认证测试工程师
高级技术测试分析师
大纲

版本：CTAL-TTA-4.0-CN-2.0

发布日期：2022 年 12 月 14 日

国际软件测试认证委员会

中文版的翻译、编辑和出版统一由 ISTQB®授权的 CSTQB®负责

版权申明

修订历史

版本	日期	备注
2012（英文版）	2012/10/19	GA 发布 2012 版本
2019 v1.0（英文版）	2019/10/18	GA 发布 2019 版本
20200620（中文版）	2020/06/20	中文版本地化工作基本结束
2021 v4.0 Alpha 评审	2020/12/07	Alpha 评审初稿更新： ● 文本整体改进。 ● 删除 K3 TTA-2.6.1（2.6 基本路径测试）相关段落，并删除学习目标。 ● 删除 K2 TTA-3.2.4（3.2.4 调用图）相关段落，并删除学习目标。 ● 重写 TTA-3.2.2 (3.2.2 数据流分析)相关段落，并定为 K3。 ● 重写 TTA-4.4.1 和 TTA-4.4.2（4.4 可靠性测试）相关段落。 ● 重写与 TTA-4.5.1 和 TTA-4.5.2 相关段落(4.5 性能测试)。 ● 添加关于操作配置文件的第 4.9 节 ● 重写 TTA-2.8.1 相关段落（2.7 选择一种白盒测试技术部分）。 ● 重写 TTA-3.2.1，使其包含圈复杂度（对考试问题没有影响）。 ● 重写 TTA-2.4.1(MC/DC)使其与其他白盒学习目标一致（对考试问题没有影响）。
2021 v4.0 Beta 评审	2021/03/01	根据 Alpha 评审意见修改初稿
v4.0	2021/04/28	根据 Beta 评审意见修改初稿
v4.0	2021/06/30	GA 发布 v4.0 版本
CTAL-TTA-4.0-CN-1.0	2022/2/17	英文大纲 4.0 本地化完成
CTAL-TTA-4.0-CN-2.0	2022/12/14	中文版修改和完善

目录

致谢

本文档 2019 版由国际软件测试认证委员会 ISTQB®高级工作组核心团队编写：Graham Bath（副主席），Rex Black,Judy McKay, Kenji Onoshi, Mike Smith(主席),Erik van Veenendaal.

以下人员参与了本大纲 2019 版的评审、讨论和投票

Dani Almog	Andrew Archer	Rex Black
Armin Born	Sudeep Chatterjee	Tibor Csöndes
Wim Decoutere	Klaudia Dusser-Zieger	Melinda Eckrich-Brájer
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 Medzihradszky
Petr Neugebauer	Ingvar Nordström	Pálma Polyák
Meile Posthuma	Stuart Reid	Lloyd Roden
Adam Roman	Jan Sabak	Péter Sótér
Benjamin Timmermans	Stephanie van Dijck	Paul Weymouth

本文档由国际软件测试认证委员会 ISTQB®高级工作组核心团队编写： Armin Born, Adam Roman, Stuart Reid.

本更新文档 V4.0 版由国际软件测试认证委员会 ISTQB®高级工作组核心团队编写：Armin Born,Adam Roman,Christian Graf, Stuart Reid.

以下人员参与了本更新大纲 V4.0 版的评审、讨论和投票

Adél Vécsey-Juhász	Jane Nash	Pálma Polyák
gota Horváth	Lloyd Roden	Paul Weymouth
Benjamin Timmermans	Matthias Hamburg	Péter Földházi Jr.
Erwin Engelsma	Meile Posthuma	Rik Marselis
Gary Mogyorodi	Nishan Portoyan	Sebastian Małyska
Geng Chen	Joan Killeen	Tal Pe'er
Gergely Ágnecz	Ole Chr. Hansen	Wang Lijuan
		Zuo Zhenlei

核心团队感谢评审组和各国分会的建议和意见

本文档于 2021 年 6 月 30 日在 ISTQB®大会正式发布

ISTQB®高级-技术测试分析师大纲 V4.0 文档中文翻译和评审的参与者（按姓氏拼音排序）：

黄文萍、李春慧、李文鹏、陶显锋、郑丹丹、周震漪（组长）、左振雷

0. 大纲简介

0.1 大纲的目的

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如下：

1. 各国家认证委员会分会将教学大纲翻译成当地语言并认可培训机构。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，也可修改参考资料以适应在当地发布。
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）开发课件并选择合适的教学方法。
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®技术测试分析师高级教学大纲概述是一个单独的文档[ISTQB®_AL_TTA_OVIEW]，其中包括以下信息：

- 商业价值。
- 展示商业价值与学习目标之间可追溯性的矩阵。
- 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该章节的具体学习目标的认知级别（K2、K3 和 K4），三个级别的具体分类如下：

- K2：理解。
- K3：应用。
- K4：分析。

0.4 所需的经验

技术测试分析师的部分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用到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，但除了本教学大纲中总结的内容，其他内容不在考试范围之内。

考试的形式是多选题，总共 45 个问题。若要通过考试，至少获得总分值的 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 认可的课程

ISTQB®成员委员会分会可对根据本教学大纲开发课程课件的培训机构进行认可。培训机构宜从成员委员会分会或机构获得认可指南。

已认可的课程视为符合本教学大纲，并允许将 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开发生命周期的内容。本教学大纲不包含任一特定软件开发模型的特定学习目标，但它讨论了这些概念如何应用于敏捷软件开发、其他类型的迭代和增量软件开发模型，以及在顺序软件开发模型中的应用。

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。对于经认可的培训课程，本教学大纲要求至少 20 小时的教学时间，并分布在以下六个章节：

- 第 1 章：基于风险的测试中技术测试分析师的任务（30 分钟）。
- 第 2 章：白盒测试技术（300 分钟）。
- 第 3 章：静态和动态分析（180 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），项目风险（project 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通常需要考虑的通用风险因素。
TTA-1.2.2	（K2）	总结在基于风险方法的测试活动中技术测试分析师进行的活动。

1.1 简介

基于风险的测试策略的建立和管理由测试经理全面负责，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内工作，提供有关项目固有的产品技术风险的信息，例如与信息安全性、系统可靠性和性能有关的风险。技术测试分析师还应该参与识别和处理与测试环境（例如性能测试、可靠性测试和信息安全性测试的测试环境的获取和搭建）有关的项目风险。

1.2 基于风险的测试任务

技术测试分析师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能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，并分析其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、技术专家和服务台技术人员）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保障。

技术测试分析师可能识别的风险通常基于本大纲第 4 章中列出的[ISO 25010][GB/T 25000.10-2016]产品质量特性。

1.2.2 风险评估

风险识别是尽可能多地识别相关风险，而风险评估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。

产品风险的可能性通常理解为所测系统发生该失效的概率。技术测试分析师能帮助了解每个产

品技术风险的概率，而测试分析师能帮助理解当问题发生时的潜在商业影响。

项目风险变成问题后可能会影响项目的整体成功，因此通常需要考虑以下通用项目风险因素：

- 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。

产品风险变成问题后可能会导致更多的缺陷，因此通常需要考虑以下通用产品风险因素：

- 技术的复杂性。
- 代码复杂度。
- 源代码变更的数量（添加、删除、修改）。
- 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提出初始的风险可能性。当考虑了所有利益相关方的意见后，测试经理可以修改此初始值。风险影响则一般由测试分析师确定。

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：

- 针对高风险区域的风险设计测试用例，并协助评价残余风险。
- 通过执行所设计的测试用例和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。

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，以确定风险缓解措施和测试策略的要素。其他信息可以从 ISTQB®专业领域教学大纲中获得，例如，信息安全性测试教学大纲[CTSL_SEC_SYL]和性能测试教学大纲[CTSL_PT_SYL]。

2. 白盒测试技术 -300 分钟

关键词

异常（Anomaly），API（应用程序接口）测试（API testing），原子条件（atomic condition），控制流（control flow），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），安全完整性等级（safety integrity level），语句测试（statement testing），白盒测试技术（white-box test technique）

白盒测试技术学习目标

2.2 语句测试

TTA-2.2.1	（K3）	通过应用语句测试，为给定的测试对象设计测试用例，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，为给定的测试对象设计测试用例，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/判定测试技术，为给定的测试对象设计测试用例，以达到完整改进的条件/判定覆盖(MC/DC)。

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，为给定的测试对象设计测试用例，以达到定义的覆盖率水平。

