SWTBOK®

软件测试知识体系

V1.0

SWTBOK®专家委员会

版权标志

版权标志© SoftWare Testing Body Of Knowledge（以下简称为SWTBOK®）。

SWTBOK®是软件测试知识体系（SoftWare Testing Body Of Knowledge）的注册商标。

版权所有

作者（不管是当前的还是将来的版权作者）同意将该软件测试知识体系授权给SWTBOK®。本软件测试知识体系作者和SWTBOK®一致同意下面的使用条款：

1） 软件测试知识体系作者和SWTBOK®是公认的原始发起者和版权拥有者。

2） 在声明承认软件测试知识体系作者和SWTBOK®作为该知识体系的原始发起者和版权拥有者的前提之下，任何个人或者团体都可以使用本软件测试知识体系的内容作为文章、书籍，或者其他资料的参考文献或者主要理论依据。

3） 在声明承认软件测试知识体系作者和SWTBOK®作为该知识体系的原始发起者和版权拥有者的前提之下，任何个人或者团体都可以使用本软件测试知识体系作为培训课程的基础。

修订历史

版本	发布日期	备注
软件测试知识体系SWTBOK® V1.0	2011-09-01	SWTBOK®专家委员会发布的第一个版本

SWTBOK®软件测试知识体系

众所周知，软件测试面临各种各样的问题。从 2008 年开始，SWTBOK®专家委员会专家针对测试过程中存在的问题进行了广泛而深入的调查和研究，最终将这些原始问题总结归类到 13 个软件测试知识域。图 1 是测试问题在不同软件测试知识域中的比例分布图。

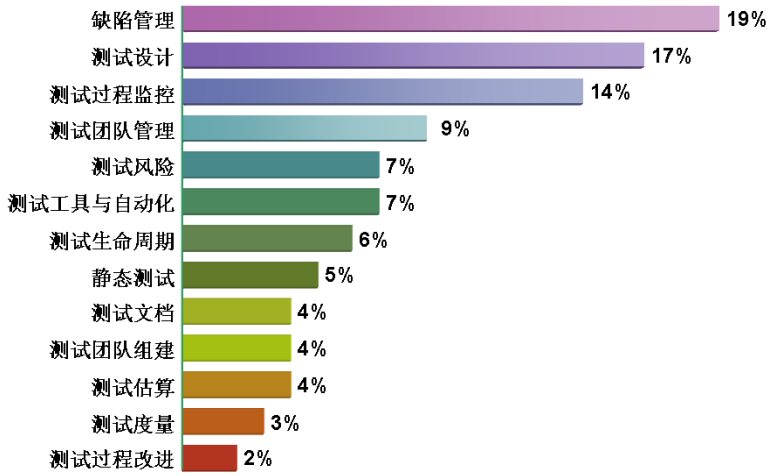

图 1 软件测试问题分布图

软件测试过程中面临的问题，需要测试人员以系统化、专业化的方式进行分析和解决。同时，软件测试作为软件工程的重要组成部分，也需要有一个知识体系与之相对应。这就是 SWTBOK®软件测试知识体系形成的初衷和目的。

SWTBOK®软件测试知识体系分别从测试过程、测试技术与方法、测试团队 3 个方向阐述了在软件测试领域被广泛接受的知识域。即该知识体系包含的知识和实践可以应用于绝大部分的项目测试中，并对其中的价值和作用达成广泛的共识。SWTBOK®软件测试知识体系提倡“好的实践”而非“最佳实践”，因此实践过程中需要对 SWTBOK®软件测试知识体系的知识域进行合理的裁剪，而非照搬应用。图 2 是 SWTBOK®软件测试知识体系的简单示意图。

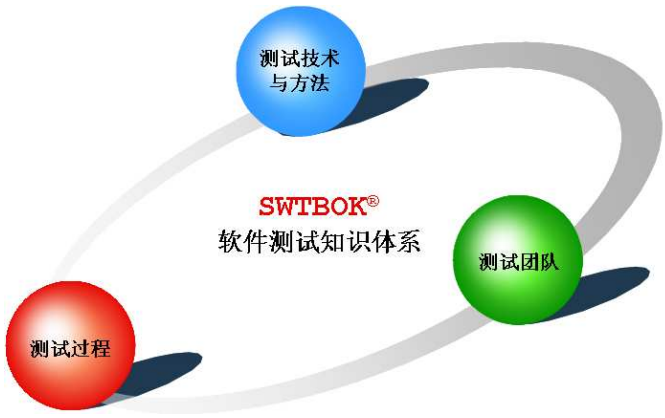

图 2 SWTBOK®软件测试知识体系

SWTBOK®软件测试知识体系由独立的 SWTBOK®专家委员会负责编写与评审，并根据软件测试发展与行业的变化进行定期的更新。SWTBOK®软件测试知识体系的主要特征包括：

- 专业性：SWTBOK®软件测试知识体系中涉及的知识域与测试过程紧密相关，从专业软件测试的角度进行提炼和归纳总结；

- 开放性：SWTBOK®软件测试知识体系对所有人完全开放。任何人都可以自由的获得本知识体系的所有内容；任何有兴趣的人都可以加入 SWTBOK®，并参与编写和评审 SWTBOK®软件测试知识体系的各个测试知识域；
- 实践性：SWTBOK®软件测试知识体系涉及的测试知识域可以应用于大部分的项目测试中，并以实际测试中存在的问题为出发点，重点体现了测试知识域的实用性；
- 独立性：SWTBOK®软件测试知识体系不涉及任何商业产品的介绍和推广，以开放和共享的方式同软件测试从业人员进行交流和沟通；

图 3 是 SWTBOK®软件测试知识体系的 13 个测试知识域。

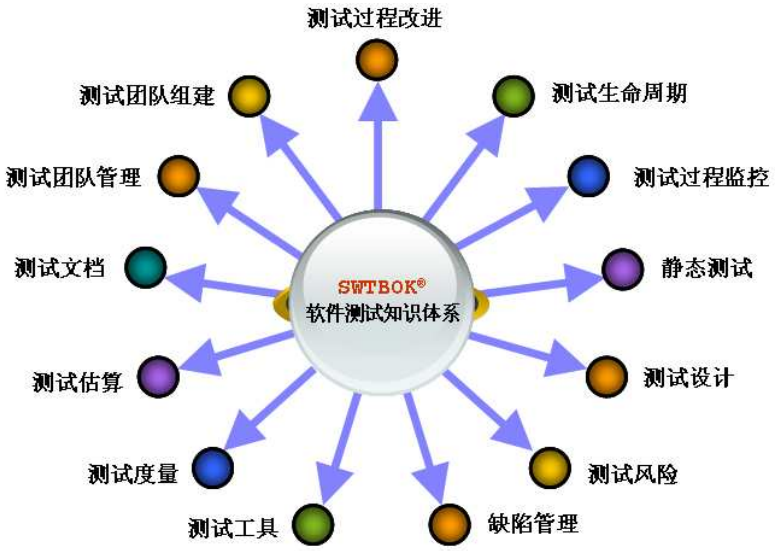

图 3 SWTBOK®软件测试知识体系的测试知识域

第 1 部分 SWTBOK®软件测试工程师

第1章 软件测试生命周期

1.1 软件测试基础

1.1.1 为什么需要软件测试

随着信息技术的不断发展，软件系统在人们日常生活中的重要性越来越大，并在各个领域对人们的生活产生影响，例如：从网上购物系统到卫星通信系统。然而，很多人都有这样的经历：软件系统工作的状态或者结果和人们的期望结果不同，从而导致许多问题，包括资金、时间、机会和商业信誉等的损失，甚至出现人员的伤亡。

人都会犯错误，因此通过人生成的各种软件工作产品（例如：文档、代码等）都会引入缺陷，并降低软件产品质量。测试除了可以发现缺陷之外，其主要目的还包括：增强对软件产品质量的信心、为决策者提供信息，以及预防缺陷。

1.1.2 什么是软件测试

软件测试是一个完整的生命周期，它由测试计划、测试设计、测试执行、测试监控和测试结束 5 个阶段组成。不同的测试对象和测试阶段，需要采用不同的静态测试和动态测试技术和方法，以满足不同的测试目的。

测试和调试是两个非常容易混淆的概念，但它们在目的、参与人员、主要活动等方面存在明显区别，同时两者在整个测试生命周期内是相辅相成的。

1.2 软件开发模型

测试不是孤立存在的，它与开发活动息息相关，不同的软件开发模型需要有不同的测试阶段、测试活动和测试方法与之相对应。

1.2.1 开发模型

- 瀑布模型：瀑布模型将软件开发生命周期划分为系统需求、软件需求、分析、程序设计、编码、测试和运行 7 个基本阶段，并且规定了它们自上而下、相互衔接的固定次序。瀑布模型将测试看作是开发过程中的一个阶段，是对软件产品的最终检查。
- V 模型：V 模型是瀑布模型的变种，它体现的思想是开发任务和测试任务是相互对等且同样重要，V 模型的左侧是开发活动，而右侧描述了测试级别。典型的 V 模型有 4 个不同的测试级别与开发活动相对应，分别是单元测试、集成测试、系统测试和验收测试。根据项目和软件产品的不同，实际采用的 V 模型测试级别也可能多于或者少于 4 个。
- RUP：RUP 是由 IBM 开发和维护的过程产品。RUP 的开发团队与顾客、合作伙伴、Rational 产品小组及顾问公司共同协作，确保开发过程持续地更新和提高，以反映新的和不断演化的实践经验。

1.2.2 模型中的测试

尽管不同的开发模型中定义了不同的测试阶段、测试活动和测试方法，但是一个良好的测试应该具备下面的特点：

- 每个开发活动都有测试活动相对应；
- 每个测试级别都有其特定的测试目的；
- 与开发活动对应的每个测试级别，都应该进行相应的测试设计和执行；
- 测试人员应尽早介入测试；
- 通过风险分析和评估确定测试的重点和优先级，以替代穷尽测试；

1.3 软件测试过程

广义的软件测试是一个完整的测试生命周期，软件测试过程主要由测试计划、测试设计、测试执行、测试结束和测试监控 5 个阶段组成。这 5 个测试阶段在整个测试过程中会有重叠，甚至同时进行。根据不同项目和组织的特点，可以合理地裁剪测试阶段中的活动以满足不同的要求。

1.3.1 测试计划阶段

该阶段的重要输出是测试计划文档。测试计划阶段的主要任务包括：

- 定义测试目标；
- 识别测试范围；
- 选择测试策略和测试方法；
- 定义测试入口准则和出口准则；
- 确定人员配备和时间进度计划；
- 定义测试工作产品；
- 测试风险和意外事件的管理。

1.3.2 测试设计阶段

测试设计是将概要的测试目标和测试依据文档中的测试条目转换为具体的测试用例的一系列的测试活动。测试设计阶段的主要任务包括：

- 参与测试依据文档的学习和评审（例如：系统需求文档、系统架构、设计文档等）；
- 识别测试依据文档中的测试条目；
- 详细设计测试用例，确定测试用例所需的测试数据，并确定测试用例和测试依据之间的可追溯性；
- 确定测试用例的优先级；
- 规划测试环境，确定测试执行所需的基础设施和工具；

- 确定自动化测试的方案，编写自动化测试脚本；

1.3.3 测试执行阶段

测试执行是在搭建的测试环境中运行测试用例的一系列活动。测试执行阶段的主要任务包括：

- 根据测试用例的优先级和执行顺序，通过手工或者自动化的方式执行测试用例；
- 记录测试执行的结果和测试环境标识，例如：测试对象的版本、测试平台类型等；
- 比较测试执行的实际结果和测试用例中的期望结果。假如两者之间有差异，分析引起差异的原因，例如：需求缺陷、代码缺陷、测试数据问题等。假如差异是由于测试对象中存在缺陷引起的，提交缺陷报告。
- 测试对象由于缺陷修改等原因出现变更，需要进行再测试和回归测试，以确认缺陷是否已经正确修改，并确认修改没有在测试对象原来工作正常的部分引入新的缺陷。

1.3.4 测试监控阶段

测试监控贯穿于整个软件测试生命周期，是持续进行的活动。测试监控的主要任务包括：

- 记录测试过程中实际的测试设计进度、测试执行进度、测试覆盖率、测试风险等状态和结果；
- 分析测试过程中记录的状态和结果，并与测试计划进行比较。假如两者存在偏离，采取合适的措施和应对计划使之回到测试计划的轨道，例如：调整测试重点和优先级；
- 根据测试过程中得到的状态和结果，在必要的时候也可以修正测试计划以满足当前的测试状态和结果，例如：变更测试时间进度和测试资源分配；
- 根据测试计划中定义的出口准则检查测试执行的状态和结果，以评估是否可以结束测试执行任务。

1.3.5 测试结束阶段

测试结束就是从已完成的测试活动中收集和分析有用的信息的一系列活动，这些信息包括测试工作产品、测试经验和教训等。测试结束阶段的主要任务包括：

- 为利益相关者提供一份测试总结报告；
- 检查测试计划中定义的测试工作产品是否已经提交并归档；
- 测试团队提交的缺陷报告是否已经关闭，假如没有关闭，是否做了合适的处理；
- 整理测试环境和测试基础设施以备将来项目的重复使用；
- 移交测试工作产品到产品的维护部门或者技术支持部门；
- 分析和记录测试过程中的经验教训；
- 收集和分析测试过程中的度量数据，分析过程中出现的问题以查找其中的根本原因，并将结果用于将来的过程改进。

