ISTQB®软件测试人员认证

高级大纲
测试分析师

2012 版
（中文版 20150601）

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

中文版本号：V3.3

英文版权声明

如果此文档的来源是公认的，则可以拷贝此完整的文档或部分。

高级测试大纲（测试分析师）子工作组：Judy McKay （主席），Mike Smith，Erik Van Veenendaal，2010-2012 年。

中文版权声明

版本历史

版本	日期	说明
ISEB v1.1	2001 年 9 月 4 日	ISEB 从业人员大纲
ISTQB® 1.2E	2003 年 9 月	ISTQB®高级课程大纲（EOQ-SG）
2007 版	2007 年 10 月 12 日	认证测试工程师高级课程大纲 2007 版
D100626	2010 年 6 月 26 日	合并 2009 版修订内容、拆分章节及模块
D101227	2010 年 12 月 27 日	对不影响句意理解的格式进行修改
D2011	2011 年 10 月 23 日	根据当地标准修改、完善大纲（参照国家标准）.
2012 Alpha 版	2012 年 3 月 9 日	对 D2011 版本中所有修改意见进行合并
2012 Beta 版	2012 年 4 月 7 日	对 Alpha 2012 版本中所有修改意见进行合并
2012 Beta 版	2012 年 4 月 7 日	2012 试行版大纲提交至成员大会（GA）
2012 Beta 版	2012 年 6 月 8 日	2012 版大纲复印件提交至国家委员会（NBs）
2012 Beta 版	2012 年 6 月 27 日	合并专家级工作组（EWG）及术语
RC 2012	2012 年 8 月 12 日	候选发布版 - 最终包括 NB 编辑
GA 2012	2012 年 10 月 12 日	最终编辑和整理版提交成员大会（GA）
中文发布版	2013 年 4 月 28 日	最终编辑和整理提交 CSTQB
中文更新版本 V3.0	2014 年 9 月 2 日	周震漪负责修改和完善，建立新版本 V3.0
中文更新版本 V3.3	2015 年 6 月 1 日	根据“软件测试专业术语中英文对照表 v2.4 修订版本 1”修订本文档中的部分术语，以保持一致性 负责人：周震漪

目录

致谢

此文档由国际软件测试认证委员会高级子工作组的核心团队编制。他们包括： Judy McKay（主席），Mike Smith，Erik van Veenendaal。

在此，工作组向参加修订的各位测试专家表示衷心感谢。

同时，参与完成高级测试大纲编写的工作组人员有（按字母顺序）：
Graham Bath，Rex Black，Maria Clara Choucair，Debra Friedenberg，Bernard Homès （副主席)，Paul Jorgensen，Judy McKay，Jamie Mitchell，Thomas Mueller，Klaus Olsen，Kenji Onishi，Meile Posthuma，Eric Riou du Cosquer，Jan Sabak，Hans Schaefer，Mike Smith （主席），Geoff Thompson，Erik van Veenendaal，Tsuyoshi Yumoto。

下列成员参与了评审、评论和大纲表决工作：

Graham Bath，Arne Becher，Rex Black，Piet de Roo，Frans Dijkman，Mats Grindal，Kobi Halperin，Bernard Homès，Maria Jönsson，Junfei Ma，Eli Margolin，Rik Marselis，Don Mills，Gary Mogyorodi，Stefan Mohacsi，Reto Mueller，Thomas Mueller，Ingvar Nordstrom，Tal Pe'er，Raluca Madalina Popescu，Stuart Reid，Jan Sabak，Hans Schaefer，Marco Sogliani，Yaron Tsubery，Hans Weiberg，Paul Weymouth，Chris van Bael，Jurian van der Laar，Stephanie van Dijk，Erik van Veenendaal，Wenqiang Zheng，Debi Zylbermann。

此文档正式由 ISTQB®大会于 2012 年 10 月 19 日颁布。

参加本大纲翻译的 CSTQB 专家有（按姓氏拼音排序）：柴阿峰、杜庆峰、马均飞、沈建雄、熊晓虹、徐文叶、郑文强、周震漪（组长）等。

0. 课程大纲引言

0.1 目的

本大纲根据国际软件测试认证委员会对高级大纲（测试分析师）的要求进行编写，ISTQB®提供此大纲的主要目的：

1. 国家认证委员会需将大纲翻译成当地语言并分发给培训机构，并可根据特定语言对大纲进行适度润色、修改，以保证语句通顺可读。
2. 考试委员会可根据特定语言，按照高级大纲（测试分析师）的学习目标设计考题。
3. 培训机构需根据高级大纲（测试分析师）准备课程并选择最适宜的教学方法。
4. 需要认证的考生，根据高级大纲准备考试（作为培训课程的一部分或独立使用）。
5. 国际软件和系统工程领域，应以此大纲为基础，推进软件和系统测试蓬勃发展。并以此为基础著书和出文章。

国际软件测试认证委员会 ISTQB®允许其他组织、机构在获得书面授权后使用此大纲内容。

0.2 概述

高级大纲由以下 3 个单独的部分组成：

- 测试经理
- 测试分析师
- 技术测试分析师

高级大纲的概述文档[ISTQB_AL_OVIEW]包括以下信息：

- 每个大纲的学习成果
- 每个大纲摘要
- 各大纲之间的关系
- 认知程度描述（K 级别）
- 附录

0.3 预期学习目标

预期学习目标是学习成果的前提保证，同时也被作为高级测试分析师认证考试题目编写的基础。被标记为 K1 级的知识点，需要考生牢记、回忆、识别和认知。K2，K3，K4 级的知识点会在之后相关章节出现。

1. 测试过程 - 300 分钟.

关键字

具体测试用例（concrete test case），出口准则（exit criteria），概要测试用例（high-level test case），逻辑测试用例（logical test case），详细测试用例（low-level test case），测试控制（test control），测试设计（test design），测试执行（test execution），测试实施（test implementation），测试计划（test planning）

测试过程的学习目标

1.2 软件开发生命周期中的测试

TA-1.2.1 （K2）解释在不同的软件开发生命周期中，测试分析师如何以及为什么在介入时间和介入程度上会有不同。

1.3 测试监视、计划和控制

TA-1.3.1 （K2）总结测试分析师为支持测试计划和控制过程而开展的活动

1.4 测试分析

TA-1.4.1 （K4）分析一个包括项目描述和生命周期模型的场景，确定测试分析师在测试分析和设计阶段的主要任务。

1.5 测试设计

TA-1.5.1 （K2）解释利益相关者应该理解测试用例的原因

TA-1.5.2 （K4）分析一个项目场景，确定最合适的详细（实际）测试用例和概要（逻辑）测试用例。

1.6 测试实现

TA-1.6.1 （K2）描述典型的测试分析和设计的出口准则，解释满足这些准则对测试实施工作量的影响。

1.7 测试执行

TA-1.7.1 （K3）针对一个给定的场景，确定执行测试时应采取的步骤和考虑因素。

1.8 评估出口准则和报告

TA-1.8.1 （K2）解释为什么准确的测试用例执行状态信息非常重要

1.9 测试结束活动

TA-1.9.1 （K2）举例说明在测试结束活动中测试分析师应该交付的工作产品

1.1 介绍

在 ISTQB®基础级大纲中，基本的测试过程包括：

- 计划和监控
- 分析和设计
- 实施和执行
- 评估出口准则和报告
- 测试结束活动

为了提供额外的细节和优化过程，高级大纲将一些活动分开进行考虑，以更好的满足软件开发生命周期，同时提高测试监控的效率。下面是高级大纲中考虑的活动：

- 计划和监控
- 分析
- 设计
- 实施
- 执行
- 评估出口准则和报告
- 测试结束活动

这些活动可以是顺序执行，其中有些活动也可以并行的执行。例如：设计可以和实施并行（例如：探索性测试）。测试分析师主要关注测试和测试用例的正确性，以及设计并执行这些测试用例。虽然理解测试过程中的其它步骤也很重要，但是测试分析师的主要工作通常集中在测试项目的分析、设计、实施和执行活动。

在将本大纲中的测试观点引入组织、团队和任务的时候，高级测试人员面临一系列的挑战。考虑不同的软件开发生命周期和被测系统的类型是非常重要的，因为它们会影响所采用的测试方法。

1.2 软件开发生命周期中的测试

长生命周期的测试方法应该在测试策略中进行定义。不同的生命周期，测试分析师介入的时间点是不一样的，同时介入的测试分析师的数量、所需要的时间、可用的信息和期望也是不同的。这是因为测试过程并不是孤立存在的，测试分析师必须了解其它相关的组织领域信息，例如：

- 需求工程和管理－需求评审
- 项目管理－进度输入
- 配置和变更管理－版本验证测试、版本控制
- 软件开发－了解进一步的行动和对应的时间
- 软件维护－缺陷管理、处理时间（例如：缺陷从发现到解决的时间）
- 技术支持－对解决方法进行准确的记录
- 生成技术文档（例如：数据库设计规格说明）－这些文档的输入以及对这些文档的评审

测试活动必须和选择的软件开发生命周期模型相适应，它们可能是顺序、迭代或者是增量模型。例如，在顺序 V 模型中，应用于系统测试级别的 ISTQB®基本测试过程可以描述如下：

- 系统测试计划与项目计划同时进行，测试控制在系统测试执行和结束活动完成之后再结束。
- 系统测试分析与设计，需要与需求规格说明、系统和架构设计规格说明（高层次的）和组件设计规格说明（低层次的）等同时进行。
- 系统测试环境（如测试台、测试装置）的部署应与系统测试执行工作一同在系统规划中启动，并一直持续到开始系统测试执行的前几天（尽管大部分部署工作应与编码和组件测试同期进行）。

- 在系统测试的入口准则已经全部满足（或放弃）时，系统测试执行才开始，这通常是指至少已经完成了组件测试并且组件之间的集成测试也已经完成。系统测试的执行会一直持续到满足系统测试的出口准则为止。
- 对系统测试出口准则的评估和系统测试结果的报告应贯穿于测试执行的整个过程，通常在项目截止日期到来前会加大评估的频率。
- 在系统测试出口准则已经满足要求并且系统测试执行已宣告完成之后，系统测试结束活动才开始。尽管这些活动有可能被推迟到验收测试结束之后或者整个项目活动结束之后才进行。

增量迭代开发模型中的任务之间的顺序可能会有所不同，也可能不包括某些任务。例如，迭代模型在每个迭代中可能对标准的测试过程进行裁剪。分析和设计、实施和执行以及评估和报告可能是每个迭代都要完成的，但是计划活动只在项目开始的时候进行，结束活动只在最后进行。在敏捷项目中，正式流程使用的更少，同时更紧密的工作关系使得项目内的变更变得更加简单。敏捷是一个“轻量级”的过程，它对完整的测试文档的要求更少，取而代之的是更快捷的沟通方式，例如“站立”会议（称为“站立”会议，是因为这些会议很短，通常 10－15 分钟，所以大家不需要坐下来，每个人都积极参与）。

和其它的生命周期模型不同，敏捷项目需要测试分析师更早的介入。测试分析师应该在项目开始就介入，在开发人员开始他们的架构和设计工作的时候与他们共同工作。不一定需要非常正式的评审，但是随着软件的变化，评审需要持续的进行。测试分析师应该专注于一个项目，同时应该全程介入到项目中去。在这种情况下，敏捷团队通常专注于一个项目，同时介入到项目的各个方面。

增量/迭代模型涵盖从适应软件演化过程变更的敏捷方法到具有增量/迭代特性的 V 模型（有时称为嵌入式的迭代）。在嵌入式的迭代模型中，测试分析师需要介入标准的计划和设计活动。但是随着软件的持续开发、测试、变更和部署，测试分析师将转变成更加互动的角色。

无论使用何种软件开发生命周期，测试人员都需要理解自己介入的目的和介入的时间。目前有很多混合式的模型在使用，例如上面提到的 V 模型中的迭代。通常测试分析师必须明确哪种角色最有效，并为之而努力，而不是依赖于模型中对测试介入时间的定义。

1.3 测试计划和监控

这一节关注测试计划和监控的过程

1.3.1 测试计划

测试计划的大部分内容会在测试工作的初始阶段开展，包括所有测试活动的识别和计划，以及用来满足在测试策略中定义的任务和目标所需的资源。在测试计划阶段，测试分析师需要和测试经理一起考虑和计划以下内容：

- 测试计划的确认不仅仅局限于功能性测试。测试计划应该考虑所有的测试类型，并为其制定进度。例如：除了功能性测试，测试分析师可能需要负责易用性测试等。该测试类型就必须包含在测试计划文档中。
- 和测试经理一起评审测试估算，确保为采购和测试环境的验证预留充足的时间
- 计划各种不同的测试配置。如果需要对各种处理器、操作系统、虚拟机、浏览器和其它相关因素的组合配置进行测试，那么需要计划测试技术以提供足够覆盖率。
- 计划对文档的测试。用户在获得软件同时，还获得了相关的文档。文档应该准确有效。测试分析师必须分配时间来对文档进行验证，还可能需要和技术写作人员一起准备屏幕截图和视频片段。
- 计划对安装规程的测试。安装规程和备份恢复规程一样，必须进行充分测试。这些规程可能比软件本身更关键。如果软件无法安装，那么根本谈不上使用。安装规程的测试有时会比较困难，因为测试分析师经常是在没有最后的安装程序的情况下，在预先配置好的系统上进行测试。

- 根据软件生命周期来计划测试。大部分的进度安排都无法通过顺序执行测试任务来完成，很多任务通常需要并行执行（至少部分）。测试分析师必须了解所选择的生命周期，以及在软件设计、开发和实现过程中自己的职责。这也包括分配时间进行确认和回归测试。
- 为识别和分析跨功能团队的风险分配充足的时间。虽然测试分析师并不负责组织风险管理活动，但是测试分析师也需要积极地参与这些活动。

在测试依据、测试条件和测试用例之间存在复杂的关系，以至于他们之间存在各种联系。理解他们之间的关系有利于有效的进行测试计划和控制。通常测试分析师是确定这些联系并将它们尽可能区分开的最合适人选。

1.3.2 测试监控

虽然通常测试监控是测试经理的工作，但是测试分析师需要为控制提供必要的度量数据。

在软件开发生命周期中，需要收集各种量化数据（例如：计划活动的完成比例、完成覆盖率的比例、通过/失败的测试用例的数目）。具体情况下，必须首先定义一个基线（例如，参考标准），然后将跟踪的进度和基线进行比较。测试经理主要关注在汇总和报告总结后的度量信息，而测试分析师关注是为每个度量数据收集信息。每个完成的测试用例、每个编写的缺陷报告、每个达到的里程碑都将会纳入到总的项目度量数据。输入各种跟踪工具的信息尽可能的准确是非常重要的，因为只有这样度量信息才能反映项目的真实情况。

正确的度量数据能够保证经理管理项目（监视），并根据需要进行调整（控制）。例如：软件的某个区域发现了大量的缺陷，那么需要在这个区域的测试中投入更多的工作量。需求和风险的覆盖率信息（可跟踪性）可以用于对剩余的工作进行优先级排序，并分配相应的资源。根本原因分析的信息可以用于确定需要改进的区域。如果记录的数据是准确的，那么可以很好的控制项目，同时利益相关者也可以收到准确的状态信息。在未来的项目中，也可以利用过去项目中所收集的数据有效地对项目进行规划。这些准确的数据有很多用途，确保数据的准确、及时和客观也是测试分析师的工作的一部分。

1.4 测试分析

在测试计划阶段，确定了测试项目的范围。测试分析师将这个范围用于：

- 分析测试依据
- 识别测试条件

测试分析师为了更有效的进行测试分析，需要满足以下入口准则：

- 应该有一个描述测试目标的文档可作为测试依据
- 这个基础文档通过了评审，并已经根据评审意见进行了相应的更新
- 合理的预算和进度，用来完成针对该测试目标的测试工作

通常测试条件是通过分析测试依据和测试目标而获得的。有些情况下，文档可能不够新或者不存在，测试条件可以通过与利益相关者的交流获得（例如：在讨论会或者项目计划时）。这些条件可以用于确定测试的内容，以及在测试策略或者测试计划中体现的测试设计技术。

