ISTQB® 认证测试工程师
基础级大纲

2018 V3.1 版

中文版本（2019 年 12 月 25 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

版权标志

ISTQB® 认证工程师基础级大纲修订历史:

版本	日期	备注
ISTQB®2018 V3.1	2019年11月11日	认证测试工程师基础级教学大纲维护发行版（含少量更新）-请参阅发行说明
ISTQB® 2018	2018年4月27日	候选一般发布版本
ISTQB® 2011	自2011年4月1日起生效	ISTQB® 认证测试工程师基础级大纲 修订版-见附录E-版本说明
ISTQB® 2010	自2010年3月30日起生效	ISTQB® 认证测试工程师基础级大纲 修订版-见附录E-版本说明
ISTQB® 2007	2007年5月1日	ISTQB® 认证测试工程师基础级大纲 修订版
ISTQB® 2005	2005年7月1日	ISTQB® 认证测试工程师基础级大纲
ASQF V2.2	2003年7月	ASQF基础级大纲 V2.2
ISEB V2.0	1999年2月25日	ISEB软件测试基础级大纲 V2.0 1999年2月25日

ISTQB® 认证测试工程师基础级大纲中文版本的修订历史:

版本	日期	备注
ISTQB® 2018V3.1	2019年12月10日	更新，有少许修改　负责人：周震漪
ISTQB® 2018中文版	2019年8月25日	质量控制 负责人：周震漪 在原版本（2019 年 3 月 18 日）基础上进行多处修改和完善；统一了与国家标准相一致的术语，例如“利益相关方”,“生存周期”,“产品负责人”,“特征/feature”等
ISTQB® 2018中文版	2019年3月18日	翻译 FL 2018 版本。 负责人：沈建雄
ISTQB® 2011中文版修订版2	2016年12月日	根据软件测试专业术语中英文对照表 v3.0 修订本文档中的部分术语，以保持一致性

		负责人：沈建雄
ISTQB® 2011中文版修订版1	2015年5月6日	根据软件测试专业术语中英文对照表 v2.4 修订本文档中的部分术语，以保持一致性 负责人：沈建雄
ISTQB® 2011中文版V13	2011年12月15日	2011 年基础级大纲和术语工作组根据软件测试专业术语中英文对照表 v2.1,修订本文档中的部分术语，以保持一致性
ISTQB® 2011中文版v12	2011年7月1日	按照 ISTQB® 2011 版进行修订 工作组组长：沈建雄
ISTQB® 2010中文版评审版 V01	2010年8月12日	按照 ISTQB® 2010 版进行修订 负责人：周震漪
ISTQB® 2007中文版	2008年8月28日	按照英文 2007 版翻译，力求与原版保持一致。 责任负责人：周震漪
CSTQB® 2007版	2007年7月1日	
CSTQB® 2005版	2005年7月1日	

目　录

致谢

本大纲由 ISTQB®成员国大会于 2019 年 11 月 11 日正式发布。

它是由 ISTQB® 的一个工作组编写制作，主要工作组人员有：Klaus Olsen (主席), Meile Posthuma 和 Stephanie Ulrich。

工作组感谢各成员国委员会对 2018 基础级教学大纲的审查意见。

2018 基础级大纲由国际软件测试认证委员会组成的工作组完成：Klaus Olsen (主席), Tauhida Parveen (副主席), Rex Black (项目经理), Debra Friedenberg, Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, 和 Eshraka Zakaria。