2.6 基础路径测试（已从本教学大纲的 v4.0 版中删除）

TTA-2.6.1 （已从本教学大纲的 v4.0 版中删除）

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

本章描述白盒测试技术。这些技术适用于代码和其他具有控制流的结构，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，并聚焦于结构的特定方面。由这些技术生成的测试用例满足被设置为目标并被度量的覆盖标准。实现完全覆盖（即 100%）并不意味着整个测试集是完整的，而是正在使用的技术不能对正在考虑的结构给出任何有用的进一步的测试建议。

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。
- 条件测试。
- 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。语句测试侧重于执行代码中的可执行语句，而判定测试则侧重于执行各种判定的结果。

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包含复合条件的判定谓词并找到相似类型的缺陷。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。

有关这些技术的规范和示例的更多详细信息，请参阅 [ISO 29119][GB/T 25000.10-2016]。

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

实现完整的语句覆盖应被视为所有被测试代码的最低要求，尽管这在实践中并不总是可能的。

局限/难点

实现完整的语句覆盖应被视为所有被测试代码的最低要求，尽管由于可用时间和/或工作量的限制，这在实践中并不总是可能的。即使是高百分比的语句覆盖率也可能无法检测到代码逻辑中的某些缺陷。在许多情况下，由于存在执行不到的代码，所以不可能实现 100% 的语句覆盖。虽然执行不到的代码通常被认为不是好的编程实践，但它还是可能会发生，例如，一个分支语句必须有默认情况的

处理，但所有可能的条件都在前面的分支被处理了，而默认情况的处理就可能无法执行到。

2.3 判定测试

判定测试执行代码中的各种判定结果。为了做到这一点，测试用例跟随从判定点发生的控制流（例如：对于 IF 语句，一个控制流针对为“真”的结果，一个控制流针对为“假”的结果；对于 CASE 语句，有多个可能的结果；对于 LOOP 语句，一个控制流针对循环条件为真的结果，一个控制流针对为假的结果）。

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。请注意，一个测试用例可能会执行多个判定结果。

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视为一个整体，只评估结果的真和假，而不管其内部结构的复杂性。

分支测试通常与判定测试互换使用，因为可以通过相同的测试实现覆盖所有分支和覆盖所有判定结果。分支测试执行代码中的分支，其中分支通常被认为是控制流图的边。对于没有判定的程序，无论运行多少测试，按上面提到的判定覆盖率的定义得出覆盖率为 0/0，这是无定义的。而从入口到出口点的单个分支（假设一个入口和出口点）将实现 100%的分支覆盖率。为了解决这两种度量之间的差异，ISO 29119-4 要求在没有判定的代码上至少运行一个测试以实现 100%判定覆盖率，因此几乎所有程序 100%判定覆盖率和 100%分支覆盖率是等价的。许多提供覆盖率测量的测试工具，包括用于测试安全相关系统的工具，都采用了类似的方法。

适用性

当被测试的代码是重要的，甚至是关键的，这时应该考虑这种覆盖级别（请参阅第 2.8.2 节中安全相关系统的表格）。这种技术可用于代码和任何涉及判定点的模型，如业务流程模型。

局限/难点

判定测试未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结果的组合引起的缺陷。

2.4 改进的条件/判定测试 MC/DC

判定测试将整个判定作为一个整体来考虑，并评估为“真”的结果和为“假”的结果。相比之下，改进的条件/判定测试考虑了当判定包含多个条件时的结构（如果判定仅由一个原子条件组成，则它仅是判定测试）。

每个判定谓词由一个或多个原子条件组成，每个原子条件的计算结果都是一个布尔值。这些条件组合在一起，以确定判定的结果。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、相互独立的原子条件，改进的条件/判定测试通常可以通过执行 N+1 个不同的测试用来来实现。改进的条件/判定测试需要结对的测试来显示单个原子条件结果的变化可以独立影响判定结果。请注意，单个测试用例可能会执行多个条件组合，因此并不总是需要运行 N+1 个单独的测试用例来实现 MC/DC。

适用性

该技术适用于航空航天工业和汽车工业，以及其他工业的安全关键系统。它用于测试可能由于软件失效而导致灾难的软件。改进的条件/判定测试可以是判定测试和复合条件测试（因为要测试的组合数量太多）之间合理的平衡点。当判定中有多个原子条件时，它比判定测试更严格，但所需要的测试条件（测试用例）比复合条件测试少得多。

局限/难点

在具有多个条件的判定中同一个变量出现多次时，达到改进的条件/判定测试可能会很复杂；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使用改进的条件/判定测试。另一种方法是逐个分析发生耦合的每个判定。

一些编译器和/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/判定测试的能力，因为某些所需的测试可能无法实现。通常，为了测试可以将编译器配置为关闭短路优化，但对于安全关键型应用程序可能不允许这样做，因为所测试的代码和交付的代码必须相同。

2.5 复合条件测试

在极少数情况下，可能需要测试一个包含所有原子条件的可能组合的判定。这种测试级别称为复合条件测试。假设有 N 个唯一的、相互独立的原子条件，可以通过执行 2^N 次来实现判定的完整复合条件覆盖。请注意，单个测试用例可能会执行多个条件组合，因此并不总是需要运行 2^N 个单独的测试用例来实现 100%的复合条件覆盖。

覆盖率以执行的测试对象中的所有判定的原子条件组合的数量进行度量，通常以百分比表示。

适用性

这种技术用于测试高风险软件和要求能长时间可靠地运行而不崩溃的嵌入式软件。

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，复合条件测试所需的大量测试用例使得改进的条件/判定测试在判定中有多个原子条件的情况下更可行。

如果编译器使用短路，可执行的条件组合数量通常会减少，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

本章已从本教学大纲的 v4.0 版中删除。

2.7 API（应用程序接口）测试

应用程序接口(API)定义了一个接口，它使程序能够调用另一个软件系统，后者为其提供服务，例如访问远程资源。典型的服务包括 Web 服务、企业服务总线、数据库、主机和网络用户界面（Web UI）。

应用程序接口（API）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（API）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（API）时，逆向测试通常是至关重要的。使用应用程序接口（API）访问自身代码外部服务的程序员，可能试图以非预料的方式使用应用程序接口（API）。这意味着强有力（健壮）的错误处理对于避免不正确的操作至关重要。可能需要对许多接口进行组合测试，因为应用程序接口（API）通常与其他应用程序接口（API）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（API）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（API）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（API）发布者进行严格的可靠性测试以及提供基础设施支持。

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（API）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（API）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（API）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（API）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（API）测试是一种测试类型的描述；它不表示任何特定的覆盖水平。应用程序接口（API）测试至少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（API）。更彻底的应用程序接口（API）测试可以确保可调用实体至少执行一次，或者所有可能的功能至少被调用一次。

表征状态转移（Representational State Transfer）是一种架构风格。RESTful Web 服务允许请求系统使用一组统一的无状态操作访问 Web 资源。RESTful Web API 存在多个覆盖准则，这是软件集成的事实标准 [Web-7]。它们可以分为两组：输入覆盖准则和输出覆盖准则。其中，输入准则可能需要执行所有可能的 API 操作、使用所有可能的 API 参数以及覆盖 API 操作序列。而输出准则可能需要生成所有正确和错误的状态代码，以及生成包含展示资源的所有属性（或所有属性类型）的响应。

缺陷的类型

通过测试应用程序接口（API）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、事务重复和异常处理中的问题也很常见。

2.8 选择一种白盒测试技术

通常白盒测试技术是由所需的覆盖水平来确定的，通过应用该测试技术来实现覆盖水平。例如，要求是实现 100%语句覆盖率，则导致会使用语句测试。然而，通常首先应用黑盒测试技术，然后度量覆盖率，并且仅在未达到所需的白盒覆盖水平时才使用白盒测试技术。在某些情况下，可能会非正式地使用白盒测试以便提示可能需要增加的覆盖范围（例如，为白盒覆盖水平特别低的区域创建额外的测试）。对于这种非正式的覆盖测量，语句测试通常就足够了。

在规定所需的白盒覆盖水平时，最好仅将其规定为 100%。这样做的原因是，如果要求的覆盖水平低于 100%，那通常意味着代码中没有经过测试的部分是最难测试的部分，而这些部分通常也是最复杂并且容易出错的。所以，例如仅要求并实现 80%的覆盖率，这可能意味着包含大部分可检测出缺陷的代码没有得到测试。因此，当在标准中规定白盒覆盖准则时，通常都是 100%。对覆盖水平的严