虽然通常的测试条件依赖于被测试的具体条目，但是测试分析师仍然有一些通用的考虑：

- 通常定义不同详细程度的测试条件是比较合理的。开始的时候，识别概要的测试条件用来定义通用的测试目标，例如“屏幕 X 的功能性”。然后，识别更详细的条件作为具体的测试用例的基础，例如“屏幕 X 拒绝比正确的长度少一位的账户”。使用这种分级的方法定义测试条件有助于确认高级别条目覆盖的充分性。
- 如果已经定义了产品风险，那么测试条件必须保证识别每个产品风险并回溯到风险条目。

总的来说，在测试分析活动中，为了满足测试项目具体区域的测试需要，测试分析师需要了解必须设计哪些具体的测试。

1.5 测试设计

依赖于测试计划中确定的范围，随着测试过程的继续，测试分析师开始设计测试用例，并实施与执行这些用例。测试设计过程包括以下活动：

- 确定详细（实际）测试用例或概要（逻辑）测试用例最适合在哪些地方使用。
- 确定能够提供必要的测试覆盖率的测试用例设计技术。
- 创建测试用例用于覆盖已经识别的测试条件。

在风险分析和测试计划中识别的优先级准则应该应用在从分析、设计到实施和执行的整个过程。

根剧需要设计的测试类型，用于设计工作的工具可用性可以作为测试设计的一个入口准则。

在设计测试时，以下方面需要重点关注：

- 有些测试条目，最好只定义测试条件，而不是更进一步定义脚本化的测试。在这种情况下，定义的测试条件用来指导非脚本化的测试。
- 明确的识别通过/失败准则。
- 设计的测试用例不仅仅需要自己理解，也需要其它测试人员能够理解。如果设计者不是将来执行测试的人，那么为了理解测试目标和相应测试的重要性，其它测试人员将需要阅读并理解之前具体定义的测试。
- 测试还必须被其它利益相关者所理解，例如：开发人员，他们将评审测试；审计员，他们将批准这些测试。
- 设计的测试应该覆盖所有软件与行为者（例如：最终用户、其它的系统等）之间的交互，而不仅仅局限于用户可见的接口。流程内部的通信、批处理执行和其它的中断也和软件有交互并可能存在缺陷，所以测试分析师必须设计测试以规避这些风险。
- 设计的测试需要覆盖各种测试对象之间的接口。

1.5.1 实际和逻辑测试用例

测试分析师的主要工作之一就是针对给定的场景确定最好的测试用例类型。具体测试用例包括所有的详细信息以及测试人员执行测试用例并验证结果所需要的规程（包括所有数据的需求）。在需求被很好的定义、测试人员经验不够丰富以及测试需要被外部人员验证（例如：审计）时，具体测试用例非常有用。具体测试用例提供了很好的可重现性（例如：其它测试人员将会获得同样的结果），但同时也需要大量的维护工作，并限制了测试人员在执行时的创造性。

逻辑测试用例为测试提供了指导，但同时允许测试分析师在执行测试的时候对测试数据甚至测试规程进行变化。相对于具体测试用例而言，逻辑测试用例提供了更好的覆盖率，因为每次执行他们的时候都有一定程度的变化。这也降低了可重现性。逻辑测试用例更适合于以下场景：需求没有很好的定义、执行测试的测试分析师具有丰富的测试和产品经验、不需要正式的文档（例如：不需要进行审计）。在需求过程的早期，当需求还没有被完全定义时，就可以定义逻辑测试用例。当需求的定义更加完善和稳定后，这些测试用例可以用来开发具体测试用例。在这种情况下，测试用例的创建是从逻辑到实际的顺序完成的，测试执行的时候只使用具体测试用例。

1.5.2 测试用例的创建

通过使用测试策略和/或测试计划中标识的测试设计技术可以识别测试条件，测试用例的设计是对这些识别出来的测试条件的逐步细化。根据使用的测试策略中的规定，测试用例应该可重复、可验证和可跟踪（可以追溯到测试依据，例如：需求）。

测试用例的设计需要识别以下内容：

- 目的
- 前置条件，例如：项目或局部测试环境的需求以及交付的计划、系统的状态等
- 测试数据的需求（既包括测试用例的输入数据，也包括执行测试用例时系统中必须已经存在的数据）

- 期望结果
- 后置条件，例如：受影响的数据，系统的状态，后续处理的触发等

测试用例的详细程度将会影响开发的成本以及执行的可重复性程度，所以应该在实际创建测试用例前就定义好。不太详细的测试用例给测试分析师的测试执行带来更大的灵活性，为研究可能的、有兴趣的区域提供了机会，但同时也降低了可重复性。

定义测试的期望结果是一个常见的挑战。手工的计算期望结果通常比较枯燥，并容易产生错误。如果可能的话，最好是能找到或创建一个自动的测试准则。在识别期望结果时，测试人员不仅需要考虑屏幕上的输出，还需要考虑后置条件的数据和环境。如果已经清晰的定义了测试依据，理论上识别正确的结果将会比较简单。但是，测试依据经常是模糊的、矛盾的、缺少关键域的覆盖或者彻底缺失的。这种情况下，测试分析师必须具备（或者能够获得）专业的技能。同时，即使具有完善的测试依据，各种影响因素和系统响应之间复杂的交互也使得定义期望结果变得很困难。因此，测试准则是至关重要的。执行无法判断结果是否正确的测试用例毫无意义，同时还会导致对系统生成错误的失效报告或错误的信心。

尽管测试依据会有所不同，但是上述活动适用于所有级别的测试。例如用户验收测试可能主要是基于需求规格说明、用例（Use case）和定义的业务过程，而组件测试可能主要是基于详细设计规格说明。

虽然这些测试的目的可能是不一样的，但是需要牢记在所有测试级别都可能发生这些活动。例如单元级别的功能性测试是为了确认一个具体组件详细设计中所描述的功能的正确性；集成级别的测试是为了验证组件间的交互以及通过他们之间的交互提供的功能。在系统测试级别，测试的对象应该是端到端的功能。在测试分析和设计时，需要记住针对的测试级别和相应的测试目的。这将有助于确定所需要的详细程度，同时确定可能使用的工具（例如：组件测试级别的驱动器和桩）。

在测试条件和测试用例的开发中，通常有不少文档输出作为测试的工作产品。在实际运用中，测试工作产品的文档化范围会有很大的差别。这取决于如下因素：

- 项目风险（必须或没有必要记录在案）
- 文档给项目带来的附加收益
- 遵循的标准
- 使用的生命周期模型（例如，敏捷开发尽量使文档最少化）
- 测试分析和设计过程中，对测试依据的可追溯性的要求

根据测试范围的不同，测试分析和设计可能会关注测试对象的不同质量特性。ISO 25000[ISO 25000]（替代了 ISO 9126）提供了必要的参考信息。在测试硬件/软件系统时，可能需要额外的质量特性。可以通过评审和静态分析来增强测试分析和测试设计的过程。事实上，实施测试分析和测试设计也是一种静态测试，因为在这个过程中可以发现一些基础文档的问题。在准备需求评审会议时，一个非常好的方式就是基于需求规格说明进行测试分析和测试设计。通过阅读需求来创建符合需求的测试，同时也是一种评估需求完备性的方法。该活动通常可以发现需求的不清晰、不可测试或者缺乏已定义的验收准则。同样的，测试工作产品，例如测试用例、需求分析和测试计划也应该是评审对象。

有些项目，例如使用敏捷生命周期，可能只有很少的文档化的需求。这些项目中通常会有“用户故事”，其中描述了少量的功能性。一个用户故事应该包括已定义的验收准则。一旦软件已经满足了验收准则，这通常意味着它已经准备好（或者已经完成了）与其它功能进行集成，以展示它的功能性。

虽然测试基础架构的具体需求可能直到测试实施时才能最终确定下来，但是在测试设计时仍然可能需要对其进行定义。需要牢记测试基础架构不仅仅包括测试对象和测试件，例如：基础架构可能包括房间、设备、人员、软件、工具、外围设备、通讯设备、用户授权和其它运行测试需要的项。

根据项目参数的不同，测试分析和测试设计的出口准则可能会不同，但是这两部分讨论的所有条目都应该包含在已经定义的出口准则中。可度量的出口准则，以及为后继步骤提供所需要的所有信息和准备都是非常重要的。

1.6 测试实施

测试实施是测试设计的下一个步骤。它包括创建自动化的测试、组织测试执行的顺序（包括手工和自动）、确定最终的测试数据和测试环境，以及形成一个包括资源分配的测试执行进度，这样可以为开始测试用例执行做准备。这也包括针对特定的测试级别对其明显的和潜在的入口准则进行检查，并确认上一个步骤中出口准则已经得到满足。如果跳过出口准则，这个测试级别或者测试过程中的某个步骤可实现的工作量将会受到影响，可能导致进度延迟、质量下降和额外的工作量。在开始测试实施之前，确认上一个步骤的所有出口准则是否已经满足是非常重要的。

在确定测试执行的顺序时，需要考虑多个因素。有些情况下，把测试用例组织成测试套件（例如：测试用例组）是有意义的。这样有助于测试的组织，相关的测试用例可以一起执行。在使用基于风险的测试策略时，风险优先级可能决定了测试用例的执行顺序。也有其它一些因素可能影响测试的顺序，例如：合适的人、设备、数据和被测试的功能是否到位。部分代码发布并不少见，此时测试活动必须和被测试对象是否可测相互协调。尤其是在增量生命周期模型中，测试分析师通过和开发团队沟通以确认软件按照可测试的顺序交付非常重要。在测试实施时，测试分析师应该最终确定手工和自动化测试的运行顺序，并认真检查需要按照特别顺序运行的测试限制。必须将他们之间的依赖关系文档化，并进行检查。

测试用例和测试条件的详细程度会影响测试实施工作的详细程度和相关工作的复杂度。有些情况下涉及到法规，测试必须提供符合相关标准的证据，例如：美国联邦航空管理署的 DO-178B/ED 12B [RTCA DO-178B/ED-12B]。

在前面提到，测试需要测试数据，而且在有些情况下可能需要非常多的测试数据。在测试实施时，测试分析师创建输入数据和测试环境，并录入数据库和其它类似的存放处。

测试实施也需要考虑测试环境。在测试执行之前，测试环境应该准备好，并通过验证。一个“满足需要”的测试环境是至关重要的。例如测试环境能够有利于通过可控的测试发现相应的缺陷；在没有缺陷发生的时候能够正常运行；如果需要的话，在高级别的测试中测试环境能够尽可能的接近产品或者终端用户的环境。针对不可预测的变更、测试结果或者其它因素，测试环境在测试执行期间可能发生变化。如果测试执行时测试环境发生了变化，那么评估这些变化对已经运行的测试的影响是非常重要的。

在测试实施时，测试员必须明确相关人员有创建和维护测试环境的职责，同时保证所有的测试件、测试支持工具和相关的流程是可用的。这包括配置管理、缺陷管理、测试记录和管理。另外，测试分析师必须为出口准则评估和测试结果报告而收集数据的流程进行验证。

根据测试计划的决定，在各种测试实施方法之间进行平衡是明智的。例如基于风险的分析性测试策略经常和动态测试策略同时使用。在这种情况下，部分测试实施的工作量并不需要完全遵循已定义的脚本（非脚本化）。

非脚本化测试并不应该是随机（ad hoc）测试或者是毫无目标的，因为这样测试的时间和测试的覆盖范围都不可预知，除非是在特定的时间段内和被特许。随着时间的推移，测试人员已经开发了多种基于经验的技术，例如：攻击、错误猜测[Myers79]和探索性测试。测试分析、测试设计和测试执行仍然是需要的，但是他们可能主要发生在测试执行阶段。在遵循这些动态测试策略时，每个测试的结果都会影响后继测试的分析、设计和实施。虽然这些策略是轻量级的，并经常能够有效的发现缺陷，但是他们仍然存在一些缺点。这些技术需要测试分析师具备专业的技能，因为测试的周期难于预测，覆盖率难于跟踪，同时因为缺乏文档或者工具支持而导致可重复性很差。

1.7 测试执行

一旦提交了测试对象并且满足了测试执行的入口准则，就可以开始测试执行。测试执行的依据是测试规程，但可以给予测试人员一定的自由度，测试分析师应该有足够的时间确保覆盖在测试过程中观察到的额外感兴趣的测试场景和行为（对于在上述的额外情况中发现的任何失效，都必须描述与已经设计的测试规程之间的差异，使得失效能够被复现）。脚本化和非脚本化（例如：探索性）测试技术的集成有助于发现在脚本覆盖中遗漏的缺陷，同时避免测试用例的杀虫剂效应。

测试执行的核心活动是对比实际的测试结果和期望的测试结果。测试人员必须高度关注这些任务，否则，如果测试结果未能识别出测试对象中的缺陷（漏报）或者错误地将正确的行为归在错误的一类（假报），则测试设计和实施的工作可能是浪费时间。如果期望的结果和实际的测试结果不匹配，将产生一个事件。必须仔细观察事件并确定产生的原因（产生事件的原因可能是测试对象中的一个缺陷，也可能不是），同时要收集数据帮助解决事件。（更多参见第 6 章关于缺陷管理的内容）

当发现失效时，需要仔细的评估测试文档（测试规格说明、测试用例等）以确认文档的正确性。测试文档本身可能因为各种原因而出现错误。如果测试文档是不正确的，应该对其进行纠正并重新运行相关的测试。即使已经成功运行很多次测试的情况下，测试依据和测试对象的变化仍然可能导致错误的测试用例。测试人员必须认识到：关注的结果可能是由于错误的测试而造成的。

测试执行时，必须对测试结果进行适当的记录。没有对已经运行的测试结果进行记录可能导致重测以识别正确的结果，这将会导致测试效率低下，同时带来延期。（注意，充分的日志记录可以帮助解决覆盖和可重现方面的技术，例如，探索性测试）由于测试对象、测试件和测试环境都可能进行更新，所以测试日志中应该识别具体的版本，同时包括具体的环境配置。针对测试执行，测试日志按照时间顺序提供了相关的详细信息。

在测试过程发生的事件和单个测试都是日志的一部分，每个测试都应当唯一标识并且记录它的状态作为测试执行的成果。任何影响测试执行的事件都要记录。应当详细记录信息用于度量测试覆盖率并且文档化测试延迟、中断的原因。另外，需要记录信息以支持测试控制、测试过程报告、度量出口准则和测试过程改进。

根据测试级别和测试策略的不同，测试日志会有所不同。例如组件测试采用自动化的方式，那么大部分的日志信息都由自动化的测试生成。如果采用手工测试的方式，测试分析师将会根据测试执行的情况记录信息，通常需要录入到一个跟踪测试执行信息的测试管理工具中。在有些情况下，根据法规或者审计的需求，可能需要更详细的记录测试执行的信息。

有时，用户或者客户会参与测试执行。如果测试发现的缺陷很少，这将有助于他们对系统建立起信心。通常这种假设在早期的测试级别中是无效的，但是在验收测试中可能是适合的。

在测试执行时，需要考虑以下方面：

- 关注和探索“不相关”的异常。貌似不相关的现象或者结果经常是由潜伏在表象之下的缺陷造成的（就像冰山一角）。
- 检查产品并没有做任何不希望做的事情。检查产品是否按照预期进行工作是正常的测试关注点，但是测试分析师还必须确信在完成一些它不应该完成的事情时没有发生不适当的行为（例如：额外的不需要的功能）
- 构建测试套件，并持续对其进行更新。代码会不断的变化，因而需要实现额外的测试以覆盖这些新的功能，同时对软件的其它区域进行回归测试。在测试执行时经常会发现有差异，所以构建测试套件是一个持续的过程。
- 为后继测试提供必要的记录。当软件提供给用户或者在市场上发布时，测试任务并没有终止。通常会开发新的版本或者发布，所以应该保存相关知识并移交给负责后继测试的测试人员。

- 不要期望重新运行所有的手工测试。重新运行所有的手工测试是不现实的。当出现可疑的问题时，测试分析师应该对之进行记录和研究，而不是期待通过随后的测试用例执行来发现这些问题。
- 挖掘缺陷跟踪工具中的数据，以设计新的测试用例。针对非脚本化或者探索性测试中发现的缺陷创建测试用例，并把他们添加到回归测试套件中。
- 在回归测试前发现缺陷。回归测试的时间通常都是有限的，所以在回归测试期间发现问题会导致进度延迟。通常回归测试不会发现大量的缺陷，很大程度上是因为这些测试之前已经被运行过了。但是这并不意味着回归测试应该被彻底的忽略，而只是在发现新的缺陷方面，回归测试的有效性要比其它测试要低。

