ISTQB®软件测试人员认证

高级教学大纲
技术测试分析师

2019 版本

国际软件测试认证委员会

英文版权声明

高级（教学大纲）工作组：Graham Bath （副组长），Rex Black，Judy McKay，KenjiOnoshi，Mike Smith（副组长），Erik van Veenendaal

中文版权声明

修订历史

版本	日期	备注
2012（英文版）	2012年10月19日	GA发布2012版本
2019 V1.0（英文版）	2019年10月18日	GA发布2019版本
20200620（中文版）	2020年6月20日	中文版本地化工作基本结束

目录

修订历史 3
目录 4
致谢 6
0. 大纲引言 7
0.1本教学大纲编写目的 7
0.2软件测试的高级认证测试人员 7
0.3可考核的学习目标和知识认知级别 7
0.4所需的经验 8
0.5高级技术测试分析师考试 8
0.6参加考试的要求 8
0.7认证的课程 8
0.8本教学大纲的详细程度 8
0.9本教学大纲的架构 9
1. 基于风险的测试中技术测试分析师的任务-30分钟 10
1.1简介11
1.2 基于风险的测试任务 11
1.2.1 风险识别 11
1.2.2 风险评估 11
1.2.3 风险缓解 12
2. 白盒测试技术 -345分钟 13
2.1 介绍 14
2.2 语句测试 14
2.3 判定测试 15
2.4 改进的条件/判定（MC/DC）测试 15
2.5 复合条件测试 16
2.6 基础路径测试 17
2.7 API（应用程序接口）测试 18
2.8 选择一种白盒测试技术 19
3. 分析技术-210分钟 21
3.1 介绍 22
3.2静态分析 22
3.2.1 控制流分析 22
3.2.2 数据流分析 22
3.2.3 使用静态分析提高可维护性 23
3.2.4 调用图 24
3.3 动态分析 25
3.3.1 概述 25
3.3.2 检测内存泄漏 26
3.3.3 检测野指针 26
3.3.4 性能效率分析 27
4. 技术测试的质量特性——345分钟 28
4.1 简介 30
4.2 总体规划问题 31
4.2.1 利益相关方的需求 31
4.2.2 所需工具的获得和人员培训 32
4.2.3 测试环境的要求 32
4.2.4 组织方面的考虑 32

4.2.5 数据安全性方面的考虑 32
4.2.6 风险和典型缺陷 32
4.3 信息安全性测试 33
4.3.1 考虑信息安全性测试的理由 33
4.3.2 信息安全性测试计划 33
4.3.3 信息安全性测试规格说明 34
4.4 可靠性测试 35
4.4.1 引言 35
4.4.2 测量软件成熟性 35
4.4.3 容错性测试 36
4.4.4 易恢复性测试 36
4.4.5 可用性测试 37
4.4.6 可靠性测试计划 37
4.4.7 可靠性测试的规格说明 37
4.5 性能效率测试 38
4.5.1 性能效率测试类型 38
4.5.2 性能效率的测试计划 38
4.5.3 性能效率测试的规格说明 39
4.5.4 性能效率的质量子特性 39
4.6 维护性测试 40
4.6.1 静态和动态维护性测试 41
4.6.2 维护性子特性 41
4.7 可移植性测试 41
4.7.1 简介 41
4.7.2 易安装性测试 42
4.7.3 适应性测试 42
4.7.4 易替换性测试 42
4.8 兼容性测试 43
4.8.1 简介 43
4.8.2 共存性测试 43
5. 评审 – 165分钟 44
5.1 技术测试分析师的评审任务 45
5.2 在评审中使用检查表 45
5.2.1 架构评审 46
5.2.2 代码评审 46
6. 测试工具与自动化-180分钟 49
6.1定义测试自动化项目 50
6.1.1 选择自动化方法 50
6.1.2 自动化的业务流程建模 52
6.2 特定的测试工具 53
6.2.1 故障植入/故障注入工具 54
6.2.2 性能测试工具 54
6.2.3 基于网页测试的工具 55
6.2.4 支持基于模型测试的工具 55
6.2.5 组件测试和构建工具 56
6.2.6 支持移动应用测试的工具 56
7. 参考资料 58
7.1 标准 58
7.2 ISTQB®文档 59
7.3 书籍 60
7.4 其他参考资料 62
8. 附录A：质量特性概述 63

致谢

本文档由国际软件测试认证委员会高级工作组的核心团队编写：Graham Bath（副组长），Rex Black, Judy McKay, Kenji Onoshi, Mike Smith（组长），Erik van Veenendaal

以下人员参加了本大纲的评审、讨论和投票：

Dani Almog	Andrew Archer	Rex Black
Armin Born	Sudeep Chatterjee	Tibor Csöndes
Wim Decoutere	Klaudia	Melinda
Peter Foldhazi	David Frei	Karol Frühauf
Jan Giesen	Attila Gyuri	Matthias Hamburg
Tamás Horváth	N. Khimanand	Jan te Kock
Attila Kovács	Claire Lohr	Rik Marselis
Marton Matyas	Judy McKay	Dénes
Petr Neugebauer	Ingvar Nordström	Pálma Polyák
Meile Posthuma	Stuart Reid	Lloyd Roden
Adam Roman	Jan Sabak	Péter Sótér
Benjamin	Stephanie van	Paul Weymouth

核心团队感谢评审小组和各国分会提出的建议和意见。

该文档于 2019 年 10 月 20 日在 ISTQB® 成员国大会上正式发布。

参加本大纲翻译和评审的 CSTQB®专家有（按姓氏拼音排序）：
黄文萍、赖晶晶、李文鹏、吴洁、许爱国、羊婷婷、于长青、翟宏宝、郑丹丹、周震漪（组长）、左振雷
郑文强（终审）

0. 大纲引言

0.1 本教学大纲编写目的

本教学大纲根据国际软件测试认证委员会对高级大纲（技术测试分析师）的要求进行编写。

ISTQB®提供本教学大纲的主要目的：

1. 各国认证委员会需将教学大纲翻译成当地语言并分享给已获认证的培训机构。各国分会可根据其当地特定的语言对教学大纲进行适度润色、修改，以保证语句通顺可读。
2. 各国考试认证委员会可根据当地语言，按照高级教学大纲（技术测试分析师）的学习目标设计考题。
3. 培训机构需根据高级教学大纲（技术测试分析师）开发课件并选择最适当的教学方法。
4. 需要认证的考生可根据高级教学大纲准备考试（作为培训课程的一部分或独立使用）。
5. 在国际软件和系统工程领域促进软件和系统测试的专业化发展，并可以此教学大纲为基础著书和写文章。

ISTQB®允许其他组织、机构在获得书面授权后使用本教学大纲的内容。

0.2 软件测试的高级认证测试人员

高级资格认证由三个独立的教学大纲组成，并分别与以下角色有关：

- 测试经理
- 测试分析师
- 技术测试分析师

ISTQB®2019 高级教学大纲概述是一个单独的文档[ISTQB_AL_OVIEW]，其中包括以下信息：

- 商业价值
- 矩阵展示商业价值与学习目标之间的可追溯性
- 摘要

0.3 可考核的学习目标和知识认知级别

学习目标是商业价值的前提保证，并作为高级技术测试分析师认证考试题目编写的基础，以实现高级技术测试分析师的认证。

在每一章节的开始部分列出 K2、K3 和 K4 三个级别的具体学习目标，具体分类如下：

- K2：理解
- K3：应用

- K4：分析

0.4 所需的经验

部分针对技术测试分析师的学习目标需要以下领域的基本经验：

- 常规的编程概念
- 常规的系统架构概念

0.5 高级技术测试分析师考试

高级技术测试分析师的考试将以本教学大纲为基础。考试问题的答案可能需要使用基于本教学大纲多个部分的材料。除了引言和附录外，教学大纲的所有部分都是可考查的。标准、书籍和其他ISTQB®教学大纲都可作为参考资料，但它们的内容不在考试范围之内，除了本教学大纲中总结的内容。

考试的形式是多选题，总共 45 个问题。若要通过考试，至少需正确回答总分值的 65%或以上。

考试可以作为认证培训课程的一部分，也可以单独参加考试（例如：在考试中心或在公开考试中进行考试）。完成认证培训课程并不是参加考试的前提条件。

0.6 参加考试的要求

参加高级技术测试分析师认证考试前，应先获得基础级认证测试工程师的证书。

0.7 认证的课程

ISTQB®成员委员会可对认证的培训机构根据本教学大纲开发的课程课件进行认证。

培训机构应从成员委员会或机构获得认证指南。

认证的课程需符合本教学大纲，并允许 ISTQB®的考试作为课程的一部分。

0.8 本教学大纲的详细程度

本教学大纲的详细程度是考虑了在国际上能够采用符合统一要求的课程和考试，为实现这一目标，教学大纲包括：

- 描述高级技术测试分析师计划的通用教学目标
- 学生必须能够记住的术语列表
- 每个知识域的学习目标，描述通过学习后获得的商业价值
- 关键概念的描述，包括对公认的文献或标准之类的资料的引用。

本教学大纲的内容并不是对整个知识领域的描述；它反映了高级培训课程所涵盖的详细程度。它关注的是可以应用于任何软件项目以及使用于任何软件生命周期的内容。

本教学大纲不包含任何与特定软件开发生命周期或方法的任何特定学习目标，但它讨论了这些概念如何应用于敏捷项目、其他类型的迭代和增量生命周期，以及在顺序生命周期中的应用。

0.9 本教学大纲的架构

本教学大纲共有六章可考核的内容。每个章节的一级标题中给出了该章节的最短（教学）时间；一级标题以下未设定时间安排。对于经认证的培训课程，本教学大纲要求至少 21 小时 15 分钟的教学时间，并分布在以下六个章节：

- 第1章：基于风险测试中技术测试分析师的任务（30分钟）
- 第2章：白盒测试（345分钟）
- 第3章：分析技术（210分钟）
- 第4章：技术测试的质量特性（345分钟）
- 第5章：评审（165分钟）
- 第6章：测试工具与自动化（180分钟）

1. 基于风险的测试中技术测试分析师的任务-30分钟

关键词

产品风险（product risk）、风险评估（risk assessment）、风险识别（risk identification）、风险缓解（risk mitigation）、基于风险的测试（risk-based testing）。

基于风险的测试中技术测试分析师的任务的学习目标

1.2 基于风险的测试任务

TTA-1.2.1（K2）总结技术测试分析师需要考虑的典型风险因素。

TTA-1.2.2（K2）总结在基于风险方法的测试活动中与技术测试分析师有关的活动。

1.1 简介

测试经理全面负责建立和管理基于风险的测试策略，但测试经理通常会要求技术测试分析师的参与以确保正确实施基于风险的方法。

技术测试分析师在由测试经理为项目而制定的基于风险的测试框架中工作。他们贡献出与项目本身密切相关的技术产品风险知识，例如与信息安全性、系统可靠性和性能效率有关的风险。

1.2 基于风险的测试任务

由于技术测试分析师特殊的技术专长，他们积极参与以下基于风险的测试任务：

- 风险识别
- 风险评估
- 风险缓解

这些任务在整个项目中迭代执行，以应对新出现的产品风险和不断变化的优先级，并定期评估和传递风险状态。

1.2.1风险识别

在风险识别过程中越广泛的收集各类项目利益相关方的样本，能最大程度识别出重大风险的可能性也就越大。 因为技术测试分析师具有独特的技术技能，所以他们特别适合专家访谈，与同事头脑风暴，并分析当前和过去的经验来确定产品风险可能存在的区域，特别重要的是，技术测试分析师与其他利益相关方（例如：开发人员、架构师、运维工程师、产品负责人、本地支持办公室和服务台技术人员）紧密合作，以确定影响产品和项目的技术风险领域。

让其他利益相关方参与能确保所有的意见都得到考虑，并且通常由测试经理提供帮助。

技术测试分析师可能识别的风险通常基于第 4 章中列出的[ISO25010]质量特性，例如包括：

- 性能效率（例如，在高负载条件下无法达到所需的响应时间）
- 信息安全性（例如，通过安全性攻击泄露敏感数据）
- 可靠性（例如，应用程序无法达到服务级别协议中指定的可用性）

1.2.2风险评估

在风险识别阶段尽可能多的识别相关风险，而风险评估阶段是对那些已识别出的风险进行研究，以便对每个风险进行分类，并确定与之相关的可能性和影响程度。发生的可能性通常解释为在被测系统中存在潜在问题的可能性。

技术测试分析师致力于发现和了解每个风险项潜在的技术产品风险，而测试分析师致力于理解当问题发生时带来的潜在商业影响。

项目风险会影响项目的整体成功，因此，通常需要考虑以下典型项目风险：

- 利益相关方之间在技术需求方面的冲突
- 由于开发组织的地域分布而导致的沟通问题
- 工具和技术（包括相关技能）
- 时间、资源和管理压力
- 缺少早期的质量保证
- 技术需求变更频率高

产品风险因素可能会导致更多的缺陷，因此通常需要考虑以下典型产品风险：

- 技术的复杂性
- 代码结构的复杂度
- 新代码与复用代码的比例
- 发现大量与技术质量特性有关的缺陷（缺陷历史记录）
- 技术接口和集成问题

在获得风险信息后，技术测试分析师根据测试经理制定的指南设置初始风险级别。例如，测试经理可能决定将风险级别划分为 1 到 10 之间的值，其中值 1 为最高风险级别。当考虑了所有利益相关方的意见后，测试经理可以修改初始值。

1.2.3风险缓解

在项目中，技术测试分析师影响着测试如何响应已识别的风险，这个影响主要包括以下几个方面：

- 通过执行最重要的测试（针对高风险区域的测试），并采取测试计划中规定的适当的风险缓解和应急措施来降低风险
- 根据项目开展过程中收集到的额外信息评估风险，并使用这些信息实施缓解措施，以降低这些风险的可能性或规避这些风险的影响

技术测试分析师将经常与信息安全性和性能效率等领域的专家合作，以定义风险缓解措施和组织测试策略的要素。其他信息可以从 ISTQB®专业领域教学大纲中获得，例如，高级-安全性测试教学大纲[ISTQB_ALSEC_SYL]和基础级-性能测试教学大纲[ISTQB_FLPT_SYL]

2. 白盒测试技术 -345分钟

关键词

API（应用程序接口）测试（API testing）、原子条件（atomic condition）、控制流测试（control flow testing）、圈复杂度（cyclomatic complexity）、判定测试（decision testing）、改进的条件/判定测试（modified condition/decision testing）、复合条件测试（multiple condition testing）、路径测试（path testing）、短路（short-circuiting）、语句测试（statement testing）、白盒测试技术（white-box test technique）。

