择用例的时候需要初步具备分析修改方法和分析修改过程对系统的影响的能力
- 选择的用例要求能够覆盖缺陷的多发区域
- 选择的用例能够覆盖到代码经常修改，变动的区域
- 产品中用户的可见部分在选择用例的时候一定要覆盖到
- 用例要覆盖到用户在需求中提及到的核心特性。

测试包的选择方式

选择回归测试策略应该兼顾效率和有效性两个方面。常用的选择回归测试的方式包括：

- **再测试全部用例**

选择基线测试用例库中的全部测试用例组成回归测试包，这是一种比较安全的方法，再测试全部用例具有最低的遗漏回归错误的风险，但测试成本最高。全部再测试几乎可以应用到任何情况下，基本上不需要进行分析和重新开发，但是，随着开发工作的进展，测试用例不断增多，重复原先所有的测试将带来很大的工作量，往往超出了我们的预算和进度。

- **基于风险选择测试**

可以基于一定的风险标准来从基线测试用例库中选择回归测试包。首先运行最重要的、关键的和可疑的测试，而跳过那些非关键的、优先级别低的或者高稳定的测试用例，这些用例即便可能测试到缺陷，这些缺陷的严重性也仅有三级或四级。一般而言，测试从主要特征到次要特征。

- **基于操作剖面选择测试**

如果基线测试用例库的测试用例是基于软件操作剖面开发的，测试用例的分布情况反映了系统的实际使用情况。回归测试所使用的测试用例个数可以由测试预算确定，回归测试可以优先选择那些针对最重要或最频繁使用功能的测试用例，释放和缓解最高级别的风险，有助于尽早发现那些对可靠性有最大影响的故障。这种方法可以在一个给定的预算下最有效的提高系统可靠性，但实施起来有一定的难度。

- **再测试修改的部分**

当测试者对修改的局部化有足够的信心时，可以通过相依性分析识别软件的修改情况并分析修改的影响，将回归测试局限于被改变的模块和它的接口上。通常，一个回归错误一定涉及一个新的、修改的或删除的代码段。在允许的条件下，回归测试尽可能覆盖受到影响的部分。

再测试全部用例的策略是最安全的策略，但已经运行过许多次的回归测试不太可能揭示新的错误，而且很多时候，由于时间、人员、设备和经费的原因，不允许选择再测试全部用例的回归测试策略，此时，可以选择适当的策略进行缩减的回归测试。

5.2.5.5回归测试的基本过程

有了测试用例库的维护方法和回归测试包的选择策略，回归测试可遵循下述基本过程进行：
(1)识别出软件中被修改的部分；
(2)从原基线测试用例库 T 中，排除所有不再适用的测试用例，确定那些对新的软件版本依然有效的测试用例，其结果是建立一个新的基线测试用例库T0。
(3)依据一定的策略从 T0 中选择测试用例测试被修改的软件。
(4)如果必要，生成新的测试用例集 T1，用于测试 T0 无法充分测试的软件部分。
(5)用 T1 执行修改后的软件。
第(2)和第(3)步测试验证修改是否破坏了现有的功能，第(4)和第(5)步测试验证 修改工作本身。

5.2.6关于单元测试

测试项目经理需要对单元测试用例和集成测试用例进行复审。可以是非正式的或者是正式的。

复审主要是关注：输入设计是否正确完整，使用的测试方法是否合理，是否还有应使用而未使用的方法。测试的预期效果是否正确。测试点是否齐全，是否还有测试点未写。

5.2.7关于α 、 β 测试

- **Alpha 测试**

Alpha 测试：是由一个用户在测试环境下进行的测试，也可以是开发机构内部的用户在模拟实际操作环境下进行的测试。软件在一个自然设置状态下使用。开发者坐在用户旁边，随时记下错误情况和使用中的问题。

Alpha 测试的目：评价软件产品的特性 (即功能、局域化、可使用性、可靠性、性能和支持) 。尤其注重产品的界面和特色。

Alpha 测试一般由最终用户或其他人员员完成，不能由程序员或测试员完成，他们提出的功能和修改意见是特别有价值的。

Alpha 测试可以从软件产品编码结束之时开始，或在模块（子系统）测试完成之后开始，也可以在确认测试过程中产品达到一定的稳定和可靠程序之后再开始。

有关的手册（草稿）等应事先准备好。

- **Beta 测试**

Beta 测试：是由软件的多个用户在一个或多个用户的实际使用环境下进行的测试。这些用户是与公司签定了支持产品预发行合同的外部客户，他们要求使用该产品，并愿意返回有关错位错误信息给开发者。与 α 测试不同的是，开发者通常不在测试现场。因而， β 测试是在开发者无法控制的环境下进行的软件现场应用。在 β 测试中，由用户记下遇到的所有问题，包括真实的以及主观认定的，定期向开发机构报告，开发机构在综合用户的报告之后，做出修改，最将软件产品交付给全体用户使用。

Beta 测试主要衡量产品的特性，着重于产品的支持性，包括文档、客户培训和支持产品生产能力。由于 β 测试的主要目标是测试可支持性，所以 β 测试应尽可能由主持产品发行的人员来管理。

只有当α测试达到一定的可靠程度时，才能开始 β 测试。由于它处在整个测试的最后阶段，不能指望这时发现主要问题。同时，产品的所有手册文本也应该在此阶段完全定稿。

5.2.8验证测试与确认测试

软件测试是验证和确认 Verfication and Validation（V & V）。验证指保证软件正确地实现了一特定功能的一系列活动。确认是指保证所生产的软件可追溯到用户需求的一系列活动。

V & V 的解释：

Verfication ： “Are we building the product right?”

Validation ： “ Are we building the right product?”

5.2.9单元/集成/系统测试的区别

三者区别可以从三方面考虑：测试方法、考察范围、评估标准。具体细节内容不再讲述。

5.2.10单元集成测试的主体分析

NASA（美国航空航天管理局）提供的经验数据：项目组执行测试，需要资源1.4 人月/千行，遗留缺陷率 20%，测试组执行测试，需要资源 2.5 人月/千行，遗留缺陷 16%。

独立测试组织在测试阶段的问题由 20%降低到了 16%，效果好，但是活动成本上升了 78.6%，说明了普通软件产品而言，单元测试、集成测试采用独立测试组织成本太高、不合适。当然，不计成本可靠性高的特殊软件，还是应该使用独立测试组织。

第 3 节测试类型

5.3.1系统测试分类（一）

- 功能测试<正确性测试/容错性测试/并发逻辑测试/关联内容测试等等>
- 安全测试
- 性能测试<疲劳测试/压力测试/响应时间>
- 强度测试<使资源出现短缺的情况，检查系统的应对>
- 可移植性<兼容性测试>
- 容量测试
- 恢复测试
- 配置测试
- 安装、卸载测试
- 用户界面测试
- 比较测试
- 接口间测试
- 数据库测试

以上测试类型的具体信息后面的章节会讲述。

5.3.2系统测试分类（二）

- 静态测试
- 动态测试

◆ **静态测试**

静态测试的基本特征是在对软件进行分析、检查和测试时不实际运行被测试的程序。它可以用于对各种软件文档进行测试，是软件开发中十分有效的质量控制方法之一。在软件开发过程中的早期阶段，由于可运行的代码尚未产生，不可能进行动态测试，而这些阶段的中间产品的质量直接关系到软件开发的成败与开销的大小，因此，在这些阶段，静态测试的作

用尤为重要。在软件开发多年的生产实践经验和教训的基础上，人们总结出了一些行之有效的静态测试技术和方法，如结构化走通、正规检视等等。这些方法和测试技术可以与软件质量的定量度量技术相结合，对软件开发过程进行监视、控制，从而保障软件质量。

主要工作：检查代码或文档。

所有的评审工作，可以理解为静态测试。

- **动态测试**

  执行被测软件或文档

5.3.3系统测试分类（三）

按照测试对象的结构来区分：可以分为C/S 结构系统测试，B/S 结构系统测试，个人软件测试等。

5.3.3.1Client/Server 软件测试

C/S 结构软件的测试发生在三个不同的层次：

- 个体的客户端应用以“分离的”模式被测试——不考虑服务器和底层网络的运行；
- 客户端软件和关联的服务器端应用被一起测试，但网络运行不被明显的考虑；
- 完整的 C/S 体系结构，包括网络运行和性能，被测试。

C/S 结构软件测试常用方法

- 应用功能测试——客户端应用被独立地执行，以揭示在其运行中的错误。
- 服务器测试——测试服务器的协调和数据管理功能，也考虑服务器性能（整体反映时间和数据吞吐量）。
- 数据库测试——测试服务器存储的数据的精确性和完整性，检查客户端应用提交的事务，以保证数据被正确地存储、更新和检索。
- 事务测试——创建一系列的测试以保证每类事务被按照需求处理。测试着重于处理的正确性，也关注性能问题。
- 网络通信测试——这些测试验证网络节点间的通信正常地发生，并且消息传递、事务和相关的网络交通无错的发生。

5.3.3.2个人软件测试

个人软件的测试需要关注：

- 基本功能测试
- 安装卸载测试
- 升级测试
- 兼容性测试
- 自我保护测试

5.3.3.3Browse/Server 软件测试

B/S 结构软件测试需要关注：

- 基本功能测试
- 性能测试
- 浏览器兼容性测试
- 数据库测试
- 安全性测试

第 6 章 系统测试分类

系统测试是基于系统需求规格说明书的黑盒测试。以功能测试为主，还包括：系统性能测试、安全性、可靠性、稳定性测试，以及对系统其它性能如负载能力、处理能力以及响应时间等进行测试。其目的是为了验证系统是否满足开发要求，是否能够提供设计所描述的功能，是否用户的需求都得到满足。

以下章节根据以前项目经验讲解各种类型测试的关注点，注意事项等等。具体的测试内容等测试报告中会有详细体现。

第 1 节功能测试

功能测试又称正确性测试，它检查软件的基本功能点功能流是否符合规格说明。按照系统功能需求规定对系统的功能、流程、数据、业务规则等进行测试，以及对系统基本特征如操作、界面、报表等的合理性、一致性进行测试。

基本的方法是构造一些合理输入，检查是否得到期望的输出。功能测试占系统测试的大部分时间，功能测试主要包括以下几个方面：

- 功能点测试（正确性，完整性，审计，追踪， 耦合性，基本安全性）
- 操作性的测试（易用性）
- 界面测试(重点美观性)
- 支持手册的测试（易用性及其他）

6.1.1GUI 测试

界面测试的一些关注点

窗口切换、移动、改变大小时正常吗？
各种界面元素的文字正确吗？（如标题、提示等）
各种界面元素的状态正确吗？（如有效、无效、选中等状态）
各种界面元素支持键盘操作吗？
各种界面元素支持鼠标操作吗？
对话框中的缺省焦点正确吗?
数据项能正确回显吗?
对于常用的功能，用户能否不必阅读手册就能使用？
执行有风险的操作时，有”确认”、”放弃”等提示吗？
操作顺序合理吗？
有联机帮助吗？
各种界面元素的布局合理吗？美观吗？
各种界面元素的颜色协调吗？
各种界面元素的形状美观吗?
字体美观吗？
图标直观吗？

6.1.2并发逻辑处理测试

需要产生新的 ID 时，多用户并发，检查 ID 的产生情况；

对同一 ID 的记录进行操作，多用户进行不同操作，检查该记录数据和状态变化情况；

6.1.3系统级关联测试

在单元测试中，可能会遇到以下情况：

程序中两个模块使用某些共享的变量，一个模块对这些变量的修改不正确，则会引起另一个模块出错，因此这是程序发现错误的一个可能的原因，应该设计测试方案，在程序的一次运行中同时检测这两个模块，特别要着重检测一个模块修改了共享变量后，另一个模块能否像预期的那样正常使用这些变量。如果两个模块相对独立，就没有必要测试它们的输入组合情况。

在系统测试中，也有同样的问题。数据库中一条记录，会对应很多显示或处理的位置。当对这样一条数据做了修改后，对应位置的显示或者处理结果都要做相应的变化。这也是系统测试需要重点检查的一个测试点。

编写测试用例时，首先要明确这条记录在什么情况下可以编辑，什么情况下不可以编辑。在可编辑时，关联位置会做什么处理，还要关注处理的一致性，正确性。

以下是一个内容管理系统的测试举例：

编号	操作	对应检查位置	预期结果
1	编辑一个栏目 <包括增删改>	内容管理>栏目树 栏目管理>快照管理 安全管理>权限分配 站点管理>站点配置 站点管理>站点内容发布 信息统计>统计栏目设定 日志分析>栏目明细修改	
2	编辑一个内容 <包括增删改>	内容管理>查看内容 内容管理>历史版本 信息审核>内容审核 站点管理>站点内容发布	
3	编辑一个用户 <包括增删改>	安全管理>用户管理 信息统计>统计用户设定 日志分析>用户明细修改 论坛管理>信息统计 个人信息显示	