1.8 评估出口准则和报告

从测试过程的角度来看，测试过程监控需要收集合适的数据以满足报告的需求。这包括针对完成情况进度的度量。在测试计划阶段定义了出口准则，这些准则应该被分成“必须”和“应该”的准则。例如，出口准则可能描述“优先级为 1 或者 2 的缺陷“必须”关闭”，和“95%的测试用例“应该”测试通过”。在这种情况下，不能满足“必须”的出口准则将导致出口准则的失败，而 93%的测试用例通过率仍然可以允许项目进入下一个级别。必须清晰的定义出口准则，这样才可以对它们进行客观的评估。

测试分析师负责为测试经理提供针对满足出口准则评估测试过程所需要的信息，同时保证这些数据的正确性。例如：如果针对测试用例的完成，测试管理系统提供以下状态：

- 通过
- 失败
- 有条件的通过

那么测试分析师需要非常清楚的了解这些状态的具体含义，同时在使用这些状态时保持一致。“有条件的通过”是否意味着发现了缺陷，但是这个缺陷并不影响系统的功能？如果是一个会导致用户疑惑的易用性的缺陷呢？如果通过率是一个“必须”的出口准则，那么将一个测试用例算作“失败”而不是“有条件的通过”将变成一个很重要的因素。还必须考虑那些虽然被标为“失败”，但并不是由于缺陷引起的测试用例（例如：没有正确的配置测试环境）。当有人对被跟踪的度量指标或者状态数据的使用出现疑惑时，测试分析师必须和测试经理一起进行澄清，以保证被跟踪的信息的正确性以及在整个项目中的一致性。

通常在测试周期中，测试分析师被要求提供状态报告，同时也会在测试结束时参与最终的测试报告编写。这需要从缺陷和测试管理系统中收集度量数据，同时评估整个覆盖率和过程。测试分析师应该能够使用这些工具，并为测试经理提供所需的额外信息。

1.9 测试结束活动

一旦决定测试执行活动结束，应该保存最重要的测试结果并且需要传递给相关人员或者归档。这些都是属于测试结束活动。测试分析师需要参与将工作产品交付给所需要的人。例如：应该将已知的、遗留的或者接受的缺陷告诉要使用和对系统的使用提供支持的人。测试和测试环境应该提交给负责维护测试的人。回归测试集（自动的或者手工的）也是一个工作产品。关于测试工作产品的信息必须清晰的文档化，包括适当的链接和授予适当的访问权限。

测试分析师也应该参与回顾会议（“经验教训”）。回顾会议上，记录重要的经验（既包括在测试项目中的，也包括贯穿整个软件开发生命周期的），并有计划的对“好的”经验进行强化，对“坏的”经验进行规避（至少要控制）。测试分析师是这些回顾会议所需的信息的重要来源，并能够提供正确的过程改进信息。如果只有测试经理参加这样的会议，那么测试分析师必须通过测试经理把相关的信息传递过去，以保证项目信息的正确性。

配置管理系统中的工作产品、结果、日志、报告和其它文档必须归档。这些任务常常由测试分析师来完成，是非常重要的测试结束活动，尤其是这些信息在将来的项目中还需要继续使用的时候。

虽然测试经理通常来决定应该归档哪些信息，但是测试分析师也需要思考如果将来这个项目需要重新开始，哪些信息是需要的。在项目结束的时候考虑这些信息要比将来这个项目重新开始时再来考虑，可以节约大量的工作量。

2. 测试管理：测试分析师的职责 – 90 分钟.

关键字

产品风险（product risk），风险分析（risk analysis），风险识别（risk identification），风险级别（risk level），风险管理（risk management），风险缓解（risk mitigation），基于风险的测试（risk-based testing），风险监督（test monitoring），测试策略（test strategy）

测试管理的学习目标：测试分析师的职责

2.2 测试进度监控

TA-2.2.1 （K2）解释测试过程中必须跟踪的各种信息类型，以进行足够的项目监控

2.3 分布式测试，外包测试与内包测试

TA-2.3.1 （K2）举例说明在 24 小时持续工作的测试环境下进行良好的沟通

2.4 测试分析师在基于风险的测试中的任务

TA-2.4.1 （K3）在给定的项目环境中，参与风险识别，开展风险评估和提供合适的风险缓解建议

2.1 介绍

测试分析师需要在很多领域与测试经理相互合作并为他提供数据，本节将关注测试分析师在测试过程中作为主要贡献者的一些特定领域。测试经理将从测试分析师得到所需要的信息。

2.2 测试过程的监视和控制

测试进度主要从 5 个维度进行监视：

- 产品（质量）风险
- 缺陷
- 测试
- 覆盖率
- 信心

在项目或者运营过程中，测试分析师可以且经常以特定的方式测度并报告产品风险、缺陷、测试和覆盖率。尽管可以通过调查的方式测度信心，但它经常是以主观方式进行报告。收集信息以支持这些度量是测试分析师日常工作的一部分。保证这些数据的正确性非常重要，因为不正确的数据会生成不正确的趋势信息，也可能导致不正确的结论。更有甚者，不正确的数据将会导致不正确的管理决策，并损害测试团队的可信性。

在采用基于风险的测试方法时，测试分析师需要跟踪：

- 哪些风险已经通过测试得到了缓解
- 哪些风险被认为是没有得到缓解

风险缓解的跟踪经常通过工具来实现，该工具也跟踪测试完成率（例如：测试管理工具）。这就要求将识别的风险映射到测试条件，再映射到测试用例，假如执行该测试用例并通过了，那么风险就得到了缓解。通过这种方式，更新测试用例就自动的更新了风险缓解信息。这既可以通过手工测试也可以通过自动化测试的方式实现。
缺陷跟踪经常是通过缺陷跟踪工具来实现。在记录缺陷的同时，缺陷的特定分类信息也会记录在案。这些信息用来生成趋势图表，以显示测试的进度与软件的质量。分类信息将会在缺陷管理章节进行详细讨论。生命周期可能会影响记录的缺陷文档数量，以及用来记录信息的方法。

在开展测试活动过程中，测试用例的状态信息必须记录在案。这通常由测试管理工具来实现，在需要的时候也可以通过手工的方式完成：

- 测试用例创建状态（例如：已设计的，已评审的）
- 测试用例执行状态（例如：通过的，失败的，被阻塞的，被忽略的）
- 测试用例执行信息（例如：日期与时间，测试人员名字，使用的数据）
- 测试用例执行附件（例如：屏幕截取，附属日志）

根据识别的风险项，测试用例应该被映射到它们验证的需求条目。测试分析师需要记住：假如测试用例 A 映射到需求 A，并且该测试用例是唯一映射到该需求，当测试用例 A 执行并且通过，那么针对需求 A 的验证就可以认为是完成了，这一点很重要。这可能是对的，也可能是不对的。在很多情况下，需要多个测试用例才能完全验证需求，由于时间限制，只创建了部分测试用例。举个例子，假如需要 20 个测试用例才能完全验证某个需求的实现，但只生成和执行了 10 个测试用例，此时需求覆盖率信息显示为 100%覆盖，而实际上只达到了 50%。正确的覆盖率跟踪与需求的评审状态跟踪可以用来对信心进行测度。
信息记录的数量与详细程度依赖于多个因素，包括软件开发生命周期模型。例如，在敏捷项目中，由于团队之间的紧密合作及更多的面对面沟通，记录的状态信息就会比较少。

2.3 分布式测试，外包测试与内包测试

在很多时候，并不是所有的测试工作都是由一个测试团队单独完成的，他们是项目团队成员的组成部分，并且和项目团队的其他成员处在同一个工作场所。如果测试工作分散在不同地点，那么可以称之为分布式测试，如果测试工作集中在一个地点，那么称之为集中式测试。若测试工作在（有别于项目团队所在地）一个（或多个）地点，由非项目成员承担，那么可以称之为外包测试。若测试工作由其他与项目团队在项目同一地点的非项目成员承担，那么称之为内包测试。

当项目的工作是由分布在不同地点或者由多个公司的测试团队完成时，测试分析师必须特别关注有效的沟通与信息共享。有些组织以“24 小时连续测试”的模式进行工作，在这种模式下，一个时区的测试团队会将工作移交到另一个时区的测试团队，以保证测试工作可以连续进行。这就要求对测试分析师的移交与接受工作进行专门的计划。良好的计划对于理解职责是重要的，但确保正确的信息可用更是至关重要。

当口头交流不可行的时候，书面沟通将是必须的。这意味着邮件、状态报告与测试管理和缺陷跟踪工具的有效使用是必须的。假如测试管理工具允许将测试任务指派给测试人员，那么它同样可以作为进度安排的工具，并且以简单的方式在测试人员之间移交工作。在需要的时候，正确记录的缺陷可以移交给同事进行后续的跟踪。对于无法依赖人与人之间直接交互的组织，有效使用这些沟通系统将是至关重要的。

2.4 测试分析师在基于风险的测试中的任务

2.4.1 概述

通常来说，测试经理全权负责基于风险的测试策略的建立与管理。测试经理经常会要求测试分析师参与这些工作，以确保基于风险的测试方法可以正确实施。

测试分析师应该主动参与下面这些基于风险的测试活动：

- 风险识别
- 风险评估
- 风险缓解

这些任务以迭代的方式开展，并贯穿于项目生命周期以处理暴露出来的风险、变更优先级，以及有规律地评估与沟通风险状态。

测试分析师应该在测试经理为项目建立的、基于风险的测试框架下工作。测试分析师应该贡献他们在业务领域风险方面的知识，这些风险根植于项目中，例如与安全、商业和经济以及政治因素相关的风险。

2.4.2 风险识别

通过广泛邀请可能的项目干系人代表，这样在风险识别过程中会发现尽可能多的重要风险。由于测试分析师经常在被测对象特定业务领域拥有独特的知识，因此他们特别适合指导业务专家和用户的评审，独立的评估，以及使用风险模板开展风险研讨会，与潜在的和当前的用户开展头脑风暴，定义测试检查表与收集以前类似系统或项目的经验。特别强调的，测试分析师应该与用户和其他领域专家紧密合作，以确定测试中需要覆盖的业务风险区域。测试分析师识别潜在风险可在用户及利益相关者的潜在影响方面提供特别的帮助。

项目中可能被识别的风险包括：

- 软件功能的正确性问题，例如不正确的计算
- 易用性问题，例如键盘快捷键不足
- 易学性问题，例如在关键决策点缺少针对使用者的指南

测试特定质量特性的内容将在本大纲的第 4 章中阐述。

2.4.3 风险评估

风险识别是尽量识别尽可能多的相关风险，而风险评估是研究这些被识别出的风险，特别是将这些风险明确分类并确定每个风险的可能性及影响。

确定风险级别通常涉及风险的评估，即针对每个风险条目，确定其可能性和影响。风险发生的可能性一般指被测系统中存在潜在问题的可能性，并且这些问题是系统处于生产阶段被发现的。换句话说，风险发生的可能性起源于技术风险。技术测试分析师应该致力于发现和理解每个风险条目的潜在技术风险级别，而测试分析师应该致力于理解风险发生时的潜在商业影响。

风险发生的影响是指风险的发生对用户、客户或其它项目干系人影响的严重性。换句话说，它起因于商业风险。测试分析师应该致力于识别与评估每个风险项的潜在商业领域的或者用户影响。影响商业风险的因素包括：

- 受影响功能的使用频率
- 商业损失
- 潜在的金融、生态或社会方面的损失或法律责任
- 民事或刑事法律制裁
- 安全方面的考虑
- 罚款，失去经营执照
- 缺少合理的变通
- 功能的可视性
- 可视的失效导致负面宣传和潜在的形象受损
- 客户流失

根据给定的有效风险信息，测试分析师需要根据测试经理建立的指南确定商业风险的级别。级别可以通过定性（例如：低，中等，高）或者定量的方式进行分类。除非有一种方式可以以定义的尺度客观的测度风险，否则它不可能是真正的定量测度。准确的测度可能性/概率和成本/后果通常是非常困难的，因此确定风险级别通常是以定性的方式开展的。数字也可以指定给定性值，但这并不代表是真正的定量测度。例如，测试经理可能决定商业风险的分类取值是从 1 到 10，其中 1 代表最高，即影响商业的风险最高。一旦指定了可能性（技术风险的评估）和影响（商业风险的评估），就可以将这些值相乘得到每个风险项的风险级别。而这些风险级别可以用来对风险缓解活动进行优先级划分。一些基于风险的测试模型，例如 PRISMA® [vanVeenendaal12]，并没有组合风险值，它允许测试方法分开处理技术与商业风险。

2.4.4 风险缓解

测试分析师在整个项目阶段，需要探索开展下面的活动：

- 降低产品风险：其手段包括使用可以明确证明测试条目通过或失败的良好的设计测试用例，积极参与软件工作产品的评审，例如，需求、设计和用户文档
- 实施在测试策略和测试计划中所定义的适当的风险缓解活动
- 根据项目演进过程中收集的额外信息，在需要的时候重新评估已知的风险，分别调整可能性和影响程度，或者同时调整。
- 根据测试过程中获取的信息识别新的风险

针对人们谈论的产品（质量）风险，测试是减轻该类风险的一种方式。测试人员可以通过发现缺陷，根据其提供的已知缺陷的认识和在发布之前修复缺陷的机会来降低风险。假如测试人员没有发现缺陷，则测试活动确保了在特定条件下（即已经测试过的条件）被测系统能正确工作，从而也降低了风险。测试分析师通过收集准确测试数据、创建和测试真实的用户场景以及开展或负责监督易用性研究的机会协助确定风险的缓解方法。

2.4.4.1 测试优先级

风险级别也可以用于确定测试优先级。测试分析师可能认为一个会计系统内交易的准确性属于高风险区域。因此，为了减轻风险，测试人员需要和其他业务领域专家紧密合作以收集一组样本数据来处理与验

证其正确性。类似地，测试分析师可能认为新产品的易用性问题是个最大风险。测试分析师可以在集成测试级别期间优先开展易用性测试，以在测试的早期帮助识别与解决易用性问题，而不是等到验收测试才发现这些问题。必须在计划阶段尽早考虑优先级，这样才能保证在进度计划规定的时间内能够覆盖必需的测试。

在某些情况下，所有高风险的测试都应该首先执行，当所有高风险的测试执行后，低风险的测试才会执行。测试执行顺序应严格地根据风险级别来制定并执行（也称作“深度优先”），在其它情况下，测试优先级可以根据抽样方法来制定。抽样方法就是选择一个根据不同风险权重的测试样本，使得测试样本可以覆盖到所有已识别的风险。用风险覆盖率评估选出的测试样本，保证每个风险至少能被覆盖到一次（通常称作“广度优先”）。

无论基于风险的测试是深度优先还是广度优先，分配给测试的时间都可能不足，无法执行所有的测试。基于风险的测试使得测试人员能以剩余风险级别的方式向管理人员报告当前测试状态，让管理人员决定是否要增加测试时间或将剩余的风险转移给用户/客户、服务/技术支持人员和/或运营人员。

2.4.4.2 为后续测试周期调整测试

在开始测试实施之前，风险评估不是一次性的活动，而是一个持续的过程。每个未来的计划测试周期都需要考虑新的风险分析，并考虑下面的因素：

- 任何新增的或有很大变化的产品风险
- 测试中发现的不稳定或易产生错误的区域
- 修复缺陷带来的风险
- 测试中发现的典型缺陷
- 已测试过，但测试不充分的区域（测试覆盖低）

假如分配了额外的测试时间，可以扩展风险覆盖到低级别的风险区域。

3. 测试技术- 825 分钟

关键字：

边界值分析（BVA）（boundary value analysis (BVA)），因果图（cause-effect graphing），基于检查表测试（checklist-based testing），分类树方法（classification tree method），组合测试（combinatorial testing），决策表测试（decision table testing），缺陷分类（defect taxonomy），基于缺陷的技术（defect-based technique），域分析（domain analysis），错误推测（error guessing），等价类划分（equivalence partitioning），基于经验的技术（experience-based technique），探索性测试（exploratory testing），正交矩阵（orthogonal array），正交矩阵测试（orthogonal array testing），结对测试（pairwise testing），基于需求的测试（requirements-based testing），基于规格说明的技术（specification-based technique），状态转换测试（state transition testing），测试章程（test charter），用例测试（use case testing），用户故事测试（user story testing）