白盒测试学习目标

注意：学习目标中的 2.2.1，2.3.1，2.4.1，2.5.1 和 2.6.1 属于参考规格说明。这包括例如代码片段、需求、用户故事、用例和功能规格说明这样的条目。

2.2 语句测试

TTA-2.2.1（K3）通过应用语句测试技术，为给定的规格说明项编写测试用例，以达到定义的覆盖率水平。

2.3 判定测试

TTA-2.3.1（K3）通过应用判定测试技术，为给定的规格说明项编写测试用例，以达到定义的覆盖率水平。

2.4 改进的条件/判定测试

TTA-2.4.1（K3）通过应用改进的条件/判定覆盖(MC/DC)测试设计技术来编写测试用例，以达到定义的覆盖率水平。

2.5 复合条件测试

TTA-2.5.1（K3）通过应用复合条件测试技术来为给定的规格说明项编写测试用例，以达到定义的覆盖率水平。

2.6 基础路径测试

TTA-2.6.1（K3）通过应用McCabe的简化基线方法为给定的规格说明项编写测试用例。

2.7 API（应用程序接口）测试

TTA-2.7.1（K2）理解API（应用程序接口）测试的适用性和它发现的缺陷类型。

2.8 选择白盒测试技术

TTA-2.8.1（K4）根据给定的项目情况选择适当的白盒测试技术。

2.1 介绍

本章主要描述白盒测试技术。这些技术适用于代码和其他结构，例如业务流程图。

每种特定的技术都能够系统地生成测试用例，并将重点放在要考虑的结构的特定方面。这些技术提供了必须被度量并与每个项目或组织定义的目标相关联的覆盖率标准。实现完全覆盖并不意味着整个测试集是完整的，而是正在使用的技术不再建议对正在考虑的结构进行任何有用的测试。

本教学大纲考虑了以下技术：

- 语句测试
- 判定测试
- 改进的条件/判定测试
- 复合条件测试
- 基础路径测试
- API（应用程序接口）测试

基础级教学大纲[ISTQB_FL_SYL]介绍了语句测试和判定测试。语句测试执行代码中的可执行语句，而判定测试执行代码中的判定并测试基于判定结果执行的代码。

上面列出的改进的条件/判定测试和复合条件测试技术都基于判定谓词，并且大体上发现相同类型的缺陷。不管判定谓词有多复杂，它的值都将是真或假，这将决定代码执行的路径。当由于判定谓词没有求得预期值而未选择预期的路径时，就会检测到缺陷。

前四种技术依次更严密（基础路径测试比语句和判定测试更严密）；更严密的技术通常需要定义更多的测试以达到预期的覆盖率，并发现更细微的缺陷。

参考[Bath14]，[Beizer90]，[Beizer95]，[Copeland03]，[McCabe96]，[ISO29119]和[Koomen06]。

2.2 语句测试

语句测试执行代码中的可执行语句。覆盖率是测试执行的语句数除以测试对象中可执行语句的总数，通常用百分比表示。

适用性

对于所有被测试的代码来说，这个覆盖率应考虑作为最低要求。

局限/难点

没有考虑判定。即使语句覆盖的百分比很高，也可能无法检测代码逻辑中的某些缺陷。

2.3 判定测试

判定测试执行代码中的判定，并测试基于判定结果执行的代码。为了做到这一点，测试用例遵循从判定点发生的控制流（例如：对于 IF 语句，一个测试用例针对为真的结果，一个测试用例针对为假的结果；对于 CASE 语句，包括默认结果在内的所有可能的结果都需要测试用例）。

覆盖率是测试执行的判定结果的数量除以测试对象中判定结果的总数，通常用百分比表示。

与下面描述的改进的条件/判定测试和复合条件技术相比，判定测试将整个判定作为一个整体来考虑，并在单独的测试用例中分别评估为真和假的结果。

适用性

当被测试的代码是重要的甚至是关键的时候，应该考虑到这一层次的覆盖率（参见第 2.8 节中的表格）。

局限/难点

因为它可能比只在语句级别上测试需要更多的测试用例，所以当时间是一个问题时，它可能会有问题。判定测试不考虑多个条件如何组成判定的细节，可能无法检测由这些条件的组合引起的缺陷。

2.4 改进的条件/判定（MC/DC）测试

判定测试将整个判定作为一个整体来考虑，并在单独的测试用例中分别评估为“真”的结果和为“假”的结果。相比之下，改进的条件/判定测试考虑的是在包含多个条件的情况下如何做出判定（否则就是简单的判定测试）。

每个判定谓词由一个或多个简单的原子条件组成，每个原子条件的计算结果都是一个离散的布尔值。这些条件组合在一起，以确定判定的最终结果。该技术检查每个原子条件独立且正确地影响总体判定的结果。

当存在包含多个条件的判定时，这种技术提供了比语句和判定更强的覆盖。假设 N 个唯一的、独立的原子条件，改进的条件/判定测试通常可以通过 N+1 个不同的测试用例来实现。改进的条件/判定测试需要成对的测试用例，这些用例对表明单个原子条件可以独立地影响判定的结果。

在下面的例子中，考虑的是“If (Aor B) and C then…”这样一个语句。

	A	B	C	(A or B) and C
测试1	真	假	真	真
测试2	假	真	真	真
测试3	假	假	真	假
测试4	真	假	假	假

在测试 1 中，A 为真，总体结果为真。如果 A 变为假（在测试 3 中，保持其他值不变），结果将变为假，从而表明 A 可以独立地影响判定的结果。

在测试 2 中，B 为真，总体结果为真。如果 B 变为假（在测试 3 中，保持其他值不变），结果将变为假，从而表明 B 可以独立地影响判定的结果。

在测试 1 中，C 为真，总体结果为真。如果 C 变为假（在测试 4 中，保持其他值不变），结果将变为为假，从而表明 C 可以独立地影响判定的结果。

注意，与语句和判定测试技术不同，改进的条件/判定测试没有“确定的覆盖水平”；它要么是达到了（100%覆盖）或者没有达到。

适用性

航空航天工业和其他工业的安全关键系统适用于该技术。它用于测试可能由于软件失败而导致灾难的软件。

局限/难点

在具有多个条件的判定中存在一个变量出现多次时，达到改进的条件/判定测试覆盖可能会很复杂；当这种情况发生时，条件可能是“耦合的”。依赖于判定的情况，可能无法仅改变一个条件的值而使判定结果发生变化。解决这个问题的一种方法是，指定只有非耦合的原子条件必须测试到改进的条件/判定测试水平。另一种方法是逐个分析发生耦合的每个判定。

一些编程语言和/或解释器的设计使得它们在评估代码中的复杂判定语句时表现出短路行为。也就是说，如果只计算表达式的一部分就可以确定计算的最终结果，那么执行代码可能不会计算整个表达式。例如，如果评估判定“A 与 B”，如果 A 已经被评估为假，就没有理由评估 B。B 的值不能改变最终结果，因此代码可以通过不评估 B 来节省执行时间。短路可能会影响获得改进的条件/判定测试覆盖的能力，因为某些所需的测试可能无法实现。

2.5 复合条件测试

在极少数情况下，可能需要测试一个判定包含的所有原子条件的可能组合。这种彻底的测试称为复合条件测试。所需测试的数量取决于判定语句中的原子条件的数量，可以通过计算 2^N 来确定，其中 N 是不耦合原子条件的数量。使用与前面相同的例子，需要以下测试来实现复合条件覆盖：

	A	B	C	(A or B) and C
测试 1	真	真	真	真
测试 2	真	真	假	假
测试 3	真	假	真	真
测试 4	真	假	假	假
测试 5	假	真	真	真

测试 6	假	真	假	假
测试 7	假	假	真	假
测试 8	假	假	假	假

覆盖率通过测试执行的唯一条件组合的数量除以测试对象中条件组合的总数来度量，通常用百分比表示。

适用性

这种技术用于测试要求能长时间可靠地运行而不崩溃的嵌入式软件（例如，期望使用 30 年的电话交换机）。

局限/难点

因为测试用例的数量可以直接从包含所有原子条件的真值表中得到，所以可以很容易地确定这个覆盖率水平。然而，所需的庞大的测试用例数量使得改进的条件/判定测试覆盖在大多数情况下更加可行。

如果编程语言使用短路，实际测试用例的数量通常会减少，这取决于对原子条件执行的逻辑操作的顺序和分组。

2.6 基础路径测试

路径测试通常包括识别代码中的路径，然后创建测试来覆盖它们。从概念上讲，测试系统中的每个唯一路径是有用的。然而，在任何实际的系统中，由于循环结构的性质，测试用例的数量可能会变得非常大。相比之下，遵循 McCabe 开发的简化基线方法进行基础路径测试是现实可行的[McCabe96]。

该技术的应用步骤如下：

1. 为给定的规格说明项（例如，代码或功能设计规格说明）创建控制流图。注意，这也可能是执行控制流分析的第一步（参见3.2.1节）。
2. 在代码中选择一条基线路径（而不是异常路径）。这个基线路径应该是最重要的需测试路径 — 可以基于风险做出该判断。
3. 通过改变路径上第一个判定的结果来生成第二个路径，同时保持最大数量的判定结果与基线路径相同。
4. 通过再次从基线路径开始并改变路径上的第二个判定的结果来生成第三条路径。当遇到多路判定（例如，case语句）时，应该在执行下一个判定之前执行当前判定的每个结果。
5. 通过改变基线路径上的每个结果来生成进一步的路径。当遇到新的判定时，首先应该执行最重要的结果。
6. 一旦覆盖了基线路径上的所有判定结果，就可以对后续路径应用相同的方法，直到执行了规格说明项中的所有判定结果。

适用性

上面定义的简化基线方法通常在任务关键型软件上执行。它是本章所涉及的其他方法的一个很好的补充，因为它关注的是贯穿软件的路径，而不仅仅是判定的方式。

局限/难点

截至教学大纲发布时，对基础路径测试的工具支持是有限的。

覆盖率

上面描述的技术应该确保完全覆盖所有线性无关的路径，并且路径的数量应该与代码的圈复杂度相匹配。根据代码的复杂性，使用工具检查路径的基本集合是否已经完全覆盖可能是有用的。覆盖率通过测试执行的线性无关路径的数量除以测试对象中线性无关路径的总数来度量，通常用百分比表示。增加数量相对较小的测试，基础路径测试就提供了比判定覆盖更全面的测试[NIST96]。

2.7API（应用程序接口）测试

应用程序接口(API)是允许不同进程、程序和/或系统之间通信的代码。应用程序接口经常用在客户端/服务器关系中，其中一个进程向其他进程提供某种功能。

应用程序接口测试是一种测试类型，而不是一种技术。在某些方面，应用程序接口测试与图形用户界面（GUI）测试非常相似。重点是对输入值和返回数据的评估。

在处理应用程序接口时，逆向测试通常是至关重要的。使用应用程序接口访问自身代码外部服务的程序员，可能会试图以非预料的方式使用应用程序接口。这意味着健壮的错误处理对于避免不正确的操作至关重要。可能需要对许多不同的接口进行组合测试，因为应用程序接口通常与其他应用程序接口一起使用，而且单个接口可能包含多个参数，这些参数的值可以多种方式组合。

应用程序接口通常是松散耦合的，导致事务丢失或计时故障的可能性非常大。这需要对恢复和重试机制进行彻底的测试。提供应用程序接口的组织必须确保所有服务具有非常高的可用性；这通常需要应用程序接口发布者进行严格的可靠性测试以及基础设施支持。

适用性

随着单个系统变成分布式，或者使用远程处理的方式将一些工作负载分给其他处理器，应用程序接口测试对于测试综合系统变得越来越重要。例子包括：

- 操作系统调用
- 面向服务的结构(SOA)
- 远程过程调用(RPC)
- Web服务

软件容器化[Burns18]的结果是将软件程序划分为几个容器，这些容器使用上面列出的机制相互通信。应用程序接口测试也应该针对这些接口。

局限/难点

直接测试应用程序接口通常需要技术测试分析师使用专门的工具。由于通常没有直接与应用程序接口关联的图形界面，因此可能需要使用工具来设置初始环境、整理数据、调用 API 并确定结果。

覆盖率

应用程序接口测试是一种测试类型的描述；它不表示任何具体的覆盖水平。至少，应用程序接口测试应该包括使用实际的输入值和用于检查异常处理的意外输入值来调用应用程序接口。更彻底的应用程序接口测试可以确保至少执行一次可调用实体，或者至少执行一次所有可能的调用。

缺陷的类型

通过测试应用程序接口可以发现的缺陷类型是完全不同的。接口问题很常见，数据处理问题、计时问题、事务丢失和事务重复也很常见。

2.8 选择一种白盒测试技术

被测系统的环境将对其产品风险和危险程度产生影响（见下文）。这些因素影响所需的覆盖率度量（因此也影响要使用的白盒测试技术）和要达到的覆盖深度。一般来说，系统越关键，产品风险水平越高，对覆盖率的要求就越严格，达到预期覆盖率所需的时间和资源也就越多。

有时候，所需的覆盖率度量可能来自应用于软件系统的适用标准。例如，如果软件要在机载环境中使用，它可能需要符合标准 DO-178C（在欧洲是 ED-12C）。本标准包含以下五种失效情况：

A. 灾难性的：失效可能会导致飞机丧失安全飞行或降落所需的关键功能

B. 危险的：失效可能对安全或性能效率有较大的负面影响

C. 重要的：失效是重要的，但没有A或B严重

D. 次要的：失效是可以察觉的，但影响比C小

E. 无影响：失效对安全无影响

如果软件系统被归类为 A 级，它必须经过 100% 改进的条件/判定覆盖的测试。如果它是 B 级，则必须测试到 100%判定水平覆盖，并且改进的条件/判定测试是可选的。级别 C 至少需要 100%的语句覆盖。

同样地，[IEC61508]是一个关于可编程的、电子的、安全相关系统的功能安全性的国际标准。这个标准已经在许多不同的领域得到应用，包括汽车、铁路、制造业、核电站和机械行业。危险程度是使用安全完整性等级（SIL）来定义的，其中 SIL1 是最不危险的，SIL4 是最危险的。此标

准给出了测试覆盖率的建议，如下表所示（注意，每个 SIL 的确切定义以及“推荐”和“高度推荐”的含义在此标准中进行了定义）。

SIL	100%语句覆盖	100%分支（判定）覆盖	100%改进的条件/判定覆盖
1	推荐	推荐	推荐
2	**高度推荐**	推荐	推荐
3	**高度推荐**	**高度推荐**	推荐
4	**高度推荐**	**高度推荐**	**高度推荐**