本大纲工作组感谢 Rex Black 和 Dorothy Graham 所做的技术编辑工作，以及评审团队、交叉评审团队和各成员国委员会的建议和输入。

下列专家参与了本大纲的评审、建议和投票: Tom Adams, Tobias Ahlgren, Xu Aiguo, Chris Van Bael, Katalin Balla, Graham Bath, Gualtiero Bazzana, Arne Becher, Veronica Belcher, Lars Hilmar Bjørstrup, Ralf Bongard, Armin Born, Robert Bornelind, Mette Bruhn-Pedersen, Geza Bujdoso, Earl Burba, Filipe Carlos, Young Jae Choi, Greg Collina, Alessandro Collino, Cui Zhe, Taz Daughtrey, Matthias Daigl, Wim Decoutere, Frans Dijkman, Klaudia Dussa-Zieger, Yonit Elbaz, Ofer Feldman, Mark Fewster, Florian Fieber, David Frei, Debra Friedenberg, Conrad Fujimoto, Pooja Gautam, Thorsten Geiselhart, Chen Geng, Christian Alexander Graf, Dorothy Graham, Michel Grandjean, Richard Green, Attila Gyuri, Jon Hagar, Kobi Halperin, Matthias Hamburg, Zsolt Hargitai, Satoshi Hasegawa, Berit Hatten, Wang Hongwei, Tamás Horváth, Leanne Howard, Chinthaka Indikadahena, J. Jayapradeep, Kari Kakkonen, Gábor Kapros, Beata Karpinska, Karl Kemminger, Kwanho Kim, Seonjoon Kim, Cecilia Kjellman, Johan Klintin, Corne Kruger, Gerard Kruijff, Peter Kunit, Hyeyong Kwon, Bruno Legeard, Thomas Letzkus, Alon Linetzki, Balder Lingegård, Tilo Linz, Hongbiao Liu, Claire Lohr, Ine Lutterman, Marek Majernik, Rik Marselis, Romanos Matthaios, Judy McKay, Fergus McLachlan, Dénes Medzihradszky, Stefan Merkel, Armin Metzger, Don Mills, Gary Mogyorodi, Ninna Morin, Ingvar Nordström, Adam Novak, Avi Ofer, Magnus C Ohlsson, Joel Oliviera, Monika Stocklein Olsen, Kenji Onishi, Francisca Cano Ortiz, Gitte Ottosen, Tuula Pääkkönen, Ana Paiva, Tal Pe'er, Helmut Pichler, Michaël Pilaeten, Horst Pohlmann, Andrew Pollner, Meile Posthuma, Vitalijs Puiso, Salvatore Reale, Stuart Reid, Ralf Reissing, Shark Ren, Miroslav Renda, Randy Rice, Adam Roman, Jan Sabak, Hans Schaefer, Ina Schieferdecker, Franz Schiller, Jianxiong Shen, Klaus Skafte, Mike Smith, Cristina Sobrero, Marco Sogliani, Murian Song, Emilio Soresi, Helder Sousa, Michael Sowers, Michael Stahl, Lucjan Stapp, Li Suyuan, Toby Thompson, Steve Toms, Sagi Traybel, Sabine Uhde, Stephanie Ulrich, Philippos Vakalakis, Erik van Veenendaal, Marianne Vesterdal, Ernst von Düring, Salinda Wickramasinghe, Marie Walsh, Søren Wassard, Hans Weiberg, Paul Weymouth, Hyungjin Yoon, John Young, Surong Yuan, Ester Zabar, 和 Karolina Zmitrowicz。

国际软件测试认证委员会基础级大纲工作组（2018 版）有： Klaus Olsen (主席), Tauhida Parveen (副主席), Rex Black (项目经理), Dani Almog, Debra Friedenberg, Rashed Karim, Johan Klintin, Vipul Kocher, Corne Kruger, Sunny Kwon, Judy McKay, Thomas Müller, Igal Levi, Ebbe Munk, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, Eshraka Zakaria,

和 Stevan Zivanovic。 编撰本大纲的核心团队感谢各成员国委员会给出的建议。

编撰本大纲的核心团队感谢评审团队：Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Rioudu Cosquier HansSchaefer, StephanieUlrich, ErikvanVeenendaal, 以及给予意见和建议的所有成员国委员会。