格定义有时会使这种覆盖水平变得不能实现。然而，ISO 29119-4 中给出了允许从计算中扣除的不可行的覆盖项目，从而使 100%覆盖成为可实现的目标。

在为测试对象规定所需的白盒覆盖率时，也只需要为其规定单个覆盖准则（例如，没有必要同时要求 100%语句覆盖率和 100% MC/DC）。退出准则为 100% 时，可以将某些退出准则关联到包含层次结构中，其中的覆盖准则表明包含了其他覆盖准则。如果对于所有组件及其规格说明，满足第一个准则的每组测试用例也满足第二个准则，则可以说第一个覆盖准则包含了第二个准则。例如，分支覆盖率包含语句覆盖率，因为如果实现了分支覆盖率（达到 100%），则保证了 100%的语句覆盖率。对于本课程大纲中涵盖的白盒测试技术，我们可以说分支和判定覆盖包含语句覆盖，MC/DC 包含判定和分支覆盖，复合条件覆盖包含 MC/DC（如果我们认为分支和判定覆盖在达到 100%时是相同的，那么我们可以说它们相互包含）。

请注意，在确定系统要达到的白盒覆盖水平时，为系统的不同部分定义不同的覆盖水平是很正常的。这是因为系统的不同部分的风险不一样。例如，在航空电子系统中，飞行中的娱乐相关的子系统的风险级别低于与飞行控制相关的子系统的风险级别。接口测试对于所有类型的系统都是通用的，并且通常是安全相关系统的所有完整性等级所必需的（有关完整性等级的更多信息，请参见第 2.8.2 节）。API 测试所需的覆盖水平通常会根据相关风险而增加（例如，与公共接口相关的风险级别越高，所需的 API 测试就越严格）。

选择使用何种白盒测试技术通常基于测试对象的性质及识别出的风险。如果测试对象被认为是与安全相关的（即失效可能对人或环境造成危害），则将适用监管标准并将定义其所需的白盒覆盖水平（参见第 2.8.2 节）。如果测试对象与安全无关，那么要达到的白盒覆盖级别的选择就更加主观，但仍应主要基于识别的风险，如第 2.8.1 节所述。

2.8.1 非安全相关系统

在为非安全相关系统选择白盒测试技术时，通常会考虑以下因素（没有特定的优先顺序）：

- 合同——如果合同要求达到特定的覆盖水平，那么未达到该覆盖水平可能会导致违约。
- 客户——如果客户要求达到特定的覆盖水平，例如作为测试计划的一部分，那么未达到该覆盖水平可能会因没有满足客户要求而与客户产生矛盾。
- 监管标准——对于某些行业部门（例如，金融），监管标准定义了适用于关键任务系统所需的白盒覆盖准则。安全相关系统的监管标准的覆盖，请参见第 2.8.2 节。
- 测试策略——如果组织的测试策略规定了白盒代码覆盖率的要求，那么与组织策略不一致可能会导致更高管理层的指责。
- 编码风格——如果编写的代码在判定中没有复合条件，那么要求诸如 MC/DC 和复合条件覆盖的白盒覆盖水平将是一种浪费。

- 历史缺陷信息——如果实现特定覆盖水平的有效性的历史数据表明了此覆盖水平适合用于此测试对象，则忽略这些现有数据将会带来风险。请注意，此类数据可从项目、组织或行业内获得。
- 技能和经验——如果可用的执行测试的测试人员对某一特定白盒技术没有足够的经验和技能，则可能会误解该技术，如果选择了该技术，可能会带来不必要的风险。
- 工具——在实践中，只有使用覆盖率工具才能度量白盒覆盖率。如果无法获得支持给定覆盖度量数据的工具，则选择要达到这样的度量将会导致高风险。

与安全相关系统相比，在为非安全相关系统选择白盒测试时，技术测试分析师可以更自由地为非安全相关系统推荐适合的白盒覆盖率。这种选择通常是在识别的风险与通过白盒测试处理这些风险所需的成本、资源和时间之间求得平衡。在某些情况下，由其他软件测试方法或其他方式（例如，不同的开发方法）实现的其他处理可能更合适。

2.8.2 安全相关系统

如果被测试的软件是安全相关系统的一部分，那么通常必须使用定义了所需达到的覆盖水平的监管标准。此类标准通常需要对系统进行风险分析，并根据风险分析为系统的不同部分指定完整性等级。每个完整性等级定义了所需的覆盖水平。

IEC 61508（可编程、电子、安全相关系统的功能安全[IEC 61508]）是用于此类目的的综合性国际标准。理论上，它可以用于任何与安全相关的系统，但是一些行业已经编写了特定的标准（例如适用于汽车系统的 ISO 26262 [ISO 26262]），一些行业已经创建了自己的标准（例如用于机载软件的 DO-178C[DO-178C]）。ISTQB® 汽车软件测试人员教学大纲[CTSL_AuT_SYL]中提供了有关 ISO 26262 的更多信息。

IEC 61508 定义了四个安全完整性等级(SIL)，每个等级都定义了提供的安全功能相对的风险水平，与识别出的危险的频率和严重程度相关。当测试对象执行与安全相关的功能时，失效风险越高意味着测试对象应该具有更高的可靠性。下表显示了与 SIL 相关的可靠性等级。请注意，连续运行情况下 SIL4 的可靠性级别非常高，因为它对应的是大于 10,000 年的平均故障间隔时间(MTBF)。

IEC 61508 SIL	连续运行 （每小时发生危险故障的概率）	请求式的 （请求时发生故障的概率）
1	10^{-6} to $<10^{-5}$	10^{-2} to $<10^{-1}$
2	10^{-7} to $<10^{-6}$	10^{-3} to $<10^{-2}$
3	10^{-8} to $<10^{-7}$	10^{-4} to $<10^{-3}$
4	10^{-9} to $<10^{-8}$	10^{-5} to $<10^{-4}$

下表显示了与每个 SIL 相关的建议的白盒覆盖水平。如果条目显示为“高度推荐”，实践中通常认为达到该覆盖水平是强制性的。相反，当条目仅显示为“推荐”时，许多从业者认为这是可选的，并通过提供合适的理由来避免实现它。因此，SIL3 测试对象的测试通常达到 100%的分支覆盖率（它自动实现 100%的语句覆盖率，如分类包含所示）。

IEC 61508 SIL	100%语句覆盖率	100%分支覆盖率	100% MC/DC
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

请注意，IEC 61508 中规定的上述 SIL 和覆盖水平要求与 ISO 26262 中的不同，与 DO-178C 中的也不同。

3. 静态和动态分析-180 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），静态分析（static analysis），野指针（wild pointer）

静态和动态分析的学习目标

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并测量圈复杂度。
TTA-3.2.2	（K3）	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）	应用静态分析来提高代码可维护性。

注意：TTA-3.2.4 已从本教学大纲的 v4.0 版中删除。

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

静态分析（参见第 3.2 节）是一种在不执行软件的情况下进行的测试形式。软件的质量是由工具或人根据其形式、结构、内容或文档来评估的。该类软件静态视角可在无需创建执行测试用例所需的数据和前置条件的情况下进行详细分析。

除了静态分析，静态测试技术还包括不同形式的评审。第 5 章中介绍了与技术测试分析师相关的内容。

动态分析（参见第 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缺陷（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中的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。

3.2.1 控制流分析

控制流分析是一种静态测试技术，通过使用控制流图（通常使用工具产生控制流图）分析程序中运行步骤。使用这种技术可以在系统中发现许多异常，包括设计不当的循环（例如，具有多个入口点或无法终止）、某些语言中函数调用的目标不明确、操作顺序不正确、不可达的代码、未调用的函数等。

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。

圈复杂度通常用作组件复杂度的指标。Thomas McCabe 的理论[McCabe76]是系统越复杂，维护起来就越困难，包含的缺陷也就越多。许多研究已经注意到复杂度与包含的缺陷数量之间的这种相关性。任何测量出来复杂度较高的模块都应进行评审以进行可能的重构，例如划分为多个模块。

3.2.2 数据流分析

数据流分析涵盖了各种收集系统中变量使用情况的技术。要分析控制流路径的每个变量的生命周期（即，变量已声明、已定义、已使用和已销毁），因为如果不按顺序进行这些操作，就可以标识为可能的异常[Beizer90]。