在现代系统中，很少在一个系统上完成所有的处理。在任何需要远程处理的情况下都应该进行 API（应用程序接口）测试。系统的危险程度决定了在应用程序接口测试中应该投入多少工作。

3. 分析技术-210分钟

关键词

控制流分析（control flow analysis）、圈复杂度（cyclomatic complexity）、数据流分析（data flow analysis）、定义-使用对（definition-use pair）、动态分析（dynamic analysis）、内存泄漏（memory leak）、成对集成测试（pairwise integration testing）、邻域集成测试（neighborhood integration testing）、静态分析（static analysis）、野指针（wild pointer）。

分析技术的学习目标

3.2 静态分析

TTA-3.2.1（K3）使用控制流分析检测代码是否有任何控制流异常

TTA-3.2.2（K2）解释如何使用数据流分析来检测代码是否有任何数据流异常

TTA-3.2.3（K3）提出通过应用静态分析来提高代码可维护性的方式

TTA-3.2.4（K2）解释如何使用调用图制定集成测试策略

3.3 动态分析

TTA-3.3.1（K3）应用动态分析达成特定目标

3.1 介绍

分析有两种类型：静态分析和动态分析。

静态分析（第 3.2 节）包含无需执行软件即可进行的分析测试。由于软件未执行，因此是通过工具或人工检查来判断软件执行时能否正确运行。该类软件静态视角无需创建数据和先决条件来执行某一场景即可进行详细分析。

请注意，第 5 章介绍了与技术测试分析师相关的不同形式的评审。

动态分析（第 3.3 节）要求实际执行代码，用于查找在执行代码时更容易检测到的失效（例如内存泄漏）。与静态分析一样，动态分析也可依赖于工具或个人监视执行系统以观察相关指标情况，如内存使用的迅速增加。

3.2 静态分析

静态分析的目的是检测代码和系统架构中实际或潜在的缺陷，并提高其可维护性。静态分析通常由工具支持。

3.2.1控制流分析

控制流分析是一种静态技术，通过使用控制流图或工具来分析程序所遵循的步骤。使用此技术可以在系统内找到多种类型的异常，包括设计不当的循环（例如，具有多个入口点）、某些语言函数调用的目标模糊（例如 Scheme 语言）、操作顺序不正确等。

控制流分析可用于确定圈复杂度。圈复杂度是一个正整数，表示强连接图中独立路径的数量。循环和迭代一旦走过一次，就会被忽略。从入口到出口的每个路径都代表通过模块的一条特定路径。每条特定路径都应进行测试。

圈复杂度值通常用于指明模块的总体复杂性。托马斯·麦凯布的理论[McCabe 76]认为，系统越复杂，维护难度越大，缺陷越多。多年来，许多研究都注意到复杂度与所包含缺陷的数量之间的这种相关性。国家标准与技术研究所（NIST）对最大复杂值的建议为 10。任何测量出来复杂值更高的模块都应尽可能划分为多个模块并经过评审。

3.2.2 数据流分析

数据流分析涵盖各种技术，用于收集系统中变量使用的情况。每个变量的整个生命周期都要接受调查（即声明、定义、读取、评估和销毁阶段），因为任何一个操作过程中以及操作顺序不正确的情况下异常都有可能发生。

有一种常见的技术，称为定义-使用符号，每个变量的生命周期被拆分为三个不同的原子操作：

- d：当变量得以声明、定义或初始化时
- u：当变量在运算或决策谓词中被使用或读取时
- k：当变量被杀死、销毁或超出范围时

d-u-k 的另一种常见的替代符号是：d（定义）- r（引用或读取）- u（未定义）。

这三个原子操作合并成对（“定义-使用对”），以说明数据流。例如，″du-路径″表示数据变量被定义且随后被使用的代码片段。

可能的数据流异常包括在错误的时间对变量执行正确的操作，或对变量中的数据执行不正确的操作。这些异常包括：

- 在使用变量之前未能为它赋值
- 由于控制谓词中的值不正确，选择了不正确的路径
- 在变量被销毁后还尝试使用变量
- 在变量超出范围（定义域）情况下引用变量
- 对一个变量进行声明和销毁，但却没有使用该变量
- 在变量使用之前重新定义变量
- 未能终止动态分配的变量（可能导致内存泄漏）
- 修改变量，导致意外的副作用（例如，在没有考虑变量的所有用途的情况下更改全局变量时产生的波及效应）

所使用的开发语言可以指导数据流分析中使用的规则。编程语言可允许程序员使用不违规的变量执行某些操作，但可能会导致系统在某些情况下的行为方式与程序员的预期不同。例如，当遵循特定路径时，变量可能会定义两次，但实际上并没有使用。数据流分析通常会将这些用途标记为″可疑″。虽然这可能是对变量赋值功能的合法使用，但它可能导致代码未来的可维护性出现问题。

数据流测试“使用控制流图来探索数据可能发生的不合理情况”[Beizer90]，从而发现不同于控制流分析得出的缺陷。技术测试分析师在规划测试时应纳入此技术，因为其中许多缺陷会导致间歇性故障，而这些故障在执行动态测试时难以发现。

数据流分析是一种静态技术，它可能会忽略了系统运行期间使用数据时发生的一些问题。例如，静态数据变量可能包含指向动态创建的数组的指针，该指针在运行之前甚至不存在。多元处理器的使用和先发的多任务可能会创建数据流或控制流分析无法找到的竞态条件。

3.2.3使用静态分析提高可维护性

可以通过多种方式使用静态分析来提高代码、架构和网站的可维护性。

编写不当、未注释和非结构化代码往往更难维护。开发人员可能需要付出更多努力来定位和分析代码中的缺陷，而修改代码以纠正缺陷或添加新功能可能会引入新的缺陷。

用工具进行静态分析，通过确保代码与编码规范和编码指南的符合程度来提高其可维护性。这些规范和指南描述了所要求的编码实践，如命名约定、注释、缩进和代码模块化。请注意，静态分析工具通常提出警告，而不是检测错误。即使代码在句法上是正确的，这些警告也可能出现。

静态分析工具可应用于实现网站所用的代码，以检查可能存在安全漏洞，如代码注入、Cookie 安全、跨站点脚本、资源篡改和 SQL 代码注入。详情见第 4.3 节和高级安全测试教学大纲[ISTQB_ALSEC_SYL]。

模块化设计通常能产生可维护性更强的代码。静态分析工具以下列方式支持模块化代码的开发：

- 它们搜索重复的代码。这部分代码可能是重构模块时的候选代码（尽管模块调用所需的运行时间可能会导致实时系统在时间上出现问题）。
- 它们生成的度量数据是代码模块化的宝贵指标。其中包括耦合度和内聚度的度量。具有良好可维护性的系统更有可能具有较低的耦合度（模块在执行期间相互依赖的程度）和较高的内聚度（模块自足并专注于单个任务的程度）。
- 在面向对象的代码中，它们指出派生对象对父类的可见性过多或过少的地方。
- 它们突出显示了代码和架构中具有高度结构复杂性的部分。

静态分析工具还可以用于支持网站的维护。目标是检查站点的树状结构是否平衡良好，或者是否存在不平衡情况将导致以下问题：

- 测试任务更难
- 维护工作量增加
- 用户导航困难

3.2.4调用图

调用图是通信复杂性的静态表示形式。它们是有向图形，其中节点表示程序模块，边表示这些模块之间的通信。

调用图可以使用在单元测试中，描述不同的函数或方法相互调用；调用图可以使用在集成和系统测试中，用于描述独立模块相互调用；调用图或可使用在系统集成测试中，用于描述独立的系统相互调用。

调用图可用于以下目的：

- 设计调用特定模块或系统的测试
- 确定软件中模块或系统被调用的位置的数量
- 评估代码的结构以及系统的架构
- 就集成顺序提出建议（如下文所述的成对和邻域集成）

基础级教学大纲[ISTQB_FL_SYL]中讨论了两种不同类型的集成测试：增量（自上而下、自下而上等）和非增量（大爆炸）。增量方法被认为是首选方法，因为它们以增量方式引入代码，所涉及的代码数量有限，从而更易进行缺陷隔离。

此高级教学大纲引入三种使用调用图的非增量方法。这些方法可能更优于增量方法，它们可能需要其他构建来完成测试，并且需要编写额外代码以支持测试。这三种方法是：

- 成对集成测试（不要与黑盒测试技术“成对测试”混淆），针对集成测试调用图中所示的协同工作的组件对。虽然此方法仅减少少量构建，但是它减少了所需的测试用具代码量。

- 邻域集成测试的对象是连接到给定节点的所有节点，作为集成测试的基础。调用图中特定节点的所有前置节点和后续节点都是测试的基础。
- 麦凯布（McCabe）的设计谓词方法使用适用于模块调用图的圈复杂度理论。这需要构建一个调用图，显示模块可以相互调用的不同方式，包括：
- 无条件调用：始终发生一个模块对另一个模块的调用
- 有条件调用：有时发生一个模块对另一个模块的调用
- 互斥条件调用：一个模块将调用多个不同模块中的一个（且仅一个）
- 迭代调用：一个模块可以调用另一个模块至少一次或多次
- 迭代条件调用：一个模块可以调用另一个模块零次或多次

创建调用图后，计算集成复杂性并创建测试以覆盖该调用图。

有关使用调用图和邻域集成测试的详细信息，请参阅 [Jorgensen07]。

3.3 动态分析

3.3.1 概述

动态分析用于检测那些仅在执行代码时症状才可显现的失效。例如，静态分析可以检测到内存泄漏的可能性（查找分配但从不释放内存的代码），但动态分析可以更容易地显示内存泄漏。

不能立即重现（间歇性）的失效可能会对测试工作以及释放或高效使用软件的能力产生重大影响。此类失效可能由内存或资源泄漏、指针使用不正确和其他损坏（例如系统堆栈）[Kaner02]引起。由于这些故障的性质（可能包括系统性能效率的逐渐恶化，甚至系统崩溃），测试策略必须考虑与此类缺陷相关的风险，并酌情执行动态分析以减少这些缺陷（通常使用工具）。由于这些故障通常查找和更正的费用最昂贵，因此建议在项目早期开始进行动态分析。

可应用动态分析解决以下问题：

- 通过检测内存泄漏（参见第3.3.2节）和野指针（参见第3.3.3节）来防止失效发生
- 分析不易重现的系统故障
- 评估网络行为
- 通过提供运行时系统的行为信息，并对其进行有效的更改，从而可以提高系统性能效率

动态分析可在任何测试级别执行，需要技术和系统技能才能执行以下操作：

- 指定动态分析的测试目标
- 确定启动和停止分析的适当时间
- 分析结果

在系统测试期间，即使技术测试分析师的技术技能水平很低，也可以使用动态分析工具，使用的工具通常会创建完整的日志，具备所需技术技能的人员可以分析这些日志。

3.3.2 检测内存泄漏

内存泄漏指的是程序分配了自身可以使用的内存空间（RAM），但在不再需要时没有释放内存空间。而此内存空间视为已分配，不可重复使用。当这种情况频繁发生或在内存不足时，程序可能会耗尽可用的内存。过去，对内存的操作是程序员的责任，任何动态分配的内存空间都必须由分配程序在正确的范围内释放，以避免内存泄漏。许多现代编程环境，包括自动或半自动“垃圾回收”，在没有程序员直接干预的情况下，分配的内存在使用后自动释放。如果自动垃圾回收释放了现有分配的内存，则隔离和检测内存泄漏可能非常困难。

内存泄漏会导致一些问题，这些问题不会立即显现出来，而是随时间推移而发展和显现。例如，当软件刚安装或系统重新启动（在测试期间经常是这样）时可能会出现（内存泄漏不会立即显现出来）这种情况。由于这些原因，在程序处于实际生产的过程中才会可能注意到内存泄漏的负面影响。

内存泄漏的主要症状是系统响应时间持续恶化，最终可能导致系统故障。虽然可以通过重新开始（重新启动）系统来解决此类故障，但这并不是真正的（实际）解决办法，有时甚至不可能重启。

许多动态分析工具识别代码中发生内存泄漏的区域，以便进行纠正。简单的内存监视器还可用于获知可用内存是否随时间而减少，但是仍然需要进行后续分析以确定减少的确切原因。

还应考虑其他类型的泄漏。例如包括文件句柄、信号量和资源连接池。

3.3.3 检测野指针

程序中的“野”指针是指不再准确且不得使用的指针。例如，野指针可能"丢失"了它应该指向的对象或函数，或者它不指向预期内存空间（例如，它指向超出数组分配边界的区域）。当程序使用野指针时，可能会出现各种后果，包括：

- 程序可以按预期执行。在这种情况下，野指针可以访问程序当前未使用的、且理论上为“空闲”和/或包含合理值的内存。
- 程序可能会崩溃。在这种情况下，野指针可能导致部分内存的不当使用，这对程序的运行至关重要（例如操作系统）。
- 程序无法正常运行，因为无法访问程序所需的对象。在这些情况下，程序可以继续运行，但可能会发出错误消息。
- 内存位置中的数据可能被指针和随后使用的错误值损坏（这也表示可能存在安全威胁）。

请注意，对程序内存使用情况所做的任何更改（例如，软件更改后的新构建）都可能引发上述四个后果中的任何一项。特别重要的是，尽管使用了野指针，但程序最初还是能按预期运行，但在软件更改后意外崩溃（甚至可能在实际生产中）。需要注意的是，此类故障通常是潜在缺陷（即野指针）的症状（请参阅 [Kaner02]，“第 74 课”）。无论野指针对程序执行的影响如何，工具可以帮助将程序中使用的野指针识别出来。某些操作系统具有内置功能用于检查运行时内存访问违规的情况。例如，当应用程序尝试访问该应用程序允许的内存空间之外的内存位置时，操作系统可能会触发异常预警。

3.3.4 性能效率分析

动态分析不仅可用于检测失效。通过对程序性能效率的动态分析，工具可帮助识别性能效率瓶颈，并生成各种性能效率指标，开发人员可以使用这些指标来调整系统性能效率。例如，它可以提供执行期间某模块被调用的次数信息。经常被调用的模块有可能成为帮助性能效率提升的备选。

通过将有关软件动态行为的信息与静态分析期间从调用图中获取的信息合并（参见第 3.2.4 节），测试人员还可以识别出需要详细深入（深度）和广泛（广度）测试的备选模块（例如，经常被调用并具有许多接口的模块）。

程序性能效率的动态分析通常在进行系统测试时进行，有时也可在测试的早期阶段使用测试工具测试单个子系统时进行。更多详细信息见基础级性能测试教学大纲［ISTQB®_FLPT_SYL］。