测试技术的学习目标

3.2 基于规格说明技术

TA-3.2.1 （K2）解释因果图的应用
TA-3.2.2 （K3）使用等价类划分的测试设计技术，对给定的规格说明项编写测试用例以达到测试覆盖的要求。
TA-3.2.3 （K3）使用边界值分析的测试设计技术，对给定的规格说明项编写测试用例以达到测试覆盖的要求。
TA-3.2.4 （K3）使用决策表分析的测试设计技术，对给定的规格说明项编写测试用例以达到测试覆盖的要求。
TA-3.2.5 （K3）使用状态转换的测试设计技术，对给定的规格说明项编写测试用例以达到测试覆盖的要求。
TA-3.2.6 （K3）使用分类树的测试设计技术，对给定的规格说明项编写测试用例以达到测试覆盖的要求。
TA-3.2.7 （K3）使用结对的测试设计技术，对给定的规格说明项编写测试用例以达到测试覆盖的要求。
TA-3.2.8 （K3）使用用例测试（use case）设计技术，对给定的规格说明项编写测试用例以达到测试覆盖的要求。
TA-3.2.9 （K3）阐述在敏捷开发项目中如何利用用户故事来指导测试
TA-3.2.10 （K3）使用域分析的测试设计技术，对给定的规格说明项编写测试用例以达到测试覆盖的要求。
TA-3.2.11 （K4）分析系统或需求规格说明，以确定发现缺陷的可能类型并选择合适的基于规格说明测试技术。

3.3 基于缺陷技术

TA-3.3.1 （K2）描述基于缺陷测试技术的应用，区分它与基于规格说明测试技术在使用上有什么不同。
TA-3.3.2 （K4）采用一个好的缺陷分类判断标准，分析一个具体的缺陷分类例子是否适用于给定的场景。

3.4 基于经验技术

TA-3.4.1 （K2）解释基于经验测试技术的原则，并列出与基于规格说明和基于缺陷技术相比较的优缺点。
TA-3.4.2 （K3）对一个给定的场景使用探索性测试并阐述如何报告一个测试结果。
TA-3.4.3 （K4）对于一个给定的项目情况，为达到特定目标，确定使用哪些基于规格说明、基于缺陷或基于经验的技术。

3.1 简介

本章涉及的测试设计技术包括如下类型：

- 基于规格说明（或基于行为，或黑盒）
- 基于缺陷
- 基于经验

这些技术都是互补的，并且可以使用在任何给定的测试活动和任何正在运行的测试级别。

注意:所有这三类测试设计技术可以用于测试功能性的和非功能性的质量特性。非功能性的测试技术将在下一章节讨论。

在如下的章节中讨论的测试设计技术主要集中在如何选取最佳的测试数据（比如:等价类的划分）或如何导出测试顺序（比如：状态模型）。通常是结合各种技术来编写完整的测试用例。

3.2 基于规格说明技术

基于规格说明的技术通过分析组件或系统的测试依据，而不关注它们的内部结构，来获得测试条件并导出测试用例的一种方法。

基于规格说明的技术其共同特性有：

- 在测试设计过程中根据测试技术构建模型，例如状态转换图和决策表
- 从这些模型中能系统地导出测试条件。

有些技术还提供覆盖准则，用来度量测试设计和执行工作。完全满足覆盖准则并不意味着整个测试集的完成，而是说明该模型在当前技术下，增加测试用例对提高测试覆盖率并没有帮助。

基于规格说明的测试通常是基于系统需求文档。因为需求规格说明中规定了系统的行为，特别在功能领域，而从需求中得到的测试常常是测试系统行为的一部分。有时候有些需求并没有明确的写在文档里，它们是隐含的需求，比如说，取代一个遗留系统的功能。

有不少基于规格说明的测试技术。这些技术针对不同的软件类型和场景。本节接下来将讨论这些技术的适用范围、局限性、测试分析师可能遇到的困难、度量测试覆盖率的方法以及对应（能发现）的缺陷类型。

3.2.1 等价类划分

等价类划分（EP）用于降低测试用例数，这类测试用例有效地测试软件的输入、输出、内部值和时间相关值的处理。划分是用来创建等价类（通常称作等价类划分），等价类划分将数据分成不同的组，软件会用相同的方式处理同组内的任何数据。假定要覆盖同一等价类中的所有数据，只需从这些数据中选取一个代表值。

适用范围

这种技术可以用于任何测试级别，当程序会采用同样方式处理待测数据集合中的任何一个数据，而这些数据是相互独立的时候尤其适用。也可简单划分有效等价类的和无效等价类（数据对被测软件来说是无效的）。这种技术与边界值分析结合，测试数据扩展包含等价类的边界值之后威力更强。这种技术普遍用于新版本或新发布的冒烟测试，因为它可以快速确定基本功能是否工作。

局限/困难

如果假设不当，等价类中数据的处理方式不完全一样，采用这样的技术可能会遗留缺陷。所以小心划分等价类非常重要。例如，输入可为正数，也可为负数的时候，最好将正数、负数划为两个不同的等价类，因为计算机对正数负数的处理有可能不同。根据是否允许零，还有可能增加一个等价类。理解隐含的处理方式以更好的划分等价类对测试分析师很重要。

覆盖率
覆盖率等于有被测代表值的等价类个数除以总的识别出的等价类个数。测试同一等价类的多个代表值不会增加测试覆盖率。

缺陷类型
这种技术可以发现在处理不同数据时的功能缺陷。

3.2.2 边界值分析

边界值分析（BVA）用于测试有序等价类边界上的数据。有两种边界值分析法：二值测试法或三值测试法。二值测试法，取一个边界值（正好在边界上的值），一个刚刚超过边界的值（可能的最小增幅）。例如，如果等价类的值域是1到10，步长是0.5，上界的边界值为10和10.5，下界的边界值为1和0.5。边界定义为等价类值域的最大值和最小值。

三值测试法，取一个不超过边界、一个在边界上、一个超过边界的值。在上面的例子中，上界为9.5、10和10.5，下界为：1.5、1和0.5。采用二值测试法还是三值测试法取决于被测项的风险大小。风险高的采用三值测试法。

适用范围
该技术可用于任何级别的测试，尤其适用于有序的等价类。要求有序是因为要求在边界上和略超过于边界。例如，一组数字的等价类是有序的，所有矩形组成的等价类是无序的并且没有边界值。除了数字范围之外，边界值分析可以用于：

- 非数值变量的数值特性（如，长度）
- 循环（包括在用例中的循环）
- 存储的数据结构
- 物理对象（包括内存）
- 由时间确定的活动

局限/困难
由于该技术的精确性取决于等价类划分的精确性，它的局限和困难与等价类相仿。测试分析师必须注意有效数据和无效数据的增幅以便于精确地定义测试数据。只有有序的等价类可以做边界值分析，但这并不妨碍对某个范围内有效输入的分析。例如，测试电子表格支持的单元格数量，一个包含可以允许的最大单元格数量（边界）的等价类，而另一个等价类则从超过这个边界 1 个单元格（超过边界）开始。
.

覆盖率
覆盖率等于测试的边界条件总数除以识别的边界条件总数（二值测试法或三值测试法）。这就提供了边界测试的覆盖百分率。

缺陷类型
边界值分析总能发现边界的位移或遗漏，也可能会发现额外的边界情况。这种技术可用于发现处理边界值，尤其是小于和大于逻辑错误（即位移）的缺陷。它也可以用来寻找非功能性缺陷，例如负载限制的容差（如，系统支持10,000个并发用户）。

3.2.3 决策表

决策表是用来测试组合条件之间的相互作用。决策表提供了一个明确的方法来测试验证所有相关条件组合和验证被测软件所有可能组合的操作。决策表测试的目标是确保每个条件、关系和约束的组合被测试到。当试图测试每个可能的组合时，决策表可能会变得非常庞大。智能地将所有可能组合减少到真正

“感兴趣”的组合，这种方法被称为精简的决策表测试。使用这种技术时，为生成不同的输出可以通过移除与结果不相关的那些条件来减少组合，冗余的测试或不可能出现的那些条件组合的测试已被删除，留下的是那些会产生不同输出的组合。采用完整的决策表，还是精简的决策表通常取决于风险。[Copeland03]

适用范围

此技术一般适用于集成、系统和验收测试级别。根据代码的不同，如果组件是负责一组决策逻辑时，此技术也可用于这类组件的组件测试。当需求定义采用流程图和业务规则表时，特别适合使用这种技术。决策表也是需求定义的一种技术，有些需求规格规范说明已经采用了这种形式。即便需求不是以流程图或决策表描述，条件组合可能用叙述的方式表达。重要的是在设计决策表时应该考虑所有的条件组合包括那些没有写明但确实存在的条件组合。为了设计一个有效的决策表，测试人员必须能够获得在规范或测试准则内所有条件组合的期望结果。只有当所有参与交互的条件都考虑到了，决策表才成为一个好的测试设计工具。

局限/困难

找出所有参与交互的条件具有挑战性，尤其当需求定义不完善，或者根本不存在时。准备了一组条件但发现期待的结果还是未知，这样的例子并不少见。

覆盖率

假定所有可能的条件组合都记录在列中，并且不存在复合条件，则决策表的最小测试覆盖是每列有一个测试用例。从决策表导出测试用例时，考虑应该测试的边界条件也很重要。这些边界条件可能会导致对需要充分测试的软件增加一定数量的测试用例。充分的边界值分析、等价类划分是与决策表技术相互补充的。

典型的缺陷

典型的缺陷包括对特定条件组合不正确的处理导致意想不到的结果。在建立决策表的时候，可以发现在规格规范说明中的缺陷。最常见的缺陷是遗漏（没有有关在特定情况下应该如何反应的信息）和矛盾。测试也能发现某些条件组合没有处理或处理不当。

3.2.4 因果图

因果图可以从各种描述程序功能逻辑（规则）的资源中导出，如从用户故事、流程图等。因果图有助于获得程序逻辑结构的概图，典型地用做决策表的基础。用因果图和/或决策表捕获决策可以让我们系统地达到程序逻辑所需要的测试覆盖。

适用范围

因果图适用的测试级别和情况与决策表一样。尤其是因果图展示了条件组合引起的结果（因果关系）和排斥的结果（非），多个条件均为真引起的特定结果（与）和某一条件为真引起的特定结果（或）。这些关系通过因果图表达显然比通过描述更直观。

局限/困难

相比其它测试技术，学习因果图需要更多的时间和精力。也要求工具的支持。因果图的作者和读者必须理解因果图的特定表达符号。

覆盖率

每个可能的因到果的线包括条件的组合都被测试到才能达到最小的覆盖。因果图包括对数据和逻辑流的限制。

缺陷类型

因果图发现的缺陷类型与决策表发现的一样。另外，生成图像有助于定义测试依据所需的详细程度，从而帮助改进测试依据的详细程度和质量，并帮助测试人员识别缺失的需求。

3.2.5 状态转换测试

状态转换测试用于测试软件通过有效或无效的转换进入和退出定义的状态的能力。事件引起软件从一个状态转移到另一个状态并执行相应的行动。事件可以是满足某些影响转换路径的条件（有时也叫关条件/guard conditions 或转换关/transition guards）。例如，用有效用户名和密码组合的登录事件与用无效密码组合的登录事件引起的转换是不同的。

状态转换可以用显示所有状态间有效转换的状态转换图来追踪，也可以用显示所有有效和无效转换的状态转换表来跟踪。

适用范围

状态转换测试用于测试有定义的状态和引起状态转换的事件（比如变化的屏幕）的软件。可以用于任一测试级别。嵌入式软件，web软件和任何类型的状态转换软件都是这类测试技术的理想候选对象。控制系统，比如交通信号控制系统，也是这类测试的理想对象。

局限/困难

定义状态表或状态图时确定状态往往是最困难的部分。当软件有一个用户界面，为用户显示的各种画面经常被用来定义状态。嵌入式软件，状态可能会依赖于硬件所处的状态。

除了状态本身，状态转换测试的基本单位是单个转换，也为0-切换。简单测试所有转换，会发现一些状态转换缺陷，通过测试转换序列可以发现更多的缺陷。连续两次转换的序列被称为1-切换，连续三次转换序列是2-切换，等等。这些转换有时也被命名为N-1切换，其中N代表将走过的转换数目。例如，一个单一的转换（0-切换），描述为一个1-1切换。 [Bath08]

覆盖率

相对与其他类型的测试技术而言，这个技术的测试覆盖率分层次。可接受的最低覆盖率是到过每个状态和遍历每一个转换。 100％的转换覆盖（又称100％的0-切换覆盖或100％的逻辑分支覆盖）将保证访问了每个状态和遍历每个状态转换，除非系统设计或状态转换模型（图或表）有缺陷。根据状态和转换之间的关系，它可能需要不止一次穿越一些转换以执行一次其他转换。

“n-切换覆盖”指覆盖状态转换的数目。比如：100%的1-切换覆盖要求每个由两次成功转换组成的有效序列至少被测试了一次。这样的测试可以发现100%的0-切换覆盖遗漏的失效。

“往返覆盖”在转换序列形成循环的情况下适用。实现100％往返覆盖意味着从任何状态出发又回到原来相同状态的所有循环被测试到。这必须测试循环中包含的所有状态。

尝试包括所有无效的转换可以达到一个更高的覆盖率。覆盖要求和状态转换测试的覆盖集必须确定是否包括无效的转换。

缺陷类型

典型的缺陷，包括在目前状态不正确的处理，这可能是以前的状态处理不正确的或不支持的转换、没有出口的状态、所需的状态与转换不存在等情况所引起的结果。在创建状态机模型时可能发现在规格说明文档中的缺陷。此种技术可以发现的缺陷中最常见的类型是遗漏（没有提供在某些情况下应该发生什么的信息）和矛盾。

3.2.6 组合测试技术

当软件测试使用多个参数，每个参数有多个值，值的组合太多，以至于要完全测试这些组合在时间上根本不可行，这时可以采用组合测试技术。这里的参数必须是独立的和兼容的，任何参数的任何选项可以与任何其他参数的任何选项组合。如果某些选项是不相容的，则分类树允许某些组合被排除在外。这并不假设组合中的因素互不影响，他们可以影响，但其影响是可接受的。

组合测试提供了一种手段，确定了适合这些组合的子集实现了预定的覆盖水平。测试分析师可以选择在组合中的值，可以是单个值、成对值、三个值或以上[Copeland03]。有许多工具可用来帮助完成这一任

务的测试分析（详见www.pairwise.org）。这些工具需要列出参数和他们的值（成对的和正交阵列）或以图形的形式表示（分类树）[Grochtmann94]。成对测试是用于测试双变量组合的方法。正交阵列预定义的精确计算的数据表，测试分析师可以选取测试值取代数组中的变量，生成组合集合，以实现定义的测试覆盖水平[Koomen06]。分类树工具可以让测试分析师决定组合的大小（即两个值的组合，三个值组合等）。

适用范围
至少有两种不同的测试情形体现了参数组合太多的问题。有些测试用例有许多参数，每个参数有一系列可取值。比如屏幕有几个输入字段，这时，测试用例的输入数据就是参数值的组合。还有一些系统的配置是多维的，因此形成了很大的配置空间。在这两种情况下，组合测试可以识别出一个大小适当的子集。

如果参数可以取很多值，应该先通过等价类分析或其他的选择机制来减少每个参数的测试值，然后再使用组合测试来降低组合的测试值子集。

这些技术通常应用在集成测试级别、系统测试级别和系统集成测试级别。

局限/困难
这些技术的主要限制是基于这样的假设，即几个测试结果代表了所有测试结果和预期。如果某些参数之间有一个意想不到的交互，假如没有测试到这样的组合，就可能遗漏某些缺陷。这些技术很难向一个非技术性人员解释，因为他们可能不明白这些用于减少测试工作量的逻辑。

有时很难确定参数和它们各自的值。手动寻找一个组合的最小集合并满足一定程度的覆盖很困难。通常会使用工具去找组合的最小集合。有些工具支持在最终选择时强迫加入或排除一些（分）组合。测试分析师根据领域知识或产品使用信息可用该功能强调或不强调某些因素。

覆盖率
覆盖分多层。覆盖率最低的被称为 1-维（1-wise）或单覆盖，它要求每个参数的每个值至少在所选择的组合中出现一次。下一个更高层的覆盖被称为 2-维（2-wise）或成对覆盖，它要求任意两个参数的每一对值至少包含在一个组合中。以此类推到 n-维覆盖，这就要求选定组合中包括任何 n 参数值的所有子组合。n 越大，要达到 100％的覆盖率的组合也就越多。这种技术的最小覆盖是对工具生成的每个组合编一个测试用例。.