1.4 软件测试级别

1.4.1 单元测试

单元测试是对编码阶段实现的软件单元进行的测试。根据不同的编程语言，单元可以是模块、函数、程序或者是功能。单元测试的主要目的是验证单元是否按照详细规格说明正确的工作，发现设计和需求中存在的缺陷，以及编码过程中引入的缺陷。

1.4.2 集成测试

集成测试是对单元之间的接口进行测试，或者检查与系统其他部分相互作用的测试，如操作系统、文件系统等。集成测试的主要目的是把经过测试的不同单元逐步集成在一起，以检查数据和信号等能否在不同单元之间正确传递和调用，以及它们是否能正确地协同工作。

1.4.3 系统测试

系统测试是将已经集成好的软件系统，在尽量模拟用户使用环境的测试环境下对被测系统进行的测试。它关注的是产品范围中定义的整个系统或者产品的行为。系统测试的主要目的是确认系统或者产品是否满足了需求规格说明中的功能和非功能需求，以及满足的程度。

1.4.4 验收测试

验收测试通常是由使用系统或者产品的用户来进行，其他利益相关者也可以参与其中。验收测试的主要目的是确认开发合同、法律法规或者规范中定义的服务是否已经满足，例如：根据合同的验收测试，发现缺陷不是它的主要目的。

第2章 测试文档

在整个软件测试生命周期中，测试团队需要输出各种不同的文档。每种类型的文档都需要包括相对应的内容。但是每个文档具体包括哪些内容，都应该根据组织和项目的特点进行合理的裁剪。

2.1 测试计划文档

2.1.1 测试对象

该部分描述了被测系统或者产品的商业目的，通常可以从系统文档中找到这方面的信息，例如：系统可行性研究报告。同时，该部分还简单描述了被测系统或者产品的系统架构和主要功能。

2.1.2 测试范围

该部分主要描述了测试的主要目的、测试范围和测试交付物。也可以包括本次的测试假定条件和限制条件。

- 测试目的：清楚定义本次测试的主要目的是什么？以发现缺陷为主，还是以增强客户对软件产品质量的信心为主？
- 测试范围：确定本次测试需要覆盖哪些功能，没有覆盖的功能等。有时候，也可以通过不同的测试类型确定测试范围，例如：覆盖哪些功能特性、非功能特性等；
- 测试交付物：确定在该测试过程中的测试工作产品输出，例如：测试计划文档、测试规格说明文档、测试报告文档等；

参考资料既可以作为测试范围的一部分进行描述，也可以放在测试计划文档的最后。参考资料可以分为外部参考资料和内部参考资料。

2.1.3 测试方法

该部分主要描述了采用的测试过程，以及与开发过程之间的关系。根据项目的项目可以对采用的测试过程和标准进行适当的裁剪。主要包括：

- 测试技术与方法：描述本次测试采用的主要测试技术与方法。根据组织和项目的特点，在不同的技术与方法之间进行选择，例如：基于规格说明的测试、基于经验的测试等；
- 测试工具和自动化：描述在测试过程中需要使用的测试工具，也可以描述用来识别和获取测试数据的工具。同时，对测试自动化的策略进行了描述；
- 判断准则：定义了测试的入口准则、出口准则、退出准则和恢复准则，用于评价测试活动结果的准则；

2.1.4 测试管理

该部分主要描述了如何获取制订测试计划的输入，以及测试过程中如何有效监控测试对象的质量和过程状态，并在需要的时候采取合适的应对措施。主要包括：

- 测试时间进度：描述了针对不同测试任务的时间进度。为每个测试任务开始和结束、检查输入和提交输出建立特定的里程碑点；

- 测试资源：识别开展测试任务所需的资源，包括人力资源、工具、设备和培训等，以及其他特殊的需求，例如：安全性需求、接入权限和文档控制等；
- 角色和职责：识别针对每个测试任务需要参与的人员以及各自的职责；
- 测试风险：识别和测试相关的管理风险和质量风险，并提供建议和应对措施以消除和降低风险；

2.2 测试规格说明文档

2.2.1 测试设计文档

测试设计文档主要确定了测试对象的测试点，并且以可以转换为测试用例的刻度进行提炼。测试设计文档主要包括：测试点标识、测试点标题、测试点目的、需求文档和需求编号、测试优先级、测试设计者、测试类型，以及手工测试/自动化测试等内容。

2.2.2 测试用例文档

测试用例文档针对测试设计文档中的测试点，详细定义了测试前置条件、测试输入、具体步骤、期望结果和后置条件等内容，同时它也包含了测试设计文档中包含的内容。

2.3 测试报告文档

2.3.1 缺陷报告文档

缺陷报告文档详细记录了测试过程中出现的缺陷。缺陷报告文档主要包括：缺陷标题、缺陷的简单描述、发现缺陷的环境配置、发现缺陷的测试阶段、缺陷的严重程度、缺陷的优先级、缺陷的提交者、缺陷复现的具体步骤以及缺陷产生时的其他的调试信息或者错误信息等。

2.3.2 测试总结报告文档

测试总结报告文档描述了测试活动的结果并根据这些结果对测试对象的质量和存在的风险进行评估。该文档主要包括：测试覆盖的功能、没有覆盖的功能、还存在的缺陷列表、对客户可能存在的影响、可能存在的风险，以及对测试对象质量的总体评价等。

2.4 测试文档实践

2.4.1 测试文档的详细程度

测试文档的详细程度受到被测试产品的特征、组织的能力、软件开发生命周期模型和团队成员的构成等多方面因素的影响。测试文档的详细程度可以分为：

- 没有文档：所有的内容都存在测试相关人员的脑袋里，信息通过口头的方式在不同人员之间传递；
- 简单的文档：文档的内容主要为作者自己所使用，不能用于评审或指导其他人进行下一步的行动；

- 较为详细的文档：文档作者详细的记录相关内容，可以用于评审，并指导团队内部人员进行下一步的工作；
- 完整的文档：文档作者完整的记录了需要的内容，该文档可以用于评审，指导团队外部人员进行下一步的工作。

2.4.2 好的缺陷报告应该具备的特点

测试人员在提交缺陷报告的时候，应该从不同的方面保证缺陷报告的质量，一个好的缺陷报告应该具有下列特征：精简的、正确的、中立的、准确的、隔离、推广、复现、证据充分和经过评审。

第3章 静态测试

与动态测试不同，静态测试是以评审（人工检查）或者静态分析（自动化分析）的方式检查代码或者其他软件工作产品，而不需要运行代码。

3.1 评审

软件工作产品可以是开发生命周期中的各种文档，如需求规格说明、测试用例规格说明等，也包括代码。

3.1.1 评审基本原则

理解和掌握评审的基本原则有利于有效开展评审活动，避免评审过程中的一些误区。评审基本原则主要包括：

- 尽早开展评审活动；
- 控制评审会议的持续时间；
- 评审对象是软件工作产品而不是作者；
- 每个评审员都有机会充分表达他们的观点；
- 评审的目的是发现缺陷而非修复缺陷；
- 评审中发现的缺陷和问题应划分不同的严重程度等级；
- 评审团队应该为评审对象给出最后的评审意见；

3.1.2 评审基本过程

典型的评审基本过程主要包括以下几个阶段：

- 计划阶段：选择合适的评审员组建评审团队；选择需要评审的文档或者章节，并为正式的评审类型定义入口和出口准则；确定评审的具体时间和地点；确定评审员需要的准备时间，以及评审对象和输入资料等的分发。
- 预备会阶段：该阶段是对计划阶段的有效补充，假如相关信息在计划阶段没有阐述清楚。其主要活动包括评审主持人简单介绍评审对象，评审的目标和过程，并核对入口准则；评审主持人解答评审员的问题和疑问。
- 个人准备阶段：该阶段评审员仔细阅读和检查评审对象，根据各自的职责要求和评审目的，尽量发现评审对象中的缺陷和问题、检查当前状态是否满足计划要求，或者是否满足定义的过程和规范等。
- 评审会议阶段：该阶段的重点是检查评审对象，发现和讨论其中的缺陷和问题（包括个人准备阶段发现的缺陷列表），记录员详细记录会议内容形成会议纪要；给出评审对象的最后评审结果。

- 返工阶段：返工阶段是作者根据评审员反馈的缺陷列表和建议对文档进行修改。假如评审的最后结果是不接受，那么作者除了进行修改之外，还需要为重新评审做好准备。
- 跟踪结果阶段：检查返工阶段的输出，并确保作者或者其他指定人正确完成了修改任务，正式的评审类型还需要核对出口准则；收集和分析评审过程中的数据和信息，以评估评审对象的质量、评审过程的有效性，以及评审本身的效率和有效性等。

3.1.3 角色和职责

评审过程中需要不同角色的参与，他们在其中承担不同的职责。涉及的主要角色和职责包括：

- 经理：决定是否需要进行评审，在项目计划中分派时间，判断是否达到评审的目标。
- 主持人：主持文档或文档集的评审活动，包括策划评审、召开会议和会议之后的跟踪。假如需要，主持人可能还需要协调评审员之间的不同观点。主持人通常是评审成功的关键。
- 作者：待评审文档的作者或主要责任人。
- 评审员：具有专门技术或业务背景的人员，他们在必要的准备后，标识和描述评审对象中的问题和缺陷。所选择的评审员应该在评审过程中代表不同的观点和角色，并且应该参与各种评审会议。
- 记录员：记录所有的事件、问题，以及在会议过程中识别的未解决的问题。

3.1.4 评审类型

评审类型选择受到开发过程成熟度、法律法规，或者审核跟踪等方面的影响。同时评审的不同目的也是影响评审类型选择的一个重要因素，例如：发现缺陷、增加理解，或者共识的讨论和决定等。根据特点和目的不同，将评审分为技术评审和管理评审 2 种类型：

- 技术评审：技术评审定位在软件工作产品的技术层面，通过同行间的小组讨论活动对所采用的技术实现方法达成共识，并识别其中的缺陷以及开展过程改进。主要包括：非正式评审、走查和审查。
- 管理评审：管理评审定位在软件工作产品或者过程的管理层面，由管理层或其代表执行的对软件采购、供应、开发、运作或维护过程的系统化评估，包括监控过程、判断计划和进度表的状态、确定需求及其系统资源分配，或评估管理方式的效用，以达到正常运作的目的。

3.2 静态分析

静态分析通常是需要工具支持的，其发现的典型缺陷有：违背语法规则、违背编程规范和标准、控制流异常和数据流异常等。

3.2.1 控制流分析

控制流指的是单元或者系统中的一系列顺序发生的事件或路径，测试对象的控制流一般通过控制流图进行直观的表示。控制流分析就是基于控制流图中的事件或者路径开展的。控制流分析可以提供测试对象的逻辑判定点和其结构复杂度的信息。

3.2.2 数据流分析

数据流指的是数据对象的顺序的和可能的状态变换的抽象表示，对象的状态可以是创建（Creation/Defined）、使用（Usage/Used）和销毁（Destruction/Killed）。数据流分析是一种基于变量定义和使用的静态分析模式，它在检测由于编码错误导致的变量赋值错误方面是一种非常有用的技术。

3.2.3 编码标准一致性检查

编码标准是软件人员编写代码的指导性规范。在静态分析过程中可以根据遵循的编码标准对测试对象进行一致性评估。编码标准包括架构方面的标准和编程结构的标准。规范的编码标准有助于软件的维护和测试，同时特定的编程语言要求也能通过静态分析的编码标准一致性进行检查。

3.2.4 生成代码度量

静态分析工具可以提供各种代码度量数据，静态测试过程中产生的这些代码度量，有助于提高代码的可维护性或可靠性。常见的代码度量包括：圈复杂度、规模、注释率、嵌套的层数、函数调用数等。

3.3 静态测试实践

3.3.1 提高评审有效性

- 技术因素
  - 保证遵循评审的基本过程，特别是针对审查这样的正式评审类型；
  - 尽早参与软件工作产品的评审，以提前识别其中的缺陷，防止缺陷的雪崩效应；
  - 合理定义评审的入口准则以提高评审的效率和有效性；
  - 针对不同的软件工作产品选择不同的评审类型，并应用不同的检查列表；
  - 鼓励发现最重要的缺陷，注重内容而非形式；
  - 持续改进评审过程；
- 组织因素
  - 管理层即使在时间和成本压力下也能大力支持评审活动；
  - 管理层给予足够的时间修改评审过程中发现的缺陷；
  - 不要将评审过程中收集的度量数据用于个人绩效评估；
  - 为评审员提供评审技能方面的培训，特别是审查这样的评审类型；
  - 选择最重要的软件工作产品进行评审；
  - 管理层认可评审过程中所取得的改进；
  - 组织内营造“无责备”的氛围，从而乐于接受识别的缺陷；
- 人员因素

- 确保能有合适的评审员参与不同类型的评审，且在技术和行业背景知识方面达到良好的平衡；
- 评审可以发现缺陷并改进软件工作产品的质量，且在利益相关者之间达成共识；
- 确保评审对每个参与人员而言都是一次正面的积极的经历；
- 确保评审员的评审意见具有建设性、有益性和客观性；
- 鼓励评审员深层次思考评审对象中最重要的内容；
- 在作者不同意或者不愿意的情况下，不进行评审；

3.3.2 评审的优点

- 提高质量：类似于动态测试，发现缺陷也是评审的最主要目的之一。尽早发现软件工作产品中的缺陷，通过修复缺陷直接提高软件产品的质量；同时尽早发现和修复缺陷，也可以减少将缺陷带入到下个阶段的机会，间接地提高质量。
- 降低成本：缺陷发现和修复的成本随着开发阶段的演进而上升，因此尽早发现和修复缺陷可以直接降低成本；同时减少缺陷的雪崩效应也可以间接地降低成本；
- 加快进度：在开发阶段的后期发现缺陷，不仅发现缺陷的难度增加，其发现的效率也将降低；同时对开发团队而言，其定位和修复缺陷的难度也将增加，从而需要花费更多的时间，导致时间进度的延后。
- 提升能力：参与评审活动，对于每个评审员而言都相当于参加了一次培训，有助于在将来的项目中输出质量更高的工作产品。通过评审员之间的分析和讨论，除了项目技术相关的知识和技能，大家还在评审过程、规则和实践等方面实现了共享。