4. 技术测试的质量特性——345分钟

关键词

可核查性（accountability*）、适应性（adaptability*）、易分析性（analyzability*）、真实性（authenticity*）、可用性（availability*）、容量（capacity*）、共存性（co-existence*）、兼容性（compatibility*）、保密性（confidentiality*）、容错性（fault tolerance*）、易安装性（installability*）、完整性（integrity*）、维护性（maintainability*）、成熟性（maturity*）、易修改性（modifiability*）、模块化（modularity*）、抗抵赖性（non-repudiation*）、运行验收测试（operational acceptance testing）、运行配置（operational profile）、性能效率（performance efficiency*）、可移植性（portability*）、质量特性（quality characteristic）、易恢复性（recoverability*）、可靠性（reliability*）、可靠性增长模型（reliability growth model）、易替换性（replaceability*）、资源利用性（resource utilization*）、可重用性（reusability*）、信息安全性（security*）、易测试性（testability*）、时间特性（time behavior*）。

说明：*表示是按“中华人民共和国国家标准 GB/T 25000.10-2016” 翻译。

技术测试的质量特性学习目标

4.2 总体策划的问题

TTA-4.2.1（K4）针对特定的场景，对非功能需求进行分析，并编写测试计划的相关部分。

TTA-4.2.2（K3）给定特定的产品风险，定义特定的最合适的非功能测试类型。

TTA-4.2.3 (K4)理解并解释在应用软件开发生命周期的哪些阶段，通常应该应用非功能测试。

TTA-4.2.4 (K3)针对给定的场景，定义你预期通过运用不同类型的非功能测试能找到的缺陷类型。

4.3 信息安全性测试

TTA-4.3.1（K2）解释在测试方法中包含信息安全性测试的理由。

TTA-4.3.2（K2）解释在计划和指定信息安全性测试时要考虑的主要方面。

4.4 可靠性测试

TTA-4.4.1（K2）解释在测试方法中包含可靠性测试的理由。

TTA-4.4.2（K2）解释在计划和指定可靠性测试时要考虑的主要方面。

4.5 性能效率测试

TTA-4.5.1（K2）解释在测试方法中包含性能效率测试的理由。

TTA-4.5.2（K2）解释在计划和指定性能效率测试时要考虑的主要方面。

4.6 维护性测试

TTA-4.6.1（K2）解释在测试方法中包含维护性测试的理由。

4.7 可移植性测试

TTA-4.7.1（K2）解释在测试方法中包含可移植性测试的理由。

4.8 兼容性测试

TTA-4.8.1（K2）解释在测试方法中包含兼容性测试的理由。

4.1 简介

通常情况下，技术测试分析师侧重于测试产品工作得“如何”，而不是产品做了“什么”的功能方面。这些测试可以在任何测试级别进行。例如，实时系统和嵌入式系统的组件测试过程中，进行性能效率基准（performance efficiency benchmarking）测试和资源利用性测试都很重要。在运行验收测试和系统测试过程中适合进行可靠性方面（例如易恢复性）的测试。这个级别的测试目的都是测试由软件和硬件组合的特定系统。被测试的特定系统可能包括各种服务器、客户端、数据库、网络和其它资源。无论在哪个测试级别，都应该根据风险的优先级和可利用的资源进行测试。

应该注意的是，动态测试和静态测试（见第 3 章）都可以用于本章所描述的对非功能质量特性的测试。

ISO25010 [ISO25010]中提供的对产品质量特性的描述是用来描述这些特性及其子特性的指南。这些特性都列举在下面的表格中，同时指出了在测试分析师和技术测试分析师教学大纲中包含了哪些特性/子特性。

特性	子特性	测试分析师	技术测试分析师
功能性	功能正确性、功能适合性、功能完备性	X	
可靠性	成熟性、容错性、易恢复性、可用性		X
易用性	可辨识性、易学性、易操作性、用户界面舒适性、用户差错防御性、易访问性	X	
性能效率	时间特性、资源利用性、容量		X
维护性	易分析性、易修改性、易测试性、模块化、可重用性		X
可移植性	适应性、易安装性、易替换性		X
信息安全性	保密性、完整性、抗抵赖性、可核查性、真实性		X
兼容性	共存性		X
	互操作性	X	

注意，附录 A 中提供了一个表格，该表格将 ISO 9126（在 2012 版教学大纲中使用）和较新的 ISO25010 中描述的特性进行了对比。

对所有在本节中讨论的质量特性和子特性，必须识别典型风险，以便形成合适的测试方法并将之文档化。质量特性测试需要特别关注生命周期时间点、所需工具、所需标准、软件和文档的可用性和技术专长。如果没有对每个特性和其独特的测试需求规划好方法，那么测试人员在制定时间计划表时可能会没有留出充分的时间进行测试计划、测试准备和测试执行[Bath14]。

这些测试中的某些测试，例如性能效率测试，需要大规模的计划、专用的设备、特定的工具、专业的测试技能以及（通常情况下还需要）大量的时间。质量特性和子特性的测试必须集成到整体测试时间表中，同时为此项工作分配充分的资源。这些测试类型中的每一种都有特定的需求，面向特定的问题，而且它们可能发生于软件开发生命周期的不同时段，如下节所讨论的。

测试经理主要关心的是编制和报告有关质量特性和子特性的度量总结信息，而测试分析师或技术测试分析师则（根据上面的表格）负责收集每个度量的信息。

技术测试分析师在软件进入生产阶段前的测试中收集的质量特性的度量，可能会构成软件系统的供应商和利益相关方（例如，客户，运营商）之间的服务等级协议（SLA-Service Level Agreement）的基础。某些情况下，可能会在软件进入生产阶段后继续执行测试，通常由另外的团队或组织进行。性能效率测试和可靠性测试经常出现这种情况，此时在生产环境中得到的结果可能会与在测试环境中得到的结果不同。

4.2 总体规划问题

如果在计划中忽略了非功能测试，就可能给应用程序的成功带来很大的风险。测试经理可能要求技术测试分析师识别相关质量特性的主要风险（参见 4.1 节中的表格），并解决任何与所提议的测试相关的规划问题。这部分信息可能用于创建主测试计划。

在执行这些任务时，会考虑以下这些综合因素：

- 利益相关方的需求
- 所需工具的获得和人员培训
- 测试环境的要求
- 组织方面的考虑
- 数据安全方面的考虑
- 风险和典型缺陷

4.2.1利益相关方的需求

非功能性需求定义通常都很粗略，有时甚至根本不存在这些定义。在计划阶段，技术测试分析师必须能够从受影响的利益相关方那里获得与技术质量特性对应的期望级别，并评估这些级别所代表的风险。

常见的方法是假设客户对当前的系统版本满意，只要系统版本能够保持所达到的质量级别，那么他们对新版本也会满意。这样系统的当前版本（的质量特性指标）就可以当做基准。对于某些非功能质量特性（比如性能效率），当利益相关方难以详细说明他们的需求的时候，采取这种方法可能特别有用。

在收集非功能需求时，广泛征求相关方面的观点是非常明智的。这些观点必须从诸如客户、产品负责人、用户、操作人员和维护人员等利益相关方那里获得。如果没有包括关键的利益相关方，很可能会忽略一些需求。关于捕获需求的更多细节，参见高级测试经理教学大纲[ISTQB_ATLTM_SYL]。

在敏捷项目中，非功能需求可能被描述为用户故事，或者作为非功能制约条件添加到用例定义的功能中。

4.2.2 所需工具的获得和人员培训

商业工具或模拟器特别适用于性能效率测试和特定的信息安全性测试。技术测试分析师应该评估采购、学习和实施工具涉及到的成本和时间。在使用专门的工具时，规划还应该考虑到使用新工具的学习曲线和/或聘请外部工具专家的成本。

而复杂的模拟器的开发可能本身就代表了一个开发项目，也应如此规划。尤其，在时间计划和资源计划中必须考虑所开发工具的测试和文档。当被模拟的产品发生变化时，应该为模拟器的更新和重新测试规划充分的预算和时间。当模拟器用于安全关键应用时，在计划中还必须考虑相应的验收测试以及通过一个独立机构对模拟器进行认证。

4.2.3 测试环境的要求

许多技术测试（例如，信息安全性测试、性能效率测试）要求有一个与生产环境类似的测试环境，以便进行实际的测量。被测系统大小和复杂度的不同可能对测试的计划和资金预算产生很大影响。由于构建此类环境的成本较高，可以考虑以下选择：

- 使用实际的生产环境
- 使用缩减版的系统。注意要让所得到的测试结果足以代表整个生产系统。
- 使用基于云的资源替代直接采购资源
- 使用虚拟化的环境

必须仔细规划执行这类测试的时机。这类测试极有可能只有在特定的时间段（例如，在较少系统使用时间段内）执行。

4.2.4 组织方面的考虑

技术测试可能包括测量完整系统中的几个组件的行为（例如，服务器、数据库、网络）。若这些组件分布在多个不同的场所和组织，则可能要花很大的精力进行测试的计划和协调。例如，某些软件组件可能只在每天或每年的特定时间段才能用于系统测试，或组织只能提供有限的天数用于支持测试。如果不能确认其它组织的系统组件或人员（即“外借”的专家）可以“待命”参与测试，将可能严重影响已排期的测试。

4.2.5 数据安全性方面的考虑

在测试计划阶段就应考虑到用于确保系统安全的手段，以确保所有的测试活动可行。例如，使用数据加密技术可能会增加创建测试数据和验证结果的难度。

数据保护政策和法令可能会禁止在实际生产数据（例如：个人数据、信用卡数据）的基础上生成任何所需的测试数据。测试数据匿名化是一件重要的任务，必须将其作为测试实现的一部分来计划。

4.2.6 风险和典型缺陷

识别和管理风险是测试计划的一个基本考虑（见第 1 章）。技术测试分析师使用对特定质量特性可能发现的典型缺陷的知识，来识别产品风险。这样就可以选择应对这些风险所需的测试类型。在本章接下来的单独介绍各个质量特性的部分中，将覆盖这些特定的方面。

4.3 信息安全性测试

4.3.1 考虑信息安全性测试的理由

信息安全性测试以破坏系统的安全性管制为目标进行攻击，从而评估系统的漏洞。下面列出了需要在信息安全性测试中寻找的潜在威胁：

- 未经授权的拷贝应用程序或数据。
- 未经授权的访问（例如，用户能够执行其没有权限执行的任务）。用户权限、访问和特权是这个测试的重点。系统的规格说明应该提供此信息。
- 当执行其预期功能的时候，软件显示预期外的副作用。例如，媒体播放器虽然能正确地播放音频，但是它是通过将文件写入到未加密的临时存储空间中来实现的，软件盗版者就能利用这种副作用。
- 插入网页的代码，可能被后续用户（跨站点脚本Cross-Site-Scripting或XSS）所使用。这种代码可能是恶意的。
- 缓冲区溢出（缓冲区超出限度），可能由于在用户界面的输入区域输入了超出能够正确处理的长度的字符串而导致。一个缓冲区溢出漏洞就代表一个运行恶意代码指令的机会。
- 拒绝服务，阻止用户与应用程序的交互。（例如，通过发送“骚扰”请求使网络服务器超载）。
- 第三方的秘密窃听、复制和/或改变然后转发通信（例如，信用卡交易），而用户根本没有意识到第三方的存在（“中间人（Man in the Middle）”攻击方式）。
- 破解用来保护敏感数据的加密密码。
- 恶意插入代码中的逻辑炸弹（有时称为“复活节彩蛋”），且只在特定条件下（例如，在某个特定日期）激活。当逻辑炸弹激活时，它们可能执行恶意操作，如文件删除或格式化磁盘。

4.3.2 信息安全性测试计划

一般而言，以下几个方面在规划信息安全性测试时非常重要：

- 因为在系统架构、设计和实施过程中都可能会引入安全问题，因此在单元、集成和系统测试级别都可能安排信息安全性测试。由于安全性威胁会不断变化，也可能在系统投入生产后定期安排信息安全性测试，对于像物联网（IoT）这样的动态开放架构尤其是如此，因为其生产阶段的特点就是会使用到很多对软件和硬件要素的更新。
- 技术测试分析师所提出的测试方法应该包括架构、设计和代码的评审，以及借助于安全性工具的静态分析。这些做法在发现动态测试中容易遗漏的安全性问题方面可能会有效。

- 可能要求技术测试分析师来设计和开展某些信息安全性“攻击”（见下文），这些“攻击”需要与利益相关方（包括信息安全性测试专家）认真地规划和协调。其它信息安全性测试可以与开发人员或测试分析师（例如，测试用户权限、访问和特权）合作进行。
- 信息安全性测试计划的一个重要方面是获得批准。对技术测试分析师而言，这意味着确保已经从测试经理处获得明确的许可来执行计划好的信息安全性测试。所执行的任何额外的计划外测试可能会被看作实际的攻击，而进行那些测试的人可能面临法律诉讼的风险。在没有书面显示意图和授权情况下，“我们在执行信息安全性测试”这种借口很难是令人信服的解释。
- 所有的信息安全性测试计划都应该与组织的信息安全官（如果组织有这样一个角色的话）协调。
- 应当指出，为系统信息安全性所做的改进可能会影响其性能效率或可靠性。在改进信息安全性方面后，考虑是否有必要进行性能效率或可靠性测试是明智的（见下文第4.4、4.5节）。

实施信息安全性测试策划时，可能有单独的适用标准，例如适用于工业自动化和控制系统[ISA/IEC 62443-3-2]。

高级安全性测试教学大纲[ISTQB_ALSEC_SYL]包含了信息安全性测试计划中关键要素的更多细节。

4.3.3 信息安全性测试规格说明

特定的信息安全性测试可以根据安全风险的来源进行分组 [Whittaker04]。包括如下的分组：

- 用户界面相关的 -- 未经授权的访问和恶意输入。
- 文件系统相关的 -- 访问文件或资料库中存储的敏感数据。
- 操作系统相关的 -- 敏感信息的存储，如密码，在内存中以非加密的形式存在；当系统由恶意输入导致崩溃时，这些信息可能会被暴露。
- 外部软件相关 -- 系统需要使用并与系统之间发生相互作用的外部组件。这可能是在网络层面（例如，不正确的数据包或消息传递）或软件组件层面（例如，该软件所依赖的软件组件的失效）。

ISO 25010 中信息安全性的子特性[ISO25010]也提供了可以用以定义信息安全性测试的依据。这些子特性关注于信息安全性的如下方面：

- 保密性——产品或系统确保数据只有在被授权时才能被访问的程度。
- 完整性——系统、产品或组件防止未授权访问或篡改计算机程序或数据的程度。
- 抗抵赖性——可证明活动或事件已发生使之之后不可被否认的程度。
- 可核查性——实体的操作可被唯一地追溯到该实体的程度。
- 真实性——可证明主体或资源的身份符合其所宣称的身份的程度。