<对于一些名词，读者可能不是很清晰，不过没关系，这个例子只是用来帮助理解>

以上只是列举了一些静态显示问题，程序中还会涉及到另外一些关联，修改某个记录后，对应位置的信息需要重新计算，状态需要重置等等，对于这样的位置，更需要重点测试。

6.1.4帮助文档的测试

用户在使用系统时候，如果出现问题，首先求助的就是在线帮助。一个糟糕的在线帮助会很大的打击用户对系统的信心。因此一个好的系统，必须要有完备的帮助体系，包括用户操作手册，实时在线帮助等。在线帮助的测试（Online Help Testing）主要用于验证系统的实时在线帮助的可用性和正确性。

帮助文档测试可以和文档测试（或资料测试）一起进行。值得注意的是，帮助文档测试需要按照帮助文档的步骤，一步步的执行，以确保文档描述的正确性和清晰性。

帮助文档测试，测试人员需要关注的问题：

- 帮助文件的索引是否正确？
- 帮助文件的内容是否正确？
- 在系统运行过程中帮助能否被正常的激活？
- 在系统不同的位置激活的帮助内容与当前操作内容是否相关联？
- 帮助是否足够详细并能解决需要被解决的问题？

第 2 节恢复测试

恢复测试是通过各种手段，让软件强制性地发生故障（如：系统崩溃、硬件损坏或其他灾难性问题），然后来验证恢复是否能正常进行的一种系统测试方法。如果恢复是自动的（由系统本身来进行的），重新初始化、检查点机制、数据恢复和重启动都要进行正确验证。如果恢复是需要人工干预的，那么要估算修复的平均时间是否在可以接受的范围之内。恢复测试时，应该参考性能测试的相关测试指标。

6.2.1测试重点

- 执行某个工作流时，出现系统问题， 系统恢复后，该工作流的数据情况。该情形注重检查数据的正确性。
- 在系统执行各个功能或操作时，突然中断，检查程序的正确性。
- 在系统执行某些操作或其他外部原因导致系统软硬件环境达到临界状态或崩溃时，软硬件环境的恢复能力。

第 3 节安全测试

验证安装在系统内的保护机构确实能够对系统进行保护，使之不受各种非常的干扰。安全测试时需要设计一些测试用例试图突破系统的安全保密措施，检验系统是否有安全保密的漏洞。

在安全测试过程中，测试者扮演着一个试图攻击系统的个人角色。测试者可以尝试去通过外部的手段来获取系统的密码，可以使用可以瓦解任何防守的客户软件来攻击系统；可以把系统“制服”，使得别人无法访问；可以有目的地引发系统错误，期望在系统恢复过程中侵入系统；可以通过浏览非保密的数据，从中找到进入系统的钥匙；等等。

只要有足够的时间和资源，好的安全测试就一定能够最终侵入一个系统。系统设计者

的任务就是要把系统设计为想要攻破系统而付出的代价大于攻破系统之后得到的信息的价值。

6.3.1安全测试的一些实例

- ✧ 用户权限安全，截取或破译口令等
- ✧ 故意导致系统失败，企图趁恢复之机非法进入
- ✧ 测试专门定做的软件破坏系统的保护机制
- ✧ 试图通过浏览非保密数据
- ✧ 非法篡改数据
- ✧ DOS 攻击
- ✧ 数据备份安全
- ✧ 网页加密安全：加密解密用的字符串等等（比如网页链接中的加密字符串）。
- ✧ 网页编写的时候得一些安全问题 ：比方说对特殊字符的过滤

6.3.2安全性测试方式

主要测试方式：

- ✧ 模拟 DOS 攻击服务器
- ✧ 以普通用户身份非法侵入数据库服务器
- ✧ 缓冲区溢出检测
- ✧ 代码的安全检查
- ✧ 数据保密性检查
- ✧ 数据备份安全性检查

6.3.3配置测试

软件系统在不同的软件配置情况下的运行和使用情况，也可以纳入兼容性测试范畴。

第 4 节容量测试

验证系统处理大量数据的能力，重点以下几个方面：

- ➢ 一定时间内，接收到大量的数据；
- ➢ 持续的数据发送和处理，累积很大的数据量；
- ➢ 最大数据量的情况下，系统诸如查询，报表等功能的使用情况。

第 5 节安装、卸载测试

安装测试有两个目的。第一个目的是确保该软件在正常情况和异常情况的不同条件下，都能进行安装。例如，进行首次安装、升级、完整的或自定义的安装。异常情况包括磁盘空间不足、缺少目录创建权限等。第二个目的是核实软件在安装后可立即正常运行。这通常是

指运行大量为功能测试制定的测试。

一般来说，升级测试也纳入安装卸载测试的范畴。

6.5.1安装正确性和完整性检查

- ✧ 对程序安装的正确性和完整性进行核对：
- ✧ 校验产品文件的完整性
- ✧ 安装的审查，追踪被记录
- ✧ 安装之前，该系统已经被证实没有问题
- ✧ 如果安装失败，系统有相应的解决方案
- ✧ 安装过程，进行了权限控制（安全性）
- ✧ 安装遵循一定的方法，步骤
- ✧ 需要的配套程序和数据已经放进了产品中
- ✧ 提供使用说明
- ✧ 相关文件已经完整(可维护性）
- ✧ 接口已经被合理调整（耦合性）
- ✧ 综合的性能达到了用户要求

6.5.2安装卸载测试兼容性检查点

- ✧ 杀毒软件的病毒检查
- ✧ 在不同系统下进行安装测试，检查程序是否可用
- ✧ 安装过程异常中断后软件的恢复性
- ✧ 和其他应用软件的兼容性、抗干扰性
- ✧ 与同类软件的攻防属性
- ✧ 反安装是否可用，数据是否保护
- ✧ 升级测试
- ✧ 比较测试
- ✧ 与竞争伙伴的产品的比较测试，如软件的弱点、优点或实力。

第 6 节兼容性测试

测试软件在一个特定的硬件/软件/操作系统/网络等环境下的性能如何。

6.6.1兼容性测试点

- ✧ 操作系统兼容性测试
- ✧ 同一平台上其他相关应用软件的兼容性
- ✧ 硬件兼容性
- ✧ 异构数据兼容性测试
- ✧ 新旧数据数据转换

✧ 异种数据兼容性测试

6.6.2相关软件市场占有率

说到兼容性，一定会考虑到其他可能存在冲突的软件，然而这样的软件太多了，如何进行有效的测试呢？每一类依照市场占有率选择有代表性的几个作为测试对象。

具体的各个软件市场占有率，请查看各大排名网站。

第 7 节接口测试

模拟不同的输入输出。需要各个接口相应的系统相互配合。

6.7.1模块接口测试要点

✧ 参数数目和由调用模块送来的变元的数目是否相等？
✧ 参数的属性和变元的属性是否匹配？
✧ 参数和变元的单位系统是否匹配？
✧ 传送给被调用模块的变元的数目是否等于那个模块的参数的数目？
✧ 传送给被调用模块的变元属性和参数的属性是否一致？
✧ 传送给被调用模块的变元的单位系统和该模块参数的单位系统是否一致？
✧ 传送给内部函数的变元属性、数目和次序是否正确？
✧ 是否修改了只做输入用的变元。
✧ 全程变量的定义和用法在各个模块中是否一致？

第 8 节数据库测试

6.8.1数据库测试内容

测试重点关注以下几个方面：
✧ 数据库操作响应时间；
✧ 数据库容量；
✧ 数据库设计检查
✧ 数据库连接

对于数据的容量和响应时间等测试，在性能测试章节会讲解到。

6.8.2基本的数据库连接测试

设备：数据库服务器、web 服务器
软件：应用系统、数据库

测试用例	操作	预期结果

A：数据库、服务、引导用户界面	正常连接	设置、运行正确
	启动前关闭数据库	提示数据库连接失败
	启动后关闭数据库	提示连接数据库错误
	关闭后连通	软件运行正确
	启动前断开网线	软件运行正确
	启动后断开	软件运行正确
	断开后连通	软件运行正确
A：数据库，B：服务、引导用户界面	正常连接	设置、运行正确
	启动前关闭数据库	提示数据库连接失败
	启动后关闭数据库	提示连接数据库错误
	关闭后连通	软件运行正确
	启动前断开网线	连接数据库失败
	启动后断开	数据库连接错误
	断开后连通	软件运行正确
A：数据库、服务，B：引导用户界面	正常连接	设置、运行正确
	启动前关闭数据库	提示数据库连接失败
	启动后关闭数据库	提示连接数据库错误
	关闭后连通	软件运行正确
	启动前断开网线	连接数据库失败
	启动后断开	数据库连接错误
	断开后连通	软件运行正确
配置中：应用服务是否由本地机控制	配置中，应用服务设置正确	软件正确运行
	配置中，应用服务设置错误	提示设置错误

6.8.3数据库设计测试

主要是为了验证数据库设计文档《数据字典》是否符合设计需要，实际数据库是否严格按照数据字典设计。

第 9 节可靠性测试

软件可靠性（Software Reliability）可以定义为：在规定环境，规定时间内（自然单元或时间单元），一个系统或其功能无故障运行的可能性。其中：规定的环境包括硬件环境和软件环境。软件环境包括允许的操作系统、应用程序、编译系统、数据库系统等；硬件环境包括 CPU 、内存、 I/O 等。规定的时间一般分为执行时间、日历时间和时钟时间。其中执行时间（ Executing Time ）是指执行一个程序所用的实际时间和中央处理器时间，或者是程序处于执行过程中的一段时间；日历时间（ Calendar time ）指的是编年时间，包括计算机可能未运行的时间；时钟时间（ Clock time ）是指从程序执行开始到程序执行完毕所经历的钟表时间。自然单元除时间外，跟软件产品处理数目相关的单元如运行、电话呼叫、API 调用等都可以看作是一个自然单元。

6.9.1软件可靠性层次划分

◆ **从用户角度来看，软件可靠性可以分四个层次来看：**

- 第一个层面：软件不出现故障；
- 第二个层面：软件即使出现故障，也仅对性能有部分影响，而设备的功能不受损失；
- 第三个层面：软件出现故障并造成部分功能受损失，但是能够很快恢复业务；
- 第四个层面：软件出现较大故障，并造成大部分功能受损、业务中断或瘫痪，但能够尽快恢复业务（无论是手工恢复还是自动恢复）；

◆ **对应于不同的可靠性层次，要求系统有相应的层次设计要求和维护要求。**

例如：

- 对于第一个层面：要求系统能够按照充分的规范来进行设计，加强各种异常处理能力和环境适应能力等；
- 对于第二个层面：要求系统有较高的容错能力，使用冗余技术和备份技术等；
- 对于第三个层面：要求系统有很好的可测试性，能迅速隔离问题和定位问题等；
- 对于第四个层面：要求系统有较高的可维护性等

第 10 节性能测试

在实时系统和嵌入式系统中，提供符合功能需求但不符合性能需求的软件是不能被接受的。性能测试就是用来测试软件在系统中的运行性能的。性能测试可以发生在各个测试阶段中，即使是在单元层，一个单独模块的性能也可以使用白盒测试来进行评估，然而，只有当整个系统的所有成分都集成到一起之后，才能检查一个系统的真正性能。

性能测试经常和压力测试一起进行，而且常常需要硬件和软件测试设备，这就是说，常常有必要的在一种苛刻的环境中衡量资源的使用（比如，处理器周期）。外部的测试设备可以监测测试执行，当出现情况（如中断）时记录下来。通过对系统的检测，测试者可以发现导致效率降低和系统故障的原因。

性能评测是一种性能测试，它对响应时间、事务处理速率和其他与时间相关的需求进行评测和评估。 性能评测的目标是核实性能需求是否都已满足。实施和执行性能评测的目的是将测试对象的性能行为当作条件(例如工作量或硬件配置)的一种函数来进行评测和微调。

6.10.1定义

◆ **性能（performance）：**

计算机系统或子系统实现其功能的能力。

对计算机系统或子系统执行其功能的能力的度最。例如，响应时间、吞吐能力、事务处理数。

◆ **性能测试：**

性能测试是为描述测试对象与性能相关的特征并对其进行评价，而实施和执行的一类测试，如描述和评价计时配置文件、执行流、响应时间以及操作的可靠性和限制等特征。

6.10.2分类

并发性能测试、压力测试、负载测试、疲劳强度测试、大数据量测试和速度测试等，其中并发性能测试是重点。

6.10.2.1并发性能测试

并发性能测试的过程是一个负载测试和压力测试的过程，即逐渐增加负载，直到系统的瓶颈或者不能接收的性能点，通过综合分析交易执行指标和资源监控指标来确定系统并发性能的过程。

并发性测试关注点：

- ✧ 以评价系统的当前性能；
- ✧ 当扩展应用程序的功能或者新的应用程序将要被部署时，负载测试会帮助确定系统是否还能够处理期望的用户负载，以预测系统的未来性能；
- ✧ 通过模拟成百上千个用户，重复执行和运行测试，可以确认性能瓶颈并优化和调整应用，目的在于寻找到瓶颈问题。

6.10.2.2负载测试

负载测试（Load Testing）是确定在各种工作负载下系统的性能，目标是测试当负载逐渐增加时，系统组成部分的相应输出项，例如通过量、响应时间、CPU 负载、内存使用等来决定系统的性能。负载测试是一个分析软件应用程序和支撑架构、模拟真实环境的使用，从而来确定能够接收的性能过程。