国际软件测试认证委员会基础级大纲工作组（2011 版）有：Thomas Müller（主席）, Debra Friedenberg。编撰本大纲的核心团队感谢评审团队：Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Rioudu Cosquier HansSchaefer, StephanieUlrich, ErikvanVeenendaal, 以及给于意见和建议的所有国家委员会。参加 2011 中文版修订的专家有（按姓氏拼音排序）静国玥、刘晓更、沈建雄、徐文叶、周震漪等。

国际软件测试认证委员会基础级大纲工作组(2010 版)有:Thomas Müller(主席), Rahul Verma, Martin Klonk and Armin Beer。编撰本书的核心团队感谢评审团队：Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, TuulaPääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Wiliams, Erik van Veendendaal 以及给于意见和建议的所有国家委员会。中国编写组合评审组是由 CSTQB 的专家组成员组成的工作组：周震漪、沈建雄、崔启亮等。

国际软件测试认证委员会基础级大纲工作组（2007 版）有：Thomas Müller（主席）, Dorothy Graham, Debra Friedenberg, and Erik van Veendendaal 和评审组成员 Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, Wonil Kwon 以及给于意见和建议的所有国家委员会。参加中文版的专家有（按姓氏拼音排序）：曹静、杜庆峰、刘琴、刘小茵、马均飞、吴晓臻、郑文强、周震漪等。

国际软件测试认证委员会基础级大纲工作组（2005 版）有：Thomas Müller（主席）,Rex Black,Sigrid Eldh,Dorothy Graham,Klaus Olsen,Maaret Pyhäjärvi,Geoff Thompson and Erik van Veendendal. 编撰本书的核心团队感谢评审团队以及对当前课程大纲提供建议的所有成员国委员会。

2018 版基础级大纲中文翻译参与者（按姓氏拼音排序）

崔哲、李华北、沈建雄（组长）、宋光照、左平

2018 版基础级大纲中文评审参与者（按姓氏拼音排序）

曹佩、郑文强

2018 版基础级大纲中文质量控制（按姓氏拼音排序）

周震漪

2018 V3.1 版基础级大纲中文翻译和质量控制（按姓氏拼音排序）

周震漪

致谢企业：上海滔瑞信息技术有限公司

0 大纲简介

0.1 本文档的目的

本大纲是国际软件测试认证基础级水平的中文版课程大纲。国际软件测试认证委员会（以下简称 ISTQB®）提供标准的课程大纲：

(1) 给各个成员国委员会，翻译成本国语言并且授权给培训机构。成员国委员会可以根据他们特定的语言调整教学大纲，以及引用当地出版物。
(2) 给考试机构，根据本大纲的学习目标，出当地语言的考试题。
(3) 给培训机构，编制课件和决定相应的授课方法
(4) 给认证考试应试者，准备认证考试（认证考试可以是作为培训课程一部分或独立准备）。
(5) 给国际软件和系统工程界，促进软件和系统测试的专业化，并且作为书籍和文章的基础之一。

ISTQB®允许其他组织为其他目的而使用本大纲，只要他们事先获得 ISTQB®书面许可。

0.2 软件测试领域认证测试工程师基础级

基础级资质认证可以针对和软件测试工作相关的任何角色，包括测试员、测试分析师、测试工程师、测试顾问、测试经理、用户验收测试员和软件开发人员等。同时本基础级资质认证也适合希望对软件测试有所了解的人，比如产品负责人、项目经理、质量经理、软件开发经理、业务分析师、IT 主管和管理顾问等。拥有基础级认证证书后，可以继续向高级软件测试资质认证努力。

ISTQB® 认证测试工程师基础级大纲概述 2018 版是一份单独的文档，它对于基础级大纲 2018 V3.1 仍然有效，它包括下列内容：

- 本大纲的商业价值
- 商业价值和学习目标之间的对应矩阵
- 本大纲的概要

0.3 可考核的学习目标和知识认知级别

学习目标支持商业价值，并且用于生成认证测试员基础级考试题。

本大纲中除了简介和附录外，考核通常包含了所有 K1 级别的内容。因此，应试者可能会被考到本大纲中要求识别（recognize）、牢记（remember）或记忆（recall）的关键词或概念。在本大纲中，每章开始都会给出相应学习目标和知识认知级别要求：