一种常见的技术将变量的使用归类为三个原子操作之一：

- 当变量已定义、已声明、或已初始化时（如 x:=3）。
- 当变量已使用或读取时（如，if x > temp）。

- 当变量已终止、已销毁，或变量超出范围时（如，text_file_1.close，退出循环时的循环控制变量(i)）。

提示可能的异常操作顺序包括：

- 定义后跟着另一个定义的变量，或终止没用使用的变量。
- 定义变量，但没有后续的终止（例如，导致动态分配的变量可能出现内存泄漏）。
- 在定义变量前使用或终止。
- 在终止变量后又使用或终止。

依赖于编程语言，编译器可能识别其中一些异常，但可能需要单独的静态分析工具来识别数据流异常。例如，在大多数编程语言中都允许在使用前重新定义，并且可能是特意如此编程，但是数据流分析工具会将其标记为应该检查的可能异常。

使用控制流路径来确定变量的操作顺序可能导致报告在实践中不可能发生的潜在异常。例如，静态分析工具无法始终确定控制流路径是否可行，因为某些路径仅根据运行时分配给变量的值来确定。还有一类数据流分析问题是工具难以识别的，如当分析的数据是动态分配变量的数据结构的一部分时，例如记录和数组。当变量在程序中的并发控制线程之间共享时，静态分析工具也难以识别潜在的数据流异常，因为对数据的操作顺序变得难以预测。

与静态测试的数据流分析相反，数据流测试是动态测试，其生成测试用例以在程序代码中执行“定义-使用对”。数据流测试使用一些与数据流分析相同的概念，因为这些“定义-使用对”是程序中变量的定义和后续使用之间的控制流路径。

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功能有可能会引入新的缺陷。

使用静态分析来验证与编码规范和编码指南的符合性，在识别出不合规的代码时，可以对其进行更新以提高其可维护性。这些规范和指南描述了所要求的编码和设计实践，如命名约定、注释、缩进和模块化。请注意，静态分析工具通常提出警告，而不是检测缺陷。即使代码在句法上是正确的，这些警告（例如，复杂程度）也可能出现。

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。这部分重复代码可能是重构为组件的候选代码（尽管组件调用所需

的运行时间开销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重要指标。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（组件在执行期间相互依赖的程度）和较高的内聚度（组件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视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，或者是否存在将导致以下问题的不平衡情况：

- 测试任务更难。
- 维护工作量增加。

除了评价可维护性外，静态分析工具也适用于实施网络所用的代码来检查可能的安全薄弱点，例如代码注入、cookie 安全性、跨站脚本攻击、资源篡改、SQL 代码注入。4.3 章节以及安全测试教学大纲[CTSL_SEC_SYL]提供了更多的信息。

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发布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问题（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；
- 通过使用代码分析器提供运行时系统的行为信息，并对其进行有根据的更改，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。

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，具备所需技术和分析技能的人员可以分析这些日志。

3.3.2 检测内存泄漏

内存泄漏是当内存空间(RAM)分配给程序，但随后不再需要时没有释放分配的内存空间。而此内存空间不可被再使用。当这种情况频繁发生或在内存容量较低时，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释放，以避免内存泄漏。许多现代编程环境都包括自动或半自动“垃圾回收”，不需要程序员直接干预，分配的内存在使用后自动释放。在分配的内存应该由自动垃圾回收释放的情况下，隔离内存泄漏可能非常困难。

内存泄漏通常会在一段时间后导致问题——当大量内存已经泄漏并变得不可用时。新安装软件，或系统重启时，内存重新分配，内存泄漏不会显现；测试是一个例子，频繁的内存分配会阻碍内存泄漏的检测。由于这些原因，程序在生产环境中可能才首次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，最终可能导致系统失效。虽然可以通过重新开始（重新启动）系统来解决此类故障，但这并不总是可行，对于某些系统来说，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。

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，野指针访问的可能是程序当前未使用的、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，野指针可能导致了对程序运行至关重要的部分（例如操作系统）的内存的不当使用。
- 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更改（例如，软件更改后的新构建）都可能触发上述四个后果中的任何一项。特别重要的是，最初的程序尽管使用了野指针，但还是能按预期运行，但在软件更改后意外崩溃（甚至可能在实际生产环境中）。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。某些操作系统具有检查运行时内存访问违规情况的内置功能。例如，当应用程序尝试访问其允许的内存空间之外的内存位置时，操作系统可能会触发一个异常预警。

3.3.4 性能效率分析

动态分析不止对检测失效和定位相关的缺陷有用。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组件被调用的次数信息。经常被调用的组件有可能成为进行性能效率提升的候选者。帕累托规则在这里通常成立：程序不成比例地将其大部分(80%)运行时间花费在少数(20%)的组件上[Andrist20]。

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用具（test harnesses）测试单个子系统时进行。更多详细信息见性能测试教学大纲[CTSL_PT_SYL]。

4. 技术测试中的质量特性——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。

技术测试中的质量特性学习目标

4.2 总体策划的问题

TTA-4.2.1	（K4）	针对特定场景分析非功能需求，并编写对应的测试计划。
TTA-4.2.2	（K3）	根据特定的产品风险，定义最适合的非功能测试类型。
TTA-4.2.3	（K2）	在应用软件开发生命周期的各个阶段，识别并解释需要应用的典型非功能测试。
TTA-4.2.4	（K3）	针对给定的场景，定义预期通过运用不同类型的非功能测试所能找到的缺陷类型。

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 性能测试

TTA-4.5.1	（K2）	解释在测试方法中包含性能测试的理由。
TTA-4.5.2	（K2）	解释在计划和指定性能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共存性测试的理由。

4.1 介绍

通常情况下，技术测试分析师侧重于测试产品工作的效果“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，在实时系统和嵌入式系统的组件测试过程中，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，都应该根据风险优先级和现有资源来执行测试。

动态测试和静态测试，包括评审（见第 2 第 3 和第 5 章）都可以用于本章所描述的对非功能质量特性的测试。

ISO25010 [GB/T 25000.10-2016]中提供的对产品质量特性的描述是关于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：附录 A 中提供了表格，该表格将已经废止的 ISO 9126-1 标准（在 2012 版大纲中使用）和较新的 ISO 25010[GB/T 25000.10-2016]标准中描述的特性进行了对比。

对所有在本节中所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进行测试计划、测试准备和测试执行的时间。

这些测试中的某些测试，例如性能测试，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。

当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测量总结信息，而测试分析师或技术测试分析师则（根据上面的表格）负责收集每个测量的信息。

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测量，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，就可能给项目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。
- 测试环境需求。
- 所需工具获取和人员培训。
- 组织层面的考虑。
- 数据信息安全方面的考虑。

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，一些需求很有可能会被忽略。关于捕获需求的更多细节，参见高级测试经理大纲[ISTQB®_ATLTM_SYL]。

在敏捷软件开发中，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 测试环境需求

很多技术测试类型（例如安全性测试、可靠性测试、性能效率测试）需要一个与产品相似的测试环境来提供可靠的度量。所测系统的规模和复杂度不同，测试环境可能对测试策划和资金产生很大的影响。因为测试环境的花费可能会很高，所以考虑以下因素：

- 使用生产环境。
- 使用系统的缩减版本，需要注意获得的测试结果能够充分代表生产系统。
- 使用云资源作为直接获取资源的替代方案。
- 使用虚拟化环境。

需要仔细计划测试执行时间，某些测试只能在特定时间执行（例如在低使用时间）。

4.2.3 所需工具的获得和人员培训

工具是测试环境的一部分。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。

4.2.4 组织层面的考虑

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成部分或人员（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 数据信息安全性和数据保护

在测试计划阶段就应考虑到用于确保系统信息安全的手段，以确保所有的测试活动是可执行的。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项重要的任务，必须将其作为测试实现的一部分来计划。

4.3 信息安全性测试

4.3.1 考虑信息安全性测试的原因

信息安全性测试以破坏系统的信息安全性制度为目标进行攻击，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：

- 未经授权的应用程序或数据拷贝。
- 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，一个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（“man in the middle 中间人”攻击方式）。
- 破解用来保护敏感数据的加密密码。
- 逻辑炸弹（有时称为“复活节彩蛋”）可能被恶意的插入代码中，而且只在特定条件下

（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，因此在组件、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（见下文中的 4.4 和 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[IEC 62443-3-2]。

安全性测试大纲[CTSL_SEC_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][GB/T 25000.10-2016]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。
- 完整性。
- 抗抵赖性。
- 可核查性。
- 真实性。

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。
- 用普遍可用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，可以为渗透测试制定攻击计划。

3.2 节（静态分析）和信息安全性测试大纲[CTSL_SEC_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 介绍

