《软件测试》学习重点归纳

目录

第 1 章 软件测试背景

1.1 软件缺陷的官方定义

（1） 软件未实现产品说明书要求的功能。

（2） 软件出现了产品说明书指明不应该出现的错误。

（3） 软件实现了产品说明书未提到的功能。

（4） 软件未实现产品说明书虽未明确提及但应该实现的目标。

（5） 软件难以理解、不易使用、运行缓慢或者——从测试员的角度看——最终用户会认为不好。（要全面，最重要的是客观评价，并非所有测试发现的缺陷都要修改）

1.2 为什么会出现软件缺陷

（1） 导致软件缺陷最大的原因是产品说明书。在许多情况下，说明书没有写。也有可能是说明书不够全面、经常更改，或者整个开发小组没有很好地沟通。

（2） 软件缺陷第二大来源是设计。

（3） 某些缺陷产生的原因是把误解当成缺陷。

（4） 还有可能缺陷多处反复出现，实际上是由一个原因造成。

（5） 一些缺陷可以归咎于测试错误。

1.3 软件缺陷的修复费用

费用指数级地增长

修复软件缺陷的费用都是随着时间推移而增加的

1.4 软件测试员究竟做什么

软件测试员的目标是尽可能早地找出软件缺陷，并确保其得以修复。

软件测试人员虽然在追求完美，确保缺陷都被修复，但软件测试的实质则是另外一回事。千万不要在无法达到的完美上纠缠和兜圈子。

1.5 优秀软件测试员应具有的素质

（1） 他们是群探索者。软件测试员不会害怕进入陌生环境。 有较强的学习能力。

（2） 他们是故障排除员。软件测试员善于发现问题的症结，喜欢解谜。

（3） 他们不放过任何蛛丝马迹。软件测试员总是不停尝试。他们可能会碰到转瞬即逝或者难以重建的软件缺陷；他们不会心存侥幸，而是尽一切可能去寻找。

（4） 他们具有创造性。测试显而易见的事实，那不是软件测试员；他们的工作是想出富有创意甚至超常的手段来寻找软件缺陷。

（5） 他们是群追求完美者。他们力求完美，但是知道某些无法企及时，不去苛求，而是尽力接近目标。

（6） 他们判断准确：软件测试员要决定测试内容、测试时间，以及看到的问题是否算作真正的缺陷。

（7） 他们注重策略和外交。软件测试员往往带来的是坏消息。 他们必须告诉程序员，你的（程序）孩子很丑。优秀的软件测试员知道怎样策略和职业地处理这些问题，也知道如何和不够冷静的程序员合作。

（8） 他们善于说服。软件测试员要善于表达观点，表明软件缺陷为何必须修复，并通过实际演示力陈观点。

（9） 在编程方面受过教育。

第 2 章 软件开发的过程

第 3 章 软件测试的实质

3.1 软件测试的原则

（1） 完全地测试程序是不可能的

- 输出量太大；
- 输出结果太多；
- 软件执行路径太多；
- 软件说明书是主观的。

（2） 软件测试是有风险的行为

因为完全地测试程序是不可能的，如果决定了不去测试所有的情况，那就是选择了冒险。

软件测试员要学会的一个关键思想是，如何把数量巨大的可能测试减少到可以控制的范围，以及如何针对风险做出明智的选择。

（3） 测试无法显示潜伏的软件缺陷

软件测试人员可以报告软件缺陷存在，却不能报告软件缺陷不存在。

（4） 找到的软件缺陷越多，就说明软件缺陷越多。

生活中的害虫和软件缺陷几乎完全一样。两者都成群出现，发现一个，附近就有可能会有一群。

（5） 杀虫剂怪事

杀虫剂怪事：软件测试越多，其对测试的免疫力就越强。

（6） 并非所有的缺陷都要修复

- 没有足够的时间
- 不算真正的软件缺陷
- 修复风险太大。软件本身是脆弱的、难以理清头绪，有点像一团乱麻，修复一个缺陷可能导致更多的缺陷出现，在紧迫的产品发布进度压力下，修改软件将冒很大的风险。
- 不值得修复。不常出现的软件缺陷和在不常用功能出现的缺陷是可以放过的，可以躲过和用户有办法预防或避免的软件缺陷通常不用修复。这都要归结为上而言风险决策。

决策过程通常由软件测试员、项目经理和程序员共同参与。

（7） 什么时候才叫缺陷难以说清

尚未发现或未观察到的软件缺陷只能说是潜在缺陷

（8） 产品说明书从没有最终版本

软件测试员必须要想到产品说明书可能改变。未曾计划测试的功能会增加，经过测试并报告软件缺陷的功能可能发生改变甚至被删除。这些都是有可能会发生的

（9） 软件测试员在产品小组中不受欢迎

软件测试员的目标是尽可能早地找出软件缺陷，并确保其得以修复。软件测试员的工作时检查和批评同事的工作、挑毛病、公布发现的问题。

保持小组成员和睦的建议：

- 早点找出缺陷。在三个月之前而不是在产品即将发布前夕找出严重的软件缺陷，会产生更小的影响，更容易让人接受。
- 控制情绪。发现严重缺陷的时候非常兴奋，但是不要兴冲冲地区找开发人员告诉他

他的代码有可怕的缺陷，他会不高兴的。

- 不要总是报告坏消息。假如发现某段代码没有软件缺陷，就大声宣扬。花一点时间找程序员聊聊天。如果总是报告坏消息，别人会对你避之唯恐不及。

（10） 软件测试是一项讲究条理的技术专业

3.2 软件测试的技术和术语

（1） 精确（precision）和准确（accuracy）

软件测试要精度还是准度很大程度上取决于产品是什么，最终取决于开发小组的目标。

（2） 确认（verification）和验证（validation）

- 确认是保证软件符合产品说明书；
- 验证是保证软件满足用户要求的过程。

（3） 质量和可靠性

如果说软件的质量高，就是指它能够满足客户要求。客户会感到产品性能卓越，优于其他产品。

而可靠性仅仅是质量的一方面。

为了确保程序质量高且可靠性强，软件测试员必须在整个产品开发过程中进行确认和验证。

（4） 测试盒质量保证（QA）

- 软件测试员的目标是尽可能早地找出软件缺陷，并确保缺陷得以修复。
- 软件质量保证人员的主要职责是创建和执行改进软件开发过程并防止软件缺陷发生的标准和方法。

当然他们存在交叉之处。重要的是了解自己的职责，并与开发小组七天成员交流。

第 4 章 检查产品说明书

假如有幸在项目早期介入，并有权修改初期的产品说明书，在此阶段找出软件缺陷极有可能为项目节省大笔开销和时间。

4.1 开始测试

除了大爆炸模式之外，每个模式中开发小组都要根据需求文档编写一份产品说明书，用以定义软件是什么样的。

产品说明书通常是利用文章和图形描述产品的书面文档。确保最终产品符合客户要求以及正确计划测试投入的唯一方法是在产品说明书中完整描述产品。

编写详细产品说明书的另一个好处是软件测试员可以将其作为测试项目的书面材料，据此可以子啊编码钱找出软件缺陷。

（1） 黑盒测试和白盒测试

在黑盒测试中，软件测试员只需知道软件要做什么——而无法看到盒子里的软件是如何运行的。只要进行一些输入，就能得到某种输出结果。他不知道软件如何运行、为什么会这样，只知道程序作了什么。

在白盒测试中，软件测试员可以访问程序员的代码，并通过检查代码的线索来协助测试——可以看到盒子里面。测试员根据代码检查结果判断多或少可能出错的数目，并据此定制测试。

（2） 静态测试盒动态测试

静态测试是指测试不运行的部分——只是检查和审核；

动态测试是指通常意义上的测试——使用和运行软件。

（3） 静态黑盒测试、测试产品说明书

测试产品说明书属于静态黑盒测试。

即使项目采用的是大爆炸模式或者松散的编写边改模式开发，没有产品说明书，总会有人知道产品是什么样的。这个人可能是开发人员、项目经理或销售人员。记下收集到的信息并反复斟酌就可以得到更详细的资料。

4.2 对产品说明书进行高级审查

测试产品说明书的第一步不是马上钻进去去找缺陷，而是站在一个高度上进行审查。审查产品说明书是为了找出根本性问题、疏忽或遗漏之处。研究产品说明书的根本是为了更好地了解软件该做什么。

（1） 假设自己是客户

当软件测试员第一次接到需要审查的产品说明书时，最容易的事实把自己当做客户。质量的定义是“满足客户的要求”，软件测试员必须了解并测试软件是否满足客户的要求。熟悉软件应用领域的相关知识会有很大帮助。最终还是得利用这个产品说明书来设计测试。

（2） 研究现有的标准和规范

可以考虑作为标准和规范的例子：

- 客户公司的习惯用语和约定
- 行业要求。医药、工业、金融行业的应用软件有其必须严格遵守的标准。
- 政府标准。政府——尤其是军事系统有严格的标准。
- 图形用户界面（GUI）。如果软件运行于 Microsoft Windows 或 Apple Macintosh 操作系统上，关于软件外观和用户感受具有公开的标准。
- 安全标准。

软件测试员的任务是管擦，“检查”采用的标准是否正确、有无遗漏。在对软件进行确认和验收时，还要注意是否与标准和规范相抵触，把标准和规范视为产品说明书的一部分。

（3） 审查和测试类似软件

了解软件最终结果的最佳方法是研究类似软件，例如竞争对手的产品或者小组开发的类似的产品。记住要阅读关于竞争对手软件的评价方面的联机或印刷的文章。

4.3 产品说明书的低层次测试技术

（1） 产品说明书属性检查清单

- 完整。是否有遗漏和丢失？完整吗？单独使用时是否包含所有内容？
- 准确。既定解决方案正确吗？目标定义明确吗？有没有错误？
- 精确、不含糊、清晰。描述是否一清二楚？是否有单独的解释？容易看懂和理解吗？
- 一致。产品功能描述是否有自相矛盾，或一起跳功能有无冲突？
- 贴切。描述功能的陈述是否必要？有没有多余信息？功能是否符合原来的客户要求？
- 合理。在规定预算和进度下，以现有人力、工具和资源能否实现？
- 代码无关。产品说明书是否坚持定义产品，而不是定义其软件设计、架构和代码？
- 可测试性。功能能否测试？给测试员提供的简历严重操作的信息是否足够？