- K1：牢记（remember）
- K2：理解（understand）
- K3：应用（apply）

更多的细节和学习目标的例子可以参考附录 B。

章节标题下面列出的所有术语的定义作为关键词需要牢记（K1），即使在学习目标中没有非常明显的涉及。

0.4 基础级认证考试

基础级认证考试的内容将基于本大纲的内容。但是考试中涉及到的问题，可能需要用到本大纲的一个甚至多个节的知识。考试的范围覆盖本大纲的所有节，除了简介和附录。标准、书籍和其他ISTQB®大纲只是作为参考,但其内容除了从这些标准、书籍和其他 ISTQB®大纲中总结出的内容之外，是不作为考试范围的。

考题的形式是单项选择题。共有 40 题。必须至少答对 65%题目（即 26 题）才算通过考试。

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：在授权的考试中心）。参加完成认证培训课程不作为考试的先决条件。

0.5 授权

ISTQB®成员国委员会（ISTQB®在中国的分会为：CSTQB®）可以授权那些遵照本大纲编写课程材料的培训机构。授权指南可以从成员国委员会（ISTQB®在中国的分会：CSTQB®）获取，或者从开展授权的机构或团体获取。被授权课程需要遵循本大纲，并且允许将 ISTQB®考试作为课程的一部分。

0.6 详细级别

本大纲的详细级别允许全球范围内一致的教学和考核。为了达到这个目标，本大纲由下面几部分组成：

- 总体教学目标，描述了基础级水平的目的；
- 列出了学生必须能记忆（recall）的术语；
- 各个知识领域的学习目标，描述需要达到的学习认知输出；
- 关键概念描述，包括来源参考，例如：已认可的文献和标准。

本大纲并没有包含软件测试的整个知识领域，只是提供了基础级培训课程需要覆盖的详细级别。本大纲关注能应用到所有软件项目（包括敏捷项目）的测试概念和技术。本大纲不包含与任何特定软件开发生存周期或方法相关的学习目标，但会讨论怎么应用这些概念到敏捷项目、其他迭代和增量生存周期类型，以及顺序生存周期类型。

0.7 课程大纲的结构

本大纲主要由包含考试内容的 6 章组成。每一章的第一级标题明确本章的授课时间，章级别以下不再提供授课时间。对认可的培训课程，本大纲要求至少 16.75 小时的授课时间，下列是整个 6 章的课时分布：

- 第 1 章：175 分钟 软件测试基础
- 第 2 章：100 分钟 软件开发生存周期中的测试
- 第 3 章：135 分钟 静态测试
- 第 4 章：330 分钟 测试技术
- 第 5 章：225 分钟 测试管理
- 第 6 章：40 分钟测试的支持工具

1. 软件测试基础（175 分钟）

关键词

覆盖(coverage)、调试(debugging)、缺陷(defect)、错误(error)、失效(failure)、质量(quality)、质量保证（quality assurance)、根本原因（root cause)、测试分析（test analysis)、测试依据（test basis)、测试用例(test case)、测试结束(test completion)、测试条件(test condition)、测试控制（test control)、测试数据（test data)、测试设计（test design)、测试执行（test execution)、测试实施（test implementation)、测试监督（test monitoring)、测试对象（test object)、测试目标(test objective)、测试结果参照物(test oracle)、测试计划(test planning)、测试规程(test procedure)、测试过程(test process)、测试套件(test suite)、测试(testing)、测试件（testware)、可追溯性（traceability)、确认（validation)、验证（verification）

软件测试基础的学习目标

1.1 什么是测试?

FL-1.1.1 (K1) 识别典型的测试目标

FL-1.1.2 (K2) 区分测试与调试的不同

1.2 为什么需要测试?