负载测试是一种性能测试。在这种测试中，将使测试对象承担不同的工作量，以评测和评估测试对象在不同工作量条件下的性能行为，以及持续正常运行的能力。 负载测试的目标是确定并确保系统在超出最大预期工作量的情况下仍能正常运行。 此外，负载测试还要评估性能特征，例如，响应时间、事务处理速率和其他与时间相关的方面。

注：事务是指”逻辑业务事务”。 这种事务被定义为将由系统的某个最终用户通过使用应用程序来执行的特定功能，例如，添加或修改。

6.10.2.3压力测试

压力测试（Stress Testing）是通过确定一个系统的瓶颈或者不能接收的性能点，来获得系统能提供的最大服务级别的测试。同时在测试系统的能力最高实际限度时，即软件在一些超负荷的情况，功能实现情况。如要求软件某一行为的大量重复、输入大量的数据或大数值数据、对数据库大量复杂的查询等。

压力测试是一种性能测试，实施和执行此类测试的目的是找出因资源不足或资源争用而导致的错误。如果内存或磁盘空间不足，测试对象就可能会表现出一些在正常条件下并不明显的缺陷。而其他缺陷则可能由于争用共享资源（如数据库锁或网络带宽）而造成的。压力测试还可用于确定测试对象能够处理的最大工作量。

压力测试的一个变种是一种被成为是敏感测试的技术。在有些情况（最常见的是在数学

算法中）下，在有效数据界限之内的一个很小范围的数据可能会引起极端的甚至是错误的运行，或者引起性能的急剧下降，这种情形和数学函数中的奇点相类似。敏感测试就是要发现在有效数据输入里可能会引发不稳定或者错误处理的数据组合。
压力测试是在一种需要反常数量、频率或资源的方式下运行系统。例如：

- 当平均每秒出现 1 个或 2 个中断的情形下，应当对每秒出现 10 个中断的情形来进行特殊的测试；
- 把输入数据的量提高一个数量级来测试输入功能会如何响应；
- 应当执行需要最大的内存或其他资源的测试用例；
  运行一个虚拟的操作系统中可能会引起大量的驻留磁盘数据的测试用例。

6.10.3性能测试（RUP）

包括以下内容：
• 基准测试 - 比较新的或未知测试对象与已知参照标准（如现有软件或评测标准）的性能。
• 争用测试 - 核实测试对象对于多个主角对相同资源（数据记录、内存等）的请求的处理是否可以接受。
• 性能配置 - 核实在操作条件保持不变的情况下，测试对象在使用不同配置时其性能行为的可接受性。
• 负载测试 （LAOD）- 核实在保持配置不变的情况下，测试对象在不同操作条件（如不同用户数、事务数等）下性能行为的可接受性。
• 强度测试（STRESS） - 核实测试对象性能行为在异常或极端条件（如资源减少或用户数过多）之下的可接受性。

6.10.4主要的性能评测包括

• 动态监测 - 在测试执行过程中，实时获取并显示正在执行的各测试脚本的状态。
• 响应时间/吞吐量 - 测试对象针对特定主角和/或用例的响应时间或吞吐量的评测。
• 百分位报告 - 数据已收集值的百分位评测/计算。
• 比较报告 - 代表不同测试执行情况的两个（或多个）数据集之间的差异或趋势。
• 追踪报告 -主角（测试脚本）和测试对象之间的消息/会话详细信息。

6.10.5性能测试的主要工具

- MI：Loadrunner，
- Compuware：Qaload
- Rational： Rational PerformanceStudio

6.10.6性能测试的原则

- 应该尽早的进行性能测试
- 性能测试的主要目的是为了系统调优。不可能对所有的系统功能都进行性能测试。在测

试设计时需要结合当时的实际系统，先分析软件可能存在的瓶颈，此时可依据80/20 原则（帕雷托原则）分析：对系统资源的利用；数据大量传输；数据转换（获取其它库表中数据转换为 XML 格式，或二进制）；再依此制定性能测试的方案

- 一个系统的性能是由很多 check point 组成的，这个在设计测试用例的时候就可以发现出来，比方说 A 用例是功能测试用的，B 用例是性能测试用的，在进行性能测试的时候，需要把相关的性能测试用例组织起来形成一个test suit 来执行性能测。

第 7 章 系统测试流程

第 1 节测试输入输出

测试生命周期中各个步骤的输入输出如下表：

步骤	输入（前提）	输出	阶段
业务培训			需求分析阶段
测试需求	软件业务规范 软件需求规格说明书 软件设计说明 <概要设计说明和详细设计说明>	软件需求的追踪矩阵 <3 份需求追踪文档>	需求分析阶段
测试计划	软件业务规范 软件开发计划 测试需求文档 <关联软件需求和设计文档>	测试计划【总的测试计划和阶段(跌代)测试计划】;【各个测试类型的测试计划】 <测试计划参见模板> 测试过程检查单	计划阶段 需求分析阶段
测试方案	软件需求规格说明书	测试方案	概要设计阶段(创建) 详细设计阶段(更新) 编码阶段(更新)
测试设计	测试方案 软件业务规范 软件测试计划 测试需求文档 <关联软件需求和设计文档>	手工测试案例	概要设计阶段(创建) 详细设计阶段(更新) 编码阶段(更新)
		测试工具评估和设计自动化测试案例及相关脚本	详细设计阶段(创建) 编码阶段(更新) 系统测试实施前完成。
环境配置	软件/硬件资源<测试计划> 环境配置文档 软件部署和升级文档	配置测试环境。 版本更新	系统测试之前
测试实施	测试计划 测试方案 测试案例 测试规程 集成测试报告	预测试报告 系统缺陷 (注意：缺陷跟踪) 测试日志 缺陷可跟踪性文档 <缺陷对应的测试案例及其他相关信息-工具可以管理>	系统测试阶段 <是个反复迭代的过程>
测试报告	测试计划 测试案例	测试天小结/周小结 任务报告	系统测试阶段

	系统缺陷记录	阶段性测试报告 系统测试报告 <测试报告参见模板>	
产生基线	系统测试文档 测试代码，工具	基线	系统测试阶段每次迭代
后期维护			

第 2 节业务培训

7.2.1必要性

开发人员和测试人员对需求的理解程度直接决定的软件的质量。

7.2.2培训机构和过程

略

第 3 节需求评审

7.3.1功能需求关注点

我们应该获得的需求信息：时间需求<计划、时间表、期限、和里程碑 >、功能需求文档<需求规格说明书、USECASE > 。

◆ **需求的内容应该包含以下信息：**

✧ 主要流程的流程图；
✧ 涉众用户分类、比例及各分类用户的详细描述；
✧ 场景（前置条件、后置条件、发生条件、主选流、备选流、错误处理、场景对应用户分类等等）；
✧ 页面各个元素的属性（各个按钮个名称、位置、格式等等、表单的输入框长度、输入数据的格式、范围、是否必填、文本框有没有对输入信息的显示格式做限制等等）；
✧ 前台页面划分（及各个部分主要元素的属性），后台模块划分（及各个部分主要元素的属性）；
✧ 功能点的优先级，是否开发完成，何时完成等信息；
✧ 功能点之间的关联、页面之间的关联（比如：一条数据可能在多个页面显示）；
✧ 一些操作并发时产生应该有什么结果，比如：增删改操作的并发、提问的并发性；
✧ 页面中的所有初始值数据；
✧ IE 自身功能的使用（比如：刷新、后退、复制等。）系统是否支持；
✧ 键盘快捷键，系统是否支持，支持哪些？
✧ 是否有系统中的保留字？
✧ 重要逻辑（设计完成后可以附带重要算法）。

7.3.2需求定义的静态测试

需求评审关注点：

- ✧ 兼容性
- ✧ 完备性
- ✧ 一致性
- ✧ 正确性
- ✧ 可行性
- ✧ 易修改性
- ✧ 健壮性
- ✧ 易理解性
- ✧ 易测试性和可验证性

第 4 节测试需求

需求测试是对需求正确性完备性等方面的检查，一般采用静态测试的方法。

方法：通过静态手工方法进行需求测试中最常使用的手段是同行评审，另外，在编写测试需求和测试案例过程时，也是对需求的一次测试。

需求评审的参与者当中，必须要有用户或用户代表参与，同时还需要包括项目的管理者，系统工程师和相关开发人员、测试人员、市场人员、维护人员等。在项目开始之初就应当确定不同级别、不同类型的评审必须要有哪些人员的参与，否则，评审可能会遗漏掉某些人员的意见，导致日后不同程度的返工。

◆ **需求跟踪矩阵文档中应该包含：**

测试需求内容、编号、测试需求优先级、是否实现、实现时间、变更等。

7.4.1如何编写测试需求

通常会有软件需求规约（以下简称 SRS）和用例（以下简称 UC）——当然，也可能是一份包含 UC 的 SRS。通过对 SRS 和 UC 的阅读，我们可以从文档对特性和业务流程的描述中获得对软件所涉及的业务的一个基本的认识。比如用户在处理实际业务时都要作些什么，多个业务之间的先后顺序是怎样的，用户在处理业务是对于哪些地方有特别的要求，等等。这部分规则，将成为我们的测试需求中最基本的一部分。也可以把想到的情况都写出来，不单单包括一个特性或一个基本流同某些备选流组成的场景，更多的强调对业务的分析，就是考虑用户处理实际业务时将做什么。

随着工作的展开，开发部门的架构设计文档和详细设计文档也将陆续提交，这时候，我们可以根据设计文档来对已有的测试需求进行增补。注意，这里我们对于设计文档中提到的内容要有选择的采用，只有同 SRS 或 UC 中已经定义的部分相符的内容，才可以用来调整我们的测试需求。而同软件需求不相符的部分，则需要同设计人员和需求人员一起讨论，确定下以哪一方作为基准，决定是否需要调整软件需求，然后对测试需求进行相应的增补或者调整。比如对于一些算法，需要考虑设计文档中定义的，同系统实现相关的那些计算公式，是否同软件需求中描述的算法表达的是否是同一个意思？而对于一些约束或者业务规则，设计文档中描述的是否同需求中的相应部分一致？

提交供内部测试的应用程序时，测试人员手头上应该已经准备好了绝大部分测试用例和测试数据，但总是有些缺陷的出现是出乎我们意料的，或者说是已有的测试需求和测试用例未能覆盖的。那么，对于这部分缺陷，也应当添加到测试需求中，并设计相应的测试用例，以便于下次版本迭代时进行参考。

7.4.2测试需求包含的内容

测试需求把需求点转化为测试点，一个需求点包含 N 个测试点。需求中还应该说明该测试点的测试方法测试策略等。注意，测试需求和需求之间应该是关联的。

第 5 节测试计划

测试并不是一个随机的活动，测试必须被计划，并且被安排足够的时间和资源。测试活动应当受到控制，测试的中间产物应当被评审并纳入配置管理。

测试计划是一个关键的管理功能，它定义了各个级别的测试所使用的策略、方法、测试环境、测试通过或失败准则等内容。测试计划的目的是要为有组织的完成测试提供一个基础。从管理的角度来看，测试计划是最重要的文档，这是由于它帮助管理测试项目。如果一个测试计划是完整并且经过深思熟虑的，那么测试的执行和分析将平滑的进行。

测试计划可以分级，也可以是一个总的计划，并且测试计划是一个不断演进的文档。如果不考虑应用软件的最初来源（复用的组件或已实现的组件），软件需求是测试活动的驱动。因此，测试计划应当关注于文档化的需求。此外，支持测试的过程应当被文档化下来以创建一个可重复的过程，该过程将保证开发工作产品的质量。

主测试计划（ Master Testing Plan ），至少包含所有预期的测试活动（单元测试、集成测试、系统测试/用户验收测试等）的总工作量和所有主要工作的责任以及在所有测试级别上应交付的物件。综合的测试计划应当结合总的项目和程序开发计划，并保证资源和责任在项目中尽可能早的被了解和分配。 其目的是要提供一个大的活动图并且协调所有的测试工作。这个文档将作为项目开发计划的一部分，合并至项目开发计划中。CMM 流程中，这个文档将会是项目测试经理和项目经理共同完成编写。

测试计划就是要从时间上、人员上、环境上、技术上、关系上、组织能力上、资金上等方面对测试工作做一个规划。

7.5.1系统测试计划内容

在文档中确定对项目进行的系统测试的策略、测试类型、资源、项目里程碑（将会执行的活动）、各个阶段测试介入时期、度量项(项目测试的周状态报告中要收集和分析度量数据)、可交付工件等。

7.5.1.1测试计划内容应该做到：

- 在检测主要缺陷方面有一个好的选择
- 提供绝大部分代码的覆盖率

- 是灵活的
- 易于执行、回归和自动化
- 定义要执行测试的种类
- 清晰的文档化了期望的结果
- 当缺陷被发现的时候，提供缺陷核对
- 清晰的定义测试的目标
- 明确测试的策略
- 清晰定义测试的出口标准
- 没有冗余
- 确认风险
- 文档化测试的需求
- 定义可交付的测试件

测试计划阶段输入输出参见第一节；测试计划内容参加测试计划模板

7.5.1.2测试计划根据[SDP 软件开发计划]的要求：

在开发 SDP 阶段，应该清楚：在进度计划中，需要确定有那些发布版本，即发布多少次给客户，每个发布版本，作为项目生命周期的重要里程碑。然后，根据发布版本的约束，和项目生命周期模型，定义项目的生命周期，项目生命周期包括项目的开始日期、结束日期，以及在此范围内所划分的阶段，每个阶段的开始日期、结束日期、阶段的目标”，这样就有利于评估测试资源（包括人力和天数）。