缺陷类型
这类测试发现的常见缺陷为：与多个参数值组合相关的缺陷。

3.2.7 用例测试

用例测试模拟系统行为提供事务性的、基于场景的测试。用例定义了参与者和系统之间为达到某种目的进行的互动。参与者可以是用户也可以是外部系统。

适用范围
用例测试一般被用于系统测试和验收测试级别。视集成水平高低也可以被用于集成测试，视组件的行为甚至也可以被用于组件测试。用例测试常作为性能测试的基础，因为它更接近系统的真实使用。用例中描述的场景有可能会被分配给虚拟用户来生成系统实际的负载。

局限/困难
用例必须与真实使用相吻合才能保证测试的有效性。这类信息应该来自用户或用户代表。用例如果不精确的反映实际用户的行为就降低了用例的价值。精确定义不同的替代路径（流）对测试的覆盖率很重要。用例可以作为指导书，但不是完整的测试定义，因为它也许不能提供所有需求的清晰定义。在其它建模过程中用例也是非常用用的，比如从用例描述到画出流程图、从用例描述到改善测试的正确性以及对用例本身的验证。

覆盖率

用例的最小覆盖为：每个主路径（正向）一个测试用例，每个替代路径（流）一个测试用例，替代路径包括例外和失效路径，替代路径有时也表示为主路径的扩展。覆盖率百分比为测试的路径除以主路径和替代路径的和。

缺陷类型
缺陷包括定义的场景处理不当、错过的替代路径处理、现有条件下处理不正确或处理不恰当或不正确的错误报告。

3.2.8 用户故事测试

在一些敏捷方法中，如Scrum，需求是以用户故事的形式呈现，主要描述在一个迭代中可以设计、开发、测试和演示的小的功能单元[Cohn04]。这些用户故事包括要实现的功能描述、非功能性标准以及用户故事完成所必须满足的验收标准。

适用范围
用户故事主要用于敏捷和类似迭代和增量的环境。它们用于功能测试和非功能测试。用于所有级别的测试，期望开发人员在交付代码给下一级测试（例如，集成测试，性能测试）之前向测试团队成员演示为用户故事实现的功能。

局限/困难
因为故事是功能的小增量，有可能要求有驱动程序和桩，以实现交付的功能件测试。这通常需要编程能力和使用有助于测试的工具，比如API测试工具。创建驱动程序和桩通常是开发人员的责任，虽然技术测试分析师也可以参与编码，并利用API测试工具。如果像大部分敏捷项目那样使用持续集成模型，对驱动和桩的需求降至最低。

覆盖率
用户故事的最低覆盖为验证每个定义的验收标准都已满足。

缺陷类型
通常是该软件没有提供指定功能的功能性缺陷。也可能在新故事与已经存在的功能集成时出现问题。因为故事可以独立开发，所以可能会发现性能、接口和错误处理的问题。对测试分析师而言执行个体的功能测试和随时进行新故事发布的集成测试是非常重要的。

3.2.9 域分析

域是一个定义的值的集。集可以是一个单变量的值域范围（一维域，例如，“在24岁以上和66岁以下的男子”），或是相互作用的几个变量的值域（多维域，如范围“男性年龄在24以上66岁以下，体重在69公斤以上，90公斤以下”）。每个多维域的测试用例对于每个参与的变量都必须定义相应的值。

一维域的分析通常使用等价类划分和边界值分析。一旦定义了区域，测试分析师从每个区域中选取一个界内值（IN），一个界外值（OUT），一个边界值（ON），一个刚好出边界的值（OFF）。通过确定这些值，也就是根据这些边界条件测试每个区域 [Black07]。

在多维域的情况下，用等价类和边界值等方法生成的测试用例数会随着变量数增长呈指数上升，而采用域分析方法测试用例数呈线性增长。此外，这种形式化的方法结合了缺陷理论（失效模式），这正是等价类划分和边界值分析所缺乏的，所以即便采用了规模较小的测试集，仍能发现多维域的缺陷，而较大的启发式测试集反而可能会遗漏这些缺陷。当处理多维域，测试模型可以构建一个决策表（或“域矩阵”）。如果是超过3维的多维域，确定测试用例输入值可能需要计算机支持。

适用范围
域分析结合决策表、等价类和边界值可以生成较小的测试集，该集覆盖了重要的和容易失效的区域。这种技术常常用在潜在互动变量太多，决策表变得很庞大的时候。任何测试级别都可以使用域分析，但大部分时候用在集成测试和系统测试级别。

局限/困难
做完整的域分析需要对软件有非常深入的理解，以确定不同域和域之间可能的相互作用。如果一个域无法识别或确认，则测试会明显不足，但由于OFF和OUT变量可能会降落在未被发现的域，使得这些域很容易被发现。当和开发人员一起定义测试范围时，域分析是一个非常有用的技术。

覆盖
域分析中的最小覆盖是对应于每个域的IN、ON、OFF和OUT值都有一个测试用例。当值出现重叠时（例如，一个域的输出值是在另一个域的输入值），没有必要重复测试。正因为如此，平均到每个域的具体测试用例数往往是小于四的。

典型缺陷
发现的缺陷包括域内的功能性问题、边界值处理，变量交互问题和错误处理（特别是当值不在有效区域内时）。

3.2.10 不同技术的组合
有时可以组合不同的技术创建测试用例。例如，对决策表中所确定的条件还可能做等价类划分，以发现可能满足其中一个条件的多种方式。测试用例不仅覆盖每个条件组合，当条件分成等价类后，可以生成额外的测试用例覆盖等价类。当选择特定的技术时，测试分析师应考虑技术的适用范围，限制和困难，测试覆盖和缺陷探测目标。有些情况下可能不存在单一的“最佳”技术。假定有足够的时间和正确运用技术的技能，组合各种技术后，往往能提供最完整的覆盖。

3.3 基于缺陷的技术

3.3.1 基于缺陷的技术
基于缺陷的测试设计技术是以发现的缺陷类型为基础，根据已知的缺陷类型来系统地获取测试用例的技术。与基于规格说明的测试，即从规格说明导出测试用例的技术不同，基于缺陷的测试根据缺陷分类（也可以是分类列表）导出测试用例，而这个分类也许和要测试的软件本身完全无关。分类可以包括缺陷类型、根源、失效症状和其他缺陷相关数据等清单。基于缺陷的测试也可以把识别的风险和风险场景清单作为测试的基础。该测试技术允许测试人员针对特定的缺陷类型，也可以通过对已知和常见的特定类型缺陷的分类系统地测试。测试分析师运用分类数据决定测试目标，找出特定类型的缺陷。通过这些信息，测试分析师生成能触发失效的测试用例和测试条件，如果缺陷存在的话。

适用范围
基于缺陷的测试可以用于任何测试级别，通常用于系统测试居多。对不同类型的软件有标准的缺陷分类。这种与产品类型无关的测试有助于利用工业标准知识获得特定的测试。坚持使用工业标准分类，就可以对所有项目，甚至是在整个组织内对缺陷有关的度量进行跟踪。

限制/困难
存在多个缺陷分类，但也许只是关注某类测试，比如易用性。重要的是要采用已有的适合被测软件的分类法。新的创新软件有可能没有现成的分类法。有些组织对可能的或常见的缺陷有自己的分类法。不管采用什么分类法，重要的是在开始测试之前定义期望的覆盖。

覆盖
基于缺陷的技术也提供覆盖准则，以决定测试是否可以结束。作为比较特殊的技术，基于缺陷技术相比基于规格说明的技术，其覆盖准则的系统性要弱一些。它先给出覆盖通则，而具体覆盖的界限如何定才有意义则要酌情而定。覆盖准则并不意味整个测试集已完成，仅仅指所考虑的缺陷已不需要做基于此技术的任何测试。

缺陷类型

发现的缺陷类型通常取决于所使用的分类法。如果采用用户界面类，大多数发现的缺陷可能与用户界面相关，但作为副产品在具体测试中可以发现其它缺陷。

3.3.2 缺陷分类法

缺陷分类法是分类的缺陷类型列表。这些列表可以很通用，作为高层次的指南，也可以是非常具体的。例如，一个用户界面缺陷的分类可能包含通用项如功能、错误处理、图形显示和性能。详细分类包括了所有可能的用户界面对象（特别是图形用户界面）的列表，并可以指定这些对象有哪些不当处理：

- 文本字段
  - 不接受有效数据
  - 接受无效数据
  - 未验证输入长度
  - 未检出特殊字符
  - 用户的错误消息不够详细
  - 用户无法纠正错误数据
  - 规则不适用
- 日期字段
  - 不接受有效日期
  - 没拒绝无效日期
  - 未验证日期范围
  - 精密数据处理不正确（例如，hh:mm:ss）
  - 用户无法纠正错误数据
  - 规则不适用（例如，结束日期必须大于起始日期）

有许多缺陷分类，从可以购买到的正式分类到那些由各种组织为特定目的而设计的分类。内部开发的缺陷分类，也可用于针对特定组织内的常见缺陷。当创建一个新的缺陷分类，或为客户定制一个缺陷分类时，首先确定目标或分类的目的很重要。例如，我们的目标可能是识别已在生产系统中发现的用户界面问题或识别输入字段处理的相关问题。

创建一个分类的步骤：
1. 创建目标，并确定所需的详细级别
2. 选择一个给定的分类作为基础
3. 定义在组织内部和/或从外部实践中经历过的值和共同的缺陷

分类越详细，开发和维护它的时间也就越多，但它会带来测试结果的高复用性。详细分类可以是有冗余的，这让测试团队能够分解测试但不损失信息或覆盖。

一旦选择了适当的分类，就可以用来生成测试条件和测试用例。以风险为基础的分类，可以帮助测试把重点放在一个特定的风险领域。分类也可用于非功能性测试，例如，易用性、性能等。分类列表可以从各种刊物、IEEE、互联网中获得。

3.4 基于经验的测试技术

基于经验的测试利用测试人员的技能和直觉，以及他在类似应用程序或技术方面的经验。这些测试能有效地发现缺陷，但不像其他技术那样适用于实现特定的测试覆盖率或生成可重复使用的测试程序。在系统文件质量很差，测试时间被严格限制或测试团队对被测系统的了解非常资深专业的情况下，基于经验

的测试相对于结构化的方法而言是一个很好的选择。基于经验的测试可能不适用于需要详细测试文档、重复性高或能够准确评估测试覆盖率的测试。

当使用动态和启发式方法时，测试人员通常使用基于经验的测试，而且测试方法不是预先计划的，更多是对事件的反应。此外，执行和评价几乎是同步进行的并发任务。一些结构化的测试方法相对于基于经验的方法而言是不完全动态的，即创建测试不是在测试人员执行测试的同时进行。

注意：虽然针对此技术也有些覆盖的想法在讨论，但基于经验的技术不具有正式的覆盖准则。

3.4.1 错误推测法

当使用错误推测法时，测试分析师运用经验来猜测代码在设计和开发中可能出现的潜在错误。当识别了预期的错误以后，测试分析师选出最好的方法来诱发缺陷。例如，如果测试分析师预计该软件在无效的密码输入时将出现失效，测试将被设计成输入各种不同的值来验证密码字段错误是否存在并形成导致测试运行时产生失效的缺陷。

除了作为测试技术，错误推测法也可以用在风险分析上，以确定潜在的失效模式。 [Myers79]

适用范围
错误推测法主要用于集成和系统测试，但可以用于任何测试级别。这种技术经常与其他技术一起使用以扩大现有测试用例的范围。错误推测法也可以在新版本发布后，在执行更严格的测试和脚本测试之前使用，用以有效地检测常见的失误和错误。清单和分类有助于指导测试。

局限/困难
覆盖率难以评估，很大程度上依赖于测试分析师的能力和经验。这种方法最好由经验丰富、熟悉被测试代码常见缺陷类型的测试人员来使用。错误推测法很常用，但经常没有记录，因此可能不如其他形式的测试有利于重现。

覆盖率
当使用分类法时，覆盖由适当的数据故障和缺陷类型来决定。没有分类的话，覆盖受限于测试人员的经验知识以及可用的测试时间。这种技术的错误发现量将取决于测试人员能否把握好问题区域。

缺陷类型
典型的缺陷通常在测试分析师定义的特定分类或“猜测”中，而这些缺陷用基于规格说明的测试方法可能不会被发现。

3.4.2 基于检查表的测试

当采用基于检查表的检测技术时，经验丰富的测试分析师使用一个高层次的、广义的检查清单，罗列针对产品验证需要注意的、检查的或提醒的事项，或验证产品的规则或标准。这些清单的建立基于一系列标准、经验和其他因素。用户界面标准检查表作为测试应用程序的基础，是基于检查表测试的一个例子。

适用范围
一个熟悉被测软件或熟悉检查表覆盖的领域并具有经验丰富的测试团队在项目中采用基于检查表的测试是非常有效的（例如，测试分析师有可能熟悉用户界面测试，但不熟悉特定的被测软件，同样可以成功的运用用户界面检查表）。因为检查表是高层次的，而且往往缺乏通用测试用例和测试过程中的具体步骤，测试人员的知识可以用来填补空白。从检查表中删除具体步骤，不仅维护成本低，而且可以应用到多个类似的版本。检查表可用于任何级别的测试。也可用于回归测试和冒烟测试。

局限/困难
高级别检查表的特性可能会影响测试结果的重复性。不同测试人员对检查表的理解有可能不一样，从而完成检查事项的方式也不一样。即使使用相同的检查表，也可能会导致不同的结果。这可能会导致覆盖面更广，但重复性有时会被牺牲。检查表也可能导致对达到的覆盖水平过于自信，然而实际上基于检查

表的测试很大程度上有赖于测试人员的判断。检查表可以从更详细的测试用例中导出，往往随着时间的推移越来越大。检查表需要维护以确保该清单覆盖了被测试软件的重要方面。

覆盖率
覆盖率与检查表一样，但因为检查表比较抽象，会因执行检查表的测试分析师而出现不同的结果。

缺陷类型
用这种方法发现的典型缺陷包括通过数据的改变、顺序内步骤的变化或工作流程变更而引起的失效。由于在测试过程中允许组合新的测试数据和流程，使用检查表可以帮助测试保持新鲜度。

3.4.3 探索性测试

探索性测试的特点是测试人员边学习有关的产品和它的缺陷，边计划、设计和执行测试，并报告结果。测试人员在执行过程中动态调整测试目标并准备少量文档。 [Whittaker09]

适用范围
良好的探索性测试是有计划的、互动的和创造性的。这种技术只需要少量的有关被测系统的文档，通常使用在没有文档或文档不适用于其他测试技术的时候。探索性测试常作为其他测试技术的补充或作为开发额外测试用例的基础。

局限/困难
探索性测试很难管理和安排时间表。覆盖率是零星的，可重复性差。在测试会话中使用章程指定覆盖区域，使用时间盒（time-boxing）规定测试所允许的时间，是一个用于管理探索性测试的方法。在测试会话或一系列会话结束时，测试经理可能会举行一次汇报会，收集测试结果，并决定下次测试会话的章程。汇报会不适合大型测试团队或大型项目。

另一个困难是在测试管理系统中准确地跟踪探索性会话，有时创建测试用例其实就是探索性会话，这允许对分配给探索性测试的时间和计划的测试覆盖与其它测试工作量一起进行跟踪。

由于探索性测试的可重复性差，这也导致发生问题时，需要记录步骤来重现失效。有些组织使用具有捕获/回放功能的自动化测试工具，记录探索性测试所采取的步骤。这提供了探索会话（或任何基于经验的测试会话）所有活动的完整记录。通过细节的挖掘，找到失效的真正原因可能很乏味，但至少有一个涉及所有步骤的记录。