第 5 章 带上眼罩测试软件

5.1 动态黑盒测试：带上眼罩测试软件

有效的动态测试需要关于软件行为的一些定义——即需求分析或产品说明书。不必了解软件“盒子”内发生的事情——而只需要知道输入 A 或输出 B 或执行操作 C 得到结果 D。好的产品说明书会提供这些信息。

接下来开始定义测试用例。

测试用例是指进行测试时使用的特定输入，以及软件测试的过程步骤。

在没有产品说明书时使用**探索测试——**了解软件、设计测试、执行测试同时进行，系统

地逐项了解软件的功能、记录软件的执行情况、详细描述功能。

5.2 通过性测试和实效性测试

通过性测试是指，实际上是确认软件至少能做什么，而不会考验其能力。软件测试员并不需要想尽办法让软件崩溃，仅仅运用最简单、最直观的测试用例。

实效性测试是指，纯粹是为了破坏软件而设计的测试用例。它是蓄意攻击软件的薄弱环节。

5.3 等价类划分

选择测试用例的方法是等价类划分。它是指分步骤地把害了（无限）的测试用例集减得很小，但过程同样有效。

一个等价类或者等价划分是指测试相同目标或者暴露相同软件缺陷的一组测试用例。

在寻找等价划分时，考虑软件具有相似输入、相似输出、相似操作的分在一组。这些组就是等价划分。

5.4 数据测试

对软件最简单的认识就是将其分为两部分：数据和程序。**数据**包括：键盘输入、鼠标单击、磁盘文件、打印输出等。**程序**是指可执行的流程、转换、逻辑和运算。软件测试常用的方法是把测试工作按同样的形式划分。

对数据进行软件测试，就是在检查用户输入信息、返回的结果以及中间计算结果是否正确。使所有的数据得以测试的技巧是，根据一些关键的原则进行等家里划分，以合理减少测试用例，这些关键的原则是：边界条件、次边界条件、空值和无效数据。

（1） 边界条件。

如果软件能够在边界条件下运行，那么在正常情况下就应该不会有什么问题。边界条件是特殊情况，因为编程从根本上说在边界容易产生问题。

提出边界条件时，一定要测试临近边界的有效数据，测试最后一个可能的有效数据，同时测试刚超过边界的无效数据。

缓冲区溢出是由边界条件缺陷引起的，它是造成软件安全问题的头号原因。

（2） 次边界条件。

有些边界在软件内部，最终用户几乎看不到，但是软件测试员仍有必要进行检查。这样的边界条件称为次边界条件。

寻找这样的边界条件不要求软件测试员成为程序员或者具有阅读源代码的能力，但是确实要求大体了解软件的工作方式。所测试的软件可能有许多其他的此边界条件，所以软件测试员应和开发小组的程序员交流，看看他们能否对其他应该测试的侧边界条件提供建议。

（3） 默认、空白、空值、零值和无

另一种看起来很明显的软件缺陷来源是当软件要求输入时——比如在文本框中——不是没有输入正确信息，而是根本没有输入任何内容，可能单单是按了 Enter 键。这种情况在产品说明书中常常忽视，程序员也经常遗忘，但是在实际使用中却时有发生。

好的软件会处理这种情况。它通常将输入内容默认为边界内最小的合法值，或者在合法划分中间的某个合法值；或者返回错误提示信息。

（4） 非法、错误、不正确和垃圾数据，这是实效性测试的对象，设法破坏软件。

5.5 状态测试

软件测试的另一方面是通过不同的状态验证程序的逻辑流程。**软件状态**是指软件当前所处的条件或者模式。

（1） 测试软件的逻辑流程

测试软件的状态和逻辑流程有同样的问题。困难在于除了极其简单的程序之外，基本上不可能走遍所有分支，达到所有状态。解决方法是运用等价划分技术选择状态和分支。因为

选择不做完全测试，所以要承担一定的风险，但是通过合理选择减少风险。

- 第一步是简历软件的状态转换图。这样的图可能作为产品说明的一部分被提供出来。否则，就需要创建一个状态图（因为是进行黑盒测试，所以不必了解代码中设置的底层变量。从短剑用户的角度建立状态图即可）。

状态转换图应该表示出以下项目：

- ✓ 软件可能进入的每一种独立状态。
- ✓ 从一种状态转入另一种状态所需的输入和条件。
- ✓ 进入或者退出某种状态时的设置条件及输出结果。

- 减少要测试的状态及转换的数量

正如对数据进行等价划分一样，需要大量的可能性减少到可以操作的测试用例集合。有以下 5 种实现方法：

- ✓ 每种状态至少访问一次。如何到达没关系，但是每一种状态都必须测试。
- ✓ 测试看起来是最常见和最普遍的状态转换。
- ✓ 测试状态之间最不常用的分支。这些分支是最容易被产品设计者和程序员忽视的。
- ✓ 测试所有错误状态及其返回值。错误没有得到正确处理、错误提示信息不正确、修复错误时未正确恢复软件等情况常有发生。
- ✓ 测试随机状态转换。

- 怎样进行具体测试

测试状态及其转换包括检查所有的状态变量——与进入和退出状态相关的静态条件、信息、值、功能等。

（2） 失败状态测试

以上探讨的状态测试都术语通过性测试，测试包括审查软件、描绘状态、尝试各种合法的可能性、确认状态及其转换正常。和数据测试一样，想法的做法是找到使测试软件失败的案例。此类案例的例子是竞争条件、重复、压迫和重负。

- 竞争条件和时序错乱

当今大多少操作系统，无论是用于个人计算机还是专用设备，都具备多任务执行能力。多任务是指操做系统设计用来同时执行多个独立的进程。

设计多任务操作系统并不繁琐，设计充分利用多任务的软件才是艰巨的任务。在真正的多任务环境中，软件设计绝对不能想当然，必须处理随时被中断的情况，能够与其他任何软件在系统中同时运行，并且共享内存、磁盘、通信以及其他硬件资源。

这一切的结果就是可能导致竞争条件问题。这些问题是指几个事件恰巧挤在一起，由于软件未预料到运行过程会被中断，以致造成混乱，即时序发生错乱。竞争条件——多个进程向前冲刺，不知道谁会首先到达。

- 重复、压迫和重负

重复测试，是不断执行同样的操作。进行这种反复测试的主要原因是检查是否存在内存泄露。如果计算机内存被分配进行某些操作，但是操作完成时没有完全释放，就会产生一个常见的软件问题。结果是最后程序耗尽了它赖以工作的内存空间。

压迫测试，是使软件在不够理想的条件下运行——内存小、磁盘空间少、CPU 速度慢|调制解调器速率低等。

重负测试，让软件处理竟可能大的数据文件。最大限度地发掘软件的能力，让它不堪重负。时间也是一种重负测试。

5.6 其他黑盒测试技术

（1） 像笨拙的用户那样做。

（2） 在已经找到的软件缺陷的地方再找找。

（3） 像黑客一样思考问题。黑客知道没有软件是100%安全的，黑客会利用这一点去寻找软件的安全漏洞。

（4） 凭借经验、直觉和感觉

第 6 章 检查代码

6.1 静态白盒测试：检查设计和代码

静态测试是指测试非运行部分——检验和审查。**白盒测试**就是指访问代码，能够查看和审查。

静态白盒测试是在不执行软件的条件下有条理地仔细审查软件*设计、体系结构* 和*代码*，。从而找出软件缺陷的过程，有时称为*结构化分析*。

进行静态白盒测试的首要原因是尽早发现软件缺陷，以找出动态黑盒测试难以发现或隔离的软件缺陷。进行静态白盒测试的另一个好处是，为黑盒测试员在接收软件进行测试时设计和应用测试用例提供思路。

注意：开发小组负责静态百合测试的人员不是固定的。在某些小组中程序员就是组织和执行审查的人员，软件测试员被邀请作为独立观察者。还有一些小组中，软件测试员是该任务的执行人，要求编写代码的程序员和其他同事帮助审查。

6.2 正式审查

正式审查 就是进行静态白盒测试的过程。

正式审查的 4 个基本要素：

- 确定问题。审查的目的是找出软件的问题——出错的项目，和遗漏的项目。全部的批评应该直指代码，而不是设计是现者。参与者之间不应该相互指责，应该把自我意识、个人情绪和敏感丢在一边。
- 遵守规则。审查要遵守一套固定的规则，规则可能设定要审查的代码量、花费多少时间，哪些内容要做评价等。其重要性在于参与者了解自己的角色目标是什么。
- 准备。每一个参与者都为审查做准备，了解自己的责任和义务，积极参与审查。
- 编写报告。审查小组必须做出审查结果的书面总结报告，并使报告便于开发小组的成员使用。

（1） 同事审查。类似于“你给我看你的，我也给你看我的”，同事审查常常仅在编写代码或设计体系结构的程序员，以及充当审查者的其他一两个程序员和测试员之间进行。这个小团体只是在一起审查代码，寻找问题和失误。

（2） 走查。走查是比同事审查更正规化的下一步。在走查的过程中通常有代码的作者来给与会的人讲解自己那份代码是干啥的，为啥那样写。在走查的过程中最好是一个有个资深程序员在场。

（3） 检验。检验是最正式的审查类型，对与会者的要求比较高，而且最好是有经验的，培训过的。审核与前两者的不同在于--讲解者或者阅读代码的人并不是那份代码的原作者。其余的与会者的职责是从不同的角度，例如用户、测试员或者产品支持人员的角度审查代码。

6.3 编码标准和规范

有三个重要的原因要坚持标准或规范：

- **可靠性**。事实证明按照某标准或规范编写的代码比不这样做的代码更加可靠和实用。
- **可读性/维护性**。符合设备标准和规范的代码抑郁阅读、理解和维护。
- **移植性**。代码经常需要在不同的硬件中运行，或者使用不同的编译器编译。如果代码符合设备标准，迁移到另一个平台就会轻而易举，甚至完全没有障碍。

6.4 通用代码审查清单

（1） 数据引用错误（*缓冲区溢出的主要原因*）

数据引用错误是指使用未经正确声明和初始化的变量、常量、数组、字符串或记录而导致的软件缺陷。

- 是否引用了未初始化的变量？查找遗漏之处与查找错误同等重要。
- 数组和字符串的下标是整数值吗？下标总是在数组和字符串长度范围之间吗？
- 在检索操作或者引用数组下标时是否包含“丢掉一个”这样的潜在错误？
- 是否在应该使用常量的地方使用了变量——例如在检查数组边界时？
- 变量食肉被赋予不同类型的值？例如，无意中使代码为整形变量赋予一个浮点数值？
- 为引用的指针分配内存了吗？
- 一个数据结构是否在多个函数或者子程序中引用，在没有一个引用中明确定义结构了吗？