7.5.2测试策略

测试策略用于说明某项特定测试工作的一般方法和目标。

7.5.2.1测试策略的内容

- 实施的测试类型和测试的目标
- 实施测试的阶段
- 技术
- 用于评估测试结果和测试是否完成的评测和标准
- 对测试策略所述的测试工作存在影响的特殊事项

7.5.2.2测试策略一般方法

- 确定测试的需求
- 评估风险并确定测试优先级
- 确定测试策略

确定测试需求

测试需求所确定的是测试内容，即测试的具体对象。在分析测试需求时，可应用以下几条规则：

- 测试需求必须是可观测、可测评的行为。如果不能观测或测评测试需求，就无法对其进行评估，以确定需求是否已经满足。
- 在每个用例或系统的补充需求与测试需求之间不存在一对一的关系。用例通常具有多个测试需求；有些补充需求将派生一个或多个测试需求，而其他补充需求（如市场需求或包装需求）将不派生任何测试需求。
- 测试需求可能有许多来源，其中包括用例、用例模型、补充需求、设计需求、业务用例、与最终用户的访谈和软件构架文档等。应该对所有这些来源进行检查，以收集可用于确定测试需求的信息。

◆ **一般需要重点关注以下测试需求：**

- 功能性测试需求
- 性能测试需求
- 可靠性测试需求
- 其他测试需求

功能性测试需求

功能性测试需求来自于测试对象的功能性行为说明。每个用例至少会派生一个测试需求。对于每个用例事件流，测试需求的详细列表至少会包括一个测试需求。

性能测试需求

性能测试需求来自于测试对象的指定性能行为。性能通常被描述为对响应时间和/或资源使用率的某种评测。性能在各种条件下进行评测，这些条件包括：

- 不同的工作量和/或系统条件
- 不同的用例功能点
- 不同的配置
- 性能需求在补充需求中说明。检查这些材料，对包括以下内容的语句要特别注意：
- 时间语句，如响应时间或定时情况
- 指出在规定时间内必须出现的事件数或用例数的语句
- 将某一项性能的行为与另一项性能的行为进行比较的语句
- 将某一配置下的应用程序行为与另一配置下的应用程序行为进行比较的语句
- 一段时间内的操作可靠性（平均故障时间或 MTTF）
- 配置或约束

可靠性需求

测试可靠性需求有若干个来源，它们通常在补充需求、用户界面指南、设计指南和编程指南中进行说明。

检查这些工件，对包括以下内容的语句要特别注意：

- 有关可靠性或对故障、运行时错误（如内存减少）的抵抗力的语句
- 说明代码完整性和结构（与语言和语法相一致）的语句

- 有关资源使用的语句

评估风险和确定测试优先级

成功的测试需要在测试工作中成功地权衡资源约束和风险等因素。为此，应该确定测试工作的优先级，以便先测试最重要、最有意义或风险最高的用例或构件。为了确定测试工作的优先级，需执行风险评估和实施概要，并将其作为确定测试优先级的基础。

评估风险和确定测试优先级的步骤

确定测试需求只是确定测试内容的一部分。还应该确定测试内容的优先级和先后顺序。之所以要执行这一步骤，是为了以下几个目的：

- 确保将测试工作的重点放在最适当的测试需求上
- 确保尽早地处理最关键、最有意义或风险最高的测试需求
- 确保在测试中考虑到了任意依赖关系（序列、数据等等）

◆ **要评估风险并确定测试优先级，可执行以下三个步骤：**

- 评估风险
- 确定实施概要
- 确定测试优先级

评估风险

在开始时可确定并说明将要使用的风险程度指标，例如：

- H - 高风险，无法忍受。极易遭受外部的风险。公司将遭受巨大的经济损失、债务或不可恢复的名誉损失。
- M - 中等风险，可以忍受，但是不希望其出现。遭受外部风险的可能性最小，公司可能会遭受经济损失，但只存在有限的债务或名誉损失。
- L - 低风险，可以忍受。根本不会或不太可能遭受外部的风险，公司只有少许经济损失或债务或根本没有损失。公司的名誉也不会受到影响。

在确定风险程度指标之后，列出测试对象中的每个用例或构件。为列表中的每一个用例或构件确定一个风险程度指标，并简要说明您选择相应值的原因。

可以从三个方面来评估风险：

- 影响 - 指定用例（需求等）失效后将造成的影响或后果
- 原因 - 用例失效所导致的非预期结果
- 可能性 - 用例失效的可能性。

选择一个方面，确定风险程度指标并说明您所作选择的原因。不必为风险的每个方面都确定一个指标。然而，如果确定了一个低风险指标，最好再从另一个方面来评估该风险，以确保它的确是低风险。

确定实施概要

在开始时可确定和说明将要使用的实施概要程度指标，例如：

- H - 使用得相当频繁，在每个时期会使用很多次，或者由多个主角或用例使用。
- M - 使用得比较频繁，在每个时期会使用若干次，或者由若干个主角或用例使用。
- L - 很少使用，或者由很少的几个主角或用例使用。

所选择的实施概要指标应该基于用例或构件的执行频率，其中包括：

一个主角（或用例）在给定时间内执行用例（或构件）的次数，或者执行用例（或构件）的主角（或用例）的数量。通常，用例或构件的使用次数越多，实施概要指标也就越高。

在确定实施概要程度指标之后，列出测试对象中的每个用例或构件。为列出的每一项确定一个实施概要指标并且说明每个指标值的理由。性能分析文档中的信息可用于此评估。

确定测试优先级

在开始时可确定和说明将要使用的测试优先程度指标，例如：

- H - 必须测试
- M - 应该测试，只有在测试完所有 H 项后才进行测试
- L - 可能会测试，但只有在测试完所有 H 和 M 项后才进行测试

在确定要使用的测试优先程度指标之后，列出测试对象中的每个用例或构件。然后，为列出的每一项确定一个测试优先级指标并且说明您的理由。以下为确定测试优先级指标提供了一些指南。

当确定每一项的测试优先级指标时，应考虑下列各项：

- 先前确定的风险程度指标值
- 先前确定的实施概要程度指标值
- 主角说明（主角是否有经验？他们是否能够接受变通方法？等等）
- 合同责任（如果不交付用例或构件，测试对象能否被接受？）

确定测试策略

略

7.5.3测试优先级

尽管我们的测试是需要按照一定的级别进行，但资源和时间是有限的，实际上我们不可能无休止的进行测试，因此在有限的时间和资源下如何有重点的进行测试是测试管理者需要充分考虑的事情。例如，在单元测试的时候，对于哪些函数我们需要重点测试，哪些函数可以粗略测试，哪些函数可以不测试；而对于系统测试，则要考虑首先应当保证哪些功能的测试，其次应当保证哪些功能的测试等等。测试的重点选择需要根据多个方面考虑，包括测试对象的关键程度，可能的风险，质量要求等等。这些考虑与经验有关，随着实践经验的增长，你的判断也会更有效。

7.5.4工作量评估

主要依据：系统需求功能点，软件开发计划书，测试的跌代次数，每次跌代的质量要求和总质量要求，经验等。

7.5.5结束准则

测试是不可能穷尽的，资源和时间是有限的。因此我们在做测试的时候需要分析哪些功能是对用户很关键的，在这些功能中出现某类型错误对用户是不可接受的，而相对其它一些功能，出现的错误是可以容忍的，这样，我们在测试的时候，重点就应当去寻找那些用户不可接受的错误，而不是漫无目的的去搜索错误。同时我们应当对测试定义合理的出口标准，这是因为测试是没有穷尽的，系统中的问题你总是可以一直发现下去，然而我们不能无休止的去寻找这些问题。当条件满足的时候，我们就应当停止测试。而测试出口条件的设置需要考虑系统的质量要求及系统的资源要求。曾经有人说过：当时间和资源用尽的时候，测试也就停止了。这是没有办法的最好办法。

不同的系统有着不同的质量要求，对于质量要求严格的系统，可能需要进行长时间的，全面的测试，尽可能的去挖掘系统中的缺陷。然而对于质量要求不是很严格的系统，系统是允许可以出现错误的，因此我们通过测试是要使得系统的缺陷数量或严重程度能够降到可接受的范围内。

◆ **可能用到的一些指标：**

- 查出了预定数目的错误；
- 达到一定覆盖率；
- 错误强度曲线下降到预定的水平；
- 达到测试计划中所规定的完备性；
- 使用了特定的测试用例设计方法；
- 其他标准。

7.5.6软件风险管理

7.5.6.1软件风险管理过程

软件风险管理过程分为五个步骤：

风险识别，风险分析，风险计划，风险跟踪和风险应对。

风险识别

风险识别过程的活动是将不确定性转变为明确的风险陈述。包括下面几项，他们在执行时可能是重复，也可能是同时进行的：

进行风险评估。在项目的初期，以及主要的转折点或重要的项目变更发生时进行。这些变更通常指成本、进度、范围或人员等方面的变更。

系统地识别风险。采用下列三种简单的方法识别风险：风险检查表，定期会议（周例会上），日常输入（每天晨会上）。

将已知风险编写为文档。通过编写风险陈述和详细说明相关的风险背景来记录已知风险，相应的风险背景包括风险问题的何事、何时、何地、如何及原因。

交流已知风险。同时以口头和书面方式交流已知风险。在大家都参加的会议上交流已知风险，同时将识别出来的风险详细记录到文档中，以便他人查阅。

风险分析

风险分析过程的活动是将风险陈述转变为按优先顺序排列的风险列表。包括以下活动：

确定风险的驱动因素。为了很好地消除软件风险，项目管理者需要标识影响软件风险因素的风险驱动因子，这些因素包括性能、成本、支持和进度。

分析风险来源。风险来源是引起风险的根本原因。

预测风险影响。如果风险发生，就将可能性和后果来评估风险影响。可能性被定义为大于 0 而小于 100，分为 5 个等级（1、2、3、4、5）。将后果分为 4 个等级（低，中等，高，关键的）。采用风险可能性和后果对风险进行分组。

对风险按照风险影响进行优先排序，优先级别最高的风险，其风险严重程度等于1，优先级别最低的风险，其风险严重程度等于20。对级别高的风险优先处理。

风险计划

风险计划过程的活动是将按优先级排列的风险列表转变为风险应对计划。包括以下内容：

- 制定风险应对策略。风险应对策略有接受、避免、保护、减少、研究、储备和转移几种方式。
- 制定风险行动步骤。风险行动步骤详细说明了所选择的风险应对途径。它将详细描述处理风险的步骤。

风险跟踪

风险跟踪过程的活动包括监视风险状态以及发出通知启动风险应对行动。包括以下内容：

- 比较阈值和状态。通过项目控制面板来获取。如果指标的值在可接受标准之外，则表明出现了不可接受的情况。
- 对启动风险进行及时通告。对要启动的风险，在每天的晨会上通报给全组人员，并安排负责人进行处理。
- 定期通报风险的情况。在定期的会议上通告相关人员目前的主要风险以及他们的状态。

风险应对

风险应对过程的活动是执行风险行动计划，以求将风险降至可接受程度。包括以下内容：

- 对触发事件的通知作出反应。得到授权的个人必须对触发事件作出反应。适当的反应包括回顾当前现实以及更新行动时间框架，并分派风险行动计划。
- 执行风险行动计划。应对风险应该按照书面的风险行动计划进行。
- 对照计划，报告进展。确定和交流对照原计划所取得的进展。定期报告风险状态，加强小组内部交流。小组必须定期回顾风险状态。
- 校正偏离计划的情况。有时结果不能令人满意，就必须换用其他途径。将校正的相关内容记录下来。

7.5.6.2风险的驱动因素

风险因素是以如下的方式定义的：

- 性能风险——产品能够满足需求且符合于其使用目的的不确定的程度。
- 成本风险——项目预算能够被维持的不确定的程度。
- 支持风险——软件易于纠错、适应及增强的不确定的程度。
- 进度风险——项目进度能够被维持且产品能按时交付的不确定的程度。

7.5.6.3风险数据库

风险数据库是用来记录风险，跟踪风险处理过程，并能够对风险进行简单查询和统计的风险管理工具。

风险内容

风险内容应该包括：

- 编号，识别日期和识别者姓名，
- 风险类别，
- 风险标题，
- 风险评估

  [风险背景、驱动因素（性能成本进度技术）、风险来源、可能性、后果、时间框架、影响]
- 风险计划

  [应对策略（风险应对策略用接受、避免、保护、减少、研究、储备和转移）、行动步骤]
- 风险跟踪

  [风险状态、批注] 、风险应对[负责人、日期、批注]

风险状态

- 风险的状态： Watch，Execute Contingency，Mitigate，Transfer，Avoid，Retired；
- 风险的另一个状态分类：潜伏/发生/释放；

风险列表

编号	类型	风险识别	规避计划	风险应对
	产品规模	与要建造或要修改的软件的总体规模相关的风险		
	商业影响	与管理或市场所加诸的约束相关的风险		
	客户	与客户的素质以及开发者		