覆盖率
可以创建章程指定任务、目标和交付，然后计划试探性会话以实现这些目标。章程也可以确定测试的重点，什么是测试会话的范围，哪些资源应致力于完成计划的测试。会话可以用来把重点放在特定的缺陷类型和无需形式化脚本就可以测试的其他潜在问题区域。

缺陷类型
探索性测试中发现的典型缺陷是脚本化功能测试所遗漏的场景问题，功能边界之间的问题和与工作流程相关的问题。探索性测试有时也能发现性能和安全问题。

3.4.4 运用最佳技术

基于缺陷和经验的技术需要将有关缺陷知识和其他测试经验应用到目标测试中，以提高缺陷发现率。形式从没有预先计划的“快速测试”到有计划的测试会话到脚本化的测试。他们几乎总是有用的，但在下列情况下有特别的价值：

- 无规格说明
- 被测系统文档质量很差
- 没有足够的时间允许设计和创建测试
- 测试人员有丰富的领域和/或专业知识和经验

- 多样化脚本测试以最大限度地提高测试覆盖率
- 对操作失效进行分析

基于缺陷和经验的测试与基于规格说明的技术结合使用非常有用，因为他们通过其它测试技术可以弥补测试覆盖中的弱点。正如基于规格说明的技术，没有哪一个技术完美到可以适用所有情况。重要的是测试分析师要了解每种技术的优缺点，并能考虑项目类型、进度、获取的信息、测试人员的技能和其他可以影响选择的因素，选择最好的技术或技术组合。

4. 测试软件质量特性- 120 分钟

关键字

可达性测试（无障碍辅助性测试）（accessibility testing）、准确性测试（accuracy testing）、吸引性（attractiveness）、启发式评估（heuristic evaluation）、互操作性测试（interoperability testing）、易学性（learnability）、易操作性（operability）、适合性测试（suitability testing）、软件易用性度量调查表（SUMI-Software Usability Measurement Inventory）、易理解性（understandability）、易用性测试（usability testing）、网站分析和度量调查表（WAMMI-Website Analysis and Measurement Inventory）

测试软件质量特性的学习目标

4.2 业务领域测试的质量特性

TA-4.2.1 （K2）举例解释哪些测试技术适用于测试准确性、适合性、互操作性和兼容性特性。
TA-4.2.2 （K2）定义准确性、适合性和互操作性特性的典型缺陷。
TA-4.2.3 （K2）在软件生命周期中定义何时应当进行准确性、适合性和互操作性特性的测试。
TA-4.2.4 （K4）对于给定的项目背景，概述能够同时验证和确认实现易用性需求并满足用户期望的合适方法。

4.1 简介

上一章中对测试人员可用的具体测试技术进行了描述，本章考虑如何将这些技术运用于评估软件应用或软件系统的主要质量特性。

本大纲对可能被测试分析师评估的质量特性进行讨论，那些被技术测试分析师评估的质量特性将被包含在高级技术测试分析师大纲中。本大纲使用了 ISO9126 以及其他诸如 ISO25000 [ISO25000]系列（已经取代了 ISO9126）标准中提供的有关产品质量特性的描述来作为指导。ISO 质量特性被分成多种产品质量特性（属性），每一种质量特性可能有多种子特性（子属性）。下表列出了这些特性和子特性，同时指出了测试分析师和技术测试分析师大纲中分别覆盖到的特性/子特性：

特性	子特性	测试分析师	技术测试分析师
功能性	准确性，适合性，互操作性，依从性	X	
	安全性		X
可靠性	成熟度（健壮性），容错性，易恢复性，依从性		X
易用性	易理解性，易学性，易操作性，吸引性，依从性	X	
效率	性能（时间行为），资源利用性，依从性		X
维护性	易分析性，易改变性，稳定性，易测试性，依从性		X
可移植性	适应性，易安装性，共存性，易替换性，依从性		X

测试分析师应当专注于软件质量特性中的功能性和易用性。可达性测试（无障碍辅助性测试）也应当由测试分析师执行。虽然无障碍辅助性没有被列为一个子特性，但是他经常也被认为是易用性测试的一部分。其他的质量特性通常被认为由技术测试分析师负责测试。虽然在不同的组织中工作的安排可能会有所变化，但是在 ISTQB®大纲中我们遵照以上描述。

依从性被显示为每一个质量特性的子特性。在特定的安全关键或者受控环境中，每种质量特性都可能必须遵从特定的标准和规则（例如：功能依从性可能指功能性需要遵从特定的标准。比如使用一种特殊的通讯协议从而可以从芯片发送/接收数据）。因为这些标准会根据行业的不同有很大的变化，本大纲不会在此对他们进行深入的讨论。如果测试分析师正在一个受到依从性需求影响的环境中工作，理解这些依从性需求并且保证测试以及测试文档都能满足依从性需求是很重要的。

对于在本章节中提到的所有质量特性和子特性，都必须进行典型风险识别从而生成合适的测试策略并使其文档化。对质量特性进行测试需要特别注意生命周期时间、所需工具、软件、文档可用性和专用技术。如果没有计划针对每种特性的处理策略，测试者可能无法进行充分的计划、提升和完善以及在进度表中为执行测试分配时间。对一些测试，比如易用性测试，可能需要分配专门的人力资源、大量的计划、专用的实验室、特殊的工具和专门的测试技巧，多数时候还需要花费大量的时间。在某些情况下，易用性测试可由易用性测试或用户体验专家组成的独立的易用性测试团队执行。

质量特性和子特性测试必须被集成到总测试进度表中并为其分配足够的资源。下文将讨论每种特性的特定的需求、针对的具体问题并可能产生于软件开发生命周期的不同时间。

虽然测试分析师可能不用负责测试那些需要运用更多技术方法才能进行测试的质量特性，但是了解其他的质量特性并且理解他们和测试分析师所关注的质量特性在测试中的重叠部分仍然是十分重要的。例如一个在性能测试中失败的产品，如果其反应速度太慢而使得用户无法有效使用，那么该产品也有很大概率会在易用性测试中失败。类似的，和某些组件有互操作性问题的产品很可能无法进行可移植性测试，因为当环境变化后可能会掩盖一些更基本的问题。

4.2 业务领域测试的质量特性

测试分析师主要关注功能测试，而功能测试主要关注产品能做“什么”。功能测试的测试基础通常是需求文档或者规格说明文档、特定领域的专业知识或者隐含的需求。功能测试可能会受到软件开发生命周期的影响，也会根据不同的测试级别有所变化。例如，在集成测试中进行的功能测试将会对一组接口模块进行功能性测试，而这组模块实现某个单一指定功能。而在系统测试级别中，功能测试包括应用程序整体的功能性测试。在综合系统中，功能测试主要集中于贯穿集成系统的端到端测试。在敏捷开发环境中，功能测试的范围通常被限定为特定迭代或者冲刺中所完成的功能，但是一个迭代的回归测试可能会覆盖所有已发布功能。

在功能测试中使用了多种测试技术（参见第三章），功能测试可以由专职的测试人员或领域专家执行，也可以由开发人员执行（通常是在组件测试级别）。

除了本章节覆盖到的功能测试以外，测试分析师负责范围内还有两个被认为是属于非功能测试范畴的质量特性（专注于产品所提交的功能工作的“怎样”）。这两个非功能属性是易用性和可达性（无障碍辅助性）。

本章节涉及以下质量特性：

- 功能质量子特性
  - 准确性
  - 适合性
  - 互操作性
- 非功能质量特性
  - 易用性
  - 可达性（无障碍辅助性）

4.2.1 准确性测试

功能准确性包括测试应用程序与具体或者隐含需求的一致性，也可能包括计算的准确性。准确性测试中使用了许多在第三章中描述的测试技术，并经常使用规格说明或者旧系统（先前的系统）作为测试准则。准确性测试的目标是找出那些对数据或者状况的错误处理。准确性测试可以在软件生命周期中的任何阶段执行。

4.2.2 适合性测试

适合性测试涉及对一组功能是否与其预定的特定任务相吻合进行评估和确认。适合性测试可以基于用例。适合性测试通常在系统测试阶段进行，但是也可能在集成测试阶段的后期进行。在适合性测试中发现问题表示系统将无法按照用户认为可接受的方式来满足用户需求。

4.2.3 互操作性测试

互操作性测试对两个或多个系统或组件之间的信息可交互程度以及交互后的信息可使用程度进行测试。互操作性测试必须覆盖所有计划的目标环境（包括不同的硬件、软件、中间件和操作系统等）以保证数据交换能够正常工作。在实际情况中，可能只用比较少的可行的环境，在这类情况下可以有选择性的挑选具有代表性的一组环境进行互操作性测试。具体实施互操作性测试首先需要识别和配置计划目标环境组合，使其对测试团队可用。使用选定的那些可访问环境中各种数据交换点的功能性测试用例来对这些环境进行测试。

互操作性和软件系统相互之间交互的差异程度有关。具有良好互操作性的软件无需大量变更就可以和许多其他的系统进行交互。需要进行变更的数量以及实现这些变更所需的工作量可以被作为互操作性的度量。

例如，软件互操作性测试可能集中于以下设计特性：

- 使用业界通讯标准，比如 XML
- 自动检测系统的交互通讯需求的能力并作出相应的调整

互操作性测试对于开发商业现货（COTS）软件和工具以及综合系统的组织可能特别重要。

在组件集成和系统测试阶段执行互操作性测试时，测试重点集中在系统和它的环境之间的交互。在系统集成级别，这类测试被用来确定开发的系统与其它系统的交互是否良好和充分。因为系统可能在多个级别上进行交互，测试分析师必须理解这些交互并且能够创造出各种交互发生的条件。例如，如果两个系统将会交换数据，测试分析师必须能够生成必要的执行数据交换所需的数据和事务。非常重要的是要记住需求文档中可能不会清晰的描述所有的交互，许多交互将只会在系统架构和设计文档中被定义。测试分析师必须具备检查这些文档的能力，并做好准备通过这些文档找出系统间以及系统和它的环境间的数据交换点，从而保证测试能够覆盖所有这些数据交换点。在互操作性测试中决策表、状态转换图、用例测试和组合测试技术都是适用的。典型的缺陷包括进行交互的组件之间错误的数据交换。

4.2.4 易用性测试

要理解“为什么用户在使用系统时会有困难？”是很重要的。要理解这一点首先必须意识到“用户”可以指各种不同类型的人，可以是 IT 专家、孩子、残疾人。

一些国立学院（例如英国皇家国立盲人学院）建议网页应该被设计成易于伤残人士、盲人、弱视者、行动障碍者、失聪者和认知能力缺失者访问的方式。检查应用程序和网站对以上用户的易用性也可以提高其对其余所有用户的易用性。后文中会进一步对可达性（无障碍辅助性）进行讨论。

易用性测试是测试用户使用系统或学习使用系统以在指定环境中达到指定目标的容易程度。易用性测试测度以下内容：

- 有效性 – 软件产品使用户可以在指定使用环境中精确和完整的实现指定目标的能力
- 效率 – 软件产品使用户可以根据指定使用环境中投入一定的资源获得相应有效性的能力
- 满意度 – 软件产品在特指定使用环境满足用户的能力

需要测度的属性包括：

- 易理解性 – 影响用户认识逻辑概念和其应用所需投入的软件属性
- 易学性 – 影响用户学习应用所需投入的软件属性
- 易操作性 – 影响用户切实有效地进行工作所需投入的软件属性
- 吸引性 – 软件吸引用户的能力

易用性测试通常分为两个步骤进行：

- 形成性易用性测试 – 在设计和原型阶段中迭代的进行，通过识别设计中的易用性缺陷来帮助指导（或者“形成”）设计
- 总结性易用性测试 – 在实现后进行测试，对易用性进行测度并且对已完成组件或系统中的易用性问题进行识别

易用性测试者的技能应当包括以下方面的专业技能或知识：

- 社会学
- 心理学
- 需要遵循的国内标准（包括可达性/无障碍辅助性标准）
- 人体工程学

4.2.4.1 进行易用性测试

应当尽可能在接近系统实际使用条件下对系统的实际实现进行确认。这可能包括建立一个具有摄像机、模拟办公室、评审小组和用户等在内的易用性实验室，开发人员对真实用户使用实际系统的效果进行观察。正式的易用性测试通常需要有一定数量的“用户”（可以是真实用户或者用户代表）参与，让这些

用户按照设定的脚本或者根据指示进行操作。其它自由形式的测试允许用户对软件进行尝试，以便观察者发现用户是和很容易还是非常困难地弄清楚如何完成自己的任务。

许多易用性测试可以由测试分析师在其他测试中执行，例如功能系统测试。为了找到一致的、能在软件生命周期的所有阶段发现和报告易用性问题的方法，易用性指导原则很有帮助。如果没有易用性指导原则，很难确定什么是“无法接受的”易用性问题。例如，用户必须点击 10 次鼠标才能登录一个应用程序是否合理？如果没有具体的指导原则，当开发人员想要用该软件是在按“设计”在工作为理由关闭缺陷时，软件测试分析师会面临困境。在需求中定义可验证的易用性规格说明很重要，同时拥有可应用于所有相似项目的一套易用性指导原则也很重要。指导原则应当包括以下内容：说明书的可达性（无障碍辅助性）、明确的提示、完成一项活动的点击数、错误信息、处理提示（某些类型的提示告诉用户系统正在进行处理，因此无法同时接受更多的输入）、屏幕布局的指导原则、颜色和声音的使用以及其它影响用户体验的因素。

4.2.4.2 易用性测试规格说明

易用性测试的主要技术包括：

- 审查、评估或评审
- 与原型的动态交互
- 验证和确认实际实现
- 调查表和调查问卷

审查、评估或评审

从易用性的角度对需求规格说明和设计进行审查或者评审可以提高用户的参与度，且是一种低成本的、能在早期发现问题的手段。启发式评估（系统化对用户接口设计进行易用性审查）可以被用来发现设计中的易用性问题，因此启发式评估可以被作为迭代设计过程中的一部分。这包括一小部分评估者对接口进行检查并且判断其与公认的易用性原则（“启发式”）的一致性。例如，对于某个特定屏幕上提供的功能，截屏通常会比叙述性描述更容易解释和理解。可视化对于适当的文档易用性评审是十分重要的。

与原型的动态交互

当原型被建立以后，测试分析师应当在原型上工作并且帮助开发人员将用户反馈加入到设计中，从而让原型不断演化。这样，原型可以被进一步提升并且用户可以对最终产品的外观和感觉获得更真实的看法。

验证和确认实际实现

如果需求中定义了软件易用性特性（例如，为了达到特定目标鼠标的点击次数），应当创建测试用例来验证软件实现中已经包括了这些易用性特性。

对于实际实现的确认，功能系统测试可以被开发成为易用性测试场景。这些测试场景测度特定的易用性特性，例如易学性或易操作性，而不是功能性结果。

易用性测试场景可以被开发成具体的测试语法和语义。语法是指接口的结构或语法（例如，在输入字段中可以输入什么），而语义是指接口的含义和目的（例如，提供给用户的合理且有意义的系统信息和输出）。

黑盒测试技术（例如，在章节 3.2 中列出的技术），特别是可以用 UML （统一建模语言）或纯文本定义的用例有时会被应用于易用性测试。

易用性测试场景也需要包括用户使用说明，测试前和测试后的访谈（目的是为了给予指导和接收反馈）的时间分配以及实施会话的约定协议。协议中包括了对如何进行测试、时间、笔录以及会话日志和访谈记录、所使用的问卷调查方法等的描述。

调查表和调查问卷

调查表和调查问卷技术可被应用于收集有关用户在系统中行为的意见和反馈。标准化和公开的调查表允许用以前的易用性度量数据库进行基准化分析，例如 SUMI（软件易用性度量调查表）和 WAMMI（网

站分析和度量调查表）。此外，由于 SUMI 提供了具体的易用性测量标准，这可以提供一套完成/验收标准。

4.2.5 可达性 / 无障碍辅助性测试

为包括残疾人在内，有特殊使用需求或限制的用户考虑软件可达性 / 无障碍辅助性是十分重要的。可达性 /无障碍辅助性测试应当考虑相关的标准，例如网页内容可达性 / 无障碍辅助性指南，以及法律法规，例如残疾歧视行为（英国，澳大利亚）和条款 508（美国）。可达性 / 无障碍辅助性同易用性一样，必须在设计阶段就被考虑。可达性 / 无障碍辅助性测试经常开始于集成测试级别，贯穿整个系统测试级别直至验收测试级别。缺陷通常在软件无法满足指定规则或标准时可被发现。