FL-1.2.1 (K2) 给出为什么需要测试的例子

FL-1.2.2 (K2) 描述测试与质量保证之间的关系，举例说明测试如何提高软件质量

FL-1.2.3 (K2) 辨别错误、缺陷和失效

FL-1.2.4 (K2) 辨别引起缺陷的根本原因及其影响

1.3 七项测试的基本原则

FL-1.3.1 (K2) 解释测试的七项基本原则

1.4 测试过程

FL-1.4.1 (K2) 解释测试过程中的环境影响

FL-1.4.2 (K2) 描述测试过程中的测试活动和各自的任务

FL-1.4.3 (K2) 区分用于支持测试过程的工作产品

FL-1.4.4 (K2) 解释在测试依据和测试工作产品之间保持可追溯性的价值

1.5 测试的心理学

FL-1.5.1 （K1）识别影响测试成功与否的心理因素

FL-1.5.2 （K2）解释测试活动所需的思维方式和开发活动所需的思维方式之间的差异

1.1. 什么是测试

软件系统是生活中不可或缺的一部分，包括从商业应用（比如银行系统）到消费产品（比如汽车）的各个领域。然而，很多人都有过这样的经历：软件并没有按照预期进行工作。不能正常工作的软件会导致许多问题，包括资金、时间和商业声誉的损失，甚至是伤害或死亡。软件测试是评估软件质量和降低软件运行中出现失效风险的一种方法。

对测试的常见误解是，它只包含了运行测试，即执行软件和检查结果。如第 1.4 节所述，软件测试是一个包含了许多不同活动的过程；测试执行(包括检查结果)只是这些活动之一。测试过程还包括诸如测试计划、测试分析、测试设计、测试实施、报告测试进度和结果，以及评估测试对象的质量等活动。

有些测试确实涉及到被测组件或系统的执行，这种测试称为动态测试。不涉及运行被测组件或系统的测试，称为静态测试。所以，测试还包括评审工作产品，例如需求、用户故事和源代码。

另一个关于测试的常见误解是，它完全关注于需求、用户故事或其他规格说明的验证。虽然测试确实涉及检查系统是否满足指定的需求，但它也包含确认，即检查系统在其运行的环境中是否满足用户和其他利益相关方的需求。

测试活动在不同的生存周期中的组织和实施是不同的(参见第 2.1 节)

1.1.1. 典型的测试目标

对于给定的任何项目，其测试目标可以包括

- 通过评估工作产品以防止缺陷，例如需求、用户故事、设计和代码
- 验证是否实现了所有指定的需求
- 检查测试对象是否完成，并确认是否按照用户和其他利益相关方期望那样工作
- 建立对被测对象质量级别的信心
- 发现缺陷和失效，从而降低软件质量不足的风险
- 为利益相关方提供足够的信息以允许他们做出明智的决策，特别是关于测试对象的质量级别
- 遵守合同、法律或法规要求或标准，和/或验证测试对象是否符合这些要求或标准

根据被测组件或系统的环境、测试级别和软件开发生存周期模型的不同，测试目标会有所变化。不同包括：

- 在组件测试时，尽可能多的发现失效，以便尽早识别和修复潜在的缺陷可能是其一个目标。而另一个目标可能是增加组件测试时的代码覆盖率。
- 在验收测试时，确认系统能够按照预期工作并且满足用户需求可能是其一个目标。而另一个测试目标可能是为利益相关方提供关于在给定时间发布系统的风险信息。

1.1.2. 测试与调试

测试和调试是两个不同的概念。执行测试可以发现由于软件缺陷引起的失效。而调试是发现、分析和修复这些缺陷的开发活动。随后的确认测试检查修复活动是否解决了缺陷。有的时候，测试员负责开始及最终的确认测试，而开发人员则负责调试、相关组件和组件的集成测试（持续集成）。然而，在敏捷开发和其他的一些软件开发生存周期中，测试员也可能会参与调试和组件测试。