	特性	和客户定期通信的能力相关的风险		
	过程定义	与软件过程被定义的程度以及它们被开发组织所遵守的程度相关的风险		
	开发环境	与用以建造产品的工具的可用性及质量相关的风险		
	技术难题	与待开发软件的复杂性以及系统所采用的新技术相关的风险		
	人员数目及经验	与参与工作的软件工程师的总体技术水平及项目经验相关的风险		
EG：				
1	环境	系统所在的服务器不能正常运行	1。备用服务器 2。定期维护系统所在的服务器	调用备用服务器
2	管理	缺乏受过技术培训（如性能测试工具）的员工	做估计时，预留适当的学习时间 维护额外的缓冲资源 定义项目详细的培训计划 4。实施交叉培训	调用其他熟悉测试工具的测试人员完成
3	管理	项目组人力资源紧张，不能及时解决测试出的软件问题	项目组分配合理的人力资源，及时解决问题	1．重新分配人力资源 2。或更改开发计划。
4	管理	项目组交付予测试组进入系统测试前，单元测试和集成测试未达到出口准则	开发组应强调做好单元测试和集成测试，达到出口准则并交付测试组进行系统测试	测试组将此问题反映给 SQA、部门经理，要求项目组按照定制的计划执行，达到出口准则
5	技术	不实际的测试进度	协商一个比较合理的进度 标识并行工作/任务 尽早资源到位	1。更新计划进度 2。重新分配人力资源
6	管理	合同中没有确定外部接口问题，这样的工作可大可小，风险大	最好早期确认所要的接口标准	
7	管理	最终用户是哪些人，有些什么样的特殊习惯	留一些渠道能够跟最终用户沟通	根据客户反馈做一点的修改
8	管理	一般客户在确认测试时是否要求我们公司的资源	希望首先确认	留一定的确认测试时间
9	技术	特殊的技术影响我的测试，公司采用先进的开发方法	对公司的新技术进行测试方面的预研究	根据优先级别判断是否采用
0	管理	项目实际开发不能按现在	项目经理首先是熟悉现有流	项目组应得到流程方

		新的流程操作，不明确各活动的进入和输出准则。	程，严格按照流程进行开发管理	面的培训，调整开发时间、进度

7.5.6.4风险管理关注点

- 对于该项目的用途而言，哪种功能最重要?
- 哪种功能对用户最明显?
- 哪种功能对安全影响最大?
- 哪种功能对用户最有用?
- 对客户来说，该应用软件的哪个部分最重要?
- 在开发过程中，该应用软件的哪个部分可以最先测试?
- 哪一部分代码最复杂，容易导致出现错误?
- 哪一部分的应用程序是在急迫或在惊恐的情况下开发出来的?
- 哪一部分程序与过去项目中引起问题的部分相类似/有关?
- 哪一部分程序与过去项目中需要大量维护的部分相类似/有关?
- 需求和设计的那些部分不清楚或不容易读?
- 开发人员认为在应用软件中哪些部分是高风险的?
- 哪些问题能造成最差的发行?
- 哪些问题最能引起用户抱怨?
- 哪些测试可以容易地覆盖多种功能?
- 哪些测试在覆盖高风险部分的测试时使用时间最少?

第 6 节测试[案例]设计

测试用例覆盖清单主要是根据确定的测试需求和产生的测试用例，在完成测试案例的同时也需要完成该文档。

◆ **如何划分测试用例的粒度**

我们是不太可能在一个测试用例包含所有测试需求的，因为众多的功能以及不同的路径组合将使这样一个测试用例像巨无霸一般，完全不具有可操作性。关注有效功能。

有效功能：就是指在被测应用所涉及的实际业务中，当用户在手工状态下进行工作时，整个业务流程中对用户来说，具有实际意义那些功能。这个功能的特征是当我们把这个功能单独从计算机软件还原到用户的原始手工状态时，它的完成可以作为用户实际业务的一个阶段性结束的标志，而不是一旦从这个业务流程中独立出来就失去了意义。而该业务完成后，可以为其他用户或业务提供所需要的信息。

区分”有效功能”的关键有如下两个：

1.这个功能是可以还原到用户原始的手工业务流程中去的。我们的计算机和软件，都是为了帮助用户解决手工业务中一些烦琐和低效的问题，而提出的一些忠实于原始工作方法或略有变通的解决方案，并不是要改变用户全部的业务流程。所以，应该从用户实际业务的角度来判断功能是否有效。

2.这个功能是否可以标志着用户实际业务的一个阶段性结束？并且这项业务完成之后，被完成的业务实体是否可以交付给其他用户或业务以供完成下面的工作？

◆ **其他注意事项**

- 测试数据和逻辑分开

- 步骤不必写得特细
- **两个错误观点**
- 每个人都可以想出很多的测试场景，没必要写
- 测试用例发现不了错误，错误都是做用例之外操作时发现的。

7.6.1测试案例分类

测试案例应包括：功能测试，性能测试，可靠性，兼容性，安全性等。其中，功能测试用例包括 ：常规测试点用例,功能流用例，并发性用例，关联性用例等等。

- **功能测试类型**
- 功能测试<正确性测试/容错性测试/并发逻辑测试/关联内容测试等等>
- 安全测试
- 性能测试<疲劳测试/压力测试/响应时间>
- 强度测试<使资源出现短缺的情况，检查系统的应对>
- 可移植性<兼容性测试>
- 容量测试
- 恢复测试
- 配置测试
- 安装、卸载测试
- 用户界面测试
- 比较测试
- 接口间测试
- 数据库测试
- 性能测试
- 压力测试
- 疲劳度测试
- 负载测试

7.6.2测试案例设计方法

- 白盒测试 注意逻辑覆盖和路径覆盖
- 性能测试 注意场景设计及监控方案
- 功能测试 注意正确性测试/容错性测试/并发逻辑测试/关联内容测试等等

7.6.3测试案例内容标准

- 准确性
- 可信度
- 有效性/数量
- 可执行性
- 可重用性
- 覆盖率/数量

- 测试数据
- 测试数据不仅包含输入数据，还包含输出数据。根据等价类、边界值、判定表、因果图等方法设计测试数据。

7.6.4测试数据的设计规则

◆ **针对性**

首先，它是针对错误本身，是依据测试点、测试目的来制定的。

第二，它必须针对题目，题目提供的输入的限制，它不但决定了题目的难度，还决定了程序应采用的算法及数据结构。而选手通过对上下限的测试，一方面可以发现疏漏的情况，更重要的是它能为你是否需要继续优化程序提供必要的依据。

第三，测试数据应针对数据结构。

第四，测试数据必须针对算法。首先，算法可以避免的错误基本无须调试，其次，应针对重点算法的缺点作调试：搜索算法应用大数据测定其效率，动态规划应密切注意其空间效率；而贪心算法则应注重提高其正确率。再次，应多针对易犯的错误设计测试数据。同时要有足够多的测试数据做支撑。

最后，应针对题目设计者的意图，在最后的调试阶段多想一些评测时可能出现的数据，尽可能使自己的程序面向评委。当某个特殊数据无法通过时，可以对其单独作特殊化处理。

◆ **广泛性**

广泛性是指测试数据应能普遍地发现错误，而不是针对某一个错误反复测试。

首先，广泛性要求测试数据应尽可能多地包含题目内容。这样一方面可以多发现错误，一方面可以检验各个模块之间协调得是否得当。

其次，在初期的调试中应多设计一些测试数据，通过变换其一部分内容可以针对不同内容进行测试。这样有利于节省时间，而且在一些较简单的题目中可以发挥事半功倍的效果。测试数据还应能抓住题目的每一个细节和每一种可能的情况。

第 7 节环境配置

把一个基线部署到相应的环境上，通常由配置管理人员和集成人员来完成。

7.7.1安装阶段的测试准备

- 安装计划
- 安装流程图
- 安装文件和程序清单
- 测试安装程序给出测试结果
- 将程序运行的软硬件要求放入产品说明中
- 对于新操作人员的使用说明书
- 对于新使用者的操作说明和操作流程
- 安装过程中的各项可能发生的结果的说明

7.7.2配置管理

7.7.2.1软件配置管理概念：

软件配置管理是通过在软件生命周期的不同时间点上对软件配置进行标识并对这些标识的软件配置项的更改进行系统控制，从而达到保证软件产品的完整性和可溯性的过程。

配置：软件系统的功能属性；配置项：软件系统的逻辑组成，即与某功能属性相对应的文档或代码等。

7.7.2.2软件配置管理的四个基本过程：

- 配置标识：标识组成软件产品的各组成部分并定义其属性，制定基线计划。
- 配置控制：控制对配置项的修改。
- 配置状态发布： 向受影响的组织和个人报告变更申请的处理过程，通过的变更及他们的实现情况等。
- 配置评审：确认受控软件配置项满足需求并就绪。

7.7.2.3配置库：

对各基线内容的存储和管理的数据库：

- 开发库：程序员工作空间，始于某一基线，为某一目的开发服务，开发完成后，经过评审回归到基线库。
- 基线库：包括通过评审的各类基线，各类变更申请的记录和统计数据。
- 产品库：是某一基线的静态拷贝，基线库进入发布阶段形成产品库。

第 8 节测试执行

项目进入测试阶段时，要求每周产生项目周状态报告，主要是收集针对项目的有关度量项和真正做到监控项目状态的目的。需要有统一的工具管理周状态报告。

7.8.1日常工作

依据测试用例执行测试，提交缺陷，跟踪缺陷。

7.8.2漏测分析

对于测试来说，进行漏测分析有助于测试的不断改进。漏测的分析不仅仅分析版本发布之后的缺陷，还可以针对内部每轮系统测试版本的漏测问题进行分析。一般对于每个缺陷我们需要从以下几个角度进行分析： 该缺陷是否能够在内部或者上一个系统测试版本中被发

现？为什么没有被发现？如何避免这类情况产生？ 该缺陷是否有相应的用例？如果没有，则设计用例，同时还需要分析是否还有类似的问题？针对这些类似的问题是否也需要补充相应的用例？ 该缺陷是否属于因开发修改其它缺陷而引入的新缺陷？为什么会引入新的缺陷？回归的时候为什么没有考虑这方面测试，是否回归分析不够全面？如何改进？

第 9 节测试报告

测试报告的关注点，参见模版。我们这里提到的报告，包括很多，比如：日报，周报，阶段报告，系统报告等。

7.9.1测试覆盖率分析

测试的目标是尽可能的去发现错误，去寻找被测对象与既定的规格不一致的地方。因此我们在进行测试用例设计的时候，首先应当从对需求和设计的了解，使用已有的经验去挖掘测试用例，包括正常的用例和异常的用例。在这个基础上，我们再使用需要的覆盖率准则来衡量已有的测试设计并补充相应的用例来达到需要的覆盖率准则。如果我们仅以覆盖率目标来指导测试的话，会丢失很多重要的信息，陷入到追求覆盖率数字的极端中去。

7.9.2测试度量和缺陷分析

在测试过程中，我们需要度量的基本数据包括：

- ✧ 测试投入的工作量和成本数据；
- ✧ 测试任务完成情况；
- ✧ 测试规模数据；
- ✧ 测试结果数据，包括缺陷数据，覆盖率数据等

有了充分的度量数据，管理人员就有了更好调整测试的依据，同时也为今后类似的项目提供了参考。但是有一点需要紧记：度量绝对不能用于对个人或组织的考核！

7.9.3测试流程中的模板

测试计划、测试案例、测试报告等等，各种报告参见模版。

第 8 章 功能自动化测试

软件企业实施测试自动化，绝对不是拍脑袋说干就能干好的，它不仅涉及测试工作本身流程上、组织结构上的调整与改进，甚至也包括需求、设计、开发、维护及配置管理等其他方面的配合。

一般来说，一个这样的软件开发团队可以优先开展自动化测试工作：测试－开发人员比例合适，比如 1：1 到 1：1。5；开发团队总人数不少于 10 个。当然，如果你的公司只有三五个测试人员，要实施自动化测试绝非易事；不过可以先让一个、两个测试带头人首先试着开展这个工作，不断总结、不断提高，并和层层上司经常汇报工作的开展情况，再最终决定是否全面推行此事。

开展自动化测试要具备从测试主管到开发工程师，测试执行者等结构合理的团队，要考虑到人员变更风险。

目前自动化测试，一般仅仅局限于回归测试。

8.1.1不适合自动化测试的领域

- 一次性项目
- 项目周期很短的项目
- 业务规则复杂的项目
- 美观、音质、易用性测试
- 系统不稳定
- 涉及物理交互

8.1.2何时开展自动化测试

◆ **何时使用自动化测试**

- 项目没有严格的时间压力
- 具有良好定义的测试策略和测试计划（知道要测试什么，知道什么时候测试）
- 对于自动化测试你拥有一个能够被识别的测试框架和候选者
- 能够确保多个测试运行的构建策略
- 多平台环境需要被测试
- 拥有运行测试的硬件
- 拥有关注在自动化过程上的资源

◆ **何时使用用手工测试**

- 没有标准的测试过程
- 没有一个测试什么、什么时候测试的清晰的蓝图
- 在一个项目中，你是一个新人，并且还不是完全的理解方案的功能性和或者设计
- 你或者整个项目在时间的压力下
- 在团队中没有资源或者具有自动化测试技能的人
- 没有硬件