5. 评审-165 分钟

关键词

无

评审的学习目标

5.1 简介

TA-5.1.1 （K2）解释为何评审准备对测试分析师来说是重要的。

5.2 在评审中使用检查表

TA-5.2.1 （K4）依据大纲中提供的检查表（checklist）信息，对一个用例或用户界面进行分析并发现问题。

TA-5.2.2 （K4）依据大纲中提供的检查表（checklist）信息，对一个需求说明或用户故事进行分析并发现问题

5.1 简介

一个成功的评审过程需要计划、参与以及追踪。测试分析师需要积极参与到评审过程中去并提供独到见解。他们需要接受正式的评审培训，以更好地理解他们在评审过程中独立的角色。所有的评审参与者都需要认可良好的评审可能到来的收益。如果操作得当，评审可以成为提升交付质量的诸多因素中性价比最高的，贡献最大的一个。

无论进行何种类型的评审，测试分析师都必须安排足够的时间进行准备。这里的时间主要用来熟悉评审材料、检查引用材料之间的一致性、以及检查待评材料中可能的缺失。如果没有足够的准备时间，在实际评审中，测试分析师就很可能花费大量的时间在文档编辑上，而不能有效地参与到评审中去，从而导致评审团队的时间不能最大化被利用，也无法提出最佳的反馈信息。一个好的评审包含对文案的理解，发现缺失内容，以及评估文案产品描述与开发中/已开发完成产品之间的一致性。例如，当评审一个集成测试级别的测试计划时，测试分析师需要考虑都有哪些模块会在此次测试中被集成。需要具备哪些条件才能开始集成？有哪些必须记录在案的模块之间的依赖？是否有测试数据用于测试模块之间的集成点？需要指出的是，评审不是独立于被评材料的过程，评审必须考虑被评审部分与系统中其它部分的交互关系。

评审对象的作者很容易会感觉到是在被批评。测试分析师应当确保用“与作者一起尽量构建最好的作品”作为出发点来提出评审意见。通过使用一些方法，评审建议就可以更有建设性，并成为针对被评审材料的意见而非针对作者本人的意见。例如，如果一个描述是含糊不清的，那我们最好说“对这部分，我不太明白如何执行测试以确认是否正确的完成了需求，你能帮我增进理解吗？”而不是说：“这个需求太含糊不清了，根本没人能帮我搞得清楚！”测试分析师在评审中的职责是保证评审对象中的信息可以足够支撑测试活动。如果缺乏这些有效的信息，或者信息不够清晰，或者表述的信息不够具体，那我们实际上就可能发现了一个需要作者做出修改的缺陷。通过使用积极正面的方法而非批评的方式，我们的意见会更容易被接受，评审会议将会更富有成效。

5.2 在评审中使用检查表

在评审中，检查表可以提醒参与者不要忘记对检查表上列出的项目进行评审。同时检查表也可以帮助减少评审的个体指向性，例如，“这是我们在每个评审中使用的同样表格，我们可不是仅仅针对你的文档进行评审。”检查表可以是一般的、具有普遍性的，适用于所有的评审，也可以是聚焦在某些特定质量特性、领域和文档类型的。例如，一个通用检查表可以用于检查一些具有普遍性的文档属性，如文档应该有唯一的标识，文档应该不包含对尚未确定的内容的引用，适当的格式以及类似的一致性项目。一个明确的需求文档检查表可能包括对“shall（必须）”和“should（应该）”这类词汇使用恰当性的检查，还可能包括对需求可测性性的检查等等。同样的，不同类型的需求文档需要使用的检查表也不同。一个针对“叙述性需求文档”的检查表和一个针对“基于图表需求文档”的检查表，应该有不同的评审标准。

检查表也可以针对开发人员、架构师、测试人员的技能。对测试分析师来说，需要在测试能力的检查内容上更加深入。检查表可能需要包含下面列出的项目。

针对需求、用例、用户故事的检查表内容与针对代码、架构的检查表是不同的。针对需求的检查表需要包含下列内容：

- 各个需求的可测试性
- 各个需求的验收准则
- 一个用例的调用结构（如果有的话）
- 各个需求、用例、用户故事需要有唯一的标识
- 各个需求、用例、用户故事需要版本号
- 每个业务需求/市场需求到需求的可追溯性
- 需求与用例之间的可追溯性

以上信息仅仅是作为一个例子。非常重要的一点是：如果需求不具备可测试性，那就意味着测试分析师无法确定测试的方法，也就意味需求的缺陷。例如，类似“软件应该是与用户非常友好的”这种需求描述不具备可测试性。测试分析师如何确定一个软件到底是用户友好的，还是用户非常友好的？相应的，如果需求表述为“软件必须符合易用性标准文档中描述的易用性标准”，而如果该易用性标准又是存在的，那么这个需求具备可测试性。而这个需求又是一个全局需求，因为该需求适用于界面中的所有元素。在这种情况下，如果被测对象是一个非核心应用，则这个需求就可以很轻易的派生出大量的测试用例。建立需求、标准文档与测试用例之间的追溯关系同样非常重要，因为一旦关联的易用性标准发生了变动，测试用例可能需要随之被重新评审或更新。

如果测试人员无法确定测试是否通过，或者无法构建出能够执行通过或者失败的测试，那与该测试有关的需求同样也不具备可测试性。例如，“系统应该全年 365（或 366）天，每周 7 天，每天 24 小时，均 100%可用”就是不可测试需求。

.

一个简单的、针对用例评审的检查表可能包括以下问题：

- 是否清晰定义了主要路径（场景）？
- 是否识别、完成了所有的可选分支（场景）并有错误处理？
- 是否定义了所有用户界面信息？
- 是否只有一个唯一主路径（一般来说我们认为一个用例应该只有一个主路径，否则就应该拆分为多个用例）？
- 每个路径都可测试？

一个简单的、针对应用程序的用户界面易用性检查表可能包括：

- 每个界面元素及其相关功能都已经定义了吗？
- 所有的错误处理信息都已经定义了吗？
- 所有的用户提示信息都已经定义并保持一致了吗？
- 界面元素的 Tab 顺序（指按键盘 Tab 键后的焦点跳转次序——译者）已经被定义了吗？
- 鼠标操作都有键盘替代操作吗？
- 快捷组合键是否已经定义了？（例如，剪切和粘贴操作）
- 界面元素之间是否有依赖关系（例如某些数据应该在某些数据展示后再出现）？
- 是否设定了屏幕布局？
- 屏幕布局是否符合规定的需求？
- 当系统正在执行某项操作时，是否有进度条？
- 界面是否符合最少鼠标点击需求（如果存在这种需求）？
- 浏览流程在逻辑上是否与用例中描述的信息符合？
- 界面中的信息是否符合易学性需求？
- 是否有针对用户的帮助文档信息？
- 是否有针对用户的鼠标浮动提示信息？
- 用户是否认为界面是“有吸引力的”（基于主观评估）？
- 界面色彩的运用是否与其他应用程序一致？是否与组织标准一致？
- 声音效果的使用是否合适？声音效果是否可配置？
- 界面是否符合本地化需求？
- 用户是否可以明白如何操作软件（易理解性）（基于主观评估）？
- 用户是否可以记住如何操作软件（易学性）（基于主观评估）？

在敏捷项目中，需求经常是采用用户故事的形式表述。这些故事代表小单位的可展现功能。与一个用例可以是一个跨越多个功能部分的事务不同，一个用户故事更加独立并且通常根据开发时间来限定范围。一个用于用户故事的检查表可能包括：

- 这个故事是否与本次迭代/冲刺的目标一致？
- 验收标准是否已经被定义并具有可测试性？
- 功能是否被清晰的定义了？
- 这个故事与其他故事之间是否有依赖性？
- 是否已经对故事设定了优先级？

- 这个故事是否包含至少一个功能点？

当然，如果这个故事定义了一个新的界面，那么除了使用通用的故事检查表（例如上面的这个）以外,还应当使用一个具体的用户界面检查表。

可以根据下述信息定制一个检查表：
- 组织机构（例如考虑公司的政策，标准，约定）
- 项目/开发因素（例如，重点，技术标准，风险）
- 评审目标（例如，针对特定的编程语言采用适合的代码评审）

好的检查表可以帮助发现问题，同时对于没有在检查表中被引用的内容，检查表也能帮助开始着手进行讨论。组合运用不同的检查表是保证评审质量和交付高质量产品的重要途径。运用基础级大纲中列出的标准检查表以及开发组织自己的检查表可以帮助测试分析师更有效地进行评审。

更多关于评审和审查的信息可以参考[Gilb93] 和 [Wiegers03]。

6. 缺陷管理—120 分钟

关键词

缺陷分类（Defect taxonomy），阶段遏制（phase containment），根本原因分析（root cause analysis）

缺陷管理的学习目标

6.2 何时缺陷会被发现？

TA-6.2.1 （K2）解释阶段遏制为何可以降低成本

6.3 缺陷报告的字段

TA-6.3.1 （K2）解释描述一个非功能缺陷的文档应该包含的内容

6.4 缺陷分类

TA-6.4.1 （K4）对于给定的缺陷，识别、收集并记录分类信息

6.5 缺陷原因分析

TA-6.5.1 （K2）解释根本原因分析的目的

6.1 简介

测试分析师从业务、用户需求等角度对系统的行为进行评估，例如，当用户看到某个提示信息或者行为时是否知道应该如何做。测试分析师通过对预期结果和实际结果的比对判断系统行为是否正确。一个异常（也称为事件）是需要进一步关注的意外现象。一个异常可能是由缺陷引起的失效。一个异常通常可能导致生成一个缺陷报告，但也可能不会。而缺陷就是需要解决的实际问题。

6.2 缺陷何时会被发现？

静态测试可以发现缺陷，而缺陷的反应，即失效可以通过动态测试被发现。软件开发生命周期的每个阶段都应提供发现和排除潜在失效的方法。例如，在开发阶段，代码评审和设计评审可以用于发现缺陷。在动态测试中，测试用例可用于发现失效。

越早发现和修正软件缺陷，系统质量整体成本越低。例如，在还不具备动态测试条件之前，就可以用静态测试发现缺陷。这也是静态测试是效费比更高的测试方法的原因之一。

缺陷追踪系统应可以允许测试分析师记录在生命周期的哪个阶段缺陷被引入，以及在哪个阶段缺陷被发现。如果缺陷的引入和缺陷的发现在同一个阶段，那么就实现了缺陷的阶段遏制。这表明缺陷在同一个阶段被引入和发现，而没有“逃脱”并隐藏到后面的阶段。例如，一个错误的需求在需求评审中被发现和纠正，就是一个缺陷阶段遏制的例子。这不仅意味着需求评审是有效的，而且这也有效避免了需求缺陷导致的额外工作以及给组织带来的昂贵成本。如果一个不正确的需求从需求评审中“逃脱”并接着被开发人员实现，并被测试分析师测试，最终在用户验收阶段被用户发现，那么所有这个错误需求导致的工作都是在浪费资源（这还没考虑由此导致的用户对系统信心的损失）。

阶段遏制（phase containment）是降低缺陷成本的有效措施。

6.3 缺陷报告的字段

缺陷报告的字段（参数）应该提供有关缺陷的足够信息，从而使这个缺陷报告具有可操作性。一个可操作的缺陷报告是：

- 完整的——报告中包含所有必要的信息
- 简洁的——报告中不包含不必要的信息
- 准确的——报告中的信息准确、清晰的表述了预期结果、实际结果以及重现缺陷的必要步骤
- 客观的——报告是专业的书面对事实的描述

缺陷报告中记录的信息应该分成不同的数据字段。被清晰定义的字段越多，报告缺陷就越容易，同时制作趋势报告和其他汇总报告越容易。如果某个字段的输入选项是确定的，那么使用下拉菜单就可以减少录入缺陷的时间。不过要注意的是下拉菜单只在备选输入的数量有限时可提高效率，而当备选选项数量过多需要大量滚动时下拉菜单的效率会降低。不同类型的缺陷报告需要不同的信息，缺陷管理系统需要足够灵活以便为不同类型的缺陷配置不同的字段。数据应当被记录在不同的字段中，理想状态下应当对数据进行确认，以避免输入错误并保证缺陷报告的有效性。

缺陷报告的编写目的是由于在功能性测试和非功能性测试中发现了失效。缺陷报告中的信息应该着眼于清晰地描述问题发现的场景，包括重现该场景所需要的步骤、数据，以及预期结果和实际结果。非功能性缺陷的报告需要包含更多关于环境、性能参数（负载规模数据）、操作步骤顺序以及预期结果等的细节信息。当我们用于记录一个易用性失效时，记录用户期望软件的操作就非常重要了。例如，如果易用性标准规定一个操作必须在四次鼠标点击内完成，那么缺陷报告就要描述实际系统中用户需要执行多少次点击才能完成操作。如果没有类似的易用性需求，或者需求文档没有覆盖这些软件的非功能特性，测试人员可以请“适当的人员”来参与测试以确定软件的易用性是否可以被接收。在这种情况下，“适当

的人员”的期望结果必须在缺陷报告中清晰地表述出来。由于非功能性需求经常会被需求文档遗漏，所以描述非功能性失效时常遇到如何记录“预期的”结果和“实际的”行为的挑战。

虽然通常编写缺陷报告的目的是为了修复问题，但是缺陷信息也必须可以被用于支持精确的分类、风险分析以及过程改进。

6.4 缺陷分类

在缺陷的生命周期中，可以从多个层次对缺陷进行分类。合理的缺陷分类是好的缺陷报告中不可缺少的一部分。缺陷分类可以用于缺陷的聚类、评估测试的有效性、评估开发过程的有效性以及对趋势做出判断。

对于新发现的缺陷，一般可以包含下列分类信息：

- 在什么项目活动中发现的缺陷— 例如评审、审计、审查、编码、测试
- 在哪个项目阶段（如果可以确定的话）引入的缺陷— 例如需求、设计、详细设计、编码
- 在哪个项目阶段发现的缺陷 — 需求、设计、详细设计、编码、代码评审、单元测试、集成测试、系统测试、验收测试
- 缺陷的可能原因 — 例如需求、设计、接口、代码、数据
- 可重现性 — 例如一次性、偶尔、可复现
- 症状 — 例如崩溃、挂起、用户界面错误、系统错误、性能问题

一旦缺陷被进一步分析，更深入的分类就成为可能：

- 根本原因 — 是什么错误导致了缺陷，例如流程、代码错误、用户错误、测试错误、配置错误、数据错误、第三方软件、外部软件问题、文档错误等
- 来源 —— 哪些工作产品导致了错误，例如需求文档、设计文档、详细设计文档、架构设计说明、数据库设计、用户手册、测试文档
- 类型 —— 例如逻辑问题、计算问题、时间问题、数据处理、升级。

在缺陷被修改后（或者被暂缓，或者确认失败），将有更多的分类信息可用，例如：

- 解决方式 —— 例如代码修改、文档变更、暂缓、不是问题、重复的缺陷
- 修正措施 —— 例如需求评审、代码评审、单元测试、配置文档、数据准备、不做修改等

除了前述的这些分类信息之外，缺陷还经常需要加上严重程度和优先级两个信息。此外，对于某些项目，还需要考虑缺陷对安全性、项目进度、项目成本、项目风险以及项目质量的影响。这些分类信息会帮助确定缺陷到底多快被修复。

根据最后解决方案对问题进行最终的分类，缺陷经常被按照他们的解决方案被分类，例如已修复/已确认的、已关闭/不是缺陷、已延迟、打开/未解决的等。这种分类通常在生命周期内跟踪整个项目的缺陷。

不同的组织，所使用的缺陷分类方法通常是不同的。上文列出的仅仅是业界普遍使用的一些例子。对于分类来说，很重要的一个宗旨就是应该确保分类方法的有效性。使用过多的分类参数会导致新建和维护缺陷报告的时间成本大幅度提升，所以，需要在维护缺陷报告的分类指标数据所需工作量与分类带来的效果之间做出权衡。在选择这类工具时，可以收集自定义分类值往往是一个重要指标。