（2） 数据声明错误

数据声明错误产生的原因是不正确地声明或使用变量和常量。

- 所有变量都赋予正确的长度、类型和存储类了吗？
- 变量是否在声明的同时进行了初始化？是否正确初始化并与其类型一致？
- 变量有相似的名称吗？这基本上不算软件缺陷，但是有可能使程序中其他地方出现名称混淆的信息。
- 存在声明过、但从未使用或者只引用过一次的变量吗？
- 所有变量在特定模块中都闲适声明了吗？如果没有，是否可以理解为该变量与更高级别的模块共享？

（3） 计算错误

计算或者运算错误实际是是糟糕的数学问题。计算无法得到预期结果。

- 计算中是否使用了不同数据类型的变量，如整数与浮点数相加？
- 计算中是否使用了数据类型相同但字节长度不同的变量？
- 计算时是否了解和考虑到编译器对类型或长度不一致的变量的转换规则？
- 赋值的目的变量是否小于赋值表达式的值？
- 在数值计算过程中是否可能出现溢出？
- 除数或模是否可能为零？
- 对于整型算术运算或某些计算，特别是除法的代码处理是否会丢失精度？
- 变量的值是否超过有意义的范围？
- 对于包含多个操作的表达式，求值次序是否混乱，运算优先级对吗？需要加括号使其清晰吗？

（4） 比较错误

小于、大于、等于、不等于、真、假、比较和判断错误很可能是边界条件问题。

- 比较得正确吗？
- 存在分数或者浮点数之间的比较吗？如果有，精度问题会影响比较吗？1.00000001 和 1.00000002 极其接近，它们相等吗？
- 每一个逻辑表达式都正确地表达了吗？逻辑计算如期进行了吗？求值次序有疑问吗？
- 逻辑表达式的操作数是逻辑值吗？

（5） 控制流程错误

控制流程错误是指编程语言中循环等控制结构未按预期方式工作，通常由计算或者比较错误直接或间接造成。

- 如果程序包含 begin…end 和 do…while 等与剧组，end 是否明确给出并与语句对应？程序中的语句组是否对应？
- 程序、模块、子程序和循环能否终止？如果不能，可以接受吗？
- 可能存在永远不停的循环吗？
- 循环可能从不执行吗？如果是这样，可能接受吗？
- 对于多分支语句，索引变量能超出可能的分支数目吗？如果超出，该情况能正确处理吗？
- 是否存在"丢掉一个"错误，导致意外进入循环？

（6） 子程序参数错误

子程序参数错误的来源是软件子程序不正确地传递数据。

- 子程序接收的参数类型和大小与调用代码发送的匹配吗？次序正确吗？
- 如果子程序有多个入口点，引用的参数是否与当前入口点没有关系？
- 常量是否当作形参传递，意外在子程序中改动？
- 子程序是更改了仅作为输入值的参数？
- 每一个参数的单位是否与相应的形参匹配？
- 如果存在全局变量，在所有引用子程序中是否有相似的定义和属性？

（7） 输入/输出错误

输入/输出错误包括文件读取、接受键盘或鼠标输入以及向输出设备写入错误等。

- 软件是否严格遵守外设读写数据的专用格式？
- 文件或者外设不存在或者未准备好的错误情况有处理吗？
- 软件是否处理外设未连接、不可用、或者读写过程中存储空间占满等情况？
- 软件以预期的方式处理预计的错误吗？
- 检查错误提示信息的准确性、正确性、语法和拼写了吗？

（8） 其他检查

- 软件是否使用其他外语？是否处理扩展 ASCII 字符？是否需用统一编码取代 ASCII？
- 软件是否需要移植到其他编译器？
- 是否考虑了兼容性，以使软件能够运行于不同数量的可用内存、不同的内部硬件、不同的外设等？
- 程序编译是否产生"警告"或者"提示"信息？这些信息通常指示语句有疑问。

第 7 章 带上 X 光眼镜测试软件

本章讲四个基本测试之中的第四个——动态白盒测试。另三个为静态黑盒（测试产品说明书）、动态黑盒（测试软件）和静态白盒（检查程序代码）。

7.1 动态白盒测试

（1） 动态白盒测试是指利用查看代码功能（做什么）和实现方式（怎么做）得到的信息来确定哪些需要测试、哪些不要测试、如何开展测试。

（2） 动态白盒测试的另一个常用名称是结构化测试（structural testing），因为软件测试员可以查看并使用代码的内部结构，从而设计和执行测试。

（3） 动态白盒测试不仅仅是查看代码的运行情况，还包括直接测试和控制软件，动态白盒测试包括以下 4 个部分：

- 直接测试底层函数、过程、子程序和库。在 windows 中称为 API。
- 以完整程序的方式从顶层测试软件，但是根据对软件运行的了解调整测试用例。
- 从软件获得读取变量和状态信息的访问权，以便确定测试与期望结果是否相符，同时，强制软件以正常测试难以实现的方式运行。

- 估算执行测试时“命中”的代码量和具体代码，然后调整测试，去掉多余的测试用例，补充遗漏的用例。

7.2 动态白盒测试和调试

不要混淆动态白盒测试和调试，两者表面上相似，因为它们都包括处理软件缺陷和查看代码的过程，但目标大不相同。

动态白盒测试的目标是寻找软件缺陷，调试的目标是修复缺陷。然而它们在隔离软件缺陷的位置和原因上确实存在交叉现象。

执行这些底层的测试，会用到许多和程序员使用的相同的工具。如果程序编译过，可能会使用相同的编译器。

可能会使用代码级的调试器来单步跟踪程序，观察变量，设置断点，等等。也可能自己编写程序来分别测试需要检查的模块代码。

7.3 分段测试

一般产生高额费用的原因有：

- 难以找出导致问题的原因；
- 某些软件缺陷掩盖了其它软件缺陷，测试可能失败。

（1） 单元测试和集成测试

解决上述问题的方法当然是一开始就不让它发生。如果代码分段构建和测试，最后合在一起形成更大的部分，那么整个产品无疑会链接在一起。

在底层进行的测试称为单元测试（unit testing）或者模块测试（module testing）。单元经过测试，底层软件缺陷被找出并修复之后，就集成在一起，对模块的组合进行集成测试（integration testing）。这个不断增加的测试过程继续进行，加入越练越多的软件片段，直至整个产品——至少是产品的主要部分——在称为系统测试（system testing）的过程中一起测试。采取这种测试策略很容易隔离软件缺陷。

在单元级发现问题时，问题肯定就在那个单元中，如果在多个单元集成时发现软件缺陷，那么它一定与模块之间的交互有关。当然也有例外。

这种递增测试有两条途径：自底向上（bottom－up）和自顶向下（top－down）。

在自底向上测试中，要编写称为测试驱动的模块调用正在测试的模块。测试驱动模块以和将来真正模块同样的方式挂接，向处于测试的模块发送测试用例数据，接受返回结果，验证结果是否正确。采取这种方式，可以对整个软件进行非常全面的测试，为它提供全部类型和数量的数据，甚至高层难以发送的数据。

自顶向下测试有点像小规模的大爆炸测试。

（2） 单元测试用例

在进行白盒测试之前，一定要根据说明书建立黑盒测试用例，用这种方式可以真正测试模块的功能和作用。

如果先从模块的白盒角度建立测试用例，检查代码，就会偏向于以模块工作方式建立测试用例。如程序员误解了说明，于是测试用例就会不对。

虽然测试用例精确完整地测试了模块，但是可能不准确，因为没有测试预期的操作。根据白盒知识增减测试用例时根据程序内部的信息对等价划分的进一步提炼。

7.4 数据覆盖

查看代码决定如何调整测试用例。合理的方法是像黑盒测试那样把软件代码分成数据和状态。从同样的角度看软件，可以相当容易地把得到的白盒信息映射到已经写完的黑盒测试用例上。

首先考虑数据。数据包括所有的变量、常量、数组、数据结构、键盘和鼠标输入、文件、屏幕输入/输出、以及调制解调器、网络等其它设备的输入和输出。

（1） 数据流（Data Flow）

数据流覆盖主要是指在软件中完全跟踪一批数据。在单元测试级，数据仅仅通过了一个模块或者函数。同样的跟踪方式可以用于多个集成模块，甚至整个软件产品。

如果在底层测试函数，就会使用调试器观察变量在程序运行时的数据。通过黑盒测试，只能知道变量开始和结束的值。通过动态白盒测试，还可以在程序运行期间检查变量的中间值。根据观察结果就可以决定更改某些测试用例。

（2） 次边界

以下是次边界导致软件缺陷最常见的例子：

- 计算税收的模块在某些财务结算处可能从使用数据表转向使用公式。
- 在 RAM 底端运行的操作系统也许开始把数据移到硬盘上的临时存储区。这种次边界甚至无法确定，它随着磁盘上剩余空间的数量而发生变化。
- 为了获得更高的精度，复杂的数值分析程序根据数字大小可能切换到不同的等式以解决问题。

如果进行白盒测试，就需要仔细检查代码，找到次边界条件，并建立能测试它们的测试用例。

（3） 公式和等式

公式和等式通常深藏于代码中。

撇开代码中的公式和等式，查看它们使用的变量，在程序正常输入和输出之外，为其建立测试用例和等价划分。

（4） 错误强制（error forcing）

如果执行在调试器中测试的程序，不仅能够观察到变量的值——还可以强制改变变量的值。

注意：在使用错误强制时，小心不要设置现实世界中不可能出现的情况。如果程序员在函数开头检查 n 值必须大于 0，而且 n 值仅用于该公式中，那么将 n 值设为 0，使程序失败的测试用例就是非法的。

如果仔细选择了错误强制情况，并和程序员一起反复检查以确认它们是合法的，错误强制就是一个有效的工具。

使用错误强制的上好方法就是迫使软件中的所有错误提示信息显示出来。

大多数软件使用内部错误代码表示错误的提示信息。

7.5 代码覆盖（Code Coverage）

测试数据只是一半的工作。为了全面地覆盖，还必须测试程序的状态及程序流程。必须设法进入和退出每一个模块，执行每一行代码，进入软件每一条逻辑和决策分支。这种类型的测试叫做代码覆盖测试。