8.1.3自动化工具使用要点

功能测试实行自动化的使用，要注意以下几点：

录制：这是必备功能，主要是看工具支持的协议以及是否有相关的 java 测试插件，是否能够录制我们的测试目标对象是最重要的一点；

分析结果：看是否具有方便的分析定制，以及结果是否在导出、统计中满足要求；

检查点：主要是看有没有对下表所示的几种检查点的支持，或者要用到的检查点

检测点类型	描述	应用举例
Page 检测点	检查 Web 页面的属性。	检查 Web 页面中是否包含”死”连接，或者检查连接一个 Web 页面所需要的时间。
Text / Text 区域检测点	检查在窗口或 Web 页面的指定位置是否显示了内容正确的文本。	检查在一个对话框中的指定位置，是否显示了内容正确的文本。 Text 检测点用于检测基于 Web 的应用，Text 区域检测点用于检测传统的 Windows 应用。
Standard 检测点	也叫做 Object 检测点，用于检测标准界面元素的状态。	检查一个编辑框的值是否正确，或者是检查一个复选框是否被选中。
Image 检测点	检查界面上图片的正确性。	检查 Web 页面上图片的正确性。
Table 检测点	检查表格中内容的正确性。	检查表格中的内容是否正确。
Database 检测点	检测 Windows 应用或 Web 应用访问数据库时，数据内容的正确性。	检查数据库查询的正确性。
XML 检测点	检查 XML 文档内容的正确性	XML 检测点有两种——XML 文件检测点和 XML 应用检测点。XML 文件检测点用于检查一个 XML 文件；XML 应用检测点用于检查一个 Web 页面的 XML 文档。
Bitmap 检测点	检查从 Web 页面或 Windows 窗口中捕获的某一部分区域。	检查一个 Windows 窗口（或这个窗口的某一部分）显示的正确性。

参数化：主要有以下 3 种情况

- 数据输入的参数化—可以循环利用我们的脚本
- 检测点的参数化——可以减少我们的维护成本
- 运行、分析测试脚本——可以方便我们调试、维护脚本

输出数值：对程序本身输出的数值进行捕获，加以利用，这样也可以调高我们维护脚本的效率；

使用通配符：我们工作的时候适当使用通配符，可以提高我们的工作效率，减少脚本的改动

组织测试脚本：适当的组织测试脚本，可以减少我们测试脚本编制和调试的次数，提高重用度。主要有几种方法：分割、添加、拷贝、调用。

第 9 章 性能测试实例

性能测试：测试软件的运行性能。这种测试常常与强度测试结合进行，需要事先对被测软件提出性能指标，如传输连接的最长时限、传输的错误率、计算的精度、记录的精度、响应的时限和恢复时限等。

第 1 节性能测试指标

9.1.1SQL 数据库

1. User Connections (用户连接数，也就是数据库的连接数量)；
2. Number of deadlocks/Sec/－Total （数据库死锁）
3. Memory\ Availalle Mbyte 内存监控 (可用内存)
4. Physicsdisk \disk time \－Total（磁盘读写总时间）（出现瓶颈时检查读磁盘的时间长还是写磁盘的时间长）
5. Butter Caile hit(数据库缓存的选取命中率)
6. 数据库的命中率不能低于 92％

9.1.2Web Server

1. Processor \ Processon time \ Tatol cpu 时间
2. Memory \ Availalle MbyteAvai 应用服务器的内存
3. Requst Quened 进入 HTTP 队列的时间；队列/每秒
4. Total request 总请求数时间
5. Avg Rps 平均每秒钟响应次数＝ 总请求时间 / 秒数
6. Avg time to last byte per terstion （mstes）平均每秒迭代次数 ； 上一个页面到下一个页面的时间是你录入角本的一个过程的执行
7. Http Error 无效请求次数
8. Send 发送请求次数字节数

9.1.3UNIX 资源监控指标和描述

- 平均负载：系统正常状态下，最后 60 秒同步进程的平均个数 ；
- 冲突率：在以太网上监测到的每秒冲突数 ；
- 进程/线程交换率：进程和线程之间每秒交换次数 ；
- CPU 利用率：CPU 占用率（％）；
- 磁盘交换率：磁盘交换速率 ；
- 接收包错误率：接收以太网数据包时每秒错误数 ；
- 包输入率：每秒输入的以太网数据包数目 ；
- 中断速率：CPU 每秒处理的中断数 ；

- 输出包错误率：发送以太网数据包时每秒错误数 ；
- 包输入率：每秒输出的以太网数据包数目 ；
- 读入内存页速率：物理内存中每秒读入内存页的数目 ；
- 写出内存页速率:每秒从物理内存中写到页文件中的内存页数目或者从物理内存中删掉的内存页数目 ；
- 内存页交换速率：每秒写入内存页和从物理内存中读出页的个数 ；
- 进程入交换率：交换区输入的进程数目 ；
- 进程出交换率：交换区输出的进程数目 ；
- 系统 CPU 利用率：系统的 CPU 占用率（％);
- 用户 CPU 利用率：用户模式下的 CPU 占用率（％） ；
- 磁盘阻塞：磁盘每秒阻塞的字节数;

9.1.4windows 资源监控指标和描述

- ProcessorTime： 指服务器 CPU 占用率，一般 平均达到 70%时，服务就接近饱和;
- Memory Available Mbyte ： 可用内存数，如果测试时发现内存有变化情况也要注意，如果是内存泄露则比较严重;
- Physicsdisk Time ： 物理磁盘读写时间情况;

具体请参见 windows 的 help 文档。

第 2 节性能测试环境

配置测试环境是测试实施的一个重要阶段,测试环境的适合与否会严重影响测试结果的真实性和正确性。测试环境包括硬件环境和软件环境，硬件环境指测试必需的服务器、客户端、网络连接设备以及打印机/扫描仪等辅助硬件设备所构成的环境；软件环境指被测软件运行时的操作系统、数据库及其他应用软件构成的环境。

一个充分准备好的测试环境有三个优点：一个稳定、可重复的测试环境，能够保证测试结果的正确；保证达到测试执行的技术需求；保证得到正确的、可重复的以及易理解的测试结果。

性能测试环境，要求和真实环境一致或可对比。

第 3 节Web 性能测试

对于 web 性能测试，做过很多次，可以好好阐述一下，提供具体的计划，方案，报告模板。

9.3.1性能测试的分类

按测试目的来分：

- 评估系统能达到什么样的性能
- 给定了性能指标，检测系统是否能符合要求

- 已知系统存在性能问题，进行测试去定位其位置。
- 检测系统是否存在性能问题。

9.3.2系统结构分析

一般都是集群方式，考虑可能出现瓶颈的位置。

最能反映系统性能情况的是：点击数、处理量、响应时间、硬件的使用情况。下面我们从这三个方面进行分析

处理量一般受以下几个方面的影响：网络带宽，服务器的配置，应用，服务的配置等。从上图来看，影响处理量的因素有以下：对外的网络带宽，服务配置（APCHE，WEBLOGIC），服务器的性能及配置（WEB，应用）。

响应时间一般受以下几个因素影响：网络带宽，服务器的配置，应用，服务的配置等。响应时间会因公众使用的带宽而异，难以用公众端的响应时间来衡量，因此，响应时间可以定义为在没有网络延迟的情况下的响应时间。影响响应时间的主要有几个方面：应用，防火墙等。

服务器资源的使用情况，也能反映出系统的性能，如：CPU 使用率，内存使用率等。使用率的增加最为理想的情况下应该是线性增长的，如果CPU 使用率，内存使用率增长比用户量的增长大得多的情况下，说明应用需要进行调优。

一些隐含的问题：如内存泄漏，日志空间不够等。针对这些问题，需要进行一个长时间的测试。

9.3.3测试指标

- **操作系统（指 Web 应用服务器、数据库服务器)指标：**
- ProcessorTime： 指服务器 CPU 占用率，一般 平均达到 70%时，服务就接近饱和；
- Memory Available Mbyte ： 可用内存数，如果测试时发现内存有变化情况也要注意，如果是内存泄露则比较严重；
- Physicsdisk Time ： 物理磁盘读写时间情况；
- **Web 服务器指标：**

项目	解释	说明	
并发用户数	是指使用自动化测试时，设定的同时上线的用户数 反映系统所能承受的压力	稳定	在长时间系统正常响应的情况下，系统所能支持的用户数
		最大	在网页出错率<10％（HTTP 500、connect 和超时错误），所能支持的虚拟并发用户量
事务	是指一连串请求动作的响应时间的统称。如登录，登录包括，验证用户密码、请求主界面及其上面的图片等。与事务相关的指标有：事务量，成功事务量，失败事务量，平均事务量。 通过事务量及测试的运行时间，可以判断系统的事务处理量是否达到系统的要求，如：一小时完成N 笔业务。		
响应时间 (Response Time)	响应时间从用户的角度来看，系统处于良好的性能状态是指系统能够快速响应用户的请求，即系统响应时间短。 具体地说，响应时间是指发出请求的时刻到用户的请求的相应结果返回用户的时间间隔。		

	响应的时间，测试工具一般会显示最大、最小、平均响应时间。 通过 LR，用户可以得到事务当中用时最长的动作。 定义事务响应时间的标准一般是： < 5s 优 5～15s 可接受 >30 不可接受
每秒点击数	系统 WebServer 每秒所能响应的请求数
每秒处理业务数	Avg time to last byte per terstion （mstes） 系统每秒响应交易的笔数
连接数	当前测试打开的 TCP/IP 连接数。一般在测试用户数据稳定时，会达到一个均衡值，如果出现突然起伏，说明系统性能存在问题。
每秒增加连接数	此参数与连接数密切相关，只有关闭数与新增数达到一致时，连接数才能保持一个均衡水平。如果新增的连接数较大，系统资源消耗会很大，同时，系统连接数存在冗余，重复利用率太低。
系统吞吐量 (Throughput)	吞吐量从系统管理员角度来看，系统的吞吐量成为系统处于良好的性能状态的指标。 具体地说，吞吐量是在给定时间段内系统完成的交易数量。即系统的吞吐量越大，说明系统在单位时间内完成的用户或系统请求越多， 系统的资源得到充分利用。 服务器的吞吐流量，单位为 Byte/s。
其他	服务器的 CPU，内存，核心进程数，WEBLOGIC 及 APCHE 的日志文件增长情况。

◆ **数据库指标**

略，请查看第一节。

9.3.4测试类型

- 负载压力测试
- 疲劳度测试

9.3.5测试需求

根据以前系统的运行纪录(并发量，访问量，点击率等)，或者根据同类产品的性能指标。

◆ **需求采集和整理**

测试需求采集和整理是性能测试最关键的步骤之一，需求不合理，后面的工作都是徒劳的。

测试需求采集，采集合理的信息才能使用等价类的方式去判断哪些功能是必须要测试的，还决定了测试中这些功能脚本对应的各项指标的设置。另外需要注意的是，客户提出了需求后，首先要验证在现有条件下，需求的可模拟性。

比如查询功能，我们是不是要考虑以下的参数：

- 数据库中相关表的数据量；
- 数据库软硬件参数；
- 用户使用频率和使用高峰期各项参数值；
- 用户常用查询条件和查询结果集的大小；

- 性能测试（正常的负载、容量变化）；压力测试（临界的负载、容量变化）；

需求采集完成后，下面的工作就是对需求进行整理和细化，使之成为可模拟的数据。根据特定的业务特定的计算公式对客户的需求进行计算。选择有代表性的功能或页面编写脚本，计算出每次测试各个脚本对应的功能所需要达到的并发数。如果客户说，我们的系统要支持 100 个用户并发，你就在 LR 上加了 100 个用户，这种做法是很不负责任的。

9.3.6测试环境

一般需要在真实环境做测试，或者与真实环境资源配制相同的环境，需要纪录所有相关服务器和测试机的详细信息。

- 服务器<web 服务器 N 台，应用服务器 N 台，数据库服务器 N 台，磁盘阵列服务器 1 台>软硬件配置信息；
- 测试机>使用 LR 做性能测试 ，一般测试机包括 ： lr_server，lr_agent(多台)，lr_监控机，lr_数据记录，lr_备份数据>软硬件配置信息；

9.3.7LR 的测试方案

一般网站是按照点击率来测试，而工作流是按照事务数来测试。

评估并发量的方式：

- 按照点击率测试：逐渐加大并发量，找出系统点击率最高时的并发量；
- 按照并发量来测试： 逐渐加大并发量，直至达到使系统的处理量及响应时间在达到用户不可接受的程度，并且记录下这个并发量，作为系统的临界点；

测试方案中，测试场景的设计是很重要的。测试场景一般包括：各个功能脚本的并发量(或百分比)，启动方式，结束方式，运行时间，监控指标，是否取缓存，脚本之间的逻辑，脚本内部的逻辑，成功和失败标准等等。

9.3.8执行测试

- 执行测试前，需要要检查：
  - 各项系统资源是否足够；
  - 是否能保证测试环境不间断运行；
- 执行测试过程，需要实时监控系统运行状况，纪录系统出现的异常，纪录测试过程中的各项数据。
- 执行完成后，对执行过程中记录的数据作分析。