ISO 25010[GB/T 25000.10-2016]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、可用性、容错性和易恢复性。可靠性测试是指系统或软件在指定条件下、指定时间内执行指定功能的能力。

4.4.2 成熟性测试

成熟性测试是系统或软件在正常操作时满足可靠性要求的程度，这些正常操作通常利用运行配置（见 4.9 节）来定义。成熟性测度通常是系统发布的标准之一。

过去，成熟性已在高可靠系统上进行规定和测量，例如那些与安全关键功能（例如飞机飞行控制系统）相关的系统，成熟性目标被定义为监管标准的一部分。这种高可靠性系统的成熟性需求可能是平均失效间隔时间（MTBF）高达 109 小时（尽管实际上不可能测量）。

测试高可靠性系统成熟性的常用方法是可靠性增长建模，并且通常在系统测试的最后执行，即其他质量特性的测试已经完成，所发现失效的相关缺陷都已经被修复。这是一种通常在与操作环境尽可能接近的测试环境中执行的统计方法。为了测量指定的 MTBF，测试输入是基于运行配置生成的，系统运行并记录故障（随后进行修复）。不断降低的失效频率能让可靠性增长模型预测 MTBF。

当成熟性用于低可靠性目标系统（例如非安全相关的），那么预期操作使用期间观察到的缺陷数量（例如每周不超过 2 次的影响度高的缺陷）可以作为系统服务水平协议（SLA）的一部分被使用和记录。

4.4.3 可用性测试

可用性通常是指系统（或软件）在正常运行条件下对用户和其他系统可用的时间。系统可能具有较低的成熟性，但仍然具有高可用性。例如，一个电话网络可能无法连接多个呼叫（因此成熟性较低），但只要系统快速恢复并允许尝试下次连接，大多数用户就会满意。然而，导致电话网络中断数小时的单个故障代表了不可接受的可用性水平。可用性通常被指定为 SLA（服务水平协议）的一部分，并用于业务系统（如网站和软件即服务（SaaS）应用）的测量。系统的可用性可能被描述为 99.999%（“5 个 9”），在这种情况下，它每年不可用的时间不超过 5 分钟；系统可用性也可以通过“不可用”来定义（例如系统不应存在每月超过 60 分钟的宕机）。

运行前的可用性测量（例如作为版本发布决定的一部分）通常使用用于成熟性测度的测试；测试基于长时间预期操作的运行配置，并在与运行环境尽可能接近的测试环境中执行。可用性通常使用 MTTF/（MTTF+MTTR）来测量，其中 MTTF 是平均失效时间，MTTR 是平均修复时间（MTTR），MTTR 也

常用于维护性测试。如果系统具有高可靠性，并具备易恢复性（见 4.4.5 节），那么当系统从故障中恢复需要一段时间时，可用平均恢复时间替换公式中的 MTTR。

4.4.4 容错性测试

具有极高可靠性要求的系统（或软件）通常包含容错设计，理想情况下允许系统在发生故障时继续运行，而不会出现明显的停机时间。系统容错的主要测度是系统容错能力。因此，容错测试包括模拟故障以确定当发生故障时系统是否可以继续运行。识别待测试的潜在故障条件是容错测试的重要组成部分。

容错设计通常会涉及一个或多个重复的子系统，即出现故障时提供一定程度的冗余。在软件方面，此类重复系统需要独立开发，以避免发生相同模式的故障；这种方法被称为 N 版本编程。飞机飞行控制系统可能包括三或四层冗余级别，最关键功能需要在几个变体中实现。如涉及硬件可靠性，嵌入式系统可能在多个不同的处理器上运行，而关键网站可能使用执行相同功能的镜像（故障转移）服务器运行，确保在主服务器故障时，镜像服务器可以接管。无论实现了哪种容错方法，测试通常都需要检测故障和故障后的响应情况。

错误注入测试是对所在环境出现缺陷（比如，一个错误的电源、一条编写糟糕的输入消息、一个不可用的过程或服务，一个找不到的文件、或者内存不可用）的系统的健壮性和系统本身的缺陷（比如，宇宙辐射引起的反转位、不好的设计或糟糕的编码）进行测试。错误注入测试是负面测试的一种形式——通过故意将缺陷注入到系统，以确保系统按照预期方式做出反应（即对于安全相关系统来说是安全的）。有时测试执行的缺陷场景是不会发生的（例如，一个软件的任务不应该“死机”或陷入无限循环），并且传统的系统测试无法模拟，但采用错误注入测试，我们创建缺陷场景并测量随后的系统行为，确保发现和处理故障。

4.4.5 易恢复性测试

易恢复性是对系统（或软件）从失效中恢复的能力的测量，根据恢复所需的时间（可能是运行状态降级）或丢失的数据量来测量。易恢复性的测试方法包括故障恢复测试、备份和恢复测试；这两种测试方法通常包括基于试运行的测试过程，并在运行环境中偶尔地、理想地、不事先宣布地进行实际测试。

备份和恢复测试的重点是测试规程，以将故障对系统数据的影响降到最低。测试评估数据备份和恢复的规程。虽然数据备份测试相对容易，但从备份数据恢复系统的测试可能更复杂，通常需要仔细规划确保对运行系统的破坏最小。测量项包括执行不同类型的备份所花费的时间（例如完全备份和增量备份）、恢复数据所花费的时间（目标恢复时间）以及可接受的数据丢失程度（目标恢复点）。

当系统架构同时包含主系统和故障转移系统时，将执行故障转移测试。如果系统必须能够从灾难性故障（例如洪水、恐怖袭击或严重勒索软件攻击）中恢复，那么故障转移测试通常被称为灾难恢复测试，该故障转移系统（或多个系统）通常位于另一个地理位置。由于风险和中断（通常是高级管理人员的休息时间，这些时间很可能被用于恢复管理）的影响，在操作系统上执行完整的灾难恢复测试需要非常仔细的计划。如果一个完整的灾难恢复测试失败，那么将立即恢复到主系统（因为它实际上并没有被破坏!）。故障转移测试包括测试转移到故障转移系统的过程，一旦故障转移系统接管后，它将提供所需的服务级别。

4.4.6 可靠性测试计划

总体上来说，策划可靠性测试时，以下因素非常重要：

- 时间——可靠性测试通常需要完整的系统进行测试，并且其他测试类型已经完成。可靠性测试需要很长时间来执行。
- 成本——因为系统必须进行长时间的无故障测试，才能预测所需的高 MTBF，因此高可靠性系统测试是非常昂贵的。
- 持续时间——使用可靠性增长模型的成熟性测试的基础是检测到的故障，对于高可靠性级别需要很长时间才能得到统计上显著的结果。
- 测试环境——测试环境需要尽可能与运行环境相似，或者可以使用运行环境。但是使用运行环境可能会对用户造成干扰并且存在很高的风险，例如，灾难恢复测试会对操作系统产生不利影响。
- 范围——可能对不同的子系统和组件进行不同类型和可靠性级别的测试。
- 出口准则——可靠性要求应按安全相关应用的法规或标准来规定。
- 故障——可靠性测量非常依赖于故障计数，因此必须事先明确故障的判定准则。
- 开发人员——对于使用可靠性增长模型的成熟性测试，需要与开发人员达成缺陷尽快修复的协议。
- 与发布前的可靠性测量相比，运行可靠性测量只需要测量故障，所以相对简单；可能需要与操作人员协调。
- 早期测试——实现高可靠性（相对于可靠性测量）需要尽早开始测试，并对早期基线文档进行严格评审，对代码进行静态分析。

4.4.7 可靠性测试的规格说明

对于成熟性和可用性测试而言，测试很大程度上是对正常运行条件下的系统进行测试。对于此

类测试，定义系统预期如何使用的运行配置是有必要的。运行配置的详细信息见 4.9 节。

对于容错性和易恢复性测试，通常需要复制环境和系统本身的失效来生成测试，并确定系统如何响应。故障注入测试就经常应用于此。各种技术和检查表可用来识别可能的缺陷和相应的故障（例如，故障树分析，失效模式和影响分析）。

4.5 性能测试

4.5.1 介绍

ISO 25010[GB/T 25000.10-2016]产品质量特性定义了以下性能效率子特性：时间特性、资源利用性和容量。性能测试（与性能效率质量特性相关联）是关于系统或软件在所使用资源条件下的性能测量。典型的资源包括运行时间、CPU 时间、内存和带宽。