代码覆盖测试是一种动态白盒测试。

代码覆盖测试最简单的形式是利用编译环境的调试器通过单步执行程序查看代码。

对于小程序或者单独模块，使用调试器就足够了。然而对大多数程序进行代码覆盖测试要用到称为代码覆盖率分析器（code coverage analyzer）的专用工具。

代码覆盖率测试器挂接在正在测试的软件中，当执行测试用例时在后台执行。每当执行一个函数，一行代码或一个逻辑决策分支时，分析器就记录相应的信息。从中可以获得指示软件哪些部分被执行，哪些部分未被执行的统计结果。利用该数据可以得到：

- 测试用例没有覆盖软件的哪些部分；
- 哪些测试用例时多余的；
- 为了使覆盖率更好，需要建立什么样的新测试用例；
- 软件质量的大致情况。

如果测试用例覆盖了软件的 90%而未发现任何软件缺陷，就说明软件质量非常好。这并非绝对。注意“杀虫剂现象”，软件测试得越多，它对测试的免疫能力越强，如果测试用例覆盖了软件的 90%而未发现任何软件缺陷，也有可能使软件的构造不好——可能是软件对测试具有了免疫能力。增加新的测试用例可能会暴露余下的 10%具有非常多的缺陷。

（1） 程序语句和代码行覆盖

代码覆盖最直接的形式称为语句覆盖（statement coverage）或者代码行覆盖（line coverage）。如果在测试软件的同时监视语句覆盖，目标就是保证程序中每一条语句最少执行一次。

遗憾的是，语句覆盖是一种误导，即使全部语句都被执行了，也不能说走遍了软件的所有路径。

（2） 分支覆盖

试图覆盖软件中的所有路径称为路径覆盖。路径测试最简单的形式称为分支覆盖测试。

大多数代码覆盖率分析器将根据代码分支，分别报告语句覆盖和分支覆盖的结果，是软件测试员更加清楚测试的效果。

（3） 条件覆盖

与分支覆盖一样，代码覆盖率分析器可以被设置为在报告结果时将条件考虑在内。如果测试条件覆盖，就能达到分支覆盖，顺带也能达到语句覆盖。

第 8 章 配置测试

8.1 配置测试综述

配置测试是指使用各种硬件来测试软件运行的过程。配置测试的目的是保证被测试的软件在尽可能多的硬件平台上运行。

- ✓ 个人计算机
- ✓ 部件
- ✓ 外设
- ✓ 接口
- ✓ 可选项和内存
- ✓ 设备驱动程序

如果开始准备进行软件的配置测试，就要考虑哪些配置与程序的关系最密切。理想情况是所有生产厂家都严格遵照一套标准来设计硬件，那么使用这些硬件的软件就会毫无疑问地正常运行。但遗憾的是，标准并没有被严格遵守。有时，标准是相当松散的——称为规范。

（1） 分离配置缺陷

判断缺陷是配置问题而不仅仅是普通缺陷最可靠的方法是，在另外一台有完全不同配置的计算机上一步步执行导致问题的相同操作，如果缺陷没有产生，就极有可能是特定的配置问题，在独特的硬件配置下才会暴露出来。

首先，要找出问题所在。这通常是动态白盒测试员和程序员调试的工作。一个配置问题产生的原因不少，全都要求有人在不同的配置中运行软件时仔细检查代码，以找出缺陷：

- 软件可能包含在多种配置中都会出现的缺陷。
- 软件可能只包含在某一个特殊配置中出现的缺陷。
- 硬件设备或者其设备驱动程序可能包含仅由软件揭示的缺陷。
- 硬件设备或者其设备驱动程序可能包含一个借助许多其它软件才能看出来的缺陷——尽管它可能对测试的软件特别明显。

前两种情况，显然要由项目小组负责修复缺陷。

后两种情况，责任不那么清晰。如果该硬件设备属于流行产品，被各界广泛使用，那么，

开发小组需要针对缺陷对软件做修改，即使软件的运行是正确的。

归根结底，无论问题出在哪里，解决问题都是开发小组的责任。

（2） 计算工作量

配置测试工作量可能非常巨大。每一类硬件都有各种生产厂商和型号。如果决定进行完整、全面的配置测试，检查所有可能的制造者和型号组合，就会面临这巨大的工作量。减少麻烦的答案是*等价划分*。需要找出一个方法把巨大无比的配置可能性减少到尽可能控制的范围。由于没有完全测试，因此存在一定的风险，但这正是软件测试的特点。

8.2 执行任务

确定测试哪些设备和如何测试的决定过程是相当直观的等价划分工作。什么重要，怎样才会成功，是决定的内容。

（1） 确定所需的硬件类型

仔细查看软件的特性，确保测试全面、彻底。

联机注册：在选择用哪些硬件来测试时容易忽略的一个特性例子是联机注册。如果软件有联机注册功能，就需要把调制解调器和网络通信考虑在配置测试之中。

（2） 确定有哪些厂商的硬件、型号和驱动程序可用

确定要测试的设备驱动程序，一般选择操作系统附带的驱动程序、硬件附带的驱动程序或者硬件或操作系统公司网站上提供的最新的驱动程序。

（3） 确定可能的硬件特性、模式和选项

每一种设备都有选项，软件没有必要全部支持。计算机游戏就是一个好例子。许多游戏要求最小验收数和显示分辨率。如果配置低于该要求，游戏就不能运行。

（4） 将确定后的硬件配置缩减成为可控制的范围

假设没有时间和计划测试所有配置，就需要把成千上万种可能的配置缩减到可以接受的范围——即要测试的范围。

一种方法是把所有配置信息放在电子表格中，列出生产厂商、型号、驱动程序版本和可选项。软件测试员和开发小组可以审查这张表，确定要测试哪些配置。

注意：用于把众多配置等价划分为较小范围的决定过程最终取决于软件测试员和开发小组。这没有一个定式，每一个软件工程都不相同，都有不同的选择标准。一定要保证项目小组中的每一个人（特别是项目经理），搞清楚什么配置要测试（什么不测试），选择它们引起的变化有哪些。

（5） 明确与硬件配置有关的软件唯一特性

这里的关键词是*唯一*。不应该也没有必要在每一种配置中完全测试软件。只需测试哪些与硬件交互时互不相同的特性即可。

选择唯一特性进行尝试并非那么容易，首先应该进行黑盒测试，通过查看产品找出明显的特性，然后与小组成员交流，了解其内部的白盒情况。最后会惊奇的发现这些特性与配置有一些紧密的关联。

（6） 设计在每一种配置中执行的测试用例

- ✓ 从清单中选择并建立下一个测试配置；
- ✓ 启动软件；
- ✓ 打开文件 configtest.doc；
- ✓ 确认显示出来的文件正确无误；
- ✓ 打印文档；
- ✓ 确认没有错误提示信息，而且打印的文档符合标准；
- ✓ 将任何不符之处作为软件缺陷记录下来。

实际上，这些步骤还有更多内容，包括具体要做什么、找什么的细节和说明。目标是建

立任何人都可以执行的步骤。

（7） 在每种配置中执行测试

执行测试用例，仔细记录并向开发小组报告结果，必要时还要向硬件生产厂商报告。明确配置问题的准确原因通常很困难，而且非常耗时，软件测试员需要和程序员紧密合作。如果软件缺陷是硬件的原因，就利用生产厂商的网站向其报告问题。

（7） 反复测试指导小组对结果满意为止

配置测试一般不会贯穿整个项目期间。最初可能会尝试一些配置，接着整个测试通过，然后在越来越小的范围内确认缺陷的修复。最后达到没有未解决的缺陷或缺陷限于不常见或不可能的配置上。

8.3 获得硬件

购买每一样硬件则费用很高昂。

- 只买可以或者将会经常使用的配置；
- 与硬件厂商联系，看能否租借甚至赠送某些硬件；
- 向全公司的人询问其家里是否有硬件。
- 如果项目经费充足，就和项目经理一起与专业配置兼容性测试实验室联系外协测试。

8.4 明确硬件标准

了解硬件说明书的一些细节，有助于做出更多清晰的等价划分决定。

8.5 对其他硬件进行配置测试

根据从设备使用者、项目经理或者销售人员那里获得的信息来建立硬件的等价划分。开发测试用例，收集所选硬件，执行测试。

第 9 章 兼容性测试

9.1 兼容性测试综述

软件兼容性测试 是指检查软件之间是否能够正确地交会和共享信息。

兼容性对于软件的意义取决于开发小组决定用什么来定义，以及软件运行的系统要求的兼容性级别。如果受命对新的软件进行兼容性测试，就需要回答以下问题：

- 软件设计要求与何种平台和应用软件保持兼容？如果要测试的软件是一个平台，那么设计要求什么应用程序在其上运行？
- 应该遵守何种定义软件之间交互的标准或规范？
- 软件使用何种数据与其他平台和软件交互和共享信息？

9.2 平台和应用程序版本

选择目标平台或者兼容的应用程序实际上是程序管理或市场定位的任务。软件设计用于某个操作系统、Web 浏览器或者其他平台要由熟悉客户基本情况的人来定。他们还要明确软件的版本或软件需要兼容的版本。

每一种平台都有自己的开发标准，并且从项目管理的立场看，使平台清单在满足客户要求的前提下尽可能小是很重要的。

（1） 向后和向前兼容

向后兼容是指可以使用软件的以前版本；向前兼容是指可以使用软件的未来版本。并非所有的软件或文件都需要向前或向后兼容。

（2） 测试多个版本的影响

测试平台和软件应用程序多个版本相互之间能否正常工作可能是一个艰巨的任务。运用等价划分，关键词是“重要”。

9.3 标准和规范

高级标准是产品普遍遵守的规则，例如外观和感觉、支持的特性等。

低级标准是本质细节，例如文件格式和网络通信协议等。

（1） 高级标准和规范

如果某个应用程序声称和某个平台兼容，就必须遵守该平台自身的标准和规范。

（2） 低级标准和规范

从某种意义上说，低级标准比高级标准更重要。通信协议编程语言语法以及程序用于共享信息的任何形式都必须符合公开的标准和规范。低级兼容性标准可以作为软件说明书的扩充部分。