第4章 测试设计

4.1 经典测试设计

4.1.1 白盒测试

白盒测试又称为基于结构的测试，是基于测试对象的代码、数据，或者系统架构而进行测试的一种技术。它关注的是测试对象的内部结构，且通过已有的测试用例可以测量测试对象的测试覆盖率。基于结构的测试主要包括：

- 语句测试：语句测试（Statement testing），指的是设计若干测试用例来执行程序代码中的语句。语句覆盖指的是被执行的语句数与所有可能的语句数之间的比值。
- 判定测试：判定测试（Decision testing）是一种针对判定结果设计测试用例的技术。判定覆盖指的是执行测试套件能够覆盖的判定结果的百分比，即被执行的判定和总的判定的比值。100%的判定覆盖可以保证 100%的语句覆盖。
- 条件测试：条件测试（Condition testing）指的是设计若干测试用例来执行不同的条件结果。条件覆盖指的是执行测试套件能够覆盖到原子条件的百分比，即被执行的原子条件和总的原子条件的比值。100％的条件覆盖要求测试覆盖到每一个原子条件语句分别取值为真和假的情况。需要注意的是，条件覆盖并不比判定覆盖更强。
- 判定条件测试：判定条件测试（Decision condition testing）指的是设计若干测试用例来执行条件结果和判定结果。判定条件覆盖指的是执行测试用例套件能够覆盖的条件结果和判定结果的百分比。100%的判定条件覆盖意味着 100%的判定覆盖和 100%的条件覆盖。
- 条件组合测试：条件组合测试（Multiple condition testing，也称为 Condition combination testing）是指设计测试用例覆盖每条语句中的原子条件所有可能的取值结果组合（即每个判定中的所有可能的原子条件取值组合至少执行一次）。条件组合覆盖指的是测试套件覆盖每条语句内的所有原子条件取值结果组合的百分比。100%条件组合覆盖意味着 100%判定条件覆盖。

4.1.2 黑盒测试

黑盒测试又称为基于规格说明的测试，是通过分析组件或系统的测试依据文档，而不是它们的内部结构，获取和选择测试用例的一种方法。黑盒测试从外部来观察测试对象的行为（其观察点也在测试对象的外部），同时控制点也处于测试对象的外部。

- 等价类划分：等价类划分测试技术可以用来减少测试用例数目，同时保证合理的测试覆盖率。
- 边界值分析：边界值分析是一种对软件或者软件系统的边界值进行测试的技术，它是等价类划分技术的有效补充。边界值分析基于这样的测试经验：程序中大量的错误出现在等价类的边界上，而不是发生在等价类内部。

- 决策表测试：决策表是分析和表达多逻辑条件下执行不同操作的表格，它能够将复杂的问题按照各种可能的情况全部列举出来，以避免测试需求的遗漏。决策表测试技术特别适用于下面的使用场景：针对不同的逻辑条件的组合，测试对象需要执行不同的操作。
- 状态转换测试：状态转换测试通常是基于状态转换图进行的，状态转换图描述了测试对象和测试数据之间的关系。测试对象的输出和行为方式不仅和当前的输入数据有关，并且与测试对象当前的状态有关。
- 结对测试：在测试过程中，测试人员经常需要将测试对象的各种输入参数进行组合之后进行测试。有时候，将各种输入参数进行组合，得到的测试用例数目将是非常庞大的。由于测试时间和成本的限制，无法对测试对象输入值的所有组合进行测试。为了更好的减少测试用例的数目而确保测试覆盖率的要求，采用结对测试（Pair-wise Testing）将是一个比较好的选择，它可以显著地减少需要生成和执行的测试用例。结对测试设计的测试用例，可以覆盖测试对象每个参数的两两组合。

4.2 基于质量特性的测试设计

质量是带有内在主观性的：对于同一个产品，不同利益相关者对质量的理解是不一样的。测试人员在测试过程中需要站在不同利益相关者的角度，检查和验证测试对象的质量。测试人员除了需要关注功能测试之外，还需要特别针对非功能特性进行测试。

4.2.1 功能测试

功能测试的关注点是软件产品能够做什么，其测试依据通常来自规格说明文档、特定领域的专业知识或者其他隐现的要求。功能测试主要包括：适合性测试、准确性测试、互操作性测试、安全保密性测试和辅助性测试。

4.2.2 非功能测试

非功能测试关注的是软件产品满足客户需求和要求的程度。软件产品实现了具体的功能后，用户并不一定满意，它还可能存在其他的问题，因此非功能质量特性是另一个重点关注的测试。非功能测试主要包括：可靠性测试、易用性测试、效率测试、可维护性测试和可移植性测试。

4.3 基于经验的测试设计

测试依据文档，如需求规格说明常常是不全的、模糊的，甚至没有任何需求文档。在这种情况下，基于经验的测试将是一种比较合适的测试策略。

4.3.1 基于缺陷分类的测试

基于缺陷分类的测试的基础是测试人员以前积累的缺陷分类，它们可以为测试人员设计测试用例提供良好的思路，并在发现缺陷方面有更好的有效性。缺陷分类在测试过程中的主要作用包括：

- 为测试人员提供新的测试思路；
- 作为没有测试经验人员培训的资料；
- 作为测试工作产品的检查表；

- 和管理层进行沟通的有效工具；

4.3.2 探索性测试

探索性测试是软件测试的一种类型，它更强调测试人员的个人自由、主观能动性和职责。在探索性测试过程中，将学习、设计、执行和结果分析作为并行且相互支持的测试活动，不断优化测试人员工作的价值。探索性测试通常是针对脚本化测试而言的，两者的主要区别表现在：

- 与脚本化测试有比较明确的测试活动时间顺序不同，探索性测试更强调四个测试活动：学习、设计、执行和结果分析的并行的特点，或者说更强调这四个测试活动的互相反馈和相互支持的特点；
- 脚本化测试和探索性测试关注的测试重点不同：脚本化测试将更多的创造性活动放在测试设计上面，而探索性测试则将更多更高认知水平的活动放在测试执行上面；

脚本化测试和探索性测试并不是两种对立的测试类型，在实际的测试过程中，它们应该是相辅相成的，它们具有各自的优点，同时可以互相弥补对方的不足；

4.4 测试设计实践

4.4.1 如何保证测试设计的覆盖率

测试设计的覆盖率主要考虑两个方面，一个是针对被测试对象的结构或者代码的覆盖，另一个是针对被测试对象的需求的覆盖。经典测试设计中的白盒测试和黑盒测试技术能够有效的保证测试设计对于代码和需求的覆盖率。同时在实施过程中，可以通过工具来辅助监控测试设计的覆盖率。例如：通过工具建立需求和测试用例的跟踪矩阵，保证每一个需求至少有一个测试用例与之对应。

4.4.2 如何平衡测试设计的深度和广度

测试设计的深度和广度的平衡应该与测试风险相结合。根据风险发生的可能性和风险严重程度得到的风险级别可以作为决定测试用例深度和广度的依据。

- 深度优先：首先设计高风险的区域，完成所有高风险的测试设计之后，才开始低风险的测试设计，即测试设计顺序严格按照风险级别来制订；
- 广度优先：测试设计优先级按照抽样的方式来制订。抽样方法可以根据不同风险级别，确定不同的权重，测试样本的选择将基于风险的权重而展开。广度优先得到的测试可以覆盖所有已经识别的风险级别，因此可以通过风险覆盖率评估得到测试样本。

4.4.3 如何满足不同客户的质量要求

对于同一个产品，不同的利益相关者对质量的理解和要求不同。因此测试人员在测试过程中，需要站在不同的利益相关者的角度对测试对象的质量进行检查和验证。例如，测试人员除了关注需求文档中明确描述的需求条目之外（有时称为“显现的需求”），还应该关注隐现的需求，如竞争对手的产品特征、用户的群体特征和使用习惯等。因此在测试过程中测试人员除了关注测试对象的功能测试之外，还需要针对其他非功能特性进行测试。

为了更好的满足不同客户的质量要求，基于质量特性的测试设计在软件测试过程中引入质量特性的概念。通过引入质量特性，可以帮助测试人员更全面地思考测试用例的设计，不断提高对质量

特性的测试覆盖率。常见的质量属性有以下 6 类：功能性、可靠性、易用性、效率、可维护性和可移植性。

4.4.4 在需求不完善的情况下如何有效测试设计

需求规格说明是测试设计的一个重要的输入，但是经常会出现需求规格说明不完善的情况。测试人员在需求规格说明中不能得到足够的信息，这种情况下就需要测试人员通过其他的方式更全面的获得测试设计的输入。通常可以考虑从以下几个方面：

- 通过与客户或者用户的沟通，更多的获得相关需求；
- 采用基于质量特性的测试设计，从质量模型中的质量特性出发进行测试设计；
- 采用基于风险的测试设计，分析被测试对象中可能存在较大风险的地方，有针对性的进行测试设计；
- 采用基于经验的方法，根据测试人员在其他项目的经验或者行业经验，选择容易出现缺陷的地方设计测试用例；
- 在测试执行的过程中，根据测试结果，对测试设计进行更新和优化。

第5章 缺陷管理

5.1 缺陷的特点

5.1.1 缺陷的雪崩效应

缺陷的雪崩效应指的是缺陷数目会随着软件开发过程的深入而不断地增加，类似于滚雪球的效应。出现缺陷雪崩效应主要是下面两个原因造成的：

- 首先，开发过程中前一阶段出现缺陷的地方，同样将在后一阶段出现，并且通常来说后一阶段的缺陷数目比前一阶段的数目更多；
- 其次，即使在前一阶段是正确的软件工作产品，由于人容易犯错误，在后续的阶段也可能引入新的缺陷，从而导致缺陷数目的增加；

5.1.2 缺陷的成本放大效应

缺陷的成本放大效应：不同阶段发现和修复缺陷的成本是不一样的；越到软件开发生命周期的后期发现缺陷，其修复的成本将是急遽增加的，而不是线性增加的。

测试人员尽早参与静态测试，例如：尽早参与软件文档的评审，可以有效地发现其中的缺陷并尽快修复缺陷，因此可以有效地减缓缺陷的雪崩效应，并降低修复缺陷的成本。

5.1.3 缺陷的集群效应

测试过程中发现的缺陷在测试对象的不同功能模块分布，同样也满足 20/80 原则：测试对象 20%模块中可以发现 80%的缺陷，即缺陷的集群效应。测试人员根据该原则，可以在后续的测试中重点关注该 20%模块。同时分析在该 20%模块中集中引入缺陷的原因，以发现开发和测试过程中存在的问题，从而帮助后续项目的过程改进。

5.2 缺陷管理生命周期

缺陷是测试过程中最重要的输出之一，缺陷的生命周期由一系列的活动和状态所组成。缺陷对评估和改进产品质量、测试效率、测试过程和开发过程等都有重要的意义，其主要表现为：

- 为开发人员和其他人员提供问题反馈，在需要的时候可以鉴别、隔离和纠正这些缺陷；
- 为项目管理人员提供被测系统的质量信息，在需要的时候作为调整测试进度的依据；
- 为测试过程改进和开发过程改进提供有用的数据和信息；

缺陷管理过程需要从缺陷发现、缺陷提交、缺陷分类、缺陷修复、解决方案验证等整个过程进行跟踪和管理。了解缺陷状态、缺陷相关的角色、缺陷严重程度和优先级是进行缺陷管理的前提条件。

5.2.1 缺陷状态

缺陷生命周期是缺陷一系列不同状态演变的过程，其包含的主要状态包括：新提交的、已分配的、已解决的、已修复的、已关闭的等。

缺陷的状态除了上面的几个主要状态之外，可以根据需要增加其他的辅助状态，例如：已拒绝的。缺陷管理过程中采用的缺陷状态可以按照组织和项目的特点进行合理的裁剪。

5.2.2 严重程度

- 严重程度 1（致命的）：产品在正常的运行环境下无法给用户提供服务，并且没有其他的工作方式可以补救；或者软件失效会造成人身伤害或危及人身安全；
- 严重程度 2（严重的）：极大地影响系统提供给用户的服务，或者严重影响系统要求或者基本功能的实现；
- 严重程度 3（一般的）：系统功能需要增强或存在缺陷，但有相应的补救方法解决这个缺陷；
- 严重程度 4（轻微的）：细小的问题，不需要补救方法或对功能进行增强；或者操作不方便，容易使用户误操作；

5.2.3 优先级

- 优先级 1（立即）：用户的业务或工作过程受阻，或运行中的测试无法继续。该问题需要立即修复，或必要的话采取临时措施（如：打补丁的方式）；
- 优先级 2（下次发布）：在下次常规的产品发布或下次（内部）测试对象版本交付时实施修正；
- 优先级 3（必要时）：在受影响的系统部件应当进行修订时进行修正；
- 优先级 4（未决）：尚无修正计划；