ISO 标准（ISO/IEC/IEEE 29119-1)）提供了更多关于软件测试概念的信息。

1.2. 为什么需要测试？

对组件和系统及其相关文档进行严格的测试，有助于降低软件运行过程中出现失效的风险。当发现缺陷并随后加以修复时，这有助于提高组件或系统的质量。此外，还可能需要进行软件测试，以满足合同或法律法规或行业具体标准的要求。

1.2.1. 测试对成功的贡献

纵观计算机的历史，软件和系统的交付使用是很常见的，但是由于缺陷的存在，随后就会导致软件和系统的失效，或者没有满足利益相关方的需求。然而，使用适当的测试技术可以减少这种有问题交付的频率，只要这些技术是在适当的测试技能水平、适当的测试级别和软件开发生存周期的适当点上得到应用。例如:

- 让测试员参与需求评审或用户故事细化，可以发现这些工作产品中的缺陷。识别和修复需求缺陷可以减少被开发（功能）特征的不正确或不可测试的风险。
- 在系统设计过程中，让测试员与系统设计人员密切合作，可以提高各方对设计和如何测试的理解。理解的增加可以降低基本设计缺陷的风险，并使测试能够尽早介入。
- 在开发代码过程中，让测试员与开发人员密切合作，可以提高各方对代码以及如何测试的理解。理解的增加可以降低代码和测试中出现缺陷的风险。
- 让测试员在发布之前对软件进行验证和确认，可能发现之前遗漏的失效，并帮助修复导致失效的缺陷(即调试)。这样就增加了软件满足利益相关方需求的可能性。

除了这些例子之外，达到已定义测试目标(参见第 1.1.1 节)有助于整个软件开发和维护的成功。

1.2.2. 质量保证和测试

人们经常使用“质量保证”(或 QA)来代指测试，虽然它们是有关联的，但是质量保证和测试是不一样的。可以用更大的概念把它们联系在一起，质量管理。质量管理包括所有指导和控制组织质量的活动。

除其他活动外，质量管理还包括质量保证和质量控制。质量保证的关注点在于遵循正确的过程，为产品能够达到合适的质量级别提供信心。当过程正确开展时，这些过程所创造的工作产品通常具有更高的质量，这也有助于缺陷的预防。另外，使用根本原因分析方法来发现缺陷并消除引起缺陷的原因，以及适当应用回顾性会议的结论来改进过程，对于有效的质量保证都也很重要。

质量控制涉及各种支持达到适当质量级别的活动，包括测试活动。测试活动是整个软件开发或维护过程的一部分。因为质量保证涉及到整个过程的正确执行，质量保证会支持正确的测试活动。如第 1.1.1 节和 1.2.1 节所述，测试在以各种方式帮助实现质量的提高。

1.2.3. 错误、缺陷和失效

所有人都会犯错误(mistake)，这样就会导致在软件代码或者其他相关工作产品中引入缺陷(fault 或 bug)。在一个工作产品中引入的缺陷就可能会导致其他相关工作产品都引入缺陷。例如，需求引发的错误就可能导致需求缺陷，需求缺陷就会导致编程错误，这样代码中就存在缺陷。

如果执行了存在缺陷的代码，就可能导致失效，但不一定在所有情况下都是这样。例如，有些缺陷需要非常特殊的输入或先决条件才能触发失效，这种失效可能很少发生，也可能永远不会发生。

发生错误的原因有很多种，例如：

- 时间压力
- 人本身容易犯错
- 缺乏经验或技能不足的项目参与者
- 项目参与者之间沟通有误，包括需求和设计之间的沟通误解。
- 代码、设计、架构的复杂度，待解决的潜在问题，和/或使用的技术
- 对系统内和系统间接口的误解，特别是当系统内和系统间的交互数量比较多的时候
- 新的不熟悉的技术