9.4 数据共享兼容性

在应用程序之间共享数据实际上是增强软件的功能。写得好的程序支持并遵守公开标准；允许用户与其他软件轻松传输数据，这样的程序可称为兼容性几号的产品。

程序之间最为人熟知的数据传输方式是读写磁盘文件。

- 文件保存和文件杜旭是人人公职的数据共享方法。
- 文件导出和文件导入时许多程序与自身以前版本、其他程序保持兼容性的方式。
- 剪切、复制和黏贴式程序之间无需借助磁盘传输数据的最常见的数据共享方式。如果对某种程序进行兼容性测试，就要确认其可以利用剪贴板与其他程序相互复制数据。
- DDE，COM 和 OLE 是 Windows 中两个程序之间传输数据的方式，DDE 飙升动态数据交换，OLE 表示对象链接和嵌入。

第 10 章 外国语言测试

如果你是一个有竞争力的软件测试员，并且熟练掌握除英语之外的一门外语，你就有了很有价值的技能。

10.1 使文字和图片有意义

软件的国际化，除了语言，还需要考虑地域（region 或 locale）——用户的国家和地理位置。使软件适应特定地域特征，照顾到语言、方言、地区习俗和文化的过程称为本地化（localization）或国际化（internationalization）。测试此类软件称为本地化测试。

10.2 翻译问题

尽管翻译只是整个本地化工作的一部分，但是从测试角度看这是重要的一环，最明显的问题是如何测试用其它语言做的产品。

软件测试员或者测试小组至少要对所测试的语言基本熟悉，能够驾驭软件，看懂软件显示的文字，输入必要的命令执行测试。

注意：软件测试小组一定要有人对测试的语言比较熟悉。对多种语言的情况，可以委托本地化测试公司进行测试。

（1） 文本扩展（text expansion）

实践证明，当英语被翻译为其它语言，用来表达同一事务时往往需要加一些字符。一个好的大拇指规则是每个单词长度预计增加 100％。因为这些扩展现象，故必须仔细测试可能受到变长了的文本影响的软件部分，要找出没有正确换行、截断的和连字符位置不对的文本；还要找到虽然文本有足够的扩展空间，但这是通过把其它的文本挤出去来实现的情况。变长了的文本还可能导致主程序失败，甚至系统崩溃。

（2） ASCII、DBCS 和 Unicode

ASCII 字符集只能表示 256 种不同的字符——远不足以表示所有语言的全部字符。当开始为不同语言开发软件时，就需要找到克服该限制的解决方案。常用的一个方法是代码页（code page）技术，实质是 ASCII 表的替换，每一种语言用一个不同的代码页。

这个方法虽然笨，但对于少于 256 个字符的语言还是可行的。但对于中文、日文则不行。

这时，使用 DBCS（双字节字符集）的系统提供对超过 256 个字符的语言的支持。用两个字节代替一个字节来表示最多可容纳 65536 个字符。

代码页和 DBCS 在许多情况下足够了，但是会遇到一些问题，最重要的是兼容性问题。解决这个麻烦的方法是使用 Unicode 标准。Unicode 为每一个字符提供唯一编号，无论何种平台，无论何种程序，无论何种语言。

（3） 热键和快捷键

在软件的本地化版本中，需要测试所有热键和快捷键工作是否正常，而且使用起来不困难。

（4） 扩展字符

本地化软件，甚至非本地化软件中存在的一个常见问题是扩展字符（extended characters）。扩展字符指的是普通英文字母 A~Z 和 a~z 之外的字符。测试扩展字符的方法是找出软件所有接受字符输入和输出之处。在每一处尝试使用扩展字符，看能否与常规字符一样处理。对话框、文本域都是合适的对象。

技巧：测试扩展字符是否被正确处理的最简单的方法是，把它们加入测试的标准字符所在的等价划分之中。

（5） 字符计算

与扩展字符有关的问题是软件在对其进行计算时如何接受解释。要弄清楚测试的语言采用什么样的排序规则，并开发测试用例专门检查排列次序的正确性。扩展字符计算打破的另一个领域是大小写转换。

（6） 从左向右和从右向左读

翻译中有一个大难题是某些语言（例如希伯莱文和阿拉伯文）从右向左读，而不是从左向右读。幸好大多数主要操作系统提供了处理这些语言的内部支持。如果没有这一点，完成任务几乎是不可能的。即便这样，翻译这样的文本也不是容易的事。

（7） 图形中的文字

另一个翻译问题是处理图形中的文字。它的影响是当软件本地化时，每一个图标都要改变，以反映新的语言。

（8） 让文本与代码脱离

所有文本字符串、错误提示信息和其它可以翻译的内容都应该存放在与源代码独立的文件中。大多数本地化人员不是程序员，也没有必要是。让其修改资源文件（resource file）的简单文本文件，该文件包含软件可以显示的全部信息。

当软件运行时，通过查找该文件来引用信息，不管信息的内容是什么，都按照原文显示。这就是说，对于白盒测试员来说，检查代码，确保没有任何嵌入的字符串未出现在外部文本文件中很重要。这个问题的另一个变化形式是当代码动态生成文本信息时。

10.3 本地化问题

翻译问题只是全部问题的一半。翻译文字和允许字符串包含不同字符和长度都不难，难的是修改软件使其适应国外市场。经过准确翻译和仔细测试的软件是精确和可靠的，但是如果程序员不考虑本地化的问题，程序就可能不够准确和高质量。

（1） 内容

这里的内容是指产品中除了代码之外的所有东西。以下清单给出了本地化问题要仔细审查的各类内容：范例文档，图标，图片，声音，视频，帮助文件，有边界争端的地图，市场宣传材料，包装，Web 链接。

（2） 数据格式

不同的地区在诸如货币、时间和度量衡上使用不同的睡觉单位格式。在本地化时需要修改程序代码。如果测试本地化软件，就需要对当地使用的度量单位非常熟悉，为了正确测试

软件，需要从原版软件创建的测试数据中建立不同的等价划分。

10.4 配置和兼容性测试

（1） 国外平台配置

键盘也许是语言依赖性最大的硬件——键盘布局。从根本上讲，软件可能会用到的任何外设都要在平台配置和兼容性测试的等价划分中去考虑。

（2） 数据兼容性

10.5 测试量有多大

如果软件从一开始就考虑到了本章所述的问题，那么本地化版本中包含更多软件缺陷和增大测试量的风险就很小。

注意：本地化测试量的要求是一个有风险的选择。

另一个问题关系到整个软件产品中什么需要改变。如果本地化工作只限于修改诸如文本和图形等内容——不是代码，——测试工作可能只是对改动进行合法性检查。如改动基本代码，则需考虑测试代码，并且检查功能和内容。

第 11 章 易用性测试

易用性（Useability）是交互的适应性、功能性和有效性的集中体现。*人体工程学*（ergonomics）是一门将日常使用的东西设计为易于使用和实用性强的学科。人体工程学的主要目标是达到易用性。

11.1 用户界面测试

用于与软件交互的方式称为用户界面或 UI。虽然各种 UI 各不相同，但是从技术上讲，他们与计算机进行同样的交互——提供输入和接受输出。

11.2 优秀 UI 由什么构成

软件测试员要负责测试软件的易用性，包括其用户界面。记住，软件测试员不需要去设计 UI，只需要把自己当作用户，然后去找出 UI 中的问题。优秀 UI 具备的七个要素：

符合标准和规范；直观；一致；灵活；舒适；正确；实用。

（1） 符合标准和规范

最重要的用户界面要素是软件符合现行的标准和规范——或者有真正站得住脚的不符合的理由。

注意：如果测试在特定平台上运行的软件，就需要把该平台的标准和规范作为产品说明书的补充内容。像对待产品说明书一样，根据它建立测试用例。

这些标准和规范由软件易用性专家开发。它们是经由大量正规测试、使用、尝试和错误而设计出的方便用户的规则。

也并非要完全遵守准则，有时开发小组可能想对标准和规范有所提高。平台也可能没有标准，也许测试的软件就是平台本身。在这种情况下，设计小组可能成为软件易用性标准的创立者。

（2） 直观

- 用户界面是否洁净、不唐突、不拥挤？UI 不应该为用户使用制造障碍。所需功能或者期待的响应应该明显，并在预期出现的地方。
- UI 的组织和布局合理吗？是否允许用户轻松地从一个功能转到另一个功能？下一步做什么明显吗？任何时刻都可以决定放弃或者退回、退出吗？输入得到确认了吗？菜单或者窗口是否太深了？
- 有多余功能吗？软件整体抑或局部是否做得太多？是否有太多特性把工作复杂化了？是否感到信息太庞杂？
- 如果其他所有努力失败，帮助系统真的能帮忙吗？

（3） 一致

被测软件本身以及与其他软件的一致是一个关键属性。如果软件或者平台有一个标准，就要遵守它。如果没有，就要注意软件的特性，确保相似的操作以相似的方式进行。在审查产品的时候想一想一下几个基本术语：

- 快捷键和菜单选项。
- 术语和命名。整个软件使用同样的术语吗？特性命名一致吗？
- 听众。软件是否一直面向同一级别的听众？
- 诸如 OK 和 Cancel 按钮的位置。大家是否注意到 Windows 中 OK 按钮总是在上方或者左方，而 Cancel 按钮总是在下方或者右方？

（4） 灵活

用户喜欢选择——不要太多，但是足以允许他们选择想要做的和怎样做。当然，灵活也会带来复杂性。

- **状态跳转**。灵活的软件在实现同一任务上有更多的选择和方式。结果是增加了通向软件各种状态的路径。状态转换图将变得更复杂，软件测试员需要花费更多时间俩决定测试哪些相互连接的路径。
- **状态终止和跳过**。当软件具有用户非常熟悉的超级用户模式时，显然能够跳过众多提示或者窗口直接到达想去的地方。测试具有这种功能的软件时，如果中间状态被跳过或提前终止，就需要保证在跳过所有状态或提前终止时变量被正确设置。
- **数据输入和输出**。用户希望有多种方法输入数据和查看结果。谁知道有多少可能的组合？测试进出软件的各种方式，将极大增加必要的工作量，使等价划分难以抉择。

（5） 舒适

软件使用起来应该舒适，不能给用户工作制造障碍和困难。

- **恰当**。软件外观和感觉应该与所做的工作和使用者相符。
- **错误处理**。程序应该在用户执行关键操作之前提出警告，并且允许用户恢复由于错误操作而丢失的数据。
- **性能**。