9.3.9测试结果分析

LR 提供了强大的监控和结果分析工具，也可以根据需要监控指标的不同使用其他的监控工具。可以查看相关工具的结果分析文档，这里不再阐述。

第 4 节安装卸载性能

主要检查不同硬件软件等资源条件下，各个部件的安装速度。

第 5 节数据库性能测试

9.5.1测试目标和基本方法

◆ **测试目标：**

✧ 验证系统瓶颈是否在数据库；
✧ 同其他大型数据库比较；
✧ 数据库的增删改查的平均响应时间；
✧ 并发用户访问时，数据库的增删改查的平均响应时间；
✧ 表分区对数据库响应时间的影响；
✧ 系统正常使用时，数据库各个监控参数变化情况；
✧ 数据库服务器的系统资源使用情况；
✧ 其他

◆ **基本方法：**

使用并发工具模拟并发用户访问各种不同类型数据库（或指定数据库），记录各种并发量下增删改查的数据库响应时间，记录IO、CUP（IDLE）等系统数据，记录数据库各个计数器的值，执行完成后，进行纵向和横向的比较。

9.5.2Oracle 性能测试计数器

◆ **常用计数器：**

编号	监控名称	描述
1	Logons current	当前的登录总数
2	Opened cursors current	当前打开的光标总数
3	User calls	在每次登录、解析或执行时，Oracle 会分配资源（Call State 对象）以记录相关的用户调用数据结构。在确定活动时，用户调用与RPI调用的比说明了因用户发往Oracle的请求类型而生成的内部工作量。
4	Opers of replaced files	由于已经不在进程文件缓存中，所以需要重新打开的文件总数

◆ **常用自定义计数器：**

编号	监控名称	SQL 基本算法	描述
1	数据高速缓存区命中率	SELECT round(1-SUM(PHYSICAL_READS)/(SUM(DB_BLOCK_GETS) + SUM(CONSISTENT_GETS)), 4) * 100 FROM (SELECT CASE WHEN NAME='physical reads' THEN VALUE END PHYSICAL_READS,CASE WH	（监控 SGA 的命中率）命中率应大于 0.90 最好

		EN NAME = 'db block gets' THEN VALUE END DB_BLOCK_GETS,CASE WHEN NAME = 'consistent gets' THEN VALUE END CONSISTENT_GETS FROM V$SYSSTAT WHERE Name IN ('physical reads','db block gets','consistent gets'))	
2	库快存命中率	SELECT 100*((sum(pins-reloads))/sum(pins)) from v$librarycache	该计数器返回当前库快存命中率
3	共享区库缓存区命中率	Select round(sum(pins-reloads)/sum(pins) * 100, 2) from v$librarycache	（监控 SGA 中共享缓存区的命中率）命中率应大于 0.99
4	监控 SGA 中字典缓冲区的命中率	Select round(sum(gets-getmisses-usage-fixed)/sum(gets) * 100, 2) from v$rowcache	（共享区字典缓存区命中率）命中率应大于 0.85
5	检测回滚段的争用	select round(sum(waits)/sum(gets) * 100, 2) from v$rollstat	小于 1%
6	检测回滚段收缩次数	select sum(shrinks) from v$rollstat, v$rollname where v$rollstat.usn = v$rollname.usn	
7	监控表空间的 I/O 读总数	select sum(f.phyrds) pyr from v$filestat f, dba_data_files df where f.file# = df.file_id	监控表空间的 I/O
8	监控表空间的 I/O 块读总数	select sum(f.phyblkrd) pbr from v$filestat f, dba_data_files df where f.file# = df.file_id	监控表空间的 I/O
9	监控表空间的 I/O 写总数	select sum(f.phywrts) pyw from v$filestat f, dba_data_files df where f.file# = df.file_id	监控表空间的 I/O
10	监控表空间的 I/O 块写总数	select sum(f.phyblkwrt) pbw from v$filestat f, dba_data_files df where f.file# = df.file_id	监控表空间的 I/O
11	监控 SGA 中重做日志缓存区的命中率	SELECT Decode(immediate_gets+immediate_misses,0,0, immediate_misses/(immediate_gets+immediate_misses)*100) ratio2 FROM v$latch WHERE name IN ('redo copy')	应该小于 1%
12	监控内存和硬盘的排序比率	select round(sum(case when name='sorts (disk)' then value else 0 end) / sum(case when name='sorts (memory)' then value else 0 end)*100,2) from (SELECT name, value FROM v$sysstat WHERE name IN ('sorts (memory)', 'sorts (disk)'))	最好使它小于 10%

第 6 节网络性能测试

在网络上性能的测试重点是利用成熟先进的自动化技术进行网络应用性能监控、网络应用能分析和网络预测。

9.6.1网络应用性能分析

网络应用性能分析的目的是准确展示网络带宽、延迟、负载和 TCP 端口的变化是如何影响用户的响应时间的。利用网络应用性能分析工具，例如 Application Expert，能够发现应用的瓶颈，我们可知应用在网络上运行时在每个阶段发生的应用行为，在应用线程级分析应用的问题。可以解决多种问题：客户端是否对数据库服务器运行了不必要的请求？当服务器从客户端接受了一个查询，应用服务器是否花费了不可接受的时间联系数据库服务器？在投产前预测应用的响应时间；利用 Application Expert 调整应用在广域网上的性能；Application Expert 能够让你快速、容易地仿真应用性能，根据最终用户在不同网络配置环境下的响应时间，用户可以根据自己的条件决定应用投产的网络环境。

9.6.2网络应用性能监控

在系统试运行之后，需要及时准确地了解网络上正在发生什么事情；什么应用在运行，如何运行；多少 PC 正在访问 LAN 或 WAN；哪些应用程序导致系统瓶颈或资源竞争，这时网络应用性能监控以及网络资源管理对系统的正常稳定运行是非常关键的。利用网络应用性能监控工具，可以达到事半功倍的效果，在这方面我们可以提供的工具是Network Vantage。通俗地讲，它主要用来分析关键应用程序的性能，定位问题的根源是在客户端、服务器、应用程序还是网络。在大多数情况下用户较关心的问题还有哪些应用程序占用大量带宽，哪些用户产生了最大的网络流量，这个工具同样能满足要求。

9.6.3网络预测

考虑到系统未来发展的扩展性，预测网络流量的变化、网络结构的变化对用户系统的影响非常重要。根据规划数据进行预测并及时提供网络性能预测数据。我们利用网络预测分析容量规划工具 PREDICTOR 可以作到：设置服务水平、完成日网络容量规划、离线测试网络、网络失效和容量极限分析、完成日常故障诊断、预测网络设备迁移和网络设备升级对整个网络的影响。从网络管理软件获取网络拓扑结构、从现有的流量监控软件获取流量信息（若没有这类软件可人工生成流量数据），这样可以得到现有网络的基本结构。在基本结构的基础上，可根据网络结构的变化、网络流量的变化生成报告和图表，说明这些变化是如何影响网络性能的。 PREDICTOR 提供如下信息：根据预测的结果帮助用户及时升级网络，避免因关键设备超过利用阀值导致系统性能下降；哪个网络设备需要升级，这样可减少网络延迟、避免网络瓶颈；根据预测的结果避免不必要的网络升级。

第 10 章 测试工具

熟悉一个工具，先要熟悉它的工作原理。

对 MI 的工具/Rational 工具/Compuware 工具系列自动化和性能测试工具，网络上流行的很多测试工具的 PDF 文档就写的都不错，这里只列出标题，不做讲解。如有需要可以联系我。

第 1 节简介

测试工具只要包括：

- 用于所有测试交付物和工作产品的中心项目数据库——测试管理系统，包括用于对测试进行保存、描述、文档化和跟踪，并且对测试目标和结果进行记录、跟踪、评审的辅助设施。
- 用于支持不同测试环境的测试床和模拟器；
- 提供变更前后分析和工作产品风险及复杂度评价的静态分析器和比较器；
- 用于测试执行和回归的测试驱动及捕获 / 回放工具；
- 度量和报告测试结果及覆盖率的动态分析工具；
- 以及其他自动化测试 /性能测试 /系统性能监控 /版本管理/缺陷管理/配置管理工具等等。

第 2 节MI 系列工具

10.2.1Loadrunner

10.2.2QTP

10.2.3Winrunner

10.2.4Testdirector

10.2.5Quality Center

第 3 节Rational 系列工具

10.3.1Clearcase

10.3.2Clearquest

10.3.3Testmanger

10.3.4Request Pro

10.3.5Rose

10.3.6Rebot

第 4 节Compuware 系列工具

10.4.1QArun

10.4.2QAload

10.4.3TrackRecord

第 5 节Java 监控工具

10.5.1Jrockit

10.5.2Jprofiles

10.5.3Borland Optimizeit Suite

第 6 节Windows 监控工具

10.6.1Winobj v2.13

对象命令空间管理器。增强了用户界面，显示更多对象类型，并集成了NT 的本地安全设置。

10.6.2CPUMon v2.0

CPU 性能监视工具。可以获取 CPU 计数器信息。该版本集成了 Perfmon。

10.6.3Process Explorer v8.52

查看进程所打开的文件，注册表和其他对象，并显示加载了那些dll。

10.6.4TCPView v2.34

监视本机 TCP 和 UDP 协议的活动情况，并显示使用该协议的进程，包括了 dos 版本。

第 7 节其他测试工具

10.7.1Webload

10.7.2Logiscope

10.7.3e-test

10.7.4JProbe_suite

10.7.5Junit

第 8 节其他相关工具

10.8.1Weblogic/ tomcat

10.8.2CVS&VSS

10.8.3Virtual Machine

10.8.4远程控制工具

10.8.5Romote Administrator &Mstsc

10.8.6QQ& KDT

10.8.7Unix -Secure-CRT

第 11 章 团队测试

随着软件开发规模的增大、复杂程度的增加，以寻找软件中的错误为目的的测试工作就显更加困难。然而，为了尽可能多地找出程序中的错误，生产出高质量的软件产品，加强对测试工作的组织和管理就显得尤为重要。团队管理的好坏决定了测试工作的质量和参与测试人员的工作量。

不是几个人负责一个项目的测试工作，就可以称作“团队测试”。

11.1.1测试经理角色定位

测试经理服务于两种完全不同的客户：测试工程师和高层管理者。对于测试工程师，测试经理帮助他们开发产品测试策略，积累产品测试经验并在测试组内充分共享。对于高层管理者，测试经理搜集尽可能全面的产品信息，供其就产品是否可以发布进行决策。但是有一点是相同的：无论是对于测试工程师还是高层管理者，测试经理将帮助其定义和校验产品发布标准。

产品发布标准的定义和校验：作为一个测试经理，应该找机会与市场、开发人员商讨产品发布标准，并根据客户的反馈对该标准进行修正和校验。开发部门的工作是如何达到公司对产品的期望，要用客户需求为开发人员勾画出客户眼中的产品以及产品应如何工作。一旦产品被清楚地定义，就可以通过测试去验证产品在多大程度上满足了客户需求。

对于测试工程师而言有一点非常重要：将测试任务按优先级划分，使产品发布标准得以满足。由于只有极少数的项目有充足的时间去完成所有事情，所以告诉测试工程师关于“测什么和何时测 ”测试经理的一个重要职责。

高层管理者需要充分理解产品发布标准，以决定产品是否可以按时发布。我不认为测试组有权利裁决产品是否应该被发布，该权利在组织高层管理者那里。在有了一个通过讨论、达成一致的产品发布标准后，项目组也可以更清楚地了解和认识产品质量。

11.1.2团队测试的意义

- 有效的测试最好由一个独立的团队来实施。
- 便于确定工作目标
- 便于人员的培养与升迁
- 利于团队建设
- 对质量的忠诚度高
- 利于新技术，新方法的产生和推广
- 工作职责明确

11.1.3团队建设

软件企业我认为应该再以下几个方面进行控制：

- 组织结构：可以保证管理的正确执行；
- 岗位职能：可以保证权责明确；

- ✧ 方针目标：是工作的根本，没目标、没方针那就什么也没有了；
- ✧ 工作流程：保证工作的按秩序、按步骤的正确执行；
- ✧ 流程控制：对流程进行控制分析，保证项目在控制之内；
- ✧ 危机处理：什么行业都有潜在的危险，软件行业更要提前做好准备；
- ✧ 纪效考核：主要是面向员工，按劳付酬，可以激励员工；
- ✧ 能力培养：IT 行业在向前发展，人也必须向前发展；
- ✧ 团队建设：好的团队更能体现出效率；

第 12 章 测试相关知识

第 1 节数据库

- ✧ Sql 语句/存储过程/视图/等概念
- ✧ MS Sql Server
- ✧ Sybase
- ✧ Oracle
- ✧ MySQL
- ✧ DB2
- ✧ Informix