以下方法 [Whittaker04] 可以用来开发信息安全性测试：

- 收集在指定的测试中可能有用的信息，如员工姓名、物理地址、内部网络相关细节、IP地址、所用软件或硬件的标识和操作系统版本。
- 用可用性广泛的工具进行漏洞扫描。这些工具不是直接用来危害系统，而是识别那些可能是或导致违背信息安全性制度的漏洞。也可以使用信息和检查表来识别特定的漏洞，比如（美国）国家标准和技术研究院（NIST- National Institute of Standards and Technology）[Web-1]和开放式Web应用程序安全项目（Open Web Application Security Project™ (OWASP)）[Web-4]提供的信息和检查表。
- 通过使用所收集的信息，开发“攻击方案”（也就是企图破坏某一特定系统的信息安全性制度的测试操作的方案）。需要在攻击方案中制定针对各种接口（如用户界面、文件系统）的输入，以检测出最严重的信息安全性缺陷。在[Whittaker04]中所描述的各种“攻击”，是一种有价值的技术来源，这些技术是专门为信息安全性测试所开发的。

注意，可以针对渗透测试制定攻击计划（参见[ISTQB_ALSEC_SYL]）。

通过评审（见第 5 章）和/或使用静态分析工具（见第 3.2 节）也能发现信息安全性问题，静态分析工具包含了大量专门针对信息安全性威胁的规则，对照这些规则对代码检查。例如，缓冲区溢出问题，是由数据分配前未能检查缓冲区大小而造成的，这类问题可以通过工具来发现。

3.2 节（静态分析）和高级安全性测试教学大纲[ISTQB_ALSEC_SYL]给出了信息安全性测试的更多细节。

4.4 可靠性测试

4.4.1 引言

ISO 25010 的产品质量特性分类中定义了以下可靠性的子特性：

- 成熟性——组件或系统在正常运行时满足可靠性要求的程度。
- 容错性——在软件有缺陷或违反了软件所规定接口的情况下，软件产品保持规定的性能效率水平的能力。
- 易恢复性——发生失效时，软件产品重新建立规定的性能效率水平并恢复直接受影响的数据的能力。
- 可用性——组件或系统在需要使用时能够进行操作和访问的程度。

4.4.2 测量软件成熟性

可靠性测试的目标之一是在一段时间内，观察软件成熟性的统计测量数据，并与期望的可靠性目标做比较。该可靠性目标可能用服务等级协议（SLA）表达。这些测量的形式可能是平均失效时间间隔（MTBF- Mean Time Between Failures）、平均修复时间（MTTR- Mean Time To Repair ）或任何其它形式的失效强度度量（例如，每周发生特定严重的失效的个数）。这些度量可能用来作为（例如，产品发布的）出口准则。

注意，不要把可靠性背景下的成熟度与整体软件测试过程的成熟度相混淆，后者在高级软件测试经理教学大纲[ISTQB_ALTM_SYL]中进行讨论。

4.4.3 容错性测试

在功能测试中，会通过输入无效数据，对软件处理意外输入值（所谓逆向测试）的容限进行评估，除此以外，还需要额外的测试来评估系统对发生于被测应用外部的故障的容错性。这些故障通常由操作系统报告（例如，磁盘已满、进程或服务不可用、未找到文件、内存不可用）。系统级别的容错性测试可由特定的工具支持。

注意，在讨论容错性（error tolerance / fault tolerance）时常常会用到另一个术语“健壮性（Robustness）”（详见 [ISTQB_GLOSSARY]）。

4.4.4 易恢复性测试

可靠性测试的进一步表现是对软件系统按照规定的方式从软硬件故障中恢复至正常状态的能力进行评估。易恢复性测试分为故障转移（Failover）和备份（Backup）/恢复（Restore）测试。

某些软件失效可引发严重后果，因而会采取专门的硬件和/或软件措施以保障系统即使在出现失效时仍能运行。针对这种情况要进行故障转移测试。如，故障转移测试适用于金融损失风险极高或存在严重安全隐患的情况。这种对于灾难性事件引发失效的易恢复性测试也被称为“灾难恢复（disaster recovery）”测试。

典型的硬件方面的措施包括平衡各个处理器负载、集群化服务器、处理器或硬盘，以便在一个硬件失效的时候，另一个马上接管（冗余系统）。典型的软件方面的措施则可在一个所谓的非相似冗余系统（Redundant dissimilar system）中实现多个独立软件系统（例如，航空飞行控制系统）。冗余系统一般结合了上述的软件和硬件措施，根据独立实例的个数（2 个、3 个或 4 个）可以分别称之为双重、三重或四重系统。令两个（或多个）互不相关的开发团队按相同的软件需求，开发不同的软件来提供相同的服务，从而达到实现软件的非相似性的目的。因而使非相似冗余系统在相似的缺陷输入情况下不至于导致相同的后果。这些为增强系统的可恢复性所采取的措施也会直接影响其可靠性，因此应在可靠性测试中予以考虑。

故障转移测试的目的是，通过模拟失效模式或在受控的环境中实际地制造失效来明确地测试系统。失效后，针对故障转移机制进行测试，可以保证没有丢失或破坏数据，且系统保持原有服务级别（例如，功能可用性和响应时间）。

备份/恢复测试关注的是将故障影响最小化的规程性措施。这类测试对进行不同形式的备份并在数据丢失或损坏时进行恢复的（通常写入手册的）规程进行评估。测试用例的设计要保证能够覆盖该规程的关键路经。可通过技术评审来预演这些场景并对照实际的规程来确认手册。在运行验收测试时，这些场景会在实际的生产环境或类似的环境中演练，以确认它们的实际效用。

备份/恢复测试的衡量指标可以包括以下几部分：

- 完成各类备份（例如全备份或增量备份）所需时间
- 恢复数据所需时间
- 确保数据备份的级别（例如，恢复不超过24小时内的所有数据、恢复不超过1小时内的具体交易数据）

4.4.5 可用性测试

任何与其他系统和/或过程（例如，为了接收输入）有接口的系统，都依赖于这些接口的可用性来保证整体的可操作性。

可用性测试主要为以下目的服务：

- 确认所需的系统组件或过程是否可用（按需或持续可用），并且对请求的响应符合预期
- 提供度量以便于从其中获得整体的可用性级别（一般以SLA中的时间百分比的形式给出）
- 确认整个系统是否已准备好运行（例如，作为运行验收测试的准则之一）
- 在进入运行服务之前和之后都进行可用性测试，并且特别适用于以下情况：
  a) 系统是由其他系统构成（也就是系统的系统/综合系统）的情况。测试重点在于所有作为单个组件的系统（在综合系统中，单个系统也作为综合系统的组件）的可用性。
  b) 系统或服务是从外部（例如，从第三方供应商）获得的情况。测试重点在于可用性级别的测量，以便于确保能遵守商定的服务等级。

可用性可以使用专门的监视工具或者通过执行特定的测试进行测量。这样的测试通常是自动化的，并且可以和正常的操作并行运行，前提是测试不影响正常的运行（例如，不降低性能效率）。

4.4.6 可靠性测试计划

总体上来说，以下方面在规划可靠性测试时非常重要：

- 在软件进入生产阶段后，可以继续监视其可靠性。当以测试规划为目的收集可靠性需求时，必须咨询负责软件操作的机构及其人员。
- 技术测试分析师可以选择可靠性增长模型（reliability growth model），该模型会显示在一段时间内的所预期的可靠性级别。可以通过可靠性增长模型对预期的和实际达到的可靠性级别进行比较，给测试经理提供有用的信息。
- 可靠性测试应在生产环境的近似环境中进行。所使用的环境应尽可能保持平稳，使得能在一个时间段内对可靠性的趋势进行监测。
- 由于可靠性测试往往需要使用整个系统，所以最常见的方式是将可靠性测试作为系统测试的一部分。不过，也可以针对单独的组件以及集成后的组件集进行可靠性测试。详细的架构、设计和代码评审也可以用于去除一部分发生于已实现的系统的可靠性问题的风险。
- 为了产生有统计学意义的测试结果，可靠性测试通常需要很长时间来执行。可能难以将其安排于其它计划好的测试时段以内执行。

4.4.7 可靠性测试的规格说明

可靠性测试的形式有可能是一组重复进行、提前确定的测试。这些测试可能随机选择自某个测试库，也可能是通过某个使用了随机或伪随机方法的统计模型生成的测试用例。测试也可能以有时被称为“运行配置（Operational Profiles）”的使用模式为依据（参见第 4.5.3 节）。

在安排可靠性测试自动地与正常操作并行运行的情况下（例如，为了测试可用性），通常将可靠性测试定义的尽可能简单，以避免对系统的性能效率产生负面影响。

某些可靠性测试可能要求重复地执行内存使用较多的动作，以便检测出可能存在的内存泄漏问题。

4.5 性能效率测试

4.5.1 性能效率测试类型

4.5.1.1 负载测试

负载测试的关注点在于系统在处理预期实际负载级别增长方面的能力，实际负载级别增长来源于大量并发用户或者进程所产生的交易/事务请求。可以测量和分析在不同的典型使用场景（运行配置 operational profiles）情况下系统的平均响应时间。请参见[Splaine01]。

4.5.1.2 压力测试

压力测试关注的是系统或组件在达到或超过其预期或指定的工作负载的界限，或在可用带宽等资源可用性减少的情况下处理峰值负荷的能力。在负载级别逐渐增加时，系统性能效率应该按照预期缓慢下降，而不致失效。特别应该在峰值负载下对系统的完整功能进行测试，以发现在功能处理或数据不一致方面的缺陷。

压力测试的另一目标是确定系统崩溃的临界点，从而发现系统中的最薄弱环节。在进行压力测试时，允许向系统中逐渐地添加额外的容量（如内存、CPU 处理能力、数据库存储量）。

4.5.1.3 可扩展性测试

可扩展性测试关注于系统满足未来效率要求的能力（可能超过目前要求）。测试的目的是确定在没有超过目前规定的性能效率要求或系统不失效的情况下系统扩展的能力（例如，容纳更多的用户，存储更多的数据）。了解这些可扩展性限度后，可以设定和监控一些阈值以便在运行过程中对可能出现的问题进行预警。此外，可以通过适量的硬件来调整生产环境以满足预期的需求。

4.5.2 性能效率的测试计划

除了 4.2 节中描述的总体规划问题外，以下因素可以影响性能效率的测试计划：

- 根据所使用的测试环境和所测试的软件，（参见4.2.3节）性能效率测试可能要求在做有效的测试之前，已经实现了整个系统。因此，通常是在系统测试阶段中安排性能效率测试。其他在组件测试级别能够有效开展的性能效率测试，可以在组件测试级别中安排。
- 一般来说，即使还不存在类似实际生产的环境，最初的性能效率测试也是越早进行越好。这些早期测试可能会发现性能效率问题（如瓶颈），且通过避免在此后的软件开发阶段或投入生产后再去从事耗时的错误修复来减少项目风险。
- 代码评审，尤其是专注于数据库的交互、组件交互和错误处理方面的评审，能识别性能效率问题（特别是关于“等待并重试（wait and retry）”的逻辑和低效的查询），这类代码评审应安排在软件开发生命周期的早期进行。

- 运行性能效率测试所需要的硬件、软件和网络带宽应在计划内得到保障并编入预算。需求主要取决于要生成的负载，该负载是基于模拟的虚拟用户的数量以及它们可能产生的网络流量。如果在计划中没有考虑这点，可能导致执行了不具代表性的性能效率测试。例如，验证大访问量的互联网网站的可扩展性需求，可能需要模拟数十万的虚拟用户。
- 生成性能效率测试所需的负载可能对硬件及工具采购费用具有显著影响。这必须在计划性能效率测试的时候考虑在内，来确保有足够的资金可用。
- 可以通过租用所需的测试基础设施，使性能效率测试生成负载的成本最小化。例如，这可能涉及到租用顶级性能工具或使用一个第三方的服务提供商以满足硬件需求（例如,云服务）。如果采取此方法，用于性能效率测试的时间受到限制，因此必须精心规划。
- 在计划阶段应小心谨慎，以确保要使用的性能工具提供了与被测系统所使用的通信协议所需的兼容性。
- 与性能效率相关的缺陷往往对被测系统有重大影响，当系统的性能效率需求是非常重要的时候，有用的做法是（通过驱动器和桩）在关键组件上进行性能效率测试，这样测试可以在生命周期的早期开始，而不是等待系统级别的测试。

基础级性能效率测试教学大纲[ISTQB_FLPT_SYL]包括性能效率测试计划更多的细节。

4.5.3 性能效率测试的规格说明

不同性能效率测试（例如，负载测试和压力测试）的测试规格说明是基于所定义的运行配置（operational profiles）的，而运行配置体现了与应用程序交互的用户行为的不同形式，对于一个给定的应用程序可以有多个运行配置。

可使用监测工具（如果实际或类似的应用程序可供使用）或通过预测来确定每个运行配置的用户数目。这些预测值可以是基于算法或者由业务部门提供，这些对于确定可扩展性测试的运行配置非常重要。

在性能效率测试过程中，运行配置是所使用测试用例数目和类型的基础，通常使用测试工具为被测运行配置生成大量“虚拟”用户或模拟用户来控制测试。(参见 6.2.2 节)

基础级性能效率测试教学大纲[ISTQB_FLPT_SYL]包括性能效率测试设计更多的细节。

4.5.4 性能效率的质量子特性

ISO 25010 产品特性质量分类标准包括以下性能效率子特性：

- 时间特性- 组件或系统在特定的时间和规定的条件下，响应用户或系统输入的能力。
- 资源利用性- 软件产品使用适当数量和类型资源的能力。
- 容量-可以处理特定参数的最大限量。

4.5.4.1 时间特性

时间行为的关注点在于组件或系统在特定的时间和规定的条件下，响应用户或系统输入的能力。时间行为的测量会随着测试目的的变化而变化。针对单独的软件组件，时间行为可以根据 CPU 周期来测量；而针对基于客户的系统，时间行为可以根据其响应特定用户请求所需的时间来测量。对那些架构是由多个组件构成的系统（例如，客户端、服务器、数据库），时间行为的测量是针对单独的组件之间的交易/事务，从而可以识别出时间行为的“瓶颈”。

4.5.4.2 资源利用性

与资源利用性有关的测试是根据定义的基准数据评估系统资源的使用情况（例如，内存的使用率、硬盘容量、网络带宽和连接）。要在正常负载下和压力条件下，如大量交易或大量数据量，对这些系统资源的利用进行比较，从而确定是否有不寻常的使用增长。例如，对嵌入式实时系统来说，内存使用情况（有时也称为“内存占用”）在性能效率测试中扮演了一个非常重要的角色。如果内存占用超过所允许的程度，系统可能因为没有足够内存而不能在特定时间内执行要求的任务，可能会拖慢系统，甚至导致系统崩溃。