（6） 正确

测试正确性，就是测试 UI 是否做了该做的事。

注意：市场定位偏差、语言和拼写、不良媒体、WYSIWYG（所见即所得）。

（7） 实用

是否实用事优秀用户界面的最后一个要素。是指具体特性是否实用。

11.3 为有残疾障碍的人员测试：辅助选项测试

易用性的一个严肃主题是*辅助选项测试*，也就是为有残疾障碍的人测试。残疾有许多张，但是只有下列几种残疾对实用计算机和软件会造成极大的困难：实力损伤、听力损伤、运动损伤、认知和语言障碍。

（1） 法律要求

开发残疾人可以使用的用户界面的软件有一些法律规定。在美国，有 3 条法律，而其他国家也在考虑采取类似的法律：

- 美国公民残疾人条例（ADA）声明
- 居民条例第 508 款
- 通信条例第 255 款

（2） 软件中的辅助特性

软件可以有两种方式提供辅助。最容易的方式是利用平台或者操作系统内置的支持。

如果测试的软件不在这些平台上运行，或者本身就是平台，就需要定义、编制和测试自己的辅助选项。

注意：如果正在测试产品的易用性，一定要专门为辅助选项建立测试用例。

如 windows 系统，提供了：粘滞键，筛选键，切换键，声音卫士，声音显示，高对比度，鼠标键，串行键。

第 18 章 编写和跟踪测试用例

测试计划过程的下一步，编写和跟踪测试用例对软件测试员的常规任务有更直接的影响。开始只能执行别人写好的测试用例，很快就嫩而过自己编写并让其他测试员使用了。

18.1 测试用例计划的目标

有条不紊地仔细计划测试用例，是达成目标的必由之路。这样做的重要性有如下 4 条原因：

- **组织**。即使在小型软件项目上，也可能有数千个测试用例。正确的计划会组织好用例，以便全体测试员和其它项目小组成员有效的审查和使用。
- **重复性**。在项目期间有必要多次执行同样的测试，以寻找新的软件缺陷。
- **跟踪**。计划执行多少个测试用例？在软件最终发行版本上执行多少个测试用例？多少个通过？多少个失败？有被忽略的测试用例吗？等等。如果测试用例没有计划，就不能回答这些问题。
- **测试（或者不测试）证实**。软件测试小组必须证明确实执行了计划执行的测试。发布忽略某些测试用例的软件实际上是不合法和危险的。

特别测试：有一种软件测试称为特别测试（ad hoc testing），描述在没有实际计划下执行测试——没有测试用例计划，有时甚至没有高级测试计划。特别测试就是测试员坐在软件前面开始乱敲键盘。有可能很快找出缺陷，作为有计划测试的补充，具有一定的价值。

18.2 测试用例计划综述

创建测试计划过程比结果文档更重要。三个等级：

- ✓ 测试设计说明（test design specification）
- ✓ 测试用例说明（test case specification）
- ✓ 测试过程说明（test procedure specification）

离最高级测试计划越远，侧重点就越倾向于产生的书面文档，而不是创建过程。最低要求是测试小组应该创建包含软件测试文档 IEEE829 大纲中所述信息的测试计划。要紧的是完成工作后满足了测试用例计划的四个目标：组织、重复性、跟踪和测试证实。

（1） 测试设计

整体项目计划在非常高的等级上编制，它把软件拆分为*具体特性* 和*可测试项* ，并将其分派到每个测试员头上。测试设计说明的目的是组织和描述针对具体特性需要进行的测试。然而，它不给出具体的用例或者执行测试的步骤。

以下为测试设计说明应有的部分内容：

- ✓ **标识符**：用于引用和标记测试设计说明的唯一标识符。
- ✓ **要测试的特性**：测试设计说明所包含的软件特性描述。该部分还将明确指出作为主要特性的辅助特性需要简洁测试的特性。还要列出不被测试的特性，及计划中由于错误分析包含进来的特性。
- ✓ **方法**：描述测试软件特性的通用方法。如果在测试计划中列出方法，就应该进行展开，描述要使用的技术，解释结果如何验证。
- ✓ **测试用例确认**：对用于检查特性的具体测试用例的高级描述和引用。它应该列出所选的等价划分，并提供测试用例的引用信息以及用于执行测试用例的程序。
- ✓ **通过/失败规则**：描述测试特性的通过和失败由什么构成。

（2） 测试用例

测试用例细节基本上应该清楚地解释要向软件发送什么值或者条件，以及预期结果。它可以由一个或多个测试用例说明来引用，也可以引用多个测试程序。包括的重要信息如下：

- ✓ **标识符**：由测试设计过程说明和测试程序说明引用的唯一标识符。

- ✓ **测试项**：描述被测试的详细特性、代码模块等，应该比测试设计说名字所列的特性更加具体。
- ✓ **输入说明**：列举送到软件执行测试用例的所有输入内容或者条件。
- ✓ **输出说明**：描述进行测试用例预期的结果。
- ✓ **环境要求**：指执行测试用例必要的硬件、软件、测试工具、实用工具、人员等。
- ✓ **特殊过程要求**：描述执行测试必须做到的特殊要求。
- ✓ **用例之间的依赖性**：如果一个测试用例依赖于其他用例，或者受其他用例影响，就应该在此说明。

表述测试用例的其它选择有简单列表、大纲甚至诸如状态表或数据流程图之类的图表。

（3） 测试程序

测试程序（test procedure）：明确指出为实现相关测试设计而操作软件系统和试验具体测试用例的全部步骤。

测试程序或者测试脚本（test script）说明详细定义了执行测试用例的每一步操作。以下是需要定义的内容：

- 标识符：把测试程序与相关测试用例和测试设计捆绑在一起的唯一标识符。
- 目的：程序的目的以及将要执行的测试用例的引用信息。
- 特殊要求：执行程序所需的其它程序、特殊测试技术或者特殊设备。
- 程序步骤：执行测试的详细步骤：
  - ✓ 日志：指出用什么方式、方法记录结果和现象；
  - ✓ 设置：说明如何准备测试；
  - ✓ 启动：说明用于启动测试的步骤；
  - ✓ 程序：描述用于运行测试的步骤；
  - ✓ 度量：描述如何判断结果；
  - ✓ 关闭：说明由于以外原因挂起测试的步骤；
  - ✓ 重启：告诉测试人员重启测试；
  - ✓ 终止：测试正常停止的步骤；
  - ✓ 重置：把环境恢复到测试前的状态；
  - ✓ 偶然事件：处理计划之外的情况。

通常不太可能需要按照最细致的程度编写测试用例。诀窍是找出最合适的详细程度。

18.3 测试用例组织和跟踪

管理和跟踪系统的方法：

- ✓ 凭脑子记。绝对不要考虑的方法。
- ✓ 书面文档。
- ✓ 电子表格。
- ✓ 自定义数据库。跟踪测试用例的理想方法是使用测试用例管理工具（test case management tool），一种为处理测试用例而专门编程设计的数据库。

第 19 章 报告发现的问题

如果回头审视软件测试的全貌，就会发现它有 3 个主要任务：测试计划、实际计划、报告发现的问题。

19.1 设法修复软件缺陷

不修复软件缺陷的原因如下：

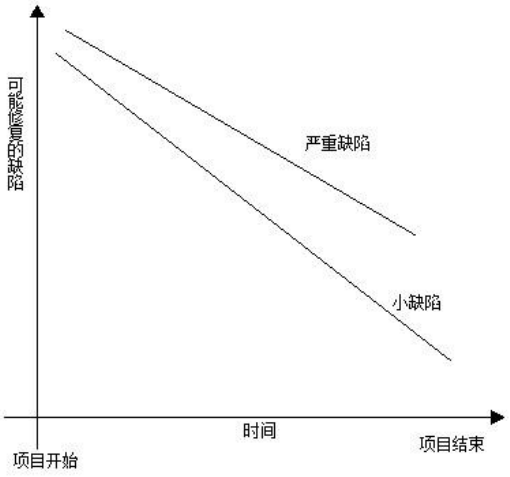
- ✓ 没有足够的时间
- ✓ 不算真正的软件缺陷

- ✓ 修复风险太大。软件本身是脆弱的、难以理清头绪，有点像一团乱麻，修复一个缺陷可能导致更多的缺陷出现，在紧迫的产品发布进度压力下，修改软件将冒很大的风险。
- ✓ 不值得修复。不常出现的软件缺陷和在不常用功能出现的缺陷是可以放过的，可以躲过和用户有办法预防或避免的软件缺陷通常不用修复。这都要归结为上而言风险决策。

决策过程通常由软件测试员、项目经理和程序员共同参与。

报告软件缺陷的基本原则：

- ✓ 尽快报告软件缺陷。修复软件缺陷的风险随着时间的推移大大增加，而且在做决定过程中的分量不断加重。



- ✓ 有效地描述软件缺陷。
  - ✧ 短小。只解释事实和演示、描述软件缺陷必须的细节。简短并抓住要点。
  - ✧ 单一。每个报告只针对一个软件缺陷。在一个报告中报告多个软件缺陷的问题时，一般只有第一个缺陷收到注意和修复——而其他的软件缺陷被忽略或忘记。分别跟踪在同一个报告中的多个软件缺陷也是不可能的。
  - ✧ 明显并通用。容易看懂的、简单易行步骤描述的软件缺陷的一个特例，得到修复的机会较大。
  - ✧ 可再现。
- ✓ 在报告软件却显示时不要做评价。软件缺陷报告应该针对产品，而不是具体的人，指陈述事实。测试员和程序员之间很容易形成对立关系。不要带任何倾向性、个人观点和煽动性的语言。*得体*、*委婉* 是关键。
- ✓ 对软件缺陷报告跟踪到底。

19.2 分离和再现软件缺陷

要想有效报告软件缺陷，就要以明显、通用和可再现的形式描述它。分离和再现软件缺陷是充分发挥才干的地方，设法找出收缩问题的具体步骤。

如果找到的软件缺陷似乎要采用繁杂步骤才能再现，或者根本无法再现，如下一些提示和技巧会打开局面。

- ✓ 不要想当然地接受任何假设。记下所做的每一件事、每一个步骤、每一次停顿、每一件工作。
- ✓ 查找时间以来和竞争条件的问题。

- ✓ 边界条件软件缺陷、内存泄露和数据溢出等白盒问题可能会慢慢自己显露出来。
- ✓ 状态缺陷仅在特定软件状态中显示出来。例子是仅在第一次运行软件时出现或者在第一次运行之后出现。
- ✓ 考虑资源依赖性和内存、网络、硬件共享的相互作用。
- ✓ 不要忽视硬件。