5.2.4 相关角色

- 测试人员：主要是指发现和报告缺陷的测试人员。通常情况下，测试人员需要对该缺陷后续相关的状态负责，包括回答相关人员对这个缺陷信息的询问，以及在正式版本上进行再测试和回归测试。
- 开发人员：主要指对缺陷进行调查和修复的开发人员。注意在提交测试人员正式再测试之前，需要对修改后的缺陷在开发环境上进行验证。
- 缺陷评审委员会：主要由项目经理、测试经理、质量经理、开发经理以及资深的开发人员、测试人员等组成。他们对缺陷进行确认，并将其分配给相应的开发人员进行修复，同时对有争议的缺陷进行仲裁。
- 版本经理：负责将已经解决的缺陷相关的配置信息合并到新的版本。

5.3 缺陷管理实践

5.3.1 如何有效处理难以重现的缺陷

测试过程中经常会发现一些难以重现的缺陷，特别是测试非功能特性的时候，例如：稳定性测试、压力测试、兼容性测试等。假如软件产品交付给用户之后，在运行过程中出现由于这样的缺陷而导致的失效，那么将会大大影响用户对产品的信心。

针对难以重现的缺陷，下面的一些建议可以改进测试的效率和有效性：

- 尽量获取系统的日志和调试信息；
- 测试人员应该报告不可重现的缺陷；
- 报告缺陷之前与开发人员进行确认
- 在产品操作指南（使用说明）中明确告知客户；
- 缺陷报告中明确该缺陷能够重现的可能性；

5.3.2 测试与开发在缺陷认定上的冲突

测试人员提交缺陷报告后，经常会出现以下情况：测试人员认为这是软件系统的一个缺陷，而开发人员认为这是正常的系统功能。到底是缺陷还是正常功能？这是测试人员和开发人员在缺陷问题上经常出现的分歧。下面的建议有助于解决该分歧：

- 首先，测试人员应该让开发团队理解提交缺陷的目的并不是对开发人员的成果进行挑刺，相反，测试人员和开发人员的目标是一致的，都是为了提高软件产品的质量，满足客户的需求。
- 其次，需要和开发人员进行有效地沟通，让开发人员理解测试的目的不仅仅只是针对软件的功能而开展，还包括了可靠性、易用性、效率、维护性、可移植性等其他质量特性。因此，发现非功能特性的缺陷时，开发人员不应该因为系统需求规格说明中没有详细定义，而否认存在的问题。
- 第三，测试人员应该站在用户的角度对软件的使用质量特性进行有效地测试，包括软件的有效性、生产率、安全性以及满意度等。
- 第四，假如测试人员和开发人员在缺陷认定沟通失败的情况下，可以通过项目的变更控制委员会，讨论确定提交的缺陷报告是不是缺陷的问题。

5.3.3 用户反馈的缺陷：过程改进的基石

穷尽测试是不可能的，通过测试并不能完全发现和修改测试对象中的缺陷。，因此，不可避免有一些缺陷会遗漏到客户的使用现场，从而触发软件产品产生令用户不满意的失效或者各种问题。

针对用户反馈的缺陷，测试人员可以从 2 个方面进行分析：

- 测试人员可以根据用户反馈的缺陷数目，来判断和分析测试人员在测试过程中的测试有效性。其中缺陷检测百分比是基于这样的目的进行定义的。
- 分析缺陷遗漏的原因以改进开发和测试过程。缺陷遗漏的原因是多方面的，包括：

- 需求不全或者不清晰；
- 测试设计问题；
- 测试资源问题；
- 回归测试问题；
- 测试环境和使用环境的问题；
- 穷尽测试是不可能的；

5.3.4 不要以缺陷数量考核测试人员

缺陷数量可以作为测试效率的度量指标，但作为考核测试人员个人能力的度量指标并不合适，因为影响发现缺陷数量的因素很多：

- 测试的对象不同；
- 测试类型的不同；
- 开发人员水平的不同；
- 功能模块复杂程度的不同；
- 缺陷可检测性的不同；
- 测试的目的不同；
- 测试的工作不同；
- 缺陷发现的阶段不同；

以缺陷数量考核测试人员除了上面的不合理和不客观之外，它可能还会引起各种问题，从而影响整个软件产品的开发。其表现为：

- 为了缺陷数量而查找缺陷；
- 降低测试团队的地位；
- 导致测试任务和工作的不好分配；
- 影响测试团队内部的合作；

5.3.5 开发人员为什么拒绝修改某些缺陷

测试过程中测试人员提交的缺陷，开发人员经常以各种理由拒绝修改。分析他们为什么不愿意花费时间和精力解决测试人员提交的缺陷，有利于测试人员提交高质量的缺陷报告，从而提高测试效率和有效性。开发人员拒绝修改某些缺陷的原因是多方面的，包括：

- 开发人员无法复现缺陷（无法复现）；
- 缺陷报告中提供的信息不足，或者复现缺陷需要奇怪而复杂的步骤（难以理解）；
- 开发人员认为是系统的一个功能点，而测试人员认为是一个缺陷（缺陷还是功能点）；
- 开发人员不理解测试人员的角色和职责定位（测试人员的角色）；

- 由于交付时间紧张，在本版本中不修复，延期到下个版本修复；
- 当前架构无法修复该缺陷，或者技术无法实现；

第6章 测试工具

6.1 测试工具类型

6.1.1 测试管理工具

测试管理工具主要涉及以下功能：管理软件需求、测试计划、测试用例以及测试过程中各类数据的统计和汇总。测试管理工具可能会和缺陷管理工具、测试执行工具等之间存在接口。测试管理工具还可以提供其他辅助功能，管理工具集成图表和报告工具能协助分析应用在测试过程的任意点上的活动情况。测试经理对需求覆盖率、进度计划、运行进度、缺陷统计、测试结果进行定量的分析，有助于决定软件版本发布的时机。

6.1.2 缺陷管理工具

缺陷管理工具是一个用于记录、跟踪、管理和分析缺陷信息的系统。它允许不同类型的用户，例如：测试人员、项目经理、开发人员、软件用户等直接将缺陷输入到数据库中，也可以对数据库内的缺陷进行查询、分析统计，从而协助开展测试监控活动。

6.1.3 静态分析工具

静态分析工具是在不运行被测系统的基础上，对被测试对象进行静态的分析的工具。有固定语法和结构的代码和文档，都有可能通过静态分析工具进行测试。

6.1.4 测试执行工具

测试执行工具可以把测试人员从机械的测试执行任务中解放出来。测试执行工具给测试对象提供测试数据，记录测试对象的反应，并将两者进行比较。测试执行工具有很多，既包括针对 WEB 或 GUI 的测试工具，也包括性能测试工具等。

6.2 测试工具引入

6.2.1 使用测试工具的益处

通过使用测试工具可以带来以下好处：

- 减少手动测试的工作量，使得一些重复性的测试活动通过工具来完成。
- 增加测试的时间，通过使用工具即使在夜晚仍然可以进行测试。
- 便于测试管理，减少沟通的成本。
- 很好的可重复性。
- 能够完成一些手动无法完成的测试。
- 更快的完成测试任务。

6.2.2 测试工具引入过程

明确工具所要支持的测试任务之后，就可以开始工具的评估和选择。由于工具的引入可能是一个庞大的投资，因此应该谨慎并且有计划地选择新工具。工具评估和选择的过程包括以下五个步骤：

- 工具应用的需求规格说明；
- 市场调研（纵览可选对象）；
- 工具演示并创建工具列表；
- 评估列表中的工具；
- 评审结果并选定工具；

选定的工具引入组织之后，通常要实施试验项目，以验证在实际的项目环境下工具是否能实现预期的优势。试验项目不应由参与工具评估的人员来进行，以防止在解释评估结果时产生利益冲突。

6.2.3 影响测试工具引入的因素

由于新工具一开始会产生额外的工作量，工具的引入需要工具用户和相关人员大量且不断的投入。初次使用时重要的成功因素是：

- 逐步的引入；
- 将工具支持集成到过程中；
- 实施用户培训和持续指导；
- 制订应用工具的规则并提供建议；
- 收集使用经验并传授给所有用户（例如：提示、技巧、常见问题等）；
- 监测工具的接受度并收集和评估成本效益数据；

6.3 测试自动化

6.3.1 录制 / 回放

没有技术背景的测试人员只要简单录制测试的操作过程，然后播放录制好的测试脚本，就可以轻松自动化所有的测试。通过录制建立的脚本，基本上都是用脚本语言以硬编码的方式编写的，当应用程序变动时，这些硬编码也随之需要更改。因此，维护这些录制好的脚本，成本是非常高的。

6.3.2 数据驱动的自动化测试

数据驱动的自动化测试是针对开发与测试之间紧密耦合问题提出的测试方法。通过建立测试与开发定义的软件元数据的关联—元数据映射表．在测试与开发之间建立松耦合关系。不论测试人员修改测试脚本，还是开发人员修改软件，只需要修改元数据映射表，就可以满足测试与开发同步进行。这样，可以减少测试脚本维护的工作量，更好的实现自动化测试。

数据驱动的自动化测试提供了这样的框架：从某个数据文件(例如 ODBC 源文件、Excel 文件、Csv 文件、ADO 对象文件等)中读取输入、输出的测试数据，然后通过变量传入事先录制好的或手工编写的测试脚本中，用来验证应用程序的测试数据。在这个过程中，数据文件的读取、测试

状态和所有测试信息都被编写进测试脚本里；测试数据只包含在数据文件中，而不是脚本里，测试脚本只是一个“驱动”，或者说是一个传送数据的机制。

6.3.3 关键字驱动的自动化测试

关键字驱动的自动化测试（也称为表驱动测试自动化），是数据驱动自动化测试的变种，可支持由不同序列或多个不同路径组成的测试。它是一种独立于应用程序的自动化框架，在处理自动化测试的同时也适合手工测试。关键字驱动的自动化测试框架建立在数据驱动手段之上，表中包含指令(关键词)，而不只是数据。这些测试被开发成使用关键字的数据表，关键字驱动的自动化测试是对数据驱动的自动化测试的有效改进和补充。

6.4 测试工具实践

6.4.1 测试工具的成本

引入新工具需要考虑选择、购买和维护工具的成本。此外，成本还可能因硬件购买、升级以及员工培训而增加。根据工具的复杂度以及要部署的工具数量，投资可能会迅速增长。新工具的引入同时也需要关注新工具开始显现成效的时间跨度。

新工具引入的成本主要包括：

- 直接购买成本：包括知识收集（工具学习曲线）、适时的评估工作（工具比较）、与其他工具的集成工作、工具的购买、修改或者二次开发。
- 后期维护成本：包括工具持有的成本（包括维护、许可费用、支持费用、持续的知识水平提升）、可移植性、可用性和依赖性（如果缺乏这方面特性）、持续成本评估、质量改进以保证对所选工具的最优使用。

6.4.2 软件测试自动化本身也是一个开发项目

软件测试自动化需要长时间的人力物力投入，因此专职的人员进行测试自动化是一个合适的选择。同时软件测试自动化也是一个独立的开发项目，需要遵循合适的开发过程并保证足够的投入。测试自动化的实施一定在软件开发的早期进行，包括自动化平台的搭建，以及是否要求被测试系统提供必要的接口以帮助实施测试的自动化。自动化平台本身的搭建，需要为自动化脚本的开发、运行和维护提供便利。

测试执行过程如果碰到因为自动化测试脚本自身的问题而导致测试失败，那么会给测试执行带来额外的调试工作量，同时也影响软件测试自动化在整个团队中的形象，不利于将来更大范围的软件测试自动化的推广。

6.4.3 使用工具并不能解决原本混乱的过程

如果测试团队当前过程混乱，通过引入工具并不能改变当前的混乱状况。混乱的过程通常是由组织架构、公司文化和团队成员技能等方面综合影响造成的。虽然很多测试工具有良好的测试过程规范，但是它很难改变过程混乱的情况。相反，在过程混乱的团队中引入测试工具，还可能导致原本的过程更加混乱。

6.4.4 选择合适的测试用例进行自动化

选择哪些测试用例进行测试自动化，需要综合考虑各种因素。即使可以自动化所有的测试用例，由于时间、成本和资源等条件的限制，也无法一次性将所有的测试用例进行自动化。在选择测试用例进行自动化的时候，需要考虑以下因素：

- 每个系统功能的典型测试用例；
- 最重要的系统功能的测试用例；
- 最容易进行自动化的测试用例；
- 可以最快得到回报的测试用例；
- 经常运行的测试用例；

第 2 部分 SWTBOK®软件测试经理

第7章 测试团队组建

7.1 测试团队和独立性

7.1.1 独立测试的策略

通过独立的测试人员进行静态测试和动态测试，其发现缺陷的效率会明显提高。但独立的测试团队并不能完全替代开发人员进行的测试，开发人员也可以有效的发现软件工作产品中的缺陷。根据组织和产品的特点，可以选择不同的独立测试的策略：

- 开发人员测试自己的代码；
- 开发人员测试他人的代码；
- 开发团队中独立的测试人员；
- 独立于开发团队的测试团队；
- 组织外的第三方测试团队；

7.1.2 独立测试的选择

不同的测试级别，可以采用不同独立测试的策略。独立测试策略选择过程中，需要考虑不同策略在独立性程度、对测试对象内部结构的了解程度、具体职责的变化、测试技能的要求、沟通成本、测试人员的归属感等因素方面的不同，并根据组织和产品的不同特点进行调整。

7.2 角色和职责

测试团队的构成与其规模没有太大的关系，假如测试团队只有一个人，那么他需要扮演不同的角色。明确的角色和职责定义是高效开展软件测试工作的前提。

7.2.1 测试经理

测试计划和控制的专家，应具备软件测试、质量管理、项目管理和人员管理等领域的知识和经验。其主要任务包括：

- 作为测试团队的代表，与其他利益相关者共同制定测试计划，并获取测试资源；
- 选择合适的测试策略和方法，积极参与缺陷管理、风险管理、配置管理等测试相关的过程，以有效管理和控制变更及其他测试活动；
- 确定测试环境和测试自动化的类型和范围，配置管理测试件以保证它们的可追溯性；
- 发起和监测测试活动和任务，例如：测试设计和执行；
- 根据测试状态和结果采取合理的应对措施，例如：调整测试计划、更新测试范围和优先级；