此外，动态分析也可应用于检测资源利用性（参见 3.3.4 节）和识别性能效率瓶颈的任务中。

4.5.4.3 容量

系统（包括软件和硬件）的容量代表了可以处理特定参数的最大限量。容量需求通常由技术和运营利益相关方指定，并可能与一些参数相关，例如在给定时间点可以使用应用程序的最大用户数量、每秒可以传输的最大数据量（即带宽）以及每秒可以处理的最大交易/事务数量。

测试容量限量的方法通常与 4.5.2 和 4.5.3 节中描述的测试性能效率的方法相似。容量测试的运行配置（operational profiles）关注于生成执行特定限制的负载。例如，这可能涉及到生成使系统达到最大数据传输量的负载。压力测试和可扩展性测试方法也可以应用于容量，用来测试超出指定容量限量的系统特性（分别参见 4.5.1.2 和 4.5.1.3 节）。

4.6 维护性测试

在软件的生命周期中，软件的维护阶段占据了大部分时间，与此相比，软件开发阶段只占有较少的时间，维护性测试是用来测试变更对操作系统或其环境的影响。为了确保对一个系统的维护尽可能高效，要执行维护性测试来探索如何对程序代码更容易地进行分析、更改和测试。

受影响的项目利益相关方（例如，软件所有者或操作者）的典型的维护性目标包括：

- 拥有或运行软件的成本降至最低
- 软件维护所需的停机时间最少

以下一个或多个因素发生时，测试方法中应包括维护性测试：

- 软件的变化可能发生在软件上线后（例如，纠正缺陷或引入计划中的更新）
- 受影响的项目利益相关方认为在软件生命周期中实现维护性目标（见上文）的收益远大于执行维护性测试的成本以及做出任何必要的变更所需的成本

- 软件维护性差的风险（例如，对用户和/或客户报告的缺陷的响应时间过长）证明进行维护性测试是正确的

4.6.1 静态和动态维护性测试

如 3.2 节和 5.2 节中所讨论的，针对维护性测试的有效方法包括静态分析和评审。维护性测试应该在设计文档就绪后就开始，并在整个编码实现过程中不断持续。由于维护性嵌入到每个组件的代码和相应的文档，因此可以在软件开发生命周期的早期对维护性进行评估，而无需等待一个完整的已在运行的系统。

动态维护性测试的关注点在于文档化规程，开发这些规程是为了维护一个特定应用程序（如进行软件升级）。测试时可以选取维护场景作为测试用例，确保使用文档化的规程就既可达到所要求的服务等级。此测试方法尤其适用于下列情况：底层基础设施相对复杂，并需要多个部门/组织协作支持。这些测试同时可以作为运行验收测试的一部分。

4.6.2 维护性子特性

系统的维护性可以根据两个方面测量：诊断系统内已经识别的问题所需的工作量（易分析性）和测试更改系统所需的工作量（易测试性）。影响易分析性和易测试性的因素包括良好编程实践的应用（例如，注释、变量名、缩进）和技术文档的可用性（例如，系统设计规格说明、接口规格说明）。

其他与维护性相关的质量子特性[ISO25010]为：

- 易修改性- 在不引入缺陷或降低现有产品质量的情况下，有效和高效地修改组件或系统的程度
- 模块化-系统、产品或组件由独立组件组成的程度，这样一个组件的变化对其他组件的影响最小
- 可重用性- 一项资产可用于多个系统或构建其他资产的程度

4.7 可移植性测试

4.7.1 简介

可移植性测试通常和软件组件或系统移植到预期环境中的难易程度相关，包括第一次的安装或从现有环境移植。

[ISO25010]包括以下可移植性的子特性：

- 易安装性-软件产品在特定环境下安装的能力
- 适应性-组件或系统能够适应不同或者是不断变化的硬件和软件环境的程度
- 易替换性-在相同的环境下，为了相同的目的，用另一个软件产品代替指定的软件产品的能力

可移植性测试可以从单独的组件开始（例如，一个特定的组件的易替换性，比如从一个数据库管理系统换到另一个），然后当更多代码可用时再扩大范围。易安装性在产品的所有组件都能正常工作之后才能测试。由于可移植性必须设计出来并且内置在产品中，因此这个质量特性在系统设计和系统架构的早期阶段就必须予以考虑。架构和设计的评审对识别潜在的可移植性需求和问题（例如：对特定操作系统的依赖性）富有成效。

4.7.2 易安装性测试

易安装性测试是在目标环境中安装软件，并文档化此安装规程。这可能包括为将操作系统安装到处理器而开发的软件，或用于在客户端 PC 电脑上安装产品的安装软件（向导 Wizard）。

典型的易安装性目标包括：

- 验证软件能根据安装手册（包括任何安装脚本的执行）上的指示，或通过使用安装向导顺利地进行安装。其中包括针对不同的软硬件配置，选择相应的选项进行安装，以及进行不同级别的安装（如初始安装或系统更新）。
- 测试安装软件是否能够正确处理安装过程中所出现的失效（例如无法安装某些DLL）现象，而不至于使系统处于某个不确定的状态（例如，软件只安装了一部分或造成错误的系统配置）
- 测试能否完成部分的安装/卸载
- 测试安装向导是否能成功识别无效的硬件平台或操作系统配置
- 测量是否能在指定的时间段内或在指定的步骤内完成整个安装过程
- 确认软件是否可以成功降级或卸载

在易安装性测试之后通常还要进行功能性测试，以检测任何在安装过程中可能引入的故障（例如，不正确的配置，不可用的功能）。在易安装性测试的同时一般还会进行易用性测试（例如，验证用户在安装过程中是否获得易懂的指示和反馈/出错信息）。

4.7.3 适应性测试

适应性测试检查一个给定的应用程序是否能在所有预期的目标环境中（硬件、软件、中间件、操作系统等）正常工作。因此，自适应系统是一个开放的系统，它的行为能适应环境的变化或适应自身部分系统的变化。在定义适用性测试时必须对预期目标环境的组合进行识别、配置并提供给测试团队。选择一组功能性的测试用例在这些环境中测试，而这些测试用例能在环境中检查应用程序的各个组成部分。

适应性还涉及到通过完成一个预定规程将软件移植到各种特定运行环境的能力。测试可以对该规程进行评估。

适应性测试还可以与易安装性测试一起进行，通常情况下，随后辅以功能测试，以检验软件在适应其他运行环境时是否会出现问题。

4.7.4 易替换性测试

易替换性测试侧重于系统中的软件组件替换成其他组件的能力。这对那些在特定的系统组件中使用商业现货软件（COTS）的系统来说尤为重要。

如果在完整系统的集成过程中存在可选的替代组件，则易替换性测试可以和功能集成测试一起进行。可以通过对系统架构或系统设计的技术评审或审查来对易替换性进行评估，重点就是为可能替换的组件定义明确的接口。

4.8 兼容性测试

4.8.1 简介

兼容性测试考虑以下几个方面[ISO25010]：

- 共存性 -在共享的环境下，测试项与其他独立的产品能一起正常工作的程度。下面将对此进行描述。
- 互操作性-系统与其他系统或组件交换信息的程度。这在ISTQB®高级水平测试分析师[ISTQB_ALTA_SYL]教学大纲中有所描述。

4.8.2 共存性测试

当互不相关的计算机系统能在同一个环境中运行（例如，相同的硬件）而不影响彼此的行为（例如，资源冲突）时，可以看作是共存的。当需要在已经安装了应用程序的环境中安装一个新的软件或进行软件的升级时，需要进行共存性测试。

在没有安装其他应用程序的环境中，可能检测不出软件的共存性问题，但如果将其部署到另一个已经安装了其他应用程序的环境（如产品环境）时，则可能会发生共存性问题。

典型的共存性测试目标包括：

- 评估在同一个运行环境中加载其他应用程序对功能性产生的负面影响（例如，当一个服务器运行多个应用程序时可能产生资源使用冲突）
- 评估由于部署操作系统的修补和升级而对每一个应用程序带来的影响。

在规划预定的目标环境时，就应该分析共存性问题，但实际测试通常是在系统测试已顺利完成后才执行。

5. 评审 - 165分钟

关键词

反模式（anti-pattern）

评审的学习目标

5.1 技术测试分析师的评审任务

TTA-5.1.1（K2）解释为何评审准备工作对技术测试分析师很重要。

5.2 在评审中使用检查表

TTA-5.2.1（K4）根据教学大纲提供的检查表来分析架构设计及识别问题。

TTA-5.2.2（K4）根据教学大纲提供的检查表来分析一段代码或伪代码并识别问题。

5.1 技术测试分析师的评审任务

技术测试分析师必须积极参与技术评审过程，并提供其独特的观点。所有评审参与者都应接受正式的评审培训，以便更好地了解在技术评审过程中他们各自扮演的角色，并且必须致力于从实施良好技术评审中受益。还包括在描述和讨论评审意见时，与作者保持建设性的工作关系。有关技术评审的完整描述说明，应包括众多评审检查表，请参阅[Wiegers02]。 技术测试分析师通常会参与技术评审和审查，会带来可能被开发人员忽略的操作（行为）方面的独特观点。此外，技术测试分析师在评审检查表和缺陷严重度信息的定义、应用和维护中也起着重要作用。

无论采用哪种类型的评审，都必须让技术测试分析师有足够的时间准备，包括评审工作产品的时间、检查交叉引用文档以验证一致性的时间、确定工作产品完整性的时间。如果没有足够的准备时间，则评审可能会成为校对作业，而不是真正的评审。良好的评审包括理解所写的内容、确定所遗漏的内容，以及验证所描述的产品与其他已开发或正在开发的产品是否一致。例如，在评审集成级别的测试计划时，技术测试分析师还必须考虑到正在集成的项，他们是否已经能够集成？是否有必须记录的依赖关系？是否有数据可用于测试集成点？评审不仅限于所评审的工作产品，它还必须考虑到与系统中其他项的交互。

5.2 在评审中使用检查表

检查表是一组在评审过程中用于提醒参与者验证的特定要点。检查表也有助于避免“个性化/针对性”的评审，例如，“我们每次评审都用相同的检查表，我们不只是针对您的工作产品。”检查表可以是通用的并且可用于所有的评审，也可以侧重于特定的质量特性或领域。例如，通用检查表可能会验证术语“应该（shall）”和“可能（should）”的正确用法，验证正确的格式和相似的一致性项。有针对性的检查表可能会集中在安全性问题或性能效率问题方面。

最有用的检查表是由组织制定和逐步发展起来的，因为它们反映了：

- 产品的本质
- 本地开发环境
  - 员工
  - 工具
  - 优先级
- 以往的成功和缺陷历史
- 特殊项（例如：性能效率，信息安全性）

应该为组织或特定的项目定制检查表。本章中所提供的检查表仅作为示例。

一些组织扩展了软件检查表的通常概念，使其包括“反模式”，这些反模式是指常见的错误、糟糕的技术以及其他无效的实践。“反模式”这一术语源自流行的“设计模式”概念，它是对常见问题可重复使用的解决方案，并在实际情况中已证明是有效的[Gamma94]。因此，反模式是一种常见的错误，通常作为权宜之计而实施的捷径。

要记住，如果一个需求是不可测试的，这意味着没有按技术测试分析师可以确定如何测试的方式来定义此需求，那么这就是一个缺陷。例如，无法测试一条描述为“软件应快速”的需求。技术测试分析师如何能确定软件是否是快速的？相反，如果这条需求是说“软件在特定负载条件

下，必须提供最大的响应时间为三秒”，那么在假设定义了“特定的负载条件”（例如并发用户的数量，用户执行的活动）的情况下，该条需求的可测试性将会大大提高。这也是一个非常重要的需求，因为这一需求很容易在一个重要的应用程序中产生大量独立的测试用例。从这个需求到测试用例的可追溯性也至关重要，因为如果需求发生变化，则所有的测试用例都根据需要进行评审和更新。

5.2.1架构评审

软件架构（architecture）由一个系统的基本组织构成，具体体现在其组件、组件之间的相互关系、环境中，以及控制其设计和演进的原则中。[ISO42010]，[Bass03]。

用于架构评审的检查表，例如：包括查证是否正确实现了以下各项（引用自[Web-2]）：

- 连接池 -通过建立一个共享的连接池减少建立数据库连接相关的执行时间
- 负载平衡 -将负载均匀地分散在一组资源之中
- 分布式处理
- 缓存 - 用本地数据拷贝来减少访问时间
- 惰性实例化
- 交易/事务并发
- 在线事务处理（OLTP）和在线分析处理（OLAP）之间的进程隔离
- 数据复制

5.2.2代码评审

代码评审的检查表必须非常详细，并且只有当它们是针对特定语言、某个项目和某个公司量身定制时是最有用的，这一点与架构评审的检查表一样。包括在代码级别引入反模式也非常有用，特别是对于那些缺少经验的软件开发人员更有帮助。

用于代码评审的检查表[1]可以包括以下六个方面：

1. 结构
   - 代码是否完整地、正确地实现了设计？
   - 代码是否符合所有相关的编码标准？
   - 代码是否结构合理、风格统一、格式一致？
   - 是否有任何未调用或不需要的过程（子程序）或任何不可达的代码（死代码）？
   - 代码中是否有任何残留的桩或测试程序？
   - 是否可以通过调用外部可重用组件或库函数来替换任何代码？
   - 是否有任何重复的代码（段）块可以压缩成单个过程（子程序）？

[1]考试问题将提供一个用于回答问题的检查表子集

- 存储使用是否高效?
- 是否使用有意义的符号而不是“幻数（magic number）”常量或字符串常量?
- 是否有任何过于复杂的模块需要重组或拆分为多个模块?