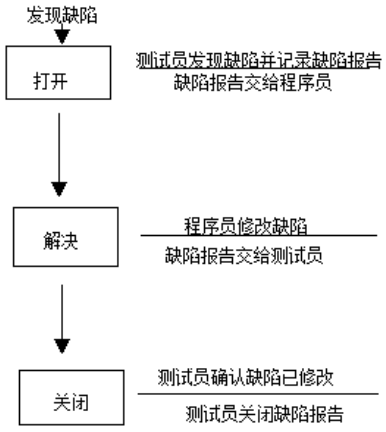
19.3 并非所有软件缺陷生来就是平等的

在报告软件缺陷时，一般要讲明他们将产生什么后果。测试员要对短剑缺陷分类，以简明扼要的方式指出其影响。常用的方法是给软件缺陷划分严重性和优先级。

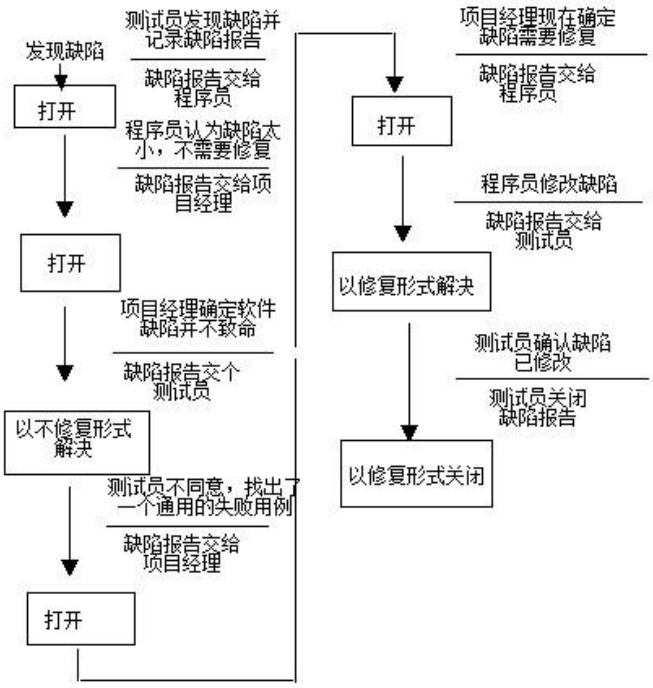
- ✓ **严重性**表示软件缺陷的恶劣程度，当用户碰到该缺陷时影响的可能性和程度。下列是常用的划分等级（当然可以灵活变动）：
  1) 系统崩溃、数据丢失、数据毁坏，安全性被破坏
  2) 操作性错误、结果错误、功能遗漏。
  3) 小问题、拼写错误、UI 布局、罕见故障。
  4) 建议。
- ✓ **优先级**表示修复缺陷的重要程度和紧迫程度。
  1) 立即修复，阻止了进一步测试，立竿见影。
  2) 在产品发布前必须修复。
  3) 如果时间允许应该修复。
  4) 可能会修复，但是即使有产品也能发布。

19.4 软件缺陷的生命周期

一般情况下，软件生命周期如图：



在有些情况下，又变得更复杂一些，如图所示：



以下是一个通用的缺陷生命周期状态图：

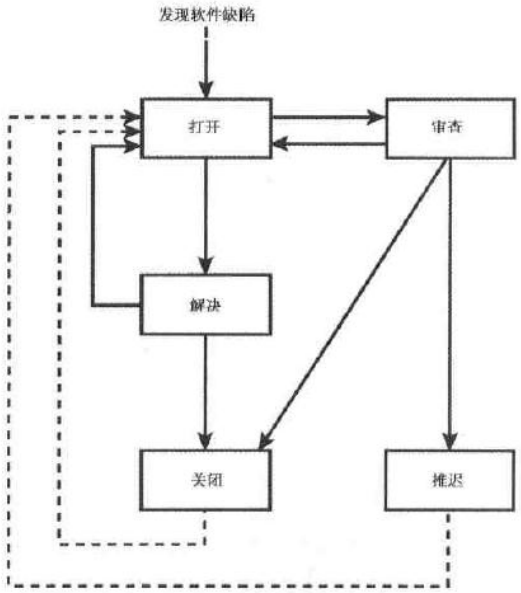


图 18-4　通用软件缺陷生命周期状态表覆盖大多数可能出现的情况

审查状态是指项目经理或者委员会决定是否应该修复。

推迟状态是指审查可能认定软件缺陷应该在某一时间考虑修复，但在软件该版本中不修复。

19.5 软件缺陷跟踪系统

（1）标准：测试事件报告

- ✓ 标识符。
- ✓ 总结。用简明扼要的事实陈述总结软件缺陷。要测试的软件及其版本的引用信息，相关的测试过程、测试用例和测试说明也应该包含在内。
- ✓ 事件描述。

- ✧ 日期和时间
- ✧ 测试人员的姓名
- ✧ 使用的硬件和软件配置
- ✧ 输入
- ✧ 过程步骤
- ✧ 预期结果
- ✧ 实际结果
- ✧ 试图再现以及尝试的描述
- ✧ 有助于程序员定位软件缺陷和其他现象或者信息

✓ 影响。

（2）手工软件缺陷报告和跟踪

（3）自动化软件缺陷报告和跟踪

第 20 章 成效评价

20.1 使用软件缺陷跟踪数据库中的信息

考虑以下问题：

- ✓ 正在测试的软件什么区域软件缺陷最多，哪里最少？
- ✓ 当前交个某个测试员多少个已经解决的软件缺陷？
- ✓ 某个测试员很快要外出度假。他发现的软件缺陷临走前能够全部修复吗？
- ✓ 本周找出了多少个软件缺陷？本月呢？整个项目期间呢？
- ✓ 愿意带一份全是打开的优先级 1 软件缺陷清单到项目审查会议上吗？
- ✓ 软件估计在符合预定发布日期的正常轨道上吗？

使用软件缺陷数据库作为度量的来源是评测项目状态和软件测试员自身进展及其有效的方式。

20.2 在日常生活中使用的度量

软件缺陷跟踪数据库最常用的特性除了输入软件缺陷之外可能是执行查询，获得感兴趣的软件缺陷清单。比如：

- ✓ 交给我来关闭的已解决软件缺陷的 ID 号是什么？
- ✓ 我在该项目中输入了多少软件缺陷？在一周内嗯？上一个月呢？在 4 月 1 日到 7 月 31 日期间呢？
- ✓ 我针对用户界面输入的哪些软件缺陷以不修复的形式解决的？
- ✓ 我发现的软件缺陷中有多少事严重性 1 或者严重性 3 的？
- ✓ 在我输入的全部软件缺陷中，修复了多少？推迟了多少？重复了多少？

20.3 常用项目级额度

从整个项目来看，管理员关注的基本问题——质量和可靠性等级是多少，是否脱离进度就绪的轨道？软件缺陷跟踪数据库是提供该信息的最佳工具。

发现的软件缺陷越多，表明未发现的软件缺陷也越多。

解决的软件缺陷不一定是修复的软件缺陷。某些软件血线可能以重复、不修复或者有意保留等形式解决。

第 21 章 软件质量保证

21.1 质量是免费的

1979 年 philp Croby 在《Quality is Free：The Art of Making Quality Certain》中写道：制造高质量产品比制造低质量产品实际上不需要额外开销（其实开销更小）。

软件缺陷发现得越晚，其处理费用越高——不是线性增长，而是按几何数级激增。

现在把质量的费用分为两类：一致性费用（costs of conformance）和非一致性费用（costs of nonconformance）。

一致性费用是指与一次性计划和执行测试相关的全部费用，用于保障软件安装预期方式运行。如果发现软件缺陷，必须花时间分离、报告和回复测试保证其得意修复，那么非一致性费用就会上涨。因为软件缺陷在发布之前发现，所以这些费用归属于内部失败。

如果软件缺陷被遗落并落到用户手里，结果就是代价高昂的产品支持电话，可能还需要修复、重新测试和发布软件——更糟糕的情况下——产品要召回或者卷入官司。解决这些外部失败的费用属于非一致性费用。

21.2 工作现场的测试盒质量保证

以下定义了一些常用的软件测试团队名称，有助于澄清星湖之间的差别。

- 软件测试

软件测试员的目标是尽早地找出软件缺陷，并确保其得以修复。

作为一个时机测试员，要对找出的软件缺陷负起责任，跟踪其整个生命周期，说服相关人员使其得以修复。

软件测试员不负责软件的质量，指是报告实事。就如，医生量体温不能退烧。气象专家测枫树不能阻止风暴。软件测试员寻找缺陷不能使质量低劣的产品变好。

- 质量保证

对于寻找软件缺陷的团队而言，另一个常用的名称是“全家质量保证（QA）”。

软件质量保证人员的主要职责是检查和评价当前软件开发的过程，找出改进过程的方法，以达到防止软件缺陷出现的目标。

既然单靠软件测试无法保证产品的质量，那么软件 QA 团队如火热实现目标呢？答案是对项目进行近乎完全的控制，建立标准和方法论，有条理地仔细监视和评价软件开发过程，对发现的软件缺陷提出解决建议，执行某些测试（或者忽略），拥有决定产品何时准备发布的授权。

全面质量管理：全面质量管理（TQM）或者全面质量控制（TQC）的内在基本原理是，用集中的质量评判团队来负责质量是不实际的，因为从事工作的人员——编写代码或者制作小工具——并不负责质量，所以他们不会设法实现质量保证的目的。想要制造高质量的产品，需要创立从管理开始自上而下的质量意识，使全体人员共同承担质量责任。

- 软件测试团队的其他名称

测试团队根据用途可能使用多个名称来标识。如：软件质量控制（SQC）最常用、软件验证和合法性检查……

21.3 测试的管理和组织结构

- 小型（小于 10 人）开发小组常用的结构。在该结构中，由测试团队向开发管理员报告，他管理程序员的工作。但是编写代码的人和在代码中寻找软件缺陷的人向同一个人报告，这有可能引起大问题。但是假如开发管理员经验丰富，认识到其目标不仅是制作软件，而且要制作高质量的软件，对测试员与程序员一视同仁，该结构也可以工作的很好。