4.5.2 时间特性测试

时间特性测试是在指定运行条件下测量系统（或软件）的以下方面：

- 从接收到请求到第一个响应所消耗的时间（即开始响应的时间，而不是完成请求活动的时间），也称为响应时间；
- 从活动启动到该活动完成的周转时间，也称为处理时间；
- 每单位时间内完成的活动数量（例如，每秒钟数据库操作的数量），也称为吞吐率。

对于许多系统，不同系统功能的最大响应时间会作为需求规定下来。在这种情况下，响应时间是消耗时间加上周转时间。当一个系统必须执行多个步骤（例如流水线）来完成一个活动时，测量每个步骤所用的时间并分析结果以确定一个或多个步骤是否导致瓶颈可能很有用。

4.5.3 资源利用性测试

资源利用性测试是对在指定运行条件下的系统（或软件）开展以下方面的测量：

- CPU 利用率，通常为一段时间内可用 CPU 的百分比；
- 内存利用率，通常为可用内存的百分比；
- I/O 设备利用率，通常为一段时间内可用 I/O 设备的百分比；
- 带宽利用率，通常为可用带宽的百分比。

4.5.4 容量测试

容量测试是测量在指定运行条件下的系统（或软件）以下方面能承受的最大限度：

- 单位时间内处理事务数（例如，每分钟最多翻译 687 字）；
- 同时访问系统的用户数（例如，最多承受 1223 个用户）；
- 单位时间内可以访问系统的新增用户数（例如，每秒最多增加 400 个用户）。

4.5.5 性能测试的共同点

由于测量的时间值可能会随系统执行的其他后台任务而波动，因此当测试时间特性、资源利用性或容量时，通常会收集多个测量值，并使用平均值作为报告的测量结果。在某些情况下，测量将以更细致的方式处理（例如使用方差或其他统计测量），或者合适的情况下开展调查或丢弃异常值。

动态分析（见 3.3.4 节）可用于识别导致瓶颈的组件，测量用于资源利用性测试的资源，以及测量容量测试的最大限制。

4.5.6 性能测试类别

性能测试不同于大多数其他形式的测试，它可以有两个不同的目标。第一个是确定被测试的软件是否满足指定的验收准则。例如，确定系统是否在指定的最大 4 秒内显示请求的网页。第二个目标是向系统开发人员提供信息，以帮助他们提高系统的效率。例如，当意外的大量用户同时访问系统时，发现瓶颈并识别系统架构的哪些部分受到不利影响。

4.5.2、4.5.3 和 4.5.4 节中描述的性能测试都可以用来确定被测软件是否满足规定的验收准则。它们还用于测量基线值，当系统后续发生变化时，这些基线值将用于进行比较。以下性能测试类型更常用于向开发人员提供关于系统在不同操作条件下如何响应的信息。

4.5.6.1 负载测试

负载测试关注系统处理不同负载的能力。这些负载通常是根据同时访问系统的用户数量或运行的并发进程数量来定义的，可被定义为运行配置（参见 4.9 节了解运行配置更多信息）。这些负载的处理通常根据系统的时间特性和资源利用性来测量（例如，确定用户数量翻倍对响应时间的影响）。在执行负载测试时，通常的做法是先从低负载开始，然后逐渐增加负载，同时测量系统的时间特性和资源利用性。负载测试中可能对开发人员有用的典型信息包括：当系统处理特定负载时的响应时间或系统资源利用的预期外变化。

4.5.6.2 压力测试

压力测试有两种类型：一种与负载测试类似，第二种是健壮性测试的一种形式。

第一种情况中，执行负载测试时，通常在最初将负载设置为预期的最大值，然后增加负载直到系统失效（例如，响应时间长得不合理或系统崩溃）。有时不会强迫系统失效，而是使用高负载对系统施加压力，然后将负载降低到正常水平，并对系统进行检查，以确保其性能水平恢复到其预期能力的水平。

第二种情况中，执行性能测试时，减少对预期资源访问（例如，减少可用内存或带宽）来故意破坏系统。压力测试的结果可以让开发人员了解系统的哪些方面是最关键的（即薄弱环节），因此可能需要进行升级。

4.5.6.3 可扩展性测试

一个可扩展的系统可以适应不同的负载。例如，一个可扩展的网站可以在需求增加时使用更多的后端服务器，在需求减少时使用更少的后端服务器。可扩展性测试与负载测试类似，但测试的是系统在面对变化的负载（例如，超过当前硬件能够处理的用户数量）时向上和向下扩展的能力。

性能测试大纲[CTSL_PT_SYL]包括更多的性能测试类型。

4.5.7 性能测试策划

总体上来说，策划性能测试时，以下因素非常重要：

- 时间——性能测试通常要求在代表性的测试环境中实现和运行整个系统，这意味着它通常作为系统测试的一部分执行。
- 评审——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代码可用（即在动态测试之前）时开展。
- 早期测试——某些性能测试（例如，确定关键组件的 CPU 利用率）可被安排为组件测试的一部分。通过性能测试确定为瓶颈的组件可以更新和作为组件测试的一部分重新单独测试。
- 架构变更——性能测试的不利结果有时会导致对系统架构的变更。如果性能测试结果可以建议对系统进行这类重大更改，则性能测试应该尽可能早地开始，以最大限度地利用可用的时间来解决这些问题。
- 成本——工具和测试环境可能非常昂贵，这意味着可能需要租用临时的云测试环境，并且可能需要使用“补充”的工具许可证。在这种情况下，测试策划通常需要优化运行测试所

花费的时间，以使成本最小化。

- 测试环境——测试环境需要尽可能代表运行环境，否则会增加测试结果从测试环境扩展到预期的运行环境的挑战。
- 出口准则——性能效率需求有时很难从客户那里得到，因此通常是从以前的基线或类似的系统来获得需求。对于嵌入式安全相关系统、，一些需求（例如使用的 CPU 和内存的最大数量）可能由监管标准指定。
- 工具——通常需要负载生成工具来支持性能测试。例如，验证一个流行网站的可扩展性可能需要模拟数十万虚拟用户。模拟资源限制的工具对于压力测试也特别有用。应小心确保为支持测试而获得的工具与被测系统使用的通信协议兼容。
- 性能测试大纲[CTSL_PT_SYL]包含了性能测试策划的更多细节。

4.5.8 性能测试的规格说明

性能测试主要是在指定的运行条件下测试系统。对于此类测试，需要定义系统预期如何使用的运行配置文件。有关运行配置的更多细节见 4.9 节。

对于性能测试，经常需要通过修改运行配置文件的一部分来更改系统上的负载，以便模拟来自系统预期操作使用的更改。例如，在进行容量测试时，通常需要在测试容量的变量区域覆盖运行配置（例如，增加访问系统的用户数量，直到系统停止响应以确定用户访问容量）。类似地，当进行负载测试时，事务量可能会逐渐增加。

性能测试大纲[CTSL_PT_SYL]包括性能效率测试设计的进一步细节。

4.6 维护性测试

在软件的生命周期中，软件的维护阶段所占的时间比软件开发阶段多。维护性测试是用来测试操作系统或其环境变化的影响。为了确保维护任务尽可能高效，要执行可维护性测试以衡量代码可以被分析、更改、测试、模块化和重用的难易程度。维护性测试不应与为操作软件变更而执行的维护测试相混淆。

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

针对静态维护性测试的有效方法，如 3.2 节和 5.2 节中所讨论的，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介入，并在整个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，维护性可以在软件开发生命周期的早期进行评估，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：基础设施相对复杂，并需要多个部门协作支持。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以下定义进行测量：

- 易分析性。
- 易修改性。
- 易测试性。

影响这些质量子特性的因素包含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，系统设计规范、接口规范）。

ISO 25010[GB/T 25000.10-2016]中其他与维护性相关的质量子特性为：

- 模块化。
- 可重用性。

模块化可以通过静态分析进行测试（见 3.2.3 节）。可重用性测试可以采用架构评审的形式（参见第 5 章）。

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某个特定的运行环境（包括第一次的安装或从现有环境上移植）、新环境或者被其他实体替代的难易程度相关。

[ISO25010][GB/T 25000.10-2016]包括以下可移植性的子特性：

- 易安装性。
- 适应性。
- 易替换性。

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的组件换到另一个），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。

由于可移植性必须在系统设计时已经考虑，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非常重要。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

易安装性测试是执行软件的安装以及为软件安装到目标环境的文档化的安装手册。例如，这可能包括为安装操作系统而开发的软件，或借助于专门的安装软件（安装向导）在客户端 PC 电脑上安装。

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）。
- 测试是否可以完成部分的安装/卸载（不是所有的都安装，例如选择部分功能的安装）。
- 测试安装向导是否能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。