6.5 根本原因分析

根本原因分析的目的是为了发现导致缺陷的根本原因，并且为过程改进提供支持数据，从而消除那些导致相当大比例缺陷的根本原因。一般是由对缺陷进行调查并且修改问题，或确定不修改问题或者是问题无法修改的人员来进行根本原因分析，通常是开发人员。确定一个可能的备选根本原因的清单的工作，通常是由测试分析师来完成，测试分析师需要对可能导致缺陷的原因做一些猜测。一旦缺陷的修复被确

认完成，测试分析师需要对开发人员所写的根本原因进行确认。在根本原因被确认那些点上，通常我们还会进一步确定和确认在哪个阶段该缺陷被引入的。

典型的根本原因包括：

- 不清晰的需求
- 需求缺失
- 错误的需求
- 错误的设计实现
- 错误的界面实现
- 代码逻辑错误
- 计算错误
- 硬件错误
- 接口错误
- 无效的数据

缺陷的根本原因信息被汇集起来后，可以分析出哪些是会导致缺陷的普遍问题。例如，如果有大量的缺陷是由于不清晰的需求导致的，那么就很有理由进行更有效果的需求评审了。同样，如果不同的开发组都遇到了接口实现的问题，那么进行跨越不同开发组并与设计人员一起的会议就很有必要了。

在过程改进中，使用根本原因信息可以帮助组织监控有效的过程改进所带来的收益，并且可以将属于特定根本原因的缺陷成本量化。这有助于为过程改进申请资金，以用于工具和设备采购，或者改变项目的进度计划。在 ISTQB®专家级大纲“测试过程改进”[ISTQB_EL_ITP]中，对根本原因分析有更详尽的讨论。

7. 测试工具—45 分钟

关键词

关键词驱动测试（keyword-driven testing）,测试数据准备工具（test data preparation tool）,测试设计工具（test design tool）,测试执行工具（test execution tool）

测试工具的学习目标

7.2 测试工具和自动化

TA-7.2.1 （K2）讲解使用测试数据准备工具、测试设计工具和测试执行工具的益处
TA-7.2.2 （K2）讲解测试分析师在关键字驱动自动化测试中的作用
TA-7.2.3 （K2）讲解针对自动化测试执行失败的故障排除步骤

7.1 简介

测试工具能显著提高测试工作的效率和精度，前提是采用合适的工具并且以合适的方法使用工具。对具有良好管理的测试机构而言，必须对测试工具进行有效管理。测试工具的复杂性和应用性变化非常广，并且测试工具市场也在不断变化中。通常可用的工具除了来自商业工具厂商外，还有很多免费的或共享的工具提供方。

7.2 测试工具和自动化

测试分析师的许多工作都需要有效地使用工具，知道用哪个工具，何时用。这样不仅能提高测试分析师的效率，而且在允许的时间内能帮助提供更好的测试覆盖率。

7.2.1 测试设计工具

测试设计工具用来帮助生成测试所需的测试用例和测试数据。这些工具可以针对特定格式的需求文档、模型（例如 UML）或由测试分析师提供的输入进行处理，测试设计工具通常被设计和构造成能与特定格式和特定产品协同工作，例如特定的需求管理工具。

测试设计工具能为测试分析师提供信息，用来决定测试类型，以达到预定的测试覆盖率、信心、或产品风险缓解活动。例如，分类树工具基于选定的覆盖准则能产生（并且显示）测试用例组合集合以达到完全覆盖，测试分析师可以利用这些信息确定必须执行的测试用例。

7.2.2 测试数据准备工具

测试数据准备工具提供以下几方面优势。一些测试数据准备工具能分析诸如需求文档、或者甚至源代码来决定测试中的数据，以达到覆盖水平。其它测试数据准备工具能从实际系统获取数据，然后对其“清洗”或“匿名化”以去除任何个人隐私信息，同时保证数据的内部完整性。处理后的数据能用于测试，并可避免安全漏洞或误用个人隐私信息的风险。这对于需要大规模的真实数据场合是非常重要的。另外一些数据生成工具能通过给定输入参数集合来生成测试数据（例如，用于随机测试中）。其中有些需要测试分析师分析数据库结构来决定需要输入的内容。

7.2.3 自动测试执行工具

在所有测试级别中，测试分析师主要使用测试执行工具运行测试和检查测试结果。使用测试执行工具的目的有以下几种：

- 为了削减成本（在工作和/或时间方面）
- 为了运行更多的测试
- 为了在多种环境中运行相同的测试
- 为了使测试执行更具可重复性
- 为了运行那些靠人工无法运行的测试（例如，大规模数据确认测试）

这些目的通常重叠成主要目的：提高覆盖率，同时降低成本。

7.2.3.1 适用性

在自动化回归测试中，测试执行工具的投资回报通常是最高的，主要是因为回归测试只需低级别维护并且被重复执行。自动冒烟测试中也可以有效地使用自动化，因为会频繁的使用冒烟测试并且需要快速的获得结果，尽管维护成本可能更高，但能通过自动化途径来评估持续集成环境中的新构建。

测试执行工具通常用于系统和集成测试级别中。有些工具也可能用于组件测试级别，特别是 API 测试工具。凭借最适用的工具将有助于提高投资回报率。

7.2.3.2 测试自动化工具基础

测试执行工具是通过执行一组编程语言写的指令来工作的，这种编程语言通常称为脚本语言。工具的指令是非常详细的，包括特定的输入、输入的顺序、使用特定值作为输入和期望的输出。这能使得详细脚本能在被测软件（SUT）中变化自如，特别当工具与图形用户界面（GUI）进行交互时。

大部分测试执行工具包括一个比较器，用于比较实际的结果与保存的期望结果。

7.2.3.3 测试自动化实施

测试执行自动化（类似编程）的趋势从详细的低层指令向更高层语言、应用库、宏和子程序方向发展。关键字驱动和动词驱动（action word-driven）设计技术捕获一系列指令并通过特定的“关键字”或“动词”来引用这些指令。这使得测试分析师可以用人类语言编写测试用例，而忽略底层的编程语言和低级别函数。当被测软件的功能和接口改变时，采用这种模块化编写技术将更容易维护。[Bath08] 下面将讨论有关自动化脚本中使用关键字的更多内容。

模型能用来指导关键字或动词的产生。通过观察业务流程模型，通常业务流程模型已经包含在需求文档中，测试分析师确定必须进行测试的关键业务流程。同时确定流程的步骤，包括在处理过程中可能会出现的决策点，这些决策点可以成为动词，并从关键字或动词电子表格中获得和使用测试自动化。业务流程建模是一种文档化业务流程的方法，能用来识别这些关键流程和决策点。可以人工建模或通过工具来建模，工具建模是基于业务规则和流程描述作为输入的操作。

7.2.3.4 提高自动化成功的工作

当决定测试自动化，必须评估每个候选的测试用例或候选的测试套件，以观察是否值得自动化。许多失败的自动化项目是由于轻易地采用人工测试用例来自动化，没有检查是否能从自动化获得实际收益。对给定的测试用例集（测试套件），可以通过优化来采用包括人工、半自动化和全自动化测试。

当实施测试执行自动化项目时，应考虑下列方面：

可能的益处：

- 自动化测试执行时间将变得更可预测
- 使用自动化测试后，在项目后期的回归测试和缺陷确认将变得更快和更可靠
- 通过使用自动化工具，加快测试人员或测试团队的地位和技术的增长
- 自动化对迭代和增量开发生命周期特别有帮助，可以对每一个构建或迭代提供更好的回归测试
- 某些测试类型的覆盖仅当采用自动化工具才有可能（例如，大规模数据确认工作）
- 对大规模数据输入、转换和比较测试工作，测试执行自动化比人工测试有更高的性价比，提供更快和一致的输入和确认。

可能的风险：

- 不完全的、无效的或不正确的人工测试可能被“原封不动”自动化
- 当被测软件变更时，测试件可能很难维护，且需要许多的更改
- 直接由测试人员参与测试的执行可能会减少和导致较少的缺陷发现
- 测试团队可能没有足够的技能来有效地使用自动化工具
- 非关键的对整个测试覆盖没有贡献的测试可能被自动化了，因为它们存在且稳定
- 当软件稳定后，测试可能成为徒劳无功了（杀虫剂悖论）

在测试执行自动化工具展开阶段，原封不动的对人工测试用例自动化通常是不明智的，应重新定义测试用例以更好地使用自动化。这包括格式化测试用例、考虑重用模式、通过使用变量替代硬编码数值来扩展输入、以及利用测试工具的所有益处。测试执行工具通常有能力横贯复合测试、聚合测试、重复测试和改变执行次序，以及提供分析和报告机制。

对许多测试执行自动化工具，要产生高效率的和有效的测试（脚本）和测试套件，编程技能是必须的。如果没有很仔细的设计，通常大型的自动化测试套件的更新和管理是非常困难的。适当的测试工具、编程技术和设计技术的培训，对于确保测试工具能发挥最大效益是非常重要的。

在测试计划阶段，花时间定期人工执行自动化测试用例以获得测试工作的知识、验证正确的操作，以及评审输入数据正确性和覆盖是很重要的。

7.2.3.5 关键词驱动自动化

关键字（有时被称为动词）经常（但不局限于）用来表示系统的高层业务交互（例如，“取消订单”）。每个关键字一般用来表示在测试中操作者和系统之间的一组详细交互。关键字序列（包括相关的测试数据）用于特定的测试用例。[Buwalda01]

测试自动化中，一个关键字可被实现为一个或多个可执行的测试脚本。工具读入用关键字序列编写的测试用例，调用相应的测试脚本来实现关键字功能。脚本被实现成高层模块方法，能够更容易映射到特定关键字。当然，实现这些模块脚本是需要编程技能的。

关键词驱动测试自动化的主要优点：

- 关键字关联到特定应用或业务领域，能被领域专家定义。可以使得编制测试用例规范说明的任务更有效。
- 主要的领域专家能从自动化测试用例执行（一旦关键字像脚本一样执行）中获益，而无需了解底层的自动化代码。
- 使用关键字编写测试用例更容易维护，因为在被测软件细节变更后很少需要修改。
- 测试用例规范说明独立于它们的实现。可以使用各种脚本语言和工具来实现关键字。

自动化脚本（实际的自动化代码）使用的关键字/动词信息通常由开发者或技术测试分析师来编写，而测试分析师则创建和维护关键字/动词数据。当关键字驱动自动化在系统测试阶段进行时，代码开发应尽早地在集成阶段开始。在迭代环境下，测试自动化开发是一个持续的过程。

一旦创建了输入关键字和数据，测试分析师通常负责执行关键词驱动测试用例，并且分析任何可能发生的失效。当发现一个异常时，测试分析师必须调查失效的原因，以确定问题是出在关键字、输入数据、自动化脚本本身，还是出在被测试的应用。通常故障排除步骤的第一步是人工使用相同的数据执行相同的测试，观察失效是否在应用本身。如果这步没有显示失效，测试分析师应该检查导致失效的测试序列，以确定问题是否发生在前面的步骤（有可能通过产生不正确的数据），但直到后面的过程才显现出问题来。如果测试分析师不能确定失效的原因，故障排除信息将转给技术测试分析师或开发人员做进一步分析。

7.2.3.6 自动化工作失败的原因

测试执行自动化项目常常不能达到它们的目标。这些失败有可能是由于测试工具的使用灵活性不够，测试团队的编程技能不足，或让测试执行自动化解决一个不切实际的问题。认识到所有测试执行自动化像软件开发项目一样，需要管理、努力、技能和专注是非常重要的。要花时间在创建可支撑的架构，遵循恰当的设计实践，提供配置管理，以及遵循好的编程实践。自动化测试脚本可能存在缺陷，因此必须被测试。为了性能，脚本有可能需要调优。除了开发人员外，使用工具执行脚本的人通常也必须考虑工具的可用性。有可能需要设计工具和使用者之间的接口，以提供访问测试用例的途径，逻辑上是给测试员使用，但也为工具提供了可存取性。

8. 参考文献

8.1 标准

- [ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation （SQuaRE）
  GB/T 25000.1-2010 软件工程软件产品质量要求与评价（SQuaRE）SQuaRE 指南
  第 1 章和第 4 章
- [ISO9126] ISO/IEC 9126-1:2001, Software Engineering - Software Product Quality, GB/T 16260.1-2006 软件工程产品质量第 1 部分：质量模型
  第 1 章和第 4 章
- [RTCA DO-178B/ED-12B]: Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE ED12B.1992.

  第 1 章

8.2 ISTQB 文档

- [ISTQB_AL_OVIEW] ISTQB Advanced Level Overview, Version 1.0
- [ISTQB_ALTM_SYL] ISTQB Advanced Level Test Manager Syllabus, Version 1.0
- [ISTQB_ALTTA_SYL] ISTQB Advanced Level Technical Test Analyst Syllabus, Version 1.0
- [ISTQB_FL_SYL] ISTQB Foundation Level Syllabus, Version 2011
- [ISTQB_GLOSSARY] Standard glossary of terms used in Software Testing, Version 2.2, 2012

8.3 文献

[Bath08] Graham Bath, Judy McKay, “The Software Test Engineer’s Handbook”, Rocky Nook, 2008, ISBN 978-1-933952-24-6

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

[Black02]: Rex Black, “Managing the Testing Process (2nd edition)”, John Wiley & Sons: New York, 2002, ISBN 0-471-22398-0

[Black07]: Rex Black, “Pragmatic Software Testing”, John Wiley and Sons, 2007, ISBN 978-0-470-12790-2

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

[Cohn04]: Mike Cohn, “User Stories Applied: For Agile Software Development”, Addison-Wesley Professional, 2004, ISBN 0-321-20568-5

[Copeland03]: Lee Copeland, “A Practitioner's Guide to Software Test Design”, Artech House, 2003, ISBN 1-58053-791-X

[Craig02]: Rick David Craig, Stefan P. Jaskiel, “Systematic Software Testing”, Artech House, 2002, ISBN1-580-53508-9
系统的软件测试，电子工业出版社，2003

[Gerrard02]: Paul Gerrard, Neil Thompson, “Risk-based e-business Testing”, Artech House, 2002, ISBN 1-580-53314-0

[Gilb93]: Tom Gilb, Graham Dorothy, “Software Inspection”, Addison-Wesley, 1993, ISBN 0-201-63181-4

[Graham07]: Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black “Foundations of Software Testing”, Thomson Learning, 2007, ISBN 978-1-84480-355-2

[Grochmann94]: M. Grochmann (1994), Test case design using Classification Trees, in: conference proceedings STAR 1994

[Koomen06]: Tim Koomen, Leo van der Aalst, Bart Broekman, Michiel Vroon "TMap NEXT, for result driven testing", UTN Publishers, 2006, ISBN 90-72194-80-2

[Myers79]: Glenford J. Myers, “The Art of Software Testing”, John Wiley & Sons, 1979, ISBN 0-471-46912-2
软件测试的艺术，机械工业出版社，2006

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

[vanVeenendaal12]: Erik van Veenendaal, “Practical risk-based testing – The PRISMA approach”, UTN Publishers, The Netherlands, ISBN 9789490986070

[Wiegers03]: Karl Wiegers, “Software Requirements 2”, Microsoft Press, 2003, ISBN 0-735-61879-8

[Whittaker03]: James Whittaker, “How to Break Software”, Addison-Wesley, 2003, ISBN 0-201-79619-8

[Whittaker09]: James Whittaker, “Exploratory Software Testing”, Addison-Wesley, 2009, ISBN 0-321-63641-4
探索式软件测试，清华大学出版社，2010

8.4 其它参考文献

下列参考指向因特网上和其它有用的信息。尽管这些参考在发布本高级大纲时核查过，ISTQB 不能保证这些参考现在还可用。

- 第 3 章
  - Czerwonka, Jacek: www.pairwise.org
  - Bug Taxonomy: www.testingeducation.org/a/bsct2.pdf
  - Sample Bug Taxonomy based on Boris Beizer’s work: inet.uni2.dk/~vinter/bugtaxst.doc
  - Good overview of various taxonomies: testingeducation.org/a/bugtax.pdf
  - Heuristic Risk-Based Testing By James Bach
  - From “Exploratory & Risk-Based Testing (2004) www.testingeducation.org"
  - Exploring Exploratory Testing , Cem Kaner and Andy Tikam , 2003
  - Pettichord, Bret, “An Exploratory Testing Workshop Report”, www.testingcraft.com/exploratorypettichord
- 第 4 章
  - www.testingstandards.co.uk

9. 索引