- 选择合适的度量评估测试过程、测试进度和产品质量；
- 选择和引入合适的测试工具，并为测试人员组织必要的培训；
- 根据测试过程收集的信息编写测试总结报告，并在测试结束后开展经验教训活动以收集和分析信息用于过程改进；

7.2.2 测试设计人员

测试方法和测试设计规格说明方面的专家，具备测试分析、设计以及软件工程等领域的知识和经验。其主要任务包括：

- 分析、评审和评估用户需求、规格说明、设计说明和模型等测试依据文档；
- 设计和创建测试用例；
- 准备和获取测试数据，根据自动化策略选择自动化哪些测试用例；

7.2.3 测试执行人员

执行测试和缺陷报告方面的专家，具备 IT 基础知识、测试基础知识，能应用测试工具，熟悉被测对象。其主要任务包括：

- 分析、评审和评估用户需求、规格说明等测试依据文档；
- 评审测试工作产品，例如：测试用例规格说明；
- 协助测试环境管理员搭建测试环境；
- 执行各种级别的测试用例，记录测试日志，评估测试结果，提交缺陷报告；
- 执行再测试和回归测试；

7.2.4 测试自动化人员

测试自动化专家，具备测试基础知识、编程经验以及丰富的测试工具和脚本语言知识，可利用测试工具有效进行测试自动化。其主要任务包括：

- 负责自动化测试框架的设计和搭建，以及自动化测试环境的搭建；
- 分析自动化测试需求，制定自动化测试计划；
- 设计、开发和维护自动化测试用例，并对自动化测试执行结果进行分析，完善自动化测试框架；

7.2.5 测试环境管理员

安装和操作测试环境方面的专家，具备系统管理知识。其主要任务包括：

- 负责测试环境所需的网络规划和建设，维护网络的正常运行；
- 建立、设置和维护测试环境所需的应用服务器和软件平台；
- 登记、分配和管理测试实验室的硬件和软件资源；
- 申请所需的硬件和软件资源，协助有关部门进行采购和验收；

- 设计和设置测试实验室的硬件和软件资源的使用权限，并保证其安全保密性；
- 协助安装测试平台和被测系统；
- 优化测试环境，提高测试环境中网络、服务器和其他设备运行的性能和稳定性；

7.3 个人技能

合格的测试人员需要具备一系列的个人技能，包括行业背景知识、软件测试过程、测试技术和方法，以及软技能等。

7.3.1 行业背景知识

测试的对象通常是具体的软件产品或者系统，因此深入了解测试对象的总体架构设计、需求、详细设计以及需要满足的标准和规范等是有效开展测试活动的基础。测试团队针对行业背景知识应该制订详细的培训计划，通过培训、内部讨论和学习等方式储备软件产品的领域知识。

7.3.2 软件测试过程

测试活动贯穿于整个软件开发生命周期，测试人员需要了解整个过程中需要做什么，也即了解软件测试过程，例如：测试阶段、测试活动、测试任务、利益相关者（角色和职责）等。

7.3.3 测试技术和方法

测试人员知道做什么之后，也需要掌握具体的测试技术和方法以有效的开展测试活动，即如何做的问题。详细的测试技术和方法，包括测试估算、风险管理、静态测试、测试设计、缺陷管理等。

7.3.4 软技能

软技能是情商的社会学术语，它由一系列能够反映个人特质的要素组成，这些要素包括一个人的人格特质、社交能力、沟通能力、语言能力、个人行为习惯等。软技能与硬技能（那些作为工作硬性要求并能够部分反映一个人智商的能力）是互补的。软技能（情商）可在一个组织的成功中扮演非常重要的角色，主要包括：怀疑精神、好奇心、创新能力、分析能力、耐心、沟通技巧、团队精神和工作热情等。

7.4 测试团队组建实践

7.4.1 如何招募测试团队成员

招募团队成员是软件测试团队组建过程中的突出问题。招募成员的基本步骤如下：

- 团队成员的规划，明确测试团队中的角色分布和职责要求；
- 根据需要从各种渠道收集候选人信息，包括招聘网站、内部推荐、其他部门调动和内部提拔等各种形式；
- 对候选人的考核，从行业背景、测试过程、测试技术和方法，以及软技能等方面进行评估；
- 确定合适人员，安排上岗。

7.4.2 分布式测试

分布式测试指的是由位于不同地方的测试团队协作完成测试任务的过程。分布式测试采用的策略包括：

- 按照测试对象功能进行划分，不同功能的测试分布在不同的地方。其优点是管理相对方便，但是可能无法有效利用测试资源。
- 将不同测试团队看作是一个整体分派测试任务，不考虑他们地点的差异。其优点是可以较好地利用测试资源，但是测试管理比较困难，需要跨区域的管理经验和良好的沟通技巧。

7.4.3 个人技能评估

测试团队中不同的角色，其承担的职责和需要具备的技能是不同的。针对测试人员的个人技能评估应该基于不同的角色开展。

个人技能评估的目的是发现测试团队成员的优缺点，以最大限度地发挥个人能力，并实现团队内部的高效合作。同时，根据个人技能评估得到的结果，可以有针对性地安排合适的培训，在提升个人能力的同时，增加测试团队的能力。

个人技能评估需要从多个角度入手，包括：

- 行业背景知识；
- 软件测试过程；
- 软件测试方法和技术；
- 软技能；

第8章 测试团队管理

8.1 有效激励

8.1.1 满足员工需求

根据员工的不同需求“投其所好”，是进行激励的有效方式。不同员工在不同时期的需求是不一样的，例如：得到尊重和认可、获得自我表现的机会、广阔的成长空间、灵活的工作时间等。

8.1.2 有效授权

通过有效授权可以调动员工的积极性和创造性，加强员工的责任感和主人翁精神，不断提高对企业和团队的认同感。同时，员工通过承担更多的责任，也可以提高自身的各项技能，获得更大的发展空间。

8.1.3 提供培训机会

由于科学技术的飞速发展，员工如果不积极学习，就很难跟上时代发展的步伐，甚至被社会淘汰。因此，每个成员自身都有危机感和紧迫感，从而都有学习和培训方面的需求。这种情况下，为员工提供良好的培训机会，可以激励员工创造更大的价值。

培训的方式可以是多种多样的，可以在团队内部组织交流和学习、相互培训，也可以聘请外部专家进行培训；既可以是课堂教学的方式，也可以通过 E-learning 的方式。

8.1.4 尊重和认可

尊重和认可能够使员工对自己更加自信、对工作更加热爱，能够鼓励员工提高工作效率。给员工的认可要及时而有效，当员工工作表现很出色时，管理人员应该立即给予称赞，让员工感受到自己受到上司的赞赏和认可。可以通过各种不同的方式认可员工的工作表现，例如：口头赞赏、书面赞美、对员工一对一的赞赏、公开的表扬等，从而不断鼓舞员工士气。

8.1.5 物质奖励

除了精神激励外，物质奖励也是必不可少的，它也是日常工作中经常使用的一种方式。薪水不仅能保证员工生存，同时能者多得的薪酬机制也能有效地起到激励效果。物质奖励的多少依赖于员工能为公司带来的价值，例如：对于为公司创造出高利润、开发出赢利新项目的核心人才，通过加薪激励是必不可少的。

在使用物质奖励方式的时候，一个重要的考虑因素是公平。如果某个员工的贡献突出，那么可以对这个员工进行单独的奖励。假如成绩是团队共同努力的结果，那么需要对整个团队进行奖励。同时物质奖励要有充足的理由，确保只有表现优秀的员工或者团队才能够获得物质奖励。

8.2 沟通技巧

沟通技巧是软技能的重要组成部分。

8.2.1 往上沟通

往上沟通指的是与上一级经理或者领导沟通。每个人都有上级或者领导。往上沟通需要注意几点：

- 尽量避免给管理层出问答题，而采用选择题的方式；
- 与管理层的沟通可以在任何地点进行；
- 针对某些问题的沟通，必须准备好自己的答案；

8.2.2 往下沟通

往下沟通指的是与下属之间的沟通。假如你是一名管理者，掌握与下属员工沟通的技巧和艺术有着举足轻重的意义。下面是 3 个建议：

- 多学习、多了解、多询问、多做功课；
- 不要只会抱怨和责骂，允许下属尝试和失败；
- 为下属提供方法，同时紧盯过程；

8.2.3 水平沟通

水平沟通指的是没有上下级关系的部门或者个人之间的沟通。水平沟通存在很多障碍，常见的就是“踢皮球”。水平沟通的几个建议：

- 主动；
- 谦让；
- 体谅；
- 协作；
- 共赢；

8.3 职业发展

软件测试的发展是一个长期的、不断完善的过程，这同样适用于软件测试的职业发展。

8.3.1 确定战略目标

根据个人的兴趣、爱好和能力等，确定职业发展的目标，即确定战略目标，例如：5 年之内成为合格的测试经理。

8.3.2 核心成功因素

为了实现战略目标，需要确定哪些因素会影响战略目标的实现，即核心成功因素，主要包括：产品背景知识、测试过程、测试技术和方法、测试团队、软技能等。

8.3.3 关键驱动因素

为了不断提高测试人员在核心成功因素方面的能力，测试过程中测试人员需要采取具体的措施和手段，即关键驱动因素，主要包括：自我学习、参加培训、实践和经验积累、软技能培养等。

8.3.4 关键绩效指标

测试过程中需要针对核心成功因素采取的措施和手段进行效能评估，衡量具体措施和手段是否满足核心成功因素，即定义关键绩效指标，以评估具体措施的有效性。

因此，测试职业规划需要首先确定战略目标，并以此确定核心成功因素；然后根据不同核心成功因素采取不同的措施（即关键驱动因素）以不断满足核心成功因素的要求；最后根据关键绩效指标评估具体措施的有效性，以及评估是否满足战略目标。

当然，测试职业规划光有远大的目标是不行的，重要的是执行和坚持，不断提高核心成功因素的竞争力，从而逐步实现战略目标。

8.4 测试团队管理实践

8.4.1 如何体现测试的价值

软件测试的价值主要体现在内部和外部两个方面：

- 内部：通过静态和动态测试提高产品质量；缺陷预防相关的测试活动，例如：制定更完善的 checklist，发现缺陷并推动缺陷的修复；
- 外部：收集客户/用户需求；利用测试团队的优势，为外部客户提供技术演示、为市场或服务人员提供技术支持。

8.4.2 将个人目标和组织目标相结合

个人目标和团队目标的结合有利于更好的提高个人技能、提高团队成员工作的积极性和主动性，同时也有利于团队目标的更好的实现。

测试经理应该积极和团队成员进行沟通，了解成员的个人目标，在安排测试活动的时候，尽量将测试工作和成员的个人目标结合起来。团队成员也应该积极主动和测试经理沟通自己的想法，在出现机会时，主动争取参与相关测试活动。同时测试经理也应该鼓励团队成员积极主动的发起一些活动。

8.4.3 测试团队的地位

测试团队在整个团队的中的地位和作用，受到以下两个方面的因素的影响：

- 测试团队的独立性：测试团队的独立性不同，导致测试团队的职责各不相同，从而影响测试团队的作用和地位；
- 测试团队的能力：测试团队自身的能力，也决定了测试团队的作用和地位，例如：当测试人员能力可以参与前期的需求和设计评审，并提出有价值的见解时，测试团队的前期介入就能发挥更多的作用。

8.4.4 如何协调开发和测试的冲突

虽然开发团队和测试团队都以发布高质量的产品为目的，但是由于角色和思维方式的差异，导致在软件生命周期中，不可避免的会出现冲突。在协调和处理这些冲突的时候要注意以下方面：

- 测试经理可以充当协调员的角色，和开发经理进行协商，尽量避免开发人员和测试人员的直接冲突；

- 可以建立第三方的仲裁机构，例如问题评审委员会，对出现的冲突进行仲裁；
- 明确测试人员和开发人员的职责；
- 平时注意增强测试团队和开发团队的交流；
- 冲突并不可怕，只要应对得当，就可以向有利于提高产品质量的方向发展。

8.4.5 测试人员的发展方向

测试人员的发展空间广阔，以下是软件测试人员可以考虑的几个发展方向和职位：

- 领域技术专家：成为行业或者特定领域的专家，例如：金融领域的测试专家，安全性、性能、自动化测试方面的专家；
- 系统架构师：成为整个产品的系统架构师，负责系统架构的设计；
- 测试经理：负责软件测试项目的管理，保证软件测试在整个软件生命周期中的活动能高质量的完成；
- 部门经理：负责人员的管理，成为部门经理，可以管理测试、质量保证甚至整个研发部门；
- 客户服务和技术支持人员：利用对产品知识的全面了解，成为专业的技术支持人员；
- 咨询和培训师：随着测试技能和实践经验的积累，可以成为专业的测试咨询和培训师；

第9章 测试估算

测试估算得到的数据，例如：测试工作量，是进行测试计划、测试资源分配和测试进度监控的基础。

9.1 测试估算过程

9.1.1 测试估算概述

为了更好的和项目利益相关者进行沟通，测试人员需要了解测试估算中的几个概念：

- 目标（target），例如：项目经理说该产品必须在 2 个月之内完成测试工作，并参与客户现场的测试。这里的 2 个月就是项目经理的目标；
- 估算（estimate），例如：测试经理认为针对该产品的测试需要的工作量是 20 个人月。这里的 20 个人月是测试人员估算的结果；
- 承诺（commitment），例如：根据测试工作量 20 个人月，以及测试团队为 5 人，得出的测试时间 4 个月。这里的 4 个月是承诺，即测试人员承诺在 4 个月之内完成该产品的测试任务；