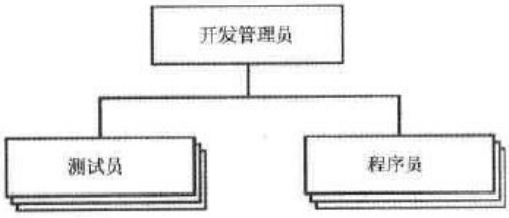


图 20-2　小型项目的组织结构通常让测试小组向开发管理员报告

- 另一种常用结构式，测试团队和开发团队都向项目管理员报告。在该组织方式下，测试团队一般拥有自己的负责人或管理员，其利益和注意力集中在测试小组及其工作上。测试小组的意见和程序员以及其他产于产品制作者的意见是同等重要的。

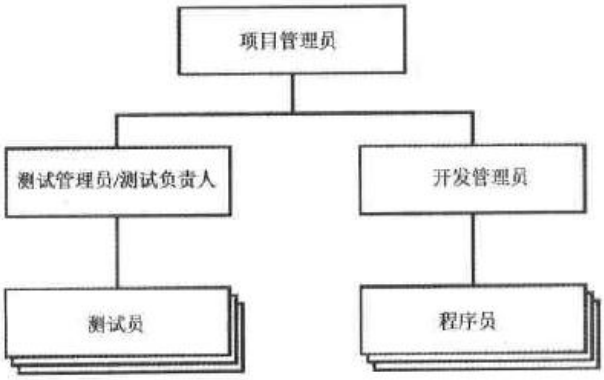


图 20-3　在测试小组向项目管理员报告的组织方式中，测试员对程序员相对独立

- 在该组织方式中负责软件质量的小组直接向高级管理员报告，相对独立，拥有与个项目同等的项目等级。

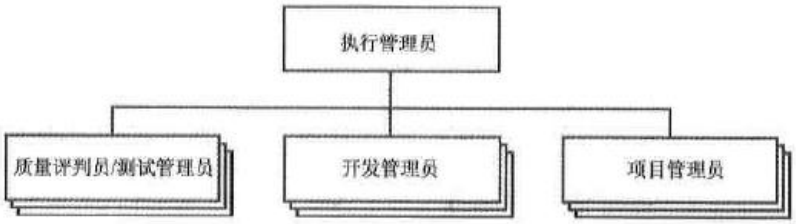


图 20-4　向执行管理员报告的质量评判团队或者测试团队
独立性最强、权限最大、职责最大

21.5 能力成熟度模型（CMM）

软件能力成熟度模型是一个行业标准模型，定义和评价软件公司开发过程的成熟度，提供怎样做才能提高软件质量的指导。

它分为 5 个等级：

- 1 级：初步。处于这个最低级的组织，基本上没有健全的软件工程管理制度。开发软件的时间和费用无法预知。测试过程与其他过程混在一起。
- 2 级：可重复的。该等级称叔父的最好描述是项目级想法。基本项目管理代替了项目费用、进度、功能和进度的跟踪。该等级有一定的组织性，使用了基本软件测试行为。
- 3 级：明确的。该等级具备了组织化思想，而不仅仅是针对具体项目。通用管理和工程活动被标准化和文档化。这些标准在其他项目采用和证实过。当压力增加时，

不会放弃规则。在测试开始之前，审查和证实测试文档和计划。测试团队与开发人员独立。测试结果用于确定软件就绪的诗句。

- 4 级：可控的。在该成熟度等级中，组织过程处于系统控制之下。产品质量实现由数量指定，软件在达到目标之前不得发布。在整个项目开发过程中收集开发过程和软件质量的详细情况，经过调整校正片场，使项目按计划进行。
- 5 级：不断优化的。该等级称为不断优化是因为他从 4 级不断提高。尝试次新的技术和处理过程，评价结果，不断增长和变革以期达到质量更佳的等级。正当所有人认为已经达到最佳时，新的想法又出现了，在次提高到下一个等级。

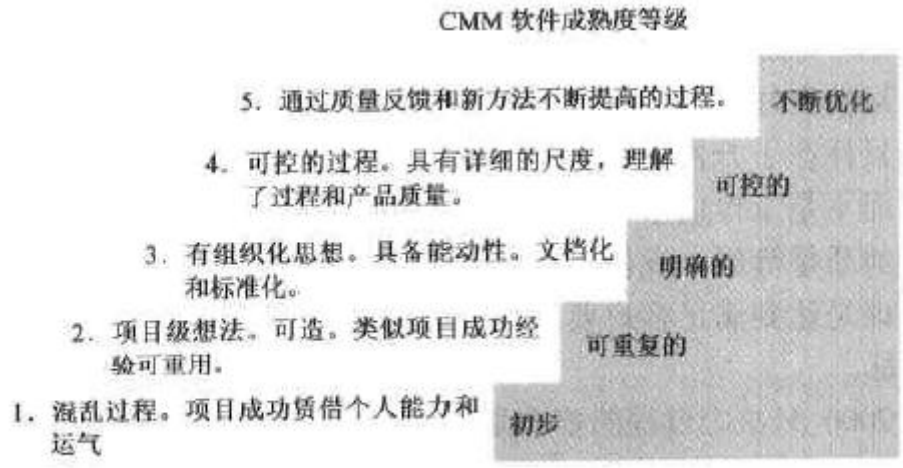


图 20-5　软件能力成熟度模型用于评测软件公司在软件开发方面的成熟度

21.5 ISO 9000

ISO 9000 是关于质量管理和质量评判的一系列标准，定义了一套基本达标的做法，有助于公司一致地交付符合客户质量要求的产品（或者服务）。

- ✓ 他的目标在于开发过程，而不是产品。
- ✓ 他决定过程的要求是什么，而不管是如何到达。

第 22 章 软件测试员的职业

22.1 软件测试员的工作

- 软件测试技术人员。这一般是真正的入门级测试职位。工性质是负责建立测试硬件和软件配置，执行简单的测试脚本或者测试自动化，可能还要利用 Bate 站点分离和再现软件缺陷。虽然某些工作可能变得多余和重复，但是称为测试技术人员是步入软件测试殿堂的号方法。
- 软件测试员或者软件测试工程师。大多数公司拥有不同经验和专长的软件测试员等级。入门级测试员可以履行技术职责，进而执行更高级和复杂的测试。在提高的过程中，可以编写自己的测试案例和测试程序，并参与设计和说明书审查。入门级测试员将进行测试，并分离、再现和报告发现的软件缺陷。若果有编程能力，可以编写测试自动化或者测试工具，在实施白盒测试时与程序员密切合作。
- 软件测试负责人。测试负责人负责软件项目主要部分的测试，有时负责整个小型项目的测试。他们通常为负责范围制定测试计划，监督其他测试员实施测试。他们常常重点手机产品的度量，向管理部门呈报。他们一般不履行软件测试员的职责。
- 软件测试管理员。测试管理员监督整个项目甚至多个项目的测试。测试负责人要向他们报告。他们和项目管理员、开发管理员一起设定进度、优先级和目标。他们负

责为项目提供合适的测试资源——人员、设备、场地等。他们为小组测试制定格调和策略。

22.2 寻求软件测试职位

22.3 获得亲身体验

选择一个熟悉的程序，并体验使用的乐趣，或者选择一个从未用过的程序。把用户手册和帮助文档当做产品说明书来阅读。整理一份测试计划，设计测试案例寻找软件缺陷。使用电子表格或者文字处理处理程序等级软件缺陷，向编写应用程序的软件公司报告发现的问题。

有了一些测试经验之后就可以注册申请成为新产品的 Bate 测试员了。

22.4 正规培训机会

22.5 网站

下列专门针对软件测试和软件缺陷的流行网址可以作为入门指导：

- ✓ Bug Net（www.bugnet.com）公布在商业软件中发现的软件缺陷，并指出相应的修复措施。
- ✓ Software Testing Hotlist（www.io.com/~wazmo/qa）列出了一些与软件测试相关的网址和文章。
- ✓ Software Testing Online Resources（www.mtsu.edu/~storm）自称“一系列软件测试联机资源……旨在成为软件测试研究者和从业者的网上第一站”
- ✓ QA Forums（www.qaforums.com）提供软件测试、自动化测试、测试管理、测试工具等主题的即使讨论。
- ✓ Comp.software.testing 新闻组及其 www.faqs.org/faqs/software-eng/testing-faq 上的 FAQ（常见问题）文档，提供测试员和测试管理员关于工具、技术和项目的即使讨论。
- ✓ Comp.risks 新闻组描述和分析近期软件失败。
- ✓ Risk Digest（catless.ncl.ac.uk/Risks）是关于计算机及其相关系统对公众风险的论坛。

22.6 关注与软件和软件质量的专业组织

- ✓ 美国质量委员会（ASQ）在 www.asq.org 及其软件分会在 www.asq-software.org 每年 10 月份国际质量月主办国际质量论坛。他们发行质量方面的刊物和文章，并管理质量认证工程师（CQE）和软件质量认证工程师（CSQE）的任命。
- ✓ 美国计算机协会（ACM）
- ✓ 美国质量委员会（ASQ）
- ✓ 软件质量委员会（SSQ）

22.7 更进一步阅读

- 《Sams Teach Yourself Beginning Programming in 24 Hours》是编程基础的优秀入门书。虽然读了本书不能成为白盒子测试员，但是从中可以洞悉软件是如何编写的——有助于更好地设计测试案例。
- 《Sams Teach Yourself HTML in 24 Hours》、《Sams Teach your self Visual Basic in 24 Hours》、《Sams Teach Yourself Java in 24 Hours》和《Sams Teach Yourself C++ in 24 Hours》是掌握编程之后很好的下一步。

- 如果目标是成为严肃的白盒子测试员，《Sams Teach Yourself Visual Basic in 21 Days》、《Sams Teach Yourself Java in 21 Days》、《Sams Teach Yourself C in 21 Days》和《Sams Teach Yourself C++ in 21 Days》将讲授具体语言编程的细节。
- 《Sams Teach Yourself Upgrading and Fixing PCs in 24 Hours》将讲授为 PC 机添加新硬件和外设的基础——对于软件测试员是一个重要课题，特别是对配置测试感兴趣时。
- 《Internationalization with Visual Basic》