2. 文档
   - 代码是否以易于维护的注释风格进行了清楚且充分的文档化?
   - 是否所有的注释都与代码相对应?
   - 文档是否符合适用的标准?
3. 变量
   - 是否所有变量的命名都是有意义的、一致的、清晰的?
   - 是否有任何多余或未使用的变量?
4. 数值运算
   - 代码中是否避免了比较浮点数相等?
   - 代码是否可以系统地防止舍入错误?
   - 代码是否避免了对巨大数量级差别的数字进行加减运算?
   - 是否测试了除数是零或噪声?
5. 循环和分支
   - 所有循环、分支和逻辑结构是否完整、正确和适当嵌套?
   - 是否首先测试了IF-ELSEIF链中最常见的情况?
   - 是否涵盖了IF-ELSEIF或CASE块中所有的情况，包括ELSE或DEFAULT分支?
   - 是否每个case语句都有default?
   - 循环终止条件是否明显且总是可以实现?
   - 循环之前是否正确初始化了索引或下标?
   - 循环体内包含的任何语句是否可以置放于循环体外?
   - 循环中的代码是否避免了操纵索引变量或在退出循环时使用它?
6. 防御性编程
   - 所有索引、指针和下标是否参照数组、记录或文件边界进行了测试?
   - 是否对导入的数据和输入的参数都进行了有效性和完整性的测试?
   - 是否赋值了所有输出变量?
   - 每个语句中是否都对正确的数据元素进行了操作?
   - 是否释放了每个内存分配?
   - 访问外部设备时是否有访问超时或错误捕获?

- 是否在尝试访问文件之前检查文件是否存在？
- 程序终止后，所有文件和设备是否都处于正确的状态？

6. 测试工具与自动化-180分钟

关键词

录制/回放（capture/playback）、数据驱动测试（data-driven testing）、调试（debugging）、仿真器（emulator）、故障植入（fault seeding）、超链接（hyperlink）、关键词驱动测试（keyword-driven testing）、性能效率（performance efficiency）、模拟器（simulator）、测试执行（test execution）、测试管理（test management）。

测试工具与自动化的学习目标

6.1 定义测试自动化项目

TTA-6.1.1（K2）总结建立一个测试自动化项目时，技术测试分析师需要执行的活动。

TTA-6.1.2（K2）总结数据驱动自动化和关键词驱动自动化的区别。

TTA-6.1.3（K2）总结造成自动化项目无法达到计划的投资回报的通用技术问题。

TTA-6.1.4（K3）根据给出的业务流程创建关键词。

6.2 特定的测试工具

TTA-6.2.1（K2）总结故障植入和故障注入工具的用途。

TTA-6.2.2（K2）总结性能测试工具的主要特性和实施方面的问题。

TTA-6.2.3（K2）解释用于基于网页测试的工具的一般用途。

TTA-6.2.4（K2）解释工具如何支持基于模型的测试。

TTA-6.2.5（K2）概述用于支持组件测试和构建（build）流程的工具的用途。

TTA-6.2.6（K2）概述用于支持移动应用测试的工具的用途。

6.1 定义测试自动化项目

应该对测试工具以及支持测试执行的工具进行仔细的构建和设计，以保证成本的有效性。如果缺少可靠的架构，实施测试执行自动化策略通常会导致这个工具集的维护成本高昂、不能充分实现既定目标，且无法实现投入产出比的目标。

测试自动化项目应该被视为一个软件开发项目，包括需要有架构文档、详细设计文档、设计和代码评审、组件和组件集成测试，以及最后的系统测试。如果使用了不稳定或不准确的测试自动化代码，测试可能会被不必要地拖延或是复杂化。

针对测试执行自动化，技术测试分析师可以执行的任务有多个，包括：

- 确定将由谁负责测试执行（可能与测试经理协调）
- 根据组织、时间表、团队技能、维护需求挑选适宜的工具（注意：可能是决定开发一个工具而不是购买工具来使用）
- 定义自动化工具和其他工具（例如测试管理工具、缺陷管理工具和用于持续集成的工具）之间的接口需求
- 开发建立测试执行工具和被测软件之间的接口时可能需要的任何适配器
- 选择自动化的方法，即关键词驱动还是数据驱动（参见6.1.1）
- 与测试经理一起估算包括培训成本在内的实施成本。对于敏捷项目，一般会在项目/冲刺（Sprint）策划会议中与整个团队一起讨论并达成一致
- 安排自动化项目的时间进度并为维护工作分配时间
- 培训测试分析师和业务分析师如何使用自动化工具，并为自动化工具提供数据
- 确定自动化测试执行的方法和时间
- 确定如何将自动化测试结果与手工测试结果相结合

对于高度强调测试自动化的项目，这些活动中有很多可能是测试自动化工程师的任务（详情参见高级测试自动化工程师教学大纲[ISTQB®_ALTAE_SYL]）。根据项目的需要和偏好，某些组织工作可能会由测试经理承担。而敏捷项目中，这些任务的角色分工一般会更灵活且不那么正式。

这些活动和活动带来的决策会影响自动化解决方案的可扩展性和维护性。必须花费足够的时间来研究不同的选择，调查可用的工具和技术，以及理解组织未来的计划。

6.1.1选择自动化方法

本章节考虑了下列影响测试自动化方法的因素：

- 基于GUI的自动化
- 应用数据驱动方法
- 应用关键词驱动方法
- 处理软件失效
- 考虑系统状态

在高级测试自动化工程师教学大纲[ISTQB_ALTAE_SYL]内有关于选择自动化方法的进一步内容。

6.1.1.1 基于 GUI 的自动化

测试自动化并不局限于基于 GUI 的测试。有一些工具在 API 层面通过一个命令行接口（CLI）和被测软件的其他接口点来实现自动化测试。技术测试分析师首先要做的决策之一就是为测试自动化确定最有效的访问接口。一般的测试执行工具都需要开发对这些接口的适配器。因此策划时应考虑适配器开发的工作量。

基于 GUI 的测试的难点之一是随着软件开发的进展，软件的 GUI 也会随着变化，这些变化又由于测试自动化代码的设计方式不同，可能会给维护工作带来很大的负担。比如，如果使用了测试自动化工具的录制/回放功能，一旦 GUI 变化，就有可能导致已经自动化的测试用例（一般称为测试脚本）不再按预期运行。这是因为录制的脚本抓取的是测试人员手动运行软件时与图形对象的交互。如果被访问的对象发生了变化，录制的脚本就需要更新以反映这种变化。

录制/回放工具可以方便地用作开发自动化脚本的起始点。测试人员录制一段测试会话，然后对录制的脚本进行修改，以提高其可维护性（例如，把所录制脚本中原有的功能替换为可重用功能）。

6.1.1.2 应用数据驱动方法

虽然执行的测试步骤几乎相同（例如，通过输入多个无效值并检查每个输入返回的错误来测试某个输入字段的错误处理），但是由于测试的软件不同，用于每个测试的数据也可能会有所不同。为每一个待测值都开发并维护一个自动化测试脚本是非常低效的。解决这一问题的通用技术方案是将数据从脚本中剥离并放置于某种外部存储，例如电子表格或数据库。创建相应的函数，以便在每次运行测试脚本时获取特定数据，从而让一个脚本处理包含输入值和预期结果值（例如文本字段显示的值或错误消息）的一整组测试数据。这种方法称为数据驱动方法。

使用这种方法时，除了处理所提供数据的测试脚本之外，还需要测试用具和基础设施来支持该脚本或该组脚本的执行。熟悉软件业务功能的测试分析师在电子表格或数据库中创建实际数据。敏捷项目中业务代表（如产品负责人）也可能参与数据的定义，尤其是用于验收测试的数据的定义。这种分工让负责开发测试脚本的人（如技术测试分析师）可以专注于实现智能的自动化脚本，而测试分析师仍然负责实际的测试。多数情况下，一旦自动化已经实现并经过测试，将由测试分析师负责执行测试脚本。

6.1.1.3 应用关键词驱动方法

另一种方法称为关键词驱动或动词驱动，这种方法更进一步将对所提供数据进行操作的行为也从测试脚本分离[Buwalda01]。为了实现这种进一步的分离，要创建一个高级的元语言，该语言是描述性的，不可直接执行。这种语言的每个语句都描述了可能需要测试的域的完整或部分业务流程。比如，业务流程的关键词可能包括“Login”、“CreateUser”和“DeleteUser”。关键词描述应用域中进行的一种高级操作。而具体的操作行为描述了与软件接口本身的交互，例如

“ClickButton”、“SelectFromList”或“TraverseTree”，这些具体行为可用来测试 GUI 的能力，但并不完全适合业务流程关键词。

一旦要使用的关键词和数据得到定义，测试自动化人员（如技术测试分析师或测试自动化工程师）就会将业务流程关键词和更具体的操作行为转换成测试自动化代码。这些关键词和操作行为，以及要使用的数据，可存储于电子表格或使用支持关键词驱动测试自动化的特定工具来导入。测试自动化的框架将这些关键词实现为一组包含一个或多个可执行的函数或脚本。工具读取使用关键词编写的测试用例，并调用实现这些关键词对应的测试函数或脚本。这些可执行程序以高度模块化的方式实现，以便能方便地映射到特定的关键词。实现这些模块化的脚本需要一定的编程技能。

这种将业务逻辑知识与实现测试自动化脚本所需的实际编程进行分工的方法让测试资源得到最有效的利用。技术测试分析师作为测试自动化人员时，能够有效地应用其编程技能，而不需要成为跨业务领域的行业专家。

通过从不断变化的数据中分离代码可以避免测试自动化的不断修改，提高代码整体的维护性并提高自动化的投资回报率。

6.1.1.4 处理软件失效

在任何测试自动化设计中，预防和处理软件失效都是一个重点。如果出现失效，测试自动化人员必须确定软件应该如何处理。是应该把失效记录下来并继续运行测试？还是应该终止测试？是否能用某个特定的操作（比如点击对话框中的一个按钮）或是在测试中增加一个延迟来处理这个失效？未处理的软件失效可能会破坏后续测试的结果，也可能会导致正在执行的测试出现问题。

6.1.1.5 考虑系统状态

同样重要的是考虑系统在测试开始和测试结束时的状态。需要确保系统在测试执行完成之后返回到一个预先定义的状态。这种做法让自动化测试套件可以反复运行，而不需要人工复位系统。要做到这点，测试自动化应删除其生成的数据或改变记录在数据库中的状态。自动化框架应该确保测试结束时实现某种适当的终止（即测试完成后退出）。

6.1.2自动化的业务流程建模

为了实现关键词驱动的测试自动化方法，必须用高级的关键词语言对要测试的业务流程进行建模。重要的是该语言对于其使用者（可能是参与该项目的测试分析师，在敏捷项目中，则是业务代表如产品负责人）来说是直观的。

关键词一般用来映射高级别业务与系统的交互。例如，“Cancel_Order”可能需要检查订单是否存在、验证要求注销的人是否有访问权限、显示应取消的订单并要求确认取消。测试分析师使用关键词的序列（例如“Login”、“Select_Order”、“Cancel_Order”）和相关的测试数据来定义测试用例。下面是一个简单的关键词驱动输入表，可以用来测试软件创建、重置和删除用户账号的能力。

关键词	用户	密码	结果

Add_User	User1	Pass1	用户已添加信息
Add_User	@Rec34	@Rec35	用户已添加信息
Reset_Password	User1	Welcome	密码重置确认信息
Delete_User	User1		无效用户名/密码信息
Add_User	User3	Pass3	用户已添加信息
Delete_User	User2		用户未找到的信息

该表的自动化脚本会寻找该自动化脚本所使用的输入值。例如，当运行到关键词“Delete_User”所在行时只需要有用户名。创建新用户时，既需要用户名也需要密码。输入值也可以引用自某个数据存储，如第二个“Add_User”关键词所示，这里输入的是数据的引用而不是数据本身，这增加了访问数据的灵活性，使得在测试执行中也能访问变化的数据。这种方式让数据驱动技术可以与关键词方案相结合。

要考虑的问题如下：

- 关键词的颗粒度越精细，能覆盖的场景就越具体，但高级别语言的维护可能变得越复杂。
- 让测试分析师来定义具体的操作行为（“ClickButton”、“SelectFromList”等）会让关键词驱动测试能更好地应对不同的情况。然而因为这些操作与GUI直接关联，也会由于变更而导致测试的高昂的维护费用。
- 使用聚合关键词可能会让开发简化，却让维护复杂化。例如，可能有六个不同的关键词共同组成了一条记录。是否应该创建一个连续调用了六个关键词的单个关键词来简化操作？
- 无论对关键词语言进行了多少分析，还是常常会需要新的和不同的关键词。一个关键词有两个不同的方面（即其背后的业务逻辑和执行该关键词的自动化功能），因此必须创建一个两方面都能处理的过程。

基于关键词的测试自动化可以显著降低测试自动化的维护成本，但这种自动化的开发成本更高、难度更大，而且需要花更多的时间来正确地设计，才能获得预期的投资回报。

高级测试自动化工程师教学大纲[ISTQB_ALTAE_SYL]包含有关自动化的业务流程建模的进一步内容。

6.2 特定的测试工具

除了基础级教学大纲[ISTQB_FL_SYL]中讨论过的工具外，本章节也包括了一些技术测试分析师很可能会用到的工具的概要信息。

注意：下列 ISTQB®教学大纲提供了关于工具的详细信息。

- 移动应用测试[ISTQB_FLMAT_SYL]
- 性能效率测试[ISTQB_FLPT_SYL]
- 基于模型的测试[ISTQB_FLMBT_SYL]

- 测试自动化工程师[ISTQB_ALTAE_SYL]

6.2.1故障植入/故障注入工具

为了检查既定测试达到的覆盖，故障植入工具实际上会修改被测试的代码（可能通过提前定义的算法）。如果以系统化的方式应用这种工具，就能够对测试的质量（即测试发现被植入缺陷的能力）进行评价，并在必要时提高测试的质量。

故障注入工具会故意向软件提供不正确的输入，以确保软件能够应对这种故障。注入这些输入来破坏代码的正常执行流程，从而扩大测试的覆盖（例如，覆盖更多的逆向测试条件和测试错误处理机制）。

这两种类型的工具一般都是技术测试分析师在使用，但在测试新开发的代码时，开发人员也有可能使用这些工具。

6.2.2性能测试工具

性能测试工具有下列主要功能：

- 生成负载
- 测量、监视、可视化和分析给定负载下的系统响应
- 提供对系统和网络组件的资源行为的深入了解

生成负载是通过预先定义的运行配置(operational profile)(参见 4.5.3)作为脚本来实现。脚本最初可能是为单个用户形成的（可能用到录制/回放工具），然后再使用性能测试工具实现特定的运行配置。实现时必须考虑每个交易/事务（或一系列交易/事务）的数据变化。

性能工具按照规定的运行配置模拟大量的并发用户（“虚拟”用户）生成一个确定量的输入数据的负载。与单个的测试执行自动化脚本不同，很多性能效率测试脚本会在通讯协议层面再现用户与系统的交互，而不是通过图形用户界面来模拟用户交互。这种方式通常会减少测试过程中需要的单独“会话”的数量。某些负载生成工具还能通过其用户界面来控制应用，从而更准确地测量系统在负载下的响应时间。