测试估算是测试计划的基础，准确的估算可以为测试活动的开展提供更好的支持，其主要作用表现在：

- 根据测试估算创建详细的测试时间进度计划；
- 识别测试的关键路径；
- 根据测试估算和目标，确定测试的优先级和重点；
- 根据测试估算和目标，确定测试资源的合理分配；

9.1.2 测试估算基本过程

合理的测试估算过程定义是提高测试估算准确性和可控性的前提，完整的测试估算过程应该包括测试估算的输入、估算步骤和测试估算的输出组成。假如输出的测试估算结果不能满足定义的项目目标，那么需要更改测试估算的输入而不是为了满足目标而有意识地调整估算结果。测试估算的输入需要考虑多个因素，例如：产品类型、测试范围、假定条件、优先级和测试重点等。

测试估算既可以先估算测试规模，然后进行测试工作量和测试进度的估算；也可以直接估算测试工作量，然后进行测试进度的估算。选择具体估算的顺序依赖于项目所处的阶段以及采用的测试估算技术。合理的测试估算过程应该具备下面的特征：

- 在可能的情况下，尽量采用计数和计算的手段进行估算，而尽量避免直觉和判断的手段；
- 在测试估算过程不同阶段采用不同的测试估算技术，并对结果进行比较；
- 测试估算过程相对稳定，假如有某些变更，应该通过文档化的方式进行记录并进行评审；

9.1.3 测试估算与测试过程的集成

测试估算应该贯穿于整个软件测试生命周期，而不是测试计划阶段的一次性活动。在不同的测试阶段，或者在测试输入有较大变更的时候，都需要重新估算测试对象。

测试生命周期不同测试阶段的测试任务，可以更好的为测试估算提供支持和帮助。同时，不同测试阶段其测试估算的准确程度范围也是不一样的，同样它可以帮助测试人员更好的分析和评估测试风险。

测试估算与测试过程的有效集成，还需要不断的分析测试估算出现偏差的原因，从而在将来的项目中不断改进测试估算的能力。测试估算与测试过程的有效集成，可以帮助测试人员了解测试估算的依据，并知道重新测试估算的原因是什么，有助于不断积累成功的经验，并总结容易出现差错的测试实践。

9.2 测试估算技术

9.2.1 基于单个专家的测试估算

基于单个专家的测试估算可能是测试过程中最常见的估算方法，这里的单个专家指的是单个人，可以是测试人员，也可以是测试经理。基于单个专家得到的测试估算，常常会以乐观的心态估算得到结果，即测试资源、测试时间和测试人员等处于理想的情况下得出的，即单点测试估算得到的结果常常是偏低的。

为了避免单个专家得到的估算结果出现偏差这个问题，采用结构化和系统化的测试估算技术是必要的，因此引入计划评审技术 PERT（Program Evaluation and Review Technique）。其基本思路是：在估算过程中引入“乐观的估算值”、“最可能的估算值”和“悲观的估算值”，分别对测试对象进行估算；然后通过 PERT 计算得到“期望的估算值”。

9.2.2 基于专家团队的测试估算

基于单个专家得到的测试估算值要么是完全高估，要么是完全低估（当然也可能由于运气好得到的是正确的估算，但这样的概率太小）。而基于专家团队的测试估算可以较好的解决这个问题，其理论基础来自于“大数法则”， 即如果测试人员估算得到一个高估的值，那么其错误将完全在偏大这侧；反之，则在偏小的一侧。如果不同测试人员估算得到了一些不同的结果，那么它们可能在偏大的一侧，也可能在偏小的一侧，结果是它们可以在一定程度上相互弥补。这实际上是概率分布在测试估算上的应用。

基于专家团队的测试估算的基本思路是：每个专家估算得到自己认为合适的期望估算值；然后在专家团队内分析和评估每个估算结果；最终得到大家一致同意的一个收敛的估算结果或者估算范围。这种基于团队的估算方式，称之为“WideBand Dephi”方法。为了更好的应用基于专家团队的测试估算，需要掌握几个基本原则：

- 每个专家成员分别对测试对象进行估算，并和其他专家估算的结果进行比较，分析其中的共同和不同之处；
- 不要直接将每个专家的估算结果进行平均，并以此作为估算的最后结果；
- 讨论并分析每个专家的估算结果，直到专家团队每个成员都可以接受的估算结果；

9.2.3 基于类似项目的测试估算

基于类似项目的测试估算的基本步骤：

1） 按照合适的工作分解结构 WBS 的方式收集以前类似项目的测试规模和工作量；

2） 获取当前项目的测试估算结果，例如：测试工作量、测试规模等；

3） 比较当前项目和类似项目的不同参数，并得到两者之间的比例关系；

4） 根据类似项目的估算结果和步骤 3）中获得的比例关系，计算当前项目的估算值；

5） 检查两个项目的不同假设条件和前提条件，以及其他可能影响测试估算的因素，对最终的估算结果进行修正；

基于类似项目的测试估算需要有一定的前提条件，如能够收集类似项目的测试规模和工作量。当前项目的测试规模，需要测试人员根据其他的信息进行估算，如系统功能的特点、系统需求的条目、回归测试用例的数目等。

9.3 测试估算实践

9.3.1 如何更好利用测试估算

由于测试过程中存在各种各样的变更和事件，它们常常会推翻测试估算中所做的各种假设条件和限制条件，即测试估算的输入，例如：测试范围的假设、测试活动的假设、测试人员技能和知识的假设、测试优先级的假设等。因此纯粹的评判估算是否正确没有太多的意义，因为最初和最后的估算对象可能并不相同。

实际上，假如利用计划的测试资源，在保证测试质量的前提之下及时的完成了测试对象的测试内容，那么就可以认为本次测试符合了测试估算，尽管可能两者之间的测试对象并不完全相同。从这个意义上，好的测试估算可以定义为“基于估算的结果，帮助测试人员控制和协调测试过程中的各种活动，使之满足测试范围、测试时间、测试资源和测试质量等方面的要求。”

因此，好的测试估算并不是如何使之达到完美和正确，而是通过好的测试估算、合理的测试目标和良好的测试控制之间的配合，使得最终的测试结果尽量接近最初估算结果。好的测试估算的主要作用包括：

- 更好的监控测试过程；
- 提高测试质量；
- 更好的风险管理；
- 增强测试团队的可信度；

9.3.2 避免测试估算的主观性

测试估算很难完全正确，测试实践中测试人员通常会以乐观的心态进行测试估算，导致得到的测试估算值偏低，从而影响测试计划的有效性和增加更多的测试工作量。

为了避免有意识地以乐观的心态进行测试估算，测试人员需要了解这样选择的压力来自哪里？

- 来自管理层：由于产品市场等方面的要求，整个测试工作必须在什么时间之前完成，从而要求测试估算必须在某个时间范围内完成。来自项目目标的压力，不应该成为测试估算的目标，因为目标和测试估算是不一样的，假如两者之间存在很大的差距，必须考虑其中的风险；
- 来自自己：有的人认为偏高的测试估算体现不出自己的专业水平，特别是担心领导认为自己能力不行。对于这样的情况，测试人员需要避免刻意的降低测试估算结果，因为这导致的后果比高估严重的多。

9.3.3 如何减少测试估算的偏差

测试对象和测试估算过程都会影响测试估算的正确程度。下面的措施可以有效的减少测试估算的偏差：

- 尽快了解测试对象的架构和功能，因为测试估算的准确程度会随着对测试对象的了解程度的提高而不断提高；
- 及时的变更沟通，以保证测试估算的输入正确性和有效性；
- 测试的尽早介入，以提高测试执行的可预测性，例如：测试对象中的缺陷数目、再测试和回归测试的工作量等；
- 不断更新测试估算的测试活动或者测试任务检查表，以避免测试估算过程中出现遗漏；
- 更好的利用组织过程数据，以避免测试估算的乐观心态，例如：和上个项目的测试相比可以有更高的效率和有效性、可以更好的避免以前的差错、接受了以前的经验教训后可以更快的完成本项目的测试；
- 尽量让参与执行测试活动和测试任务的测试人员进行估算，并通过测试团队估算能力进行估算结果的优化；
- 避免管理层对测试估算结果的干扰；

9.3.4 测试估算出现偏差怎么办

测试估算出现偏差的时候，需要选择合适的应对措施使得整个测试过程处于可控状态。例如：整体的时间进度计划是 4 个月，完成第一个里程碑的测试任务的时间为 4 个星期，但是实际测试过程中完成第一个里程碑花费了 6 个星期。这种情况下可以采取的应对措施包括：

- 第 1 个选项是要求测试人员在后续的进度中弥补延期的这 2 个星期；
- 第 2 个选项是在整个测试进度计划中增加 2 个星期；
- 第 3 个选项是在整个测试进度计划中都考虑该延期，即增加 50%的时间；
- 第 4 个选项是裁剪部分优先级较低的测试任务，或者将它们放到版本发布后再进行。

第10章 测试风险

只要有可能发生某些问题，使客户、用户、参与者或利益相关者减少对产品质量或项目成功的信心，即可认为存在风险。不同的风险可能会有不同的风险级别，它由下面两个因素决定：问题发生的可能性（风险的可能性）和问题发生后的影响程度（风险的严重程度）。

10.1 风险基本概念

10.1.1 管理风险

管理风险指的是影响项目成功的潜在问题，它会对软件产品的及时交付构成威胁，例如：项目失败、项目延期、项目成本的严重超支等。和测试相关的管理风险有：

- 在测试执行之前测试环境和测试工具没有准备就绪；
- 具备必要技能的测试人员无法及时到位；
- 测试范围或者测试对象范围定义的频繁变更；
- 供应商方面的风险，例如：测试团队采购的测试仪表不能及时到货、分包商无法及时交付产品；

10.1.2 质量风险

质量风险指的是影响测试对象质量的潜在问题，例如：产品存在可靠性的缺陷可能会导致系统运行过程中出现重启。常见的质量风险有：

- 软件产品的可靠性差；
- 使用软件产品过程中会导致人员的伤亡；
- 软件产品没有实现预期的功能导致用户无法使用；
- 交付的软件产品功能不完善；

10.2 风险管理过程

风险管理的主要目的是通过管理软件开发生命周期中的风险，减少对测试目标造成不利影响的事件发生的可能性及严重程度。风险管理过程是系统的、主动的、持续的过程。

10.2.1 风险识别

风险识别主要是来确定在软件开发生命周期中存在哪些风险，并以文档的形式进行记录。根据不同的组织和项目特征，可以采用不同的风险识别方法，主要包括：

- 专家咨询；
- 独立评估；
- 风险模板；

- 经验教训（例如：项目评估会议）；
- 风险研讨会（例如：失效模式和影响分析）；
- 头脑风暴法；
- 风险分类（或检查表）；
- 过去的经验；
- 问卷调查法；

10.2.2 风险分析

风险分析通过研究被识别出的风险，确定每个风险的可能性及严重程度，并得到风险级别。风险级别是风险可能性和风险严重程度的乘积。风险分析既可以是定性的也可以是定量。

风险分析过程中，确定每个风险发生的可能性和严重程度是其主要的活动。影响风险发生可能性的因素包括：

- 技术和团队的复杂性；
- 业务分析人员、设计人员和程序员的技能；
- 团队内部矛盾；
- 与供应商的合同问题；
- 开发团队的地理分布；
- 老方法和新方法对立；
- 采用的工具和技术；
- 不良的管理领导和技术领导；
- 时间、资源和管理压力；
- 缺乏初期的质量保证；
- 频繁的变更；
- 初期的高缺陷率；
- 接口和集成问题；

风险发生的严重程度是指当风险发生后，对用户、客户或其他利益相关者影响的严重性。影响风险发生的严重程度的因素包括：

- 使用受影响的功能的频率；
- 对组织形象的损害；
- 商业损失；
- 潜在的金融、生态或社会方面的损失或法律责任；
- 民事或刑事法律制裁；

- 缺少合理的变通；
- 明显的失效导致负面宣传；

10.2.3 风险应对

根据风险分析得到的风险级别，需要对风险采取合适的应对措施。经常采用的风险应对策略包括：

- 风险减轻：通过降低风险可能性或者严重程度（或者同时降低可能性和严重程度）就可以降低风险的级别；
- 风险避免：将风险可能性或者严重程度降低为 0 就可以达到风险避免的目的；
- 风险转移：将风险转移到第三方，例如：项目的某个功能模块，项目成员没有相关的开发经验，为了保证质量和进度，将该功能模块的开发外包给了第三方，这实际上就采取了风险转移的策略；
- 风险接受：识别的风险经过分析和评估之后，认为风险的级别在可以接受的范围之内（例如：在组织定义的风险阈值之内），这个时候就可以接受风险；或者针对风险进行应对的成本，比不进行应对的成本还要高，这个时候，也可以接受风险；
- 应急计划：当风险变成现实，即风险的可能性达到了 100%，这个时候需要相应的应急计划，降低事件产生的影响程度；