除了代码中的缺陷导致的失效之外，环境条件也可能导致失效。例如：辐射、电磁场和污染等都有可能引起固件中的失效，或者由于硬件环境的改变而影响软件的执行。

但并非所有意外的测试结果都属于失效。由于测试执行方式的错误，或者由于测试数据、测试环境或其他测试件中的缺陷，又或者由于其他的原因，可能会出现假阳性结果（误报）。相反的情况也有可能发生，即相似的错误或缺陷会导致假阴性结果（缺陷的漏报）。假阴性结果指的是没有发现测试应该要发现的缺陷；假阳性结果记录为缺陷，但实际上并不是缺陷。

1.2.4. 缺陷、根本原因和影响

缺陷的根本原因是导致缺陷产生的最早的行为或条件。可以分析缺陷并找出其根本原因，以减少类似的缺陷以后再发生。通过将关注点放在最重要的根本原因，根本原因的分析可以促进过程的改进，从而防止将来引入大量的缺陷。

例如，假设由于一行不正确的代码，支付了错误的利息，导致了客户投诉。由于产品负责人对如何计算利息有误解，所以为模糊的用户故事编写了有缺陷的代码。如果在利息计算中存在很大比例的缺陷，并且引发这些缺陷的根本原因来源于类似的误解，那么需要为产品负责人进行利息计算相关主题的培训，以便在未来减少这类缺陷。

在这个例子中，客户投诉的是缺陷导致的影响。支付错误的利息属于失效。代码中的错误计算属于缺陷，它是由模糊的用户故事中的原始缺陷造成的。原始缺陷产生的根本原因是产品负责人知识的缺乏，导致产品负责人在编写用户故事时犯了错误。在 ISTQB-CTEL-TM 和 ISTQB-CTEL-ITP 大纲中讨论了根本原因分析过程。

1.3. 七项测试的基本原则

过去 50 年来，人们提出了一些测试原则，并为所有测试提供了通用的指南。

原则 1 测试说明缺陷的存在，而不能说明缺陷不存在

测试可以证明存在缺陷，但不能证明不存在缺陷。测试降低了软件中存在未发现缺陷的可能性，但即使没有发现缺陷，测试也无法证明其对象的正确性。

原则 2 穷尽测试是不可能的

进行穷尽测试（输入和前提条件的所有组合）是不可行的，除非是小型的案例。应利用风险分析、测试技术和优先级确定测试工作量，而不是试图进行穷尽测试。

原则 3 测试的尽早介入可以节省时间和成本

为了尽早发现缺陷，应该在软件开发生存周期中尽早启动静态和动态测试活动。测试的尽早介入有时被称为测试的左移。在软件开发生存周期的早期进行测试有助于减少或消除代价高昂的变更（参见第 3.1 节）。

原则 4 缺陷的群集效应

通常在少数模块中包含了大部分在发布前测试中发现的缺陷，或者是造成大部分运行失效的原因。预测的缺陷集群和在测试或操作中实际观察到的缺陷集群，应该作为风险分析的重要输入，并用来集中测试工作量（如原则 2 所述）。

原则 5 杀虫剂悖论

如果多次重复同样的测试，最终这些测试将不再能够发现任何新的缺陷。为了发现新的缺陷，可能需要更改现有的测试用例和测试数据，并且可能需要编写新的测试。（测试不再能有效发现缺陷，就像杀虫剂在一段时间后对杀死昆虫不再有效一样）。但是在某些情况下，杀虫剂悖论也有好处，例如在自动化回归测试中，发现的回归缺陷数量相对较少。

原则 6 测试活动依赖于测试周境

测试在不同周境下是不同的。比如，安全关键工业控制软件的测试不同于电子商务移动应用。另一个例子，在敏捷项目中进行的测试不同于在顺序软件开发生存周期项目中进行的测试（见第 2.1 节）。