- 验证软件是否可以降级（安装一个早期的版本）或卸载。

在易安装性测试之后通常还要进行功能适合性测试，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。在定义适用性测试时必须对预定目标环境进行识别、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过程将软件移植到各种特定运行环境的能力。测试可以对该过程进行评估。

4.7.4 易替换性测试

易替换性测试侧重于系统中的现有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或 IoT 应用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平行进行。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，重点就是可用于替换的组件有定义明确的接口。

4.8 兼容性测试

4.8.1 简介

兼容性测试考虑以下方面[ISO25010][GB/T 25000.10-2016]：

- 共存性。
- 互操作性。

4.8.2 共存性测试

互不相关的计算机系统当它们能在相同的环境中运行（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。当需要在一个新的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，这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兼容性问题，但如果将其配置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所导致的功能适合性的负面影响（例如，当一个服务器运行多个应用程序时可能产生资源冲突）。
- 评估由于操作系统的修补和升级造成对每一个应用程序的影响。

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

4.9 运行配置

运行配置用作几种非功能测试类型（包括可靠性测试和性能测试）测试规格说明中的一部分。当被测试的需求包含“在特定条件下”的约束时运行配置特别有用，因为它们可以用来定义这些条件。

运行配置定义了系统的使用模式，通常是根据系统的用户和系统执行的操作来规定的。通常根据预期使用数量（和使用时间）和类型（例如主要用户、次要用户）来定义用户。期望被系统执行的不同操作通常是根据频率（和发生的概率）来规定的。此类信息可以通过使用监视工具（在实际或类似的应用程序已经可用的情况下）或基于业务组织提供的算法或估计的预测使用情况来获得。

工具可用于生成基于运行配置的测试输入，并通常使用伪随机生成测试输入的方法。这类工具可用于创建“虚拟”或模拟用户，其数量可与运行配置相匹配（例如，用于可靠性和可用性测试）或超过运行配置（例如，用于压力或容量测试）。有关这些工具的详细信息，请参阅 6.2.3 节。

5. 评审-165 分钟

关键词

评审（review），技术评审（technical review）

评审的学习目标

5.1 技术分析师的评审任务

TTA-5.1.1	（K2）	解释为何评审准备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）	根据教学大纲中提供的检查表分析架构设计并识别问题。
TTA-5.2.2	（K4）	根据教学大纲中提供的检查表，分析一段代码或伪代码并识别问题。

5.1 评审中的技术测试分析师任务

技术测试分析师必须积极参与技术评审过程，提供他们独特的观点。所有评审参与者都应接受正式的评审培训，以更好地了解在技术评审过程中他们各自扮演的角色，并且必须致力于能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。有关技术评审的详细说明，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的定义、应用和维护以及提供缺陷严重度信息方面发挥着重要作用。

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、验证技术方面的正确性，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，他们是否已经具备集成条件是否有必须记录的依赖关系？集成点的测试数据是否可用？评审不仅限于所评审的工作产品，它还必须考虑到评审项与系统中其他项的交互。

5.2 在评审中使用检查表

检查表在评审期间用于提醒参与者在评审期间验证特定点。检查表还可以帮助去除评审的随意性，例如，“这是我们在每次评审中使用的同一检查表，我们不仅仅针对您的工作产品。”检查表可以是通用的，用于所有评审，也可以是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单词“shall”和“should”的正确使用，验证正确的格式和类似项的一致性。检查表也可能集中于安全问题或性能效率问题。

最有用的检查表是由个性化组织自己逐步制定的，因为它们反映了：

- 产品的性质。
- 本身发展环境。
  - 工作人员。
  - 工具。
  - 优先级。
- 以往成功和失败（缺陷）的历史。
- 特殊问题（例如，性能效率、安全性）。

检查表应该为组织定制，也可以为特定的项目定制。本章中给出了一些检查表的示例。

5.2.1 架构评审

软件架构由系统的基本概念或属性组成，体现在其元素、关系及其设计和演进原则中。[ISO42010]。

例如，用于对网站的时间行为（例如，反应速度）进行架构评审的检查表可以包括对以下项的正确实施的验证（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 – 使用地数据副本来减少访问时间。
- 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）之间的隔离。
- 数据复制。

5.2.2 代码审查

代码审查清单必然是低级别的，并且在特定于语言时最有用。包括在代码级别的检查表中引入反模式也非常有用，尤其是对于经验不足的软件开发人员更有帮助。

用于代码审查的清单可能包括以下项目：

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）？
- 存储使用是否高效？
- 是否使用符号而不是“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. 测试工具和自动化-180 分钟

关键词

录制/回放（capture/playback），数据驱动测试（data-driven testing），仿真器（emulator），故障注入(fault injection)，故障植入(fault seeding)，关键字驱动测试(keyword-driven testing)，基于模型的测试(model-based testing / MBT)，模拟器（simulator），测试执行（test execution）

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）	根据给出的业务流程创建关键词。
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

为了具有成本有效性，测试工具（尤其是那些支持测试执行的工具）必须经过精心构建和设计。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。
- 开发在测试执行工具和被测软件之间创建接口可能需要的适配器。
- 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。在敏捷软件开发中，一般会在项目/冲刺计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，（有关详细信息，请参阅测试自动化工程师教学大纲 [CTSL_TAE_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。在敏捷软件开发中，这些任务的角色分工一般会更灵活且不那么正式。

这些活动及其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、API 和 CLI 实现自动化。
- 应用数据驱动的方法。
- 应用关键字驱动的方法。
- 处理软件失效。
- 考虑系统状态。

在测试自动化工程师教学大纲[CTSL_TAE_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，API 和 CLI 的自动化

测试自动化不仅限于通过 GUI 进行测试。 还存在通过命令行界面（CLI）和被测软件中的其他接口点帮助在 API 级别自动测试的工具。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

很多测试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），但是根据测试的软件，每个测试的数据可能会有所不同。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

捷软件开发中业务代表（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

另一种称为关键字或动作词驱动的方法更进一步，还将对提供的测试数据执行的动作与测试脚本 [Buwalda01] 分开。为了实现这种进一步的分离，创建了一种描述性的而不是直接可执行的高级语言。可以为高级和低级操作定义关键字。例如，业务流程关键字可以包括“登录”、“创建用户”和“删除用户”。此类关键字描述将在应用程序域中执行的高级操作。较低级别的操作表示与软件界面本身的交互。诸如“ClickButton”、“SelectFromList”或“TraverseTree”之类的关键字可用于测试不完全适合业务流程关键字的 GUI 功能。关键字可以包含参数，例如关键字“LogIn”可以有两个参数：user_name 和 password。

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低级别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

要实现关键字驱动的测试自动化方法，必须使用高级关键字语言对要测试的业务流程进行建模。 重要的是，该语言对使用者是直观的，使用者可能是从事项目的测试分析师，或者在敏捷软件开发的情况下可能是业务代表（例如，产品负责人）。

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、验证要求取消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。

需要考虑的问题包括：

- 关键字越详细，可涵盖的场景就越具体，但高级语言的维护可能会变得更加复杂。
- 允许测试分析师指定低级操作（“ClickButton”、“SelectFromList”等）使关键字测试更有能力处理不同的情况。但是，由于这些操作直接与 GUI 相关联，因此在发生变更时也可能导致测试需要更多维护。
- 使用聚合关键字可能会简化开发，却让维护复杂化。例如，可能有六个关键字共同创建一个记录。然而， 创建一个连续调用六个关键字的高级关键字可能不是总体上最有效的方法。
- 无论对关键字语言进行多少分析，通常都会有需要新的和更改的关键字的时候。一个关键字有两个独立的域（即背后的业务逻辑和执行它的自动化功能）。因此，必须创建一个过程来处理这两个域。

基于关键字的测试自动化可以显着降低测试自动化的维护成本。最初启动和实施的成本可能更高，但如果项目持续时间足够长，整体成本可能会更低。

测试自动化工程师教学大纲[CTSL_TAE_SYL]包含有关自动化的业务流程建模的进一步详细内容。

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[CTSL_MAT_SYL]。
- 性能测试[CTSL_PT_SYL]。
- 基于模型的测试[CTSL_MBT_SYL]。
- 测试自动化工程师[CTSL_TAE_SYL]。