10.2.4 风险监控

风险监控的目的包括：

- 评审并更新单个风险的状态和风险管理环境；
- 评估风险应对的有效性；
- 识别新的风险以及分析引起该风险的原因；

风险监控是对风险状态的变化过程进行跟踪和管理。经常采用的风险状态有：

- 新的：新的风险加入到风险列表中，还没有进行风险的分析和应对；
- 正在进行的：对风险进行了分析，并制订了风险的应对计划；
- 关闭的：风险应对活动已经完成；
    - 接受的：风险已经被接受，不再进行风险的应对活动；
    - 风险成真的：风险变成了事件。需要采取应急计划处理和控制该事件；
    - 陈旧的：由于项目的变化或者一些条件的变化，风险已经过时，或者不再存在；
    - 转移的：风险已经转移给第三方；

10.3 测试风险实践

10.3.1 如何选择测试重点

- 测试对象的测试任务总是非常庞大的，而测试时间却是非常有限的。为了提高测试的有效性和效率，基于风险选择测试重点和安排测试优先级将是一个很好的选择。其核心思想是：针对每个测试对象的功能或者特征，根据潜在风险列表进行风险评估，然后将潜在的风险列表评估的数值，按照某种合理的算法进行计算，即可得到该功能或者特征的风险级别，从而得到其测试重点，并基于此选择合适的测试策略。

10.3.2 基于风险开展测试活动

测试不仅可以发现测试对象当前存在的风险，也可以帮助降低风险的不确定性，有助于风险评估和识别新的风险。基于测试对象的风险可以有效的开展测试活动：

- 首先，根据质量风险的级别，分配测试资源、选择测试技术和方法、安排测试活动顺序以及修复缺陷；
- 其次，根据管理风险的级别，计划和监控测试工作以有效实现风险的减轻和缓解；
- 第三，根据剩余风险进行测试结果和项目状态的报告；

10.3.3 风险与测试过程的集成

风险活动应该贯穿于整个测试生命周期。测试计划定义了质量风险和管理风险之后，后续的测试活动将基于这些风险进行迭代的风险管理活动，包括识别新的风险、重新评估风险级别和评估风险应对活动的有效性等。

同时，测试人员通过分析遗留的风险以及风险级别，可以评估测试对象的质量并帮助确定测试对象的发布时间。测试过程中可以运用各种技术和方法控制风险，包括：

- 评审软件工作产品；
- 选择合适的测试设计技术；
- 不同的测试级别采用不同的测试独立性策略；
- 最有经验的人员执行最重要的测试任务；
- 再测试和回归测试的策略选择；

10.3.4 如何应用测试风险

采用基于风险的测试，可以有效的指导测试活动的展开，更好的使测试活动在进度、成本、质量等方面进行平衡，从而提高测试质量、降低测试成本、缩短测试时间等。基于风险的测试的主要优点体现在：

- 确定测试重点和优先级；
- 确定测试的完备性；
- 确定测试资源的分配；

- 易于测试进度的监控；
- 加速测试信心的提升；

10.3.5 风险管理的挑战

基于风险的测试同样也存在一些挑战，主要包括：

- 测试风险分析自身的风险；
- 完成测试之后的风险；
- 测试风险列表不清晰；
- 项目成员不愿意谈论风险；
- 风险级别难以达成一致；
- 风险管理仅仅依靠管理团队；

第11章 测试度量

没有度量就没有控制，度量是整个软件测试监控的基础。

11.1 测试度量的意义

11.1.1 测试过程监控

测试过程监控通过监视当前的测试状态和测试结果，与测试计划定义的准则进行比较，并在合适的时候采取合适的测试控制方法以降低测试的实际情况与测试计划之间的偏差。而度量是测试过程监控的基础，无论是静态测试还是动态测试，都可以通过度量提供的信息对它们进行监控。

11.1.2 为决策提供数据支持

度量可以帮助相关人员评估测试的入口和出口准则。实施度量之后，测试的入口准则和出口准则可以变得更加清晰，例如：集成测试入口准则的条目之一是单元测试的代码覆盖率达到100%。通过对单元测试中的代码覆盖率的度量，可以更加清晰地定义和评估集成测试的入口准则。

11.1.3 测试过程改进

度量本身并不能改进测试过程，但是测试过程中收集的度量数据可以作为测试过程改进的输入，通过对这些数据的分析和评估，可以帮助实施测试过程改进。

11.1.4 测试团队激励

度量还可以达到激励测试团队的目的。度量能够以直观和量化的方式反映测试团队的绩效，使得测试团队的工作能够得到项目团队中其他成员的认可。

11.1.5 组织能力数据

测试活动中碰到的很多问题都依赖于度量，如测试团队的生产率、测试对象的质量等。同时度量在发现问题和诊断问题也都有着不可替代的作用。分析度量数据不仅可以发现测试过程中的问题，也可以有效地诊断测试团队遇到的问题。通过不断积累度量数据，逐步形成组织的度量数据库，并作为将来项目测试活动的重要输入。

11.2 测试度量过程

11.2.1 测试度量计划

测试度量计划将统筹安排整个测试度量活动，并指导度量的实施和评估。度量数据库中的信息和评估结果是测试度量计划的重要输入，同时它们对当前的测试度量计划有重要的指导意义。测试度量计划的主要任务包括识别信息需求，选择度量，定义度量数据的收集、分析和报告过程，定义信息产品和度量过程评估准则，以及评审/批准/提供度量任务的资源。

11.2.2 测试度量实施

测试度量实施的主要活动是实施测试度量计划中的各种任务，并形成最终的信息产品。实施测试度量的过程中需要考虑和应用度量数据库中的信息产品和评估结果。测试度量的实施主要包括收集数据、分析数据并获得信息产品以及沟通结果。

11.2.3 测试度量评估

测试度量评估主要包括两个方面的任务：评估信息产品和度量过程、识别改进点。

评估信息产品和度量过程是根据具体的评估准则对信息产品和度量过程进行评估，了解整个信息产品的优缺点以及度量过程的有效性。评估的输入是度量过程的各个活动信息、最终的信息产品以及度量用户的反馈意见。具体的度量评估准则来源于测试度量计划。

识别改进点可以分别从信息产品和度量过程入手，分别得到信息产品的改进点（例如：信息产品对应的度量指标的调整、产品规模的计算从代码规模换成功能点规模、缺陷类型的重新定义等）和度量过程改进点（例如：将度量过程集成到项目过程中是否可以改进、各个度量角色的培训的质量改进等）。

11.3 测试度量技术

测试度量指标需要通过合适的测试度量技术来获取，它是测试度量过程中的一个重要的内容。测试度量指标的确立能够解决“度量什么”的问题。

11.3.1 自底向上

自底向上的方式适合建立一个完整的测试度量系统，可以在 PSM、CMMi 或者其他系统的分类方法的基础上，构建属于自己团队的度量指标集合。使用该方法可以快速而全面构建测试团队的度量指标集合。

PSM（Practical Software and Systems Measurement）可以用来解决软件和系统有关技术和管理的挑战。PSM 是根据美国政府、国防部和企业界实际的度量经验发展而成的，它认为度量是一个弹性的过程，而不是预先定义好的一系列的图标和报告，它把度量需求集成到软件和系统供应商的流程中去。PSM 定义了一个信息驱动的分析方法，它帮助项目经理在软件和系统方面作出正确的决策。

CMMI（Capability Maturity Model Integration）， 即软件能力成熟度模型集成，是由美国国防部与卡内基-梅隆大学和美国国防工业协会共同开发和研制的。CMMI 是一套融合多学科的、可扩充的产品集合， 其研制的初步动机是为了利用两个或多个单一学科的模型实现一个组织的集成化过程改进。CMMI 分为 5 个等级，22 个过程域（PA）。这些过程域可以作为选择测试度量指标的基础。

11.3.2 自顶向下

自顶向下的方式适合目标比较明确的度量，例如：需要分析团队的工作负荷是否合适。此时目标明确，那么采用 GQM(GQM：goals-questions-metrics)的方法可以更快捷准确的选择合适的度量指标。

GQM 由德国马里兰大学(University of Maryland)的巴士利博士(Dr.Victor Basili)及其助手提出，用以告诉组织或者机构应该采集哪些数据。Dr.Victor Basili 认为度量应该采用从上至下的形式展开，以度量的目标和模型为基础。

GQM 的过程主要由计划阶段、定义阶段、收集数据阶段和解释阶段四个阶段组成。

11.4 测试度量实践

11.4.1 如何选择合适的测试度量指标

测试团队经常会遇到如何选择度量指标的问题。由于公司文化、团队结构、项目类型和开发过程等各不相同，很难有一个放之四海而皆准的统一的测试度量指标的集合。但是通过分类总结不同的度量指标，可以有效地帮助测试团队选择合适的度量指标。其中自底向上测试度量技术ＰＳＭ和ＣＭＭＩ与自顶向下的ＧＱＭ中提供的测试度量指标，是测试团队选择度量指标的非常重要的输入。在测试度量指标选择过程中，也可以将自底向上和自顶向下两种技术进行有效的结合。

11.4.2 如何综合应用度量数据

尽管有时候单个度量数据就可以达到目的，但是在很多情况下，准确全面的掌握所需的信息需要多个度量数据的配合。例如：判断测试是否充分的时候，通常需要综合分析测试用例执行通过率、测试执行的需求覆盖率、测试发现的缺陷密度、测试发现的缺陷按触发原因分类和缺陷发现数量的收敛情况等度量数据。

11.4.3 效率测试和相关度量

效率指的是在规定条件下，相对于所用的资源的数量，软件产品可提供适当性能的能力。资源可能包括其他软件产品或系统的软件和硬件配置，以及其他相关的资源（例如：打印纸、磁盘等）。效率测试主要关注产品的时间和资源相关的特性。时间相关的特性是指，在规定条件下，软件产品执行其功能时，提供适当的响应和处理时间以及吞吐量的能力，例如：用户打开某个网页需要等待的时间；资源相关的特性是指，在规定条件下，软件产品执行其功能时，使用合适数量和类别的资源的能力，例如：用户在进行相关操作时，系统的内存和 CPU 的变化情况。

11.4.4 如何保证度量数据的正确性

测试和测试过程是可以度量的，尽管存在各种困难。正确应用度量（例如：通过图表、数据等方式）可以提供足够的信息来帮助测试过程的监控，以及项目开发过程和测试过程的改进。针对测试过程的度量也可以为项目的其他活动提供信息。同时，通过定期对度量的信息进行收集、分析、更新和总结，度量可以帮助判断相关活动的趋势。为了保证度量数据的正确性以更好地提高测试过程的效率和有效性，在度量应用过程中需要掌握以下几个基本原则：

- 简单易收集；
- 可测量；
- 目的明确；
- 不针对个人；
- 完善的度量过程；

第12章 测试过程监控

测试过程监控应该贯穿整个测试生命周期以有效地监视和控制每个测试阶段，包括测试过程监视和控制两个方面。测试过程监视的主要目的是获取当前测试的状态和测试结果，例如：测试执行进度、产品的质量信息等。监视过程中需要以合适的度量方式呈现其中的信息，主要包括：

- 测试进度；
- 产品质量；
- 资源和成本；
- 团队能力；

12.1 测试进度

12.1.1 测试设计进度

测试用例设计进度主要用来控制测试设计是否按照计划的时间进度要求进行，其监控的主要参数包括总的测试用例数目、原来计划的测试用例数目、实际计划的测试用例数目、实际设计的测试用例数目等。

12.1.2 测试执行进度

测试用例执行进度主要用来控制测试执行是否按照计划的时间进度要求进行，其监控的主要参数包括总的测试用例数目、原来计划的测试用例数目、实际计划的测试用例数目、实际执行的测试用例数目等。

12.1.3 自动化测试脚本实现进度

自动化测试脚本实现进度主要用来控制自动化测试脚本是否按照计划的时间进度要求进行，其监控的主要参数包括总计划的自动化的测试用例数目、原来计划的自动化测试用例数目、实际计划的自动化测试 用例数目、当前实现的自动化测试用例数目等。

12.2 产品质量

12.2.1 测试用例执行通过率

测试用例执行通过率随着时间的变化可以帮助说明测试执行的状态和软件产品的质量情况。该趋势图包括成功执行的测试用例数目、失败的测试用例数目和被阻塞的测试用例数目。该指标常常用作测试执行出口准则的一个要求，例如：测试用例执行通过率必须达到95%以上。

12.2.2 缺陷发现趋势

测试执行过程中发现的累计缺陷数目的趋势可以为测试过程监控提供详细的输入，其中累计的缺陷包括新发现的缺陷数目、正在修复的缺陷数目、已经解决待验证的缺陷数目和已经关闭的缺陷数目等。对于质量趋于稳定的软件产品而言，其测试发现缺陷状态变化的趋势图应该是处于收敛状态的。

12.2.3 缺陷密度

缺陷密度可以用来判断产品的质量，同时也有助于判断测试的充分性。通常缺陷目的可以表示为千行代码发现的缺陷数量。在一个成熟的团队中，每千行代码的缺陷数目是在一个比较稳定的区间中。

12.2.4 缺陷按照各种属性的划分

- 缺陷发现阶段分布：基于软件开发生命周期不同阶段发现的缺陷数目的统计分布，通过它可以清晰地了解各个阶段发现的缺陷数目，从而分析开发过程和测试过程的有效性。主要包括的参数有缺陷发现的目标值、实际发现的缺陷数目、不同的阶段。
- 缺陷按不同功能模块的分布：可以用来表示在某个测试阶段在不同软件功能模块中发现的缺陷数目，以此为基础确定后续开发和测试的重点。
- 缺陷按测试类型分布：可以用来表示在某个测试阶段，针对软件进行的不同测试类型发现的缺陷数目。根据发现缺陷的测试类型的分布，结合执行的测试用例数目等数据，可以分析开发过程和测试过程中可能存在的问题，并帮助决定后续开发和测试的重点。