除了对数据库的原理了解之外，掌握简单的数据库调优知识是很重要的。

第 2 节网络知识

计算机的发展依赖于网路，需要掌握一些基础的网络知识，在编程，性能测试等情况下都是由帮助的，比如要了解一些网络通信端口，网络流量 ，带宽等知识。

12.2.1一些名词

- ➢ LAN、MAN、WAN、Internetwork、网关（gateway）；
- ➢ 子网：指把分组从源主机传送到目的主机的路由器和通信线路的集合。它包含了与网络寻址相关的含义。
- ➢ 子网就是当某一段默认网段的主机数量不够用时，通过修改默认子网掩码的方法划分出不同的子网段，以增加通信的主机数量。
- ➢ 拓扑结构：网络的结构，个个设备路由之间的连接方式。
- ➢ 网关：一台机器，用来连接相互不兼容的网络，并提供硬件和软件的转换。
- ➢ 互联网：通过 WAN 把 LAN 连接起来的集合。
- ➢ 服务：在形式上是由一组据原语（操作）来描述的，包括：请求、提示、响应、证实。
- ➢ 是各层向它的上层提供的一组原语，尽管能够代表上一层完成的操作，但不涉及上层是如何完成的。
- ➢ 协议：定义了同层之间对等实体之间交换的桢、分组和报文的格式及意义的一组规则。实体使用协议来实现服务。

12.2.2网络协议

七层协议：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。其中，前 3 层是链接起来的，4-7 层是端对端的。

- ✧ 物理层（physical layer）

通信在信道上传输的比特流（多少伏特代表 0 或 1）

- ✧ 数据链路层（data link layer）
  为网络层提供无错的线路；防止重复桢；调节流量；借道（回复桢和数据桢）。
- ✧ 网络层（network layer）
  确定如何选择路由（动态和静态）；防止分组导致的阻塞；网络记费。
- ✧ 传输层（transport layer）
  从会话层接收数据，分割为教小单元在适当时机传递给网络层。
- ✧ 会话层（session layer）
  允许不同机器上的用户建立会话关系，允许普通数据的传输；可以控制单向传输或双向传输；令牌管理（不能同时做同样操作）；同步（断点续传）。
- ✧ 表示层（presentaltion layer）
  完成一些通用功能[比如各种数据代码（ASCII、Unicode）之间的转换]，表示层关心所传输的信息的语法与意义，而以下的层关心的是可靠的传输比特流。
- ✧ 应用层（application layer）
  包含大量普遍需要的协议，比如各种终端的型号（终端的型号不同问题用网络虚拟终端的方式来解决；文件传输（包括不同文件系统之间的传输控制）；电子邮件；远程作业输入、名录查询等。

12.2.3TCP/IP 协议

模型结构：应用层、传输层、互连层（网络层）、主机至网络。

应用层（application layer）包含了所有的高层协议：虚拟终端协议 telnet、文件传输协议 ftp、电子邮件协议 smtp、域名系统协议 DNS（domain name service）把主机名映射为网络地址、NNTP 协议用于传递新闻、HTTP 协议用语在 WWW 上获取主页。

TCP 和 UDP 的区别：是否控制流量。

第 3 节操作系统监控参数

- ✧ Windows 自身的监控计数器很强大，需要认真去了解它。可以查看 windows 的帮助文档获取到所有需要的知识。
- ✧ UNIX 自身也有一些监控，可以使用 top 等命令进行简单的查询。也可以自己写一些 shell 脚本对系统监控到的这些数据做一些统计和分析。

第 13 章 附录

第 1 节附录一：自动化工具清单

生产厂商	工具名称	测试功能简介	网址链接
Mercury Interactive Corporation	winrunner	功能测试	http://www.mercury.com/us/products/
	Loadrunner	性能测试	
	QuickTest Pro	功能测试	
	Astra LoadTest	性能测试	
	testdirector	测试管理	
IBM Rational	Rational robot	功能测试和性能测试	http://www-900.ibm.com/cn/software/rational/products/index.shtml
	Rational xde tester	功能测试	
	Rational testmanager	测试管理	
	Rational purifyplus	白盒测试	
compuware corporation	QARun	功能测试	http://www.compuware.com/products/
	QALoad	性能测试	
	QADirecto	测试管理	
	DevPartner Studio Professional Edition	白盒测试	
Segue software	SilkTest	功能测试	http://www.segue.com/products/index.asp
	SilkPerformer	性能测试	
	SilkCentral Test/Issue Manager	测试管理	
Empirix	e-Tester	功能测试	http://www.empirix.com/Empirix/Web+Test+Monitoring/Testing+Solutions Integrated+Web+Testing.html
	e-Load	性能测试	
	e-Monitor	测试管理	
parasoft	Jtest	Java 白盒测试	http://www.parasoft.com/jsp/products.jsp?itemId=12
	C++test	C/C++ 白盒测试	
	。test	.NET 白盒测试	

RadView	WebLOAD	性能测试	http://www.radview.com/products/index.asp
	WebFT	功能测试	
MicroSoft	Web Application Stress Tool	性能测试	http://www.microsoft.com/technet/archive/itsolutions/intranet/downloads/webtutor.mspx
Quest Software	Benchmark Factory	性能测试	http://www.quest.com/benchmark_factory/
Minq Software	PureTes	功能测试	http://www.minq.se/products/

第 2 节附录二：Windows 监控工具

- **CPUMon v2.0**
  CPU 性能监视工具。可以获取 CPU 计数器信息。该版本集成了 Perfmon。
- **DebugView v4.31**
  截取 Win32 设备驱动程序发出的消息，允许通过本机或网络查看和录制调试信息而不打开一个活动的调试器。
- **Diskmon v2.01**
  显示硬盘的活动信息。
- **Filemon v6.12**
  实时监视操作系统中活动的文件。
- **Handle v2.20**
  显示进程及其打开的文件等信息。
- **ListDLLs v2.23**
  列出当前系统加载的所有 dll 文件、调用它的执行程序及 dll 版本路径等详细信息。
- **NTFSInfo v1.0**
  查看 NTFS 卷的详细信息，包括大小、文件分配表的大小起止位置，还有元数据文件的大小等。
- **PMon v1.0**
  监视进程的创建、删除，也包括显示多 CPU 机器或 checked kernel 机器上的上下文交换信息。
- **Portmon v3.02**
  端口监视工具，监视端口收发的信息等。
- **Process Explorer v8.52**
  查看进程所打开的文件，注册表和其他对象，并显示加载了那些dll。
- **PsTools v2.1**
  包含一套命令行工具，包括显示本机或远程机器上运行的进程，在远程机器上运行进程，重启机器，记录日志等。
- **Regmon v6.12**
  实时监视注册表的活动。
- **TCPView v2.34**
  监视本机 TCP 和 UDP 协议的活动情况，并显示使用该协议的进程，包括了 dos 版本。
- **TDImon v1.01**

通过网络 API 实时监视 TCP 和 UDP 协议的活动情况。

- **Tokenmon v1.01**

  令牌监视器，监视与信息安全相关的活动，比如登录，退出等。

- **Winobj v2.13**

  对象命令空间管理器。增强了用户界面，显示更多对象类型，并集成了NT 的本地安全设置。

第 3 节附录三：测试基础知识

13.3.1测试与质量的关系

• 质量的概念：过程质量、交付件质量
• 质量管理概念：质量管理发展的三个阶段
• 全面质量管理：质量管理活动
• 质量管理基本思想

13.3.2测试原则与方法

• 为什么要尽早测试？ • 缺陷成本的阶段性增长
• 缺陷引入的阶段性分布
• 木桶原理 • 20/80 原则 • Good-Enough 原则
• 错误、缺陷、故障、失效 • 测试成本分析
• 测试、调试、测试与调试的关系和区别
• 验证与确认 • 广义的测试和狭义的测试
• 测试重心 • 测试投入 • 测试结束准则
• 测试方法分类 • 白盒测试、黑盒测试、灰盒测试
• 动态测试、静态测试 • 手工测试、自动测试
• 测试策略 • 单元测试、集成测试、系统测试
• ALPHA 测试、BETA 测试、验收测试
• 回归测试 • 测试、缺陷、质量的关系
• 测试的任务、必要性、局限性
• 测试生命周期模型、测试和开发的并发性

13.3.3产品测试流程

13.3.3.1整体介绍

• 市场驱动的产品研发 • 结构化的产品开发流程
• 跨部门的产品开发团队 • 测试代表的职责定义
• 测试代表与相关职能领域代表的关系描述

• 产品开发阶段划分 • 技术评审点
• 决策评审点 • 三级测试计划体系
• 测试计划的分层控制 • 产品测试业务框架
• 产品版本化管理 • 测试学习曲线
• 产品测试管理框架 • 产品测试管理过程
• 各个产品开发阶段的关键开发与测试活动

13.3.3.2概念阶段测试活动介绍

• 需求的可测试性 • 可测试性需求
• 生产的可测试性 • 客户服务的可诊断性
• 产品测试策略的确定 • 初步测试端到端计划

13.3.3.3计划阶段测试活动介绍

• 测试团队的扩充方法 • 优化端到端测试计划
• 制定验证测试计划 • 集成测试方案确定
• 测试工具开发策略 • 可测试性设计的监督实施

13.3.3.4开发阶段测试活动介绍

• 测试工具选型 • 集成测试工作的开展
• 测试工具的实现 • SDV 测试 • SIT 测试
• 生产测试设备的开发 • BETA 测试用户确定
• 渐增的产品开发与测试方法、框架
• 基于产品组件的测试 • 产品组件集成与测试

13.3.3.5验证阶段测试工作介绍

• SVT 测试 • BETA 测试 • 标竿测试
• 外协测试 • ESP 客户支持 • 测试总结
• 基于样机的系统测试 • 基于初始产品测试
• 基于客户交付的产品测试 • 专业实验测试
• 运营测试 • 准入测试 • 专项测试

13.3.3.6发布阶段测试工作介绍

• 客户问题跟踪 • 系统升级
• 收集分析新需求 • 招标支撑
• 重点客户支撑 • 培训客户 • 在线诊断

• 软件项目级的测试流程
• 硬件项目级的测试流程

13.3.4测试组织结构

• 公司级别测试组织
• 跨部门产品开发测试团队
• 产品测试组涉及的角色和组织结构
• 测试代表的职责 • 测试外围组的职责
• 测试部在产品测试中的职责
• 测试人员核心素质 • 测试人员的职业发展
• 测试人员技术等级介绍
• 测试技术等级管理存在问题与避免办法
• 国内测试组织存在的问题及解决办法
• 组织定位与职责 • 测试组织的五步改进过程
• IBM 测试人员素质要求介绍
• 微软的测试发展历程
• 测试经理的职业素质要求 • 测试经理的培养
• 测试人员为什么缺少成就感?
• 如何提高测试人员的成就感?

13.3.5测试度量过程

• 度量的概念 • 测试度量的必要性
• 测试度量的过程
• 识别目标、定义过程、数据收集、
• 数据分析、过程改进
• 组织过程能力基线 PCB
• 过程能力基线 • 建立方法和过程、应用
• 测试度量角色介绍
• 测试度量的目的 • 理解、预测、评估、改进
• 常用的测试测量项 • 测试度量的常见问题
• 产品质量度量 • 缺陷密度 • 缺陷总数
• 客户问题的度量 • 客户满意度度量
• 产品测试缺陷到达模式 • 发现缺陷趋势
• 确认缺陷趋势 • 累计缺陷趋势
• 缺陷修复周期
• 基于阶段的缺陷去除模式 • 缺陷去除率
• 产品缺陷分析方法与模型 • Rayleigh 方法
• Gompertz 方法

13.3.6测试工具与技术

• 自动化设计的三个阶段 • 自动化策略
• 自动化测试成熟度
• TestFrame 测试思维介绍
• 测试系统与被测系统关系的发展趋势
• 基于镜像的测试技术 • 实体镜像 • 过程镜像
• 脚本在自动化测试中的应用 • 语法分析技术
• 插装技术 • 测试工具的开发策略
• 测试技术开发
• 代码质量度量核心技术介绍

13.3.7测试工程过程

13.3.7.1测试过程

• V&V 测试模型
• 测试计划 • 测试设计 • 测试开发
• 测试执行 • 测试评估 • 测试报告
• 缺陷跟踪

13.3.7.2单元测试过程

• 单元测试基本概念 • 单元测试的意义
• 单元测试的定义
• 单元测试过程 • 单元测试阶段输出
• 单元测试计划及评审 • 单元测试准备
• 单元测试执行 • 单元测试报告
• 单元测试成败关键因素分析
• 演示讲解：测试计划、测试规程、测试用例、测试方案、测试报告、测试指导、需求跟踪表、测试计划和用例评审查检表、测试记录、缺陷报告

13.3.7.3集成测试过程

• 集成测试基本概念 • 集成测试对象
• 集成测试的特点
• 集成测试过程 • 集成测试计划及评审
• 集成测试准备
• 集成测试执行 • 缺陷跟踪
• 集成测试过程报告

• 集成测试质量目标 • 集成测试报告
• 演示讲解：集成测试方案模板
• 集成测试成败关键因素分析

13.3.7.4系统测试过程

• 系统测试基本概念（定义、对象、依据）
• 系统测试过程 • 系统测试输入、输出
• 系统测试计划
• 系统测试准备 • 系统测试执行
• 系统测试报告 • 转测试操作
• 问题跟踪反馈 • 回归测试
• 系统测试成败关键因素分析 •”黑白唱”

13.3.7.5验收测试过程

• 验收测试基本概念 • 验收测试对象
• 验收测试的特点
• 验收测试过程 • 验收测试计划及评审
• 验收测试准备
• 验收测试执行 • 缺陷跟踪
• 验收测试过程报告
• 验收测试质量目标 • 验收测试报告
• 验收测试成败关键因素分析