性能测试工具会提供多种度量数据，可以用于测试执行期间或测试执行之后的分析。记录的数据和提供的报告通常包括：

- 整个测试中模拟的用户数量
- 由模拟用户产生的交易/事务的数量和类型，以及交易/事务的输入率
- 对用户提出的特定交易/事务请求的响应时间
- 负载对应响应时间的报告和图表
- 资源使用情况的报告（如随时间推移的使用情况，包含最小值和最大值）

性能测试工具的实施中要考虑的主要因素包括：

- 生成负载所需的硬件和网络带宽
- 待测系统所使用的通讯协议与工具的兼容性

- 工具的灵活性，以保证不同的运行配置易于执行
- 监视、分析和报告所需的功能

由于开发性能测试工具需要很大的投入，因此这些工具通常都是采购而不是内部进行开发。然而，如果因为技术的限制而不能使用现有的产品，或者所需的负载配置和功能都相对简单，也可以专门开发一个性能测试工具。基础级性能效率测试教学大纲[ISTQB_FLPT_SYL]有关于性能测试工具的进一步内容。

6.2.3基于网页测试的工具

网页测试可以使用各种开源的和商业定制的工具。下面的列表体现了部分常用的基于网页测试工具的用途：

- 超链接测试工具用来扫描和检查网站上是否有损坏或缺失的超链接
- HTML和XML检查工具用来检查网站创建的页面是否符合HTML和XML标准
- 负载模拟器用来测试当大量用户连接时服务器将如何应对
- 与不同浏览器一起工作的轻量化自动化执行工具
- 扫描整个服务器检查孤儿（未链接）文件的工具
- HTML专用拼写检查器
- 风格样式表（CSS）检查工具
- 检查是否违反了标准的工具，例如美国的第508条无障碍（可达性）标准或欧洲的M/376
- 发现各种信息安全性问题的工具

一些好的开源网页测试工具来源包括：

- 万维网联盟（W3C）[Web-3] 。该组织建立了英特网的标准并提供各种工具来检查违反了这些标准的错误。
- Web超文本应用技术工作组（WHATWG）[Web-5] 。该组织建立了HTML标准，有一个工具用来进行HTML确认[Web-6]。

一些包含网络爬虫引擎（web spider engine）的工具也可以提供页面大小、下载这些页面所需的时间、页面是否存在（例如 HTTPerror404）方面的信息。这为开发人员、网站管理员和测试人员提供了有用的信息。

测试分析师和技术测试分析师主要在系统测试阶段使用这些工具。

6.2.4支持基于模型测试的工具

基于模型的测试（MBT）是使用形式化模型的一种技术，用类似有限状态机来描述由软件控制的系统的预期执行时间行为。商用 MBT 工具（参见[Utting07]）通常会提供一个引擎，让用户可以“执行”该模型。值得注意的执行线程可以保存下来并用作测试用例。佩特里网（PetriNets）和状态图（Statecharts）等其他可执行模型也支持 MBT。

MBT 模型（和工具）可以用来生成大量的不同执行线程。MBT 工具也有助于减少模型产生大量可能路径。使用这些工具进行测试可以提供关于待测软件的不同看法，从而可能发现一些功能测试未发现的缺陷。

基础级基于模型的测试教学大纲[ISTQB_FLMBT_SYL]中有关于基于模型测试工具的进一步内容。

6.2.5组件测试和构建工具

尽管组件测试和自动化构建工具都是开发人员的工具，但在很多情况下，都是由技术测试分析师使用和维护，尤其是在敏捷开发的环境中。

组件测试工具通常是根据软件模块编码的编程语言确定的。例如，如果使用 Java 作为编程语言，可能会用 JUnit 来做自动化单元测试。很多其他语言有它们自己专门的测试工具，这些工具统称为 xUnit 框架。这种框架为创建的每一个类生成测试对象，从而简化自动化组件测试时程序员需要进行的任务。

调试工具在一个非常低（细致）的层面可协助手动组件测试，让开发人员和技术测试分析师可以在执行过程中改变变量的值，并且在测试过程中逐行地执行代码。当测试团队报告了失效时，调试工具也可用来协助开发人员分离和识别代码中的问题。

自动化构建工具通常可以在组件每次变化时，自动触发一个新的构建。该构建完成后，其他工具会自动执行组件测试。围绕内部版本流程的自动化通常出现在持续集成的环境中。

这套工具在设置正确的情况下能对即将测试的内部版本的质量有积极的影响。程序员所做的改变导致的缺陷，通常会引起自动化测试的失败，在内部版本发布到测试环境前，立即触发失败原因调查。

6.2.6支持移动应用测试的工具

模拟器和仿真器是经常用于支持移动应用测试的工具。

6.2.6.1 模拟器

移动模拟器会模拟移动平台运行时的环境。在模拟器上进行测试的应用会被编译为一个专用版本，该版本能在模拟器上运行但不能在真实的设备上运行。测试中有时候会用模拟器代替真实的设备。但是，在模拟器上测试的应用跟将要发布的应用之间存在差异。

6.2.6.2 仿真器）

移动仿真器会模拟硬件并使用和物理硬件相同的运行时环境。经过编译要在仿真器上部署和测试的应用也可以用在真实的设备上。

仿真器常被用来代替真实的设备从而降低测试环境的成本。然而，由于仿真器的行为方式可能与其模仿的移动设备不同，因此仿真器无法完全代替该设备，类似（多点）触摸、加速度计的一些特征可能不受支持。这种情况部分是由用于运行仿真器的平台的限制造成的。

6.2.6.3 通用方面

由于模拟器和仿真器通常与开发环境集成，而且可以实现应用的快速部署、测试和监视，所以在开发的早期阶段是有用的。使用仿真器或模拟器需要启动该仿真器或模拟器，在上面安装必要的应用，然后像在实际设备上一样测试该应用。每种移动操作系统的开发环境一般都自带绑定的仿真器和模拟器。也有第三方的仿真器和模拟器可供使用。

仿真器和模拟器通常都可以设置各种使用参数。这些设置可能包括仿真不同的网络速度、信号强度和丢包率，改变方向，产生中断以及 GPS 位置数据。由于部分设置（比如全球 GPS 位置或信号强度）用真实设备再现较为困难或成本高昂，所以有时非常有用。

基础级移动应用测试教学大纲[ISTQB_FLMAT_SYL]包含了进一步的内容。

7. 参考资料

7.1 标准

以下标准分别在所列出的章节中涉及到。

- [RTCA DO-178C/ED-12C]: Software Considerations in Airborne Systems and Equipment Certification/机载系统和设备认证中的软件考虑, RTCA/EUROCAE ED12C.2013.第2章

- [ISO9126] ISO/IEC 9126-1:2001, Software Engineering - Software Product Quality/软件工程－软件产品质量 第4章

- [ISO25010] ISO/IEC 25010 (2014) Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) System and software qualitymodels

  系统和软件工程－系统和软件质量要求和评价（SQuaRE）系统与软件质量模型第2章和第4章

- [ISO29119] ISO/IEC/IEEE 29119-4 International Standard for Software and Systems Engineering - Software Testing Part 4: Test techniques.2015

  软件和系统工程国际标准-软件测试第4部分：测试技术， 2015年第2章

- [ISO42010] ISO/IEC/IEEE42010:2011

  Systems and software engineering - Architecture description 系统和软件工程-架构描述第5章

- [IEC61508] IEC 61508-5 (2010) Functional Safety of Electrical/Electronic/Programmable ElectronicSafety-relatedSystems,Part5:Examplesofmethodsforthedeterminationof safety integritylevels

  电气/电子/可编程电子安全相关系统的功能安全，第5部分：确定安全完整性等级的方法示例第2章

7.2ISTQB®文档

- [ISTQB_AL_OVIEW]Advanced Level Overview, Version2019 高级教学大纲-概述，2019版
- [ISTQB_ALSEC_SYL]Advanced Level Security Testing Syllabus, Version2016高级-安全性测试教学大纲，2016版
- [ISTQB_ALTAE_SYL]Advanced Level Test Automation Engineer Syllabus, Version2017 高级-测试自动化工程师教学大纲，2017版
- [ISTQB_FL_SYL] Foundation Level Syllabus, Version2018 基础级教学大纲，2018版
- [ISTQB_FLPT_SYL]Foundation Level Performance Testing Syllabus, Version2018 基础级-性能效率测试教学大纲，2018版
- [ISTQB_FLMBT_SYL]Foundation Level Model-Based Testing Syllabus, Version2015 基础级-基于模型测试教学大纲，2015版
- [ISTQB_ALTA_SYL]Advanced Level Test Analyst Syllabus, Version2019 高级-测试分析师教学大纲，2019版
- [ISTQB_ALTM_SYL]Advanced Level Test Manager Syllabus, Version2012 高级-测试经理教学大纲，2012版
- [ISTQB_FLMAT_SYL] Foundation Level Mobile Application Testing Syllabus,2019 基础级-移动应用测试教学大纲，2019版
- [ISTQB_GLOSSARY] Glossary of Terms used in Software Testing, Version 3.2,2019 软件测试中使用的术语表，3.2版，2019

7.3 书籍

[Bass03] Len Bass, Paul Clements, Rick Kazman “Software Architecture in Practice (2nd edition)”软件架构实践 （第二版）,Addison-Wesley 2003, ISBN 0-321-15495-9

[Bath14] Graham Bath, Judy McKay, “The Software Test Engineer’s Handbook (2nd edition)”软件测试工程师手册（第二版）, Rocky Nook, 2014, ISBN 978-1-933952-24-6

[Beizer90] Boris Beizer, "Software Testing Techniques Second Edition"《软件测试技术第二版》, International Thomson Computer Press, 1990, ISBN 1-8503-2880-3

[Beizer95] Boris Beizer, “Black-box Testing”黑盒测试, John Wiley & Sons, 1995, ISBN 0-471-12094-4

[Burns18]BrendanBurns,“DesigningDistributedSystems:Patterns and Paradigms for Scalable, Reliable Services”设计分布式系统:可扩展的模式和范例，可靠的服务, O’Reilly, 2018, ISBN 13: 978-1491983645

[Buwalda01]: Hans Buwalda, “Integrated Test Design and Automation”集成测试设计与自动化“, Addison-Wesley Longman, 2001, ISBN 0-201-73725-6

[Copeland03]: Lee Copeland, “A Practitioner’s Guide to Software Test Design”软件测试设计的实践者指南, Artech House, 2003,ISBN 1-58053-791-X

[Gamma94] Design Patterns: Elements of Reusable Object-Oriented Software 设计模式:可重用的面向对象软件的元素, Addison-Wesley, 1994,ISBN 0-201-63361-2

[Jorgensen07]: Paul C. Jorgensen, “Software Testing, a Craftsman’s Approach third edition”软件测试,《工匠的方法》第三版, CRC press, 2007, ISBN-13:978-0-8493-7475-3

[Kaner02]: Cem Kaner, James Bach, Bret Pettichord; “Lessons Learned in Software Testing”软件测试中的经验教训; Wiley, 2002, ISBN: 0-471-08112-4

[Koomen06]: Tim Koomen, Leo van der Aalst, Bart Broekman, Michael Vroon, "TMap Next for result- driven testing TMap Next 用于结果驱动的测试"; UTN Publishers, 2006, ISBN: 90-72194-79-9

[McCabe76] Thomas J. McCabe, "A Complexity Measure"复杂性度量”, IEEE Transactions on Software Engineering,Vol. SE-2, No. 4, December 1976. PP 308-320

[McCabe96]ArthurH.WatsonandThomasJ.McCabe."StructuredTesting:ATestingMethodology Usingthe Cyclomatic Complexity Metric"结构化测试：使用圈复杂度的测试方法 (PDF), 1996, NIST Special Publication500-235.

[NIST96]ArthurH.WatsonandThomasJ.McCabe,"StructuredTesting:ATestingMethodologyUsing the Cyclomatic Complexity Metric"结构化测试：使用圈复杂度的测试方法, NIST Special Publication 500-235, Prepared under NIST Contract 43NANB517266, September1996.

[Splaine01]: Steven Splaine, Stefan P. Jaskiel, “The Web-Testing Handbook” web测试手册, STQE Publishing, 2001,ISBN 0-970-43630-0

[Utting07] Mark Utting, Bruno Legeard, “Practical Model-Based Testing: A Tools Approach”实用的基于模型的测试：一种工具方法, Morgan-Kaufmann, 2007, ISBN: 978-0-12-372501-1

[Whittaker04]: James Whittaker and Herbert Thompson, “How to Break Software Security”如何破解软件安全, Pearson / Addison-Wesley, 2004, ISBN 0-321-19433-0

[Wiegers02] Karl Wiegers, "Peer Reviews in Software: A Practical Guide"软件中的同行评审:实用指南, Addison-Wesley, 2002, ISBN 0-201-73485-0

7.4 其他参考资料

以下参考地址指向因特网上可用的信息。在本高级教学大纲发布时已对这些参考资料进行了检查，但如果这些参考地址不再有效，ISTQB®也不承担任何责任。

[Web-1] http://www.nist.gov NIST National Institute of Standards and Technology,

[Web-2]http://www.codeproject.com/KB/architecture/SWArchitectureReview.aspx

[Web-3] http://www.W3C.org

[Web-4]https://www.owasp.org/index.php/Category:OWASP Top Ten Project

[Web-5] https://whatwg.org

[Web-6] https://whatwg.org/validator/

Chapter 4: [Web-1] [Web-4]

Chapter 5: [Web-2]

Chapter 6: [Web-3] [Web-5] [Web-6]

8. 附录A：质量特性概述

下表比较了 ISO 9126（在 2012 版技术测试分析师教学大纲中使用的质量特性）和新版[ISO25010]中描述的质量特性（在 2019 版教学大纲中使用的质量特性）。仅显示与技术测试分析师相关的特性。

ISO/IEC 25010	ISO/IEC 9126-1	备注
性能效率	**效率**	
时间特性	时间特性	
资源利用率	资源利用率	
容量		新的子特性
兼容性		新特性
共存性	共存性	从可移植性中移除
互操作性		从功能性移除（测试分析师）
可靠性	**可靠性**	
成熟性	成熟性	
可用性		新的子特性
容错性	容错性	
易恢复性	易恢复性	
信息安全性	**安全性**	以前不存在的子特性
保密性		新的子特性
完整性		新的子特性
抗抵赖性否认性		新的子特性
可核查性		新的子特性
真实性		新的子特性
可维护性	**可维护性**	
模块化		新的子特性
可重用性		新的子特性
易分析性	易分析性	
易修改性	稳定性	合并了易改变性和稳定性
	可变性	
易测试性	易测试性	
可移植性	**可移植性**	
适应性	适应性	

易安装性	易安装性	
	共存性	移到兼容性
易替换性	可替换性	
	合规性	在25010中删除