6.2.1 故障植入工具

为了检查既定测试达到的覆盖，故障植入工具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。

故障植入工具通常由技术测试分析师使用，但也可能由开发人员在测试新开发的代码时使用。

6.2.2 故障注入工具

故障注入工具故意向软件提供不正确的输入，以确保软件能够应对缺陷。注入的输入会导致负面条件，这应该会导致错误处理运行（并被测试）。代码正常执行流程的这种中断也增加了代码覆盖率。

故障注入工具通常由技术测试分析师使用，但也可能由开发人员在测试新开发的代码时使用。

6.2.3 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 4.9）作为脚本来实现。脚本最初可能是为单个用户生成的（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/事务的到达率。
- 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，或者如果要提供的负载配置文件和设施与商业工具提供的负载配置文件和设施相比简单，则开发特定的性能工具可能是合适的。性能测试大纲[CTSL_PT_SYL]中提供了性能测试工具的更多详细信息。

6.2.4 测试网站的工具

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针对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。
- 性能测试工具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。

- 扫描服务器代码的工具，检查网站以前访问过的孤立（未链接）文件。
- 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 HTTP error 404）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。

6.2.5 支持基于模型的测试

基于模型的测试（MBT）是一种使用诸如有限状态机之类的模型来描述软件控制系统在执行时行为的一种技术。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。值得关注的执行线程可以保存下来并用作测试用例。佩特里网（Petri Nets）和状态图（Statecharts）等其他可执行模型也支持 MBT。

MBT 模型（和工具）可以用来生成大量不同的执行线程。MBT 工具也可用于减少模型中可能产生大量路径。使用这些工具进行测试可以提供关于待测软件的不同视图，从而能发现一些功能测试遗漏的缺陷。

基于模型的测试大纲[CTSL_MBT_SYL]中提供了基于模型的测试工具的更多详细信息。

6.2.6 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。

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发布到测试中的构建的质量有积极的影响。程序员所做的改变导致在构建中引入回归缺陷，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

6.2.7 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。

6.2.7.1 模拟器（Simulators）

移动模拟器对移动平台的运行环境进行建模。在模拟器上测试的应用程序被编译成一个专用版本，它可以在模拟器中运行，但不能在真实设备上运行。模拟器在测试中用作真实设备的替代品，但通常仅限于初始功能测试和在负载测试中模拟许多虚拟用户。模拟器相对简单（与仿真器相比）并且可以比仿真器更快地运行测试。但是，在模拟器上测试的应用程序不同于将要分发的应用程序。

6.2.7.2 仿真器（Emulators）

移动仿真器对硬件进行建模，并使用和物理硬件相同的运行时的环境。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。

然而，仿真器无法完全替代硬件设备，因为仿真器的行为方式可能与其试图模仿的移动设备不同。此外，可能会不支持某些功能，例如，类似（多点）触摸、加速计等。这在一定程度上是由于运行仿真器所使用的平台的局限性造成的。

6.2.7.3 通用方面

模拟器和仿真器通常用于通过替换真实设备来降低测试环境的成本。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。

移动应用测试大纲[CTSL_MAT_SYL]包括进一步的细节。

7. 参考资料

7.1 标准

在这些相应的章节中提到了以下标准。

[DO-178C] DO-178C – 机载系统和设备认证中的软件注意事项，RTCA，2011。
第 2 章。

[ISO 9126] ISO/IEC 9126-1:2001，软件工程——软件产品质量。
第 4 章和附录 A。

[ISO 25010] ISO/IEC 25010: 2011，系统和软件工程——系统和软件质量要求和评估（SQuaRE）——系统和软件质量模型。
第 1、4 章和附录 A。

[ISO 29119] ISO/IEC/IEEE 29119-4:2015，软件和系统工程 – 软件测试 – 第 4 部分：测试技术。
第 2 章。

[ISO 42010] ISO/IEC/IEEE 42010:2011，系统和软件工程 – 架构描述。
第 5 章。

[IEC 61508] IEC 61508-5:2010，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。

[ISO 26262] ISO 26262-1:2018，道路车辆 – 功能安全，第 1 至 12 部分。
第 2 章。

[IEC 62443-3-2] IEC 62443-3-2:2020，工业自动化和控制系统的安全 – 第 3-2 部分：系统设计的安全风险评估。
第 4 章。

7.2 ISTQB®文档

[ISTQB®_AL_TTA_OVIEW] ISTQB®技术测试分析师高级概述 v4.0

[CTSL_SEC_SYL] 高级安全测试大纲，2016 版

[CTSL_TAE_SYL] 高级测试自动化工程师教学大纲，2017 版

[ISTQB®_FL_SYL] 基础级教学大纲，2018 版

[CTSL_PT_SYL] 基础级性能测试大纲，2018 版

[CTSL_MBT_SYL] 基础级基于模型的测试大纲，2015 版

[ISTQB®_ALTM_SYL] 高级测试经理教学大纲，2012 版

[CTSL_MAT_SYL] 基础级移动应用测试大纲，2019

[ISTQB®_GLOSSARY] 软件测试术语表，版本 3.2，2019

[CTSL_AuT_SYL] 基础级汽车软件测试员，2018 版

7.3 书籍和文章

[Andrist20] Björn Andrist and Viktor Sehr, C++ High Performance: Master the art of optimizing the functioning of your C++ code, 2nd Edition, Packt Publishing, 2020

[Beizer90] Boris Beizer, "Software Testing Techniques Second Edition", International Thomson Computer Press, 1990, ISBN 1-8503-2880-3

[Buwalda01] Hans Buwalda, "Integrated Test Design and Automation", Addison-Wesley Longman, 2001, ISBN 0-201-73725-6

[Kaner02] Cem Kaner, James Bach, Bret Pettichord; "Lessons Learned in Software Testing"; Wiley, 2002, ISBN: 0-471-08112-4

[McCabe76] Thomas J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, December 1976, pp. 308-320

[Utting07] Mark Utting, Bruno Legeard, "Practical Model-Based Testing: A Tools Approach", Morgan-Kaufmann, 2007, ISBN: 978-0-12-372501-1

[Whittaker04] James Whittaker and Herbert Thompson, "How to Break Software Security", Pearson / Addison-Wesley, 2004, ISBN 0-321-19433-0

[Wiegers02] Karl Wiegers, "Peer Reviews in Software: A Practical Guide", Addison-Wesley, 2002, ISBN 0-201-73485-0

7.4 其他参考资料

以下参考资料指向 Internet 上可用的信息。即使在本高级课程大纲发布时检查了这些参考资料，如果这些参考资料不再可用，ISTQB®也不承担任何责任。

[Web-1] http://www.nist.gov（NIST 国家标准与技术研究所）

[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://validator.w3.org/

[Web-7] https://dl.acm.org/doi/abs/10.1145/3340433.3342822

第 2 章： [Web-7]

第 4 章： [Web-1] [Web-4]

第 5 章： [Web-2]

第 6 章： [Web-3] [Web-5] [Web-6]

8. 附录 A：质量特性概述

下表比较了已被替换的 ISO 9126-1 标准（2012 版《技术测试分析师教学大纲》中使用的标准）与更新的 ISO 25010 标准（最新版教学大纲中使用的标准）中描述的质量特性。

请注意，功能性和易用性是测试分析师教学大纲的一部分。

ISO/IEC 25010	ISO/IEC 9126-1	备注
功能性	**功能**	新名称更准确，避免与“功能”的其他含义混淆
功能完备性		对所述需求的覆盖
功能正确性	准确性	比准确性更通用
功能适合性	适应性	隐含需求的覆盖范围
	互操作性	移至兼容性
	安全性	现在是一项质量特性
性能效率	**效率**	重命名以避免与 ISO/IEC 25062 中的效率定义发生冲突
时间特性	时间特性	
资源利用性	资源利用性	
容量		新的子特性
兼容性		新特性
共存性	共存性	从便携性转移
互操作性		从功能转移（测试分析师）
易用性		隐式的质量问题作出了明确
易理解性	易懂	新名称更准确
易学性	易学性	
易操作性	易操作性	
用户差错防御性		新的子特性

用户界面舒适性	吸引力	新名称更准确
易访问性		新的子特性
可靠性	**可靠性**	
成熟度/成熟性	成熟度/成熟性	
可用性		新的子特性
容错性	容错性	
容错性	容错性	
信息安全	**信息安全**	以前没有，新的子特性
保密性		以前没有，新的子特性
完整性		以前没有，新的子特性
抗抵赖性		以前没有，新的子特性
可核查性		以前没有，新的子特性
真实性		以前没有，新的子特性
维护性	**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	更准确的名称结合可变性和稳定性
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	转移到兼容性

易替换性	易替换性	