12.3 资源和成本

12.3.1 测试人员工作量分布

测试人员在不同的测试阶段（测试计划、测试设计、测试执行、测试结束活动和测试监控）的工作分布的监控，有利于及时发现人力资源的分配是否合理，及时对测试人员工作量进行调整。也可以针对测试人员在不同模块上的工作量分布进行监控。

12.3.2 测试硬件设备有效性和利用率

测试活动越来越越依赖于硬件环境。对测试硬件设备的有效性和利用率的监控，有利于测试团队更早的发现测试硬件设备是否充足，有利于最大化的利用测试硬件资源。

12.4 团队能力

12.4.1 测试设计的效率

测试设计效率的主要指标是单位时间能够设计的测试用例的数量。例如：每人天能够设计 2 个测试用例。

12.4.2 测试执行的效率

测试执行效率的主要指标是单位时间能够执行的测试用例的数量。例如：每人天能够执行 5 个测试用例。

12.4.3 评审的效率

评审的效率根据被评审对象的不同，相应的指标也会有所差异。当被评审对象是测试计划时，可以用每人天评审的页数来衡量；对于评审效果的评估，还可以监控每页测试计划发现的缺陷数。

12.4.4 自动化测试效率

自动化测试效率的监控可以针对自动化测试用例的设计、编码和执行等分别进行，可以考虑使用每人天的测试用例数作为指标。

12.5 测试过程监控实践

测试过程控制的主要目的是比较当前的测试状态和结果与测试计划之间的差距，例如：测试进度、测试质量等与出口准则之间的比较，并在合适的时候采取合适的测试控制方法以降低测试的实际情况与测试计划之间的偏差。

12.5.1 获取额外的测试资源

在时间压力大或者测试工作量明显大于当前测试资源的时候，首先想到的是如何获取更多的测试资源，例如：让更多的人参与测试工作、提供更多的测试设备和测试平台等。

12.5.2 改变测试范围和测试优先级

在规定时间内无法完成测试任务的时候，改变测试范围和测试优先级是经常采用的一种策略。该策略的关键是如何确定测试对象中的不同测试重点，并按照测试重点的不同，可以采用广度优先或者深度优先的测试策略继续后续的测试工作。

12.5.3 变更测试出口准则

评估测试进度、测试对象质量与出口准则之间差距的过程中，由于现实的条件限制等原因无法达到出口准则的要求，而市场要求或者商业机会等又不得不发布软件产品，这时候就可能采取变更测试出口准则的策略。在该过程中，变更测试出口准则导致的风险是必须面对的问题。

12.5.4 推迟测试结束的时间

测试任务无法及时完成，还可以采用推迟测试结束的时间。这是应该比较方便的策略，但是这会导致测试成本的增加，甚至市场机会的散失。所以作为该策略的折中的方案：测试人员常常被要求加班，以弥补测试时间上面的不足。

第13章 测试过程改进

13.1 测试过程改进概述

13.1.1 戴明改进循环 PDCA

分析和吸取以前的经验教训，有助于组织的过程改进，而软件开发和测试也能在该过程中受益。戴明改进循环 PDCA，即“戴明环” 已经在业界使用了多年，它由计划（Plan）、实施（Do）、检查（Check）和行动（Act）4 个阶段组成。目前，PDCA 仍是测试人员进行过程改进的首选。

戴明改进循环 PDCA 的 4 个阶段不是运行一次就结束的，而是需要周而复始的进行，一个循环完成后，可以解决一些问题，而未解决的问题则进入下个循环，阶梯式上升。戴明改进循环 PDCA 是有效进行任何一项工作的合乎逻辑的工作程序，因此有人称其为质量管理的基本方法。

13.1.2 测试过程改进步骤

基于戴明改进循环 PDCA，测试过程改进的步骤如下：

- 过程改进计划和准备：明确改进目标和范围，明确改进的方法，定义改进参与人员的角色和职责，制订相应的进度计划；
- 组织测试过程评估：评估主要包括数据收集和数据分析两个活动；
- 制定改进行动计划：根据发现的改进点，制定过程改进行动计划；
- 实施测试过程改进：根据制定的行动改进计划，召集相关的过程改进人员，实施测试过程改进；
- 测试过程改进反馈：评估过程改进的实施效果，总结经验教训，将未完成的改进目标纳入后期的改进计划；

13.2 测试过程改进模型

13.2.1 TMMi

TMMi(Test Maturity Model integration)是一种用于测试过程改进的模型，它可以作为 CMMi 有效的补充。TMMi 的理论原理和 CMMi 一样，通过成熟度级别进行过程的评估和改进。测试成熟度的演化，反映了测试组织经历的不同的阶段，从调试和探测为导向的测试阶段到预防为导向的阶段。另外，许多软件行业中的最佳测试实践为 TMMi 提供了很多实用的信息。

TMMi 有 5 个成熟度等级组成，它们分别反映了不同的测试过程成熟度，每个等级包括了特定的成熟度目标（初始级除外）。TMMi 的 5 个等级，可以帮助测试组织确定组织当前所处的测试过程成熟度级别，同时可以明确下一步的改进目标，以达到更高一级的测试成熟度等级。

13.2.2 TPI

测试过程改进 TPI（Test Process Improvement）提供了测试过程的一个框架，是一种更加容易进行测试过程改进的方法。通过 TPI 可以确定测试过程的弱项和强项，同时 TPI 可以用来评估测试过程的成熟度等级和确定过程改进的测试活动。

TPI 认为测试过程是由一系列需要被评估的任务和活动组成的。因此，TPI 定义了 20 个关键域；为了评估测试过程，每个关键域对应一个特定的级别，每个关键域有多个等级组成；检查点用来评价测试过程的优缺点；每个等级中还包括了相应的建议以帮助测试过程提高到更好的水平。测试改进过程从低级别到高级别的转变，需要综合衡量测试时间、测试成本和测试质量等因素。

13.3 测试过程改进实践

13.3.1 合适的测试过程改进目标

合适的测试过程改进目标是进行测试过程改进的前提。合适的改进目标至少需要满足：

- 目标明确：明确的目标有利于更好的集中各方面的力量用于过程改进，避免资源的浪费；
- 可行性：改进目标必须具备可行性，必要的情况下可以将宏大的目标拆分成多个小的目标；
- 和组织目标相结合：测试过程改进的目标要与整个组织的改进目标相结合，使得测试过程更好的融入整个开发生命周期；

13.3.2 是否需要采用模型

测试过程改进模型可以系统的指导测试过程改进活动，并根据一些指标量化过程改进的成果，但是测试过程改进模型并不一定是必需的。基于模型的测试过程改进通常会带来投入增加，并且使得过程改进过于复杂。尽管模型本身是很多业界经验的总结，但是每个组织和团队都有自身的特点，因此模型很难完全满足一个组织的测试过程改进目标。所以测试过程改进过程中，应该根据组织和测试团队的特点，确定选择测试过程改进的模型，或者自己分析定义改进点，或者两者兼而有之。

13.3.3 测试过程改进和项目过程改进的集成

通过测试过程改进能够提高测试团队的能力，但是仅从测试的角度开展测试过程改进活动很难达到理想的效果。软件测试过程是软件开发生命周期的重要组成部分，测试过程改进不仅受到开发活动的影响，同时也受到组织结构、文化和团队成员的能力的约束。因此，测试过程改进目标应该与组织或者项目的过程改进目标相结合，以达到两者的有效集成。

参考文献

[01] A.H.Maslow.动机与人格.许金声等译.北京：华夏出版社，1987。

[02] Adler, M.J.,Van Doren, C., How to Read a Book,Touchstone,1972。

[03] ANDREAS SPILLNER, TILO LINZ, HANS SCHAEFER. SOFTWARE TESTING FOUNDATIONS.北京：人民邮电出版社, 2008-4-1。

[04] B. Beizer, Software Testing Techniques(Second Edition), Van Nostrand Reinhold Company Limited,1990。

[05] B. Littlewood, Software Reliability: Achievement and Assessment，Blackwell Scientific Publications， November 1987。

[06] B. M. Subraya, Integrated Approach to Web Performance Testing: A Practitioner's Guide, IGI Global,2006。

[07] Bill Hetzel, The Complete Guide to Software Testing(2 edition), John Wiley & Sons, September 1993。

[08] Cem Kaner,Jack Falk,Hung QuocNguyen，王峰等译《计算机软件测试》[第 2 版]。

[09] Chris Ford,Ido Gileadi,Sanjiv Purba,Mike Moerman, Patterns for Performance and Operability: Building and Testing Enterprise Software, Auerbach Publications,2008。

[10] Clifton A. Ericson，Hazard Analysis Techniques for System Safety，John Wiley & Sons，2005。

[11] Gerald M, Weinberg, Quality Software Management Volume 1: Systems thinking, Dorset House Publishing,1992。

[12] Glenford J.Myers，王峰等译,《软件测试的艺术》机械工业出版社 2006。

[13] Hung Q. Nguyen,Testing Applications on the Web: Test Planning for Mobile and Internet-Based Systems(Second Edition),John Wiley & Sons,2003。

[14] I. Burnstein, A. Homyen, R. Grom, C. R. Carlson, A Model for Assessing Testing Process Maturity,CrossTalk: Journal of Department of Defense Software Engineering, Vol. 11, No. 11, Nov., 1998, pp. 26-30。

[15] IEC 60812 failure mode and effects analysis (FMEA)。

[16] IEC 61025 Fault tree analysis (FTA)。

[17] IEEE 1008-1987 IEEE Standard for Software Unit Testing。

[18] IEEE 1028-2008 Peer Review。

[19] IEEE 1044 -1993 Incident Management。

[20] IEEE 1059-1993 IEEE Guide for Software Veri?cation and Validation Plan。

[21] IEEE 828-2005 Software Configuration Management Plan。

[22] IEEE 829-2008 Standard for Software Test Documentation。

[23] IEEE Std 1061 1998 Software Quality Metrics Methodology。

[24] Ilene Burnstein，Practical Software Testing: A Process-Oriented Approach，Springer，2003。

[25] Ilene Burnstein，Taratip Suwannasart，C.R. Carlson，Developing a Testing Maturity Model, Crosstalk， 1996。

[26] ISO/IEC 16085-2006 Risk Management。

[27] ISO/IEC 9126-1-2001 Quality Model。

[28] ISO/IEC TR 19760-2003 Risk Management Process。

[29] James A. Whittaker,Herbert H. Thompson,How to Break Software Security,Addison Wesley,2003。

[30] James Bach,Session-Based Test Management,Software Testing and Quality Engineering magazine, 11/00。

[31] Johann Rost，The Insider's Guide to Outsourcing Risks and Rewards，Auerbach Publications，2006。

[32] Jonathan Bach，Session-Based Test Management，Software Testing and Quality Engineering magazine, 11/00。

[33] Koomen，Pol，TEST PROCESS IMPROVEMENT：A PRACTICAL STEP-BY-STEP GUIDE TO STRUCTURED TESTING (FIRST EDITION).北京：高等教育出版社，2003。

[34] Larry R. Williams，Keep'em Motivated: A Practical Guide to Motivating Employees，Marshall Cavendish，2003。

[35] Lee Copeland, A Practitioner’s Guide to Software Test Design, Artech House, 2004。

[36] Len Sandler, Becoming an Extraordinary Manager: The 5 Essentials for Success, AMACOM, 2008。

[37] Lydia Ash,The Web Testing Companion: The Insider's Guide to Efficient and Effective Tests,John Wiley & Sons,2003。

[38] Mauro Pezz, Michal Young, Software Testing and Analysis: Process, Principles and Techniques, John Wiley & Sons, 2008。

[39] MO JAMSHIDI,SYSTEM OF SYSTEMS ENGINEERING: innovations for the 21st century,A John Wilsy &Sons,2009。

[40] Mo Jamshidi，System-of-Systems Engineering - a Definition，IEEE SMC，October 2005。

[41] N. Fenton and S. Pfleeger，Software Metrics.2nd ed.， PWS Publishing，1997。

[42] Penny Grubb, Armstrong A. Takang , Software Maintenance: Concepts And Practice(Second Edition), World Scientific Publishing Co, 2003。

[43] Qin Liu, Wenqiang Zheng, JunFei Ma, Improving Test Quality by a Test Type Analysis Based Method,ssiri, pp.325-328, 2009 Third IEEE International Conference on Secure Software Integration and Reliability Improvement, 2009。

[44] Rex Black, Advanced Software Testing Vol.1, Rocky Nook, 2008。

[45] Rex Black.核心测试过程：计划、准备、执行和完善.李华隽译.北京：中国电力出版社，2007。

[46] Rick D. Craig ,Stefan P. Jaskiel,Systematic Software Testing,Artech House,2002。

[47] Royce W., Managing the Development of Large Software Systems: Concepts and Techniques, Proc. IEEE WESCON, 1970。

[48] Sagar Naik ,Piyu Tripathy,Software Testing and Quality Assurance: Theory and Practice,Wiley-Spektrum,2008。

[49] SCAMPI Upgrade Team，Appraisal Requirements for CMMI（Version 1.2），www.sei.cmu.edu/cmmi，Augest 2006。

[50] Steven M. Bragg, Cost Accounting: Comprehensive Guideby,John Wiley & Sons,2001。

[51] Tom Alexander, Optimizing and Testing WLANs: Proven Techniques for Maximum Performance, Newnes, 2007。

[52] W.Richard Stevens，范建华等翻译《TCP/IP 详解 卷 1：协议》机械工业出版社 2000。

[53] Wilbert O. Galitz ,The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques(Third Edition),John Wiley & Sons,2007。