原则 7 不存在缺陷的谬论

有些组织期望测试员能够运行所有可能的测试并发现所有可能的缺陷，但是原则 2 和原则 1 分别告诉了我们这是不可能的。另外，期望仅仅发现并修复大量缺陷就能确保系统的成功，这是一个谬论（即错误的信念）。例如，穷尽测试所有指定的需求并修复发现的所有缺陷，仍然可能会生产出一个难以使用，或无法满足用户需求和期望，或与其他竞争产品相比更差的系统。

更多测试原则的实例及其他测试原则，请参见 Myers 2011，Kaner 2002，Weinberg 2008，和 Beizer 1990。

1.4. 测试过程

尽管没有统一的软件测试过程，但是有一些常见的测试活动，如果没有这些测试活动就不太可能实现既定的目标。这些测试活动就组成了一个测试过程。在任何给定的情况下，适当的、特定的软件测试过程取决于很多因素。

测试过程中涉及哪些测试活动，如何实施这些测试活动，以及何时开始这些测试活动，都可以在组织的测试策略中进行讨论。

1.4.1. 周境中的测试过程

影响组织测试过程的周境因素包括，但不仅限于以下：

- 使用的软件开发生存周期模型及项目所使用方法
- 考虑的测试级别和测试类型
- 产品风险和项目风险
- 业务领域
- 运行限制，包括但不仅限于：
  - 预算和资源
  - 时间
  - 复杂度
  - 合同和法规要求
- 组织方针和实践
- 所需的内部和外部标准

后面章节就以下几点描述了组织中测试过程的通用方面：

- 测试活动和测试任务
- 测试工作产品
- 测试依据和测试工作产品之间的可追溯性

如果测试依据（对于正在考虑的任何测试级别或类型）具有可度量的覆盖标准，那是非常有用的。覆盖标准可以有效地作为关键绩效指标(KPIs)，推动显示软件测试目标实现的活动(参见第 1.1.1 节)

例如，对于移动应用程序，测试依据可能包括需求列表和支持的移动设备列表。每个需求都是测试依据的一个元素。覆盖标准可能要求测试依据的每个元素至少有一个测试用例。执行测试时，这些测试的结果将告诉利益相关方是否实现了指定的需求，以及是否在受支持的设备上观察到了失效。

有关于测试过程的更多信息参见 ISO 标准(ISO/IEC/IEEE 29119-2)。

1.4.2. 测试活动和测试任务

测试过程主要由以下主要的活动组所组成：

- 测试计划
- 测试监督与控制
- 测试分析
- 测试设计
- 测试实施
- 测试执行
- 测试结束

每个活动组都是由多个活动构成，具体内容将在下面的小节中加以说明。每个组成的活动都可能由多个单独的任务组成，这些任务可能因项目或版本发布而不同。

此外，虽然这些主要活动组中的许多活动在逻辑上看起来是有顺序的，但它们通常是迭代实现的。例如，敏捷开发涉及到软件设计、构建和测试的小迭代，这些迭代都是在持续计划的支持下连续进行的。所以，在这种软件开发的方法中，测试活动也是在迭代的、持续的基础上进行。即使在顺序软件开发中，主要活动的阶梯式逻辑顺序也会涉及重叠、组合、并发或省略，所以必须根据系统和项目的周境因素裁剪这些主要活动。

测试计划

测试计划包括的活动有定义测试目标以及在周境因素限制下达到测试目标的方法(例如，指定适合的测试技术和适当的测试任务，并制定满足截止期限的测试进度表)。根据测试监督与控制活动的反馈，可以重新审议测试计划。测试计划会在第 5.2 节中进一步讲述。

测试监督与控制

测试监督包括使用测试计划中定义的测试监督度量，对实际进度与计划的进展进行持续的比较。测试控制包括采取必要的措施来满足测试计划的目标(目标可能会随时间的变化有所更新)。测试监督与控制是通过评估出口准则来支持的，出口准则在一些软件开发生存周期模型中被称为“完成的定义”(参见 ISTQB-CTFL-AT)。例如，作为给定测试级别的一部分，对测试执行的出口准则评估可能包括:

- 根据指定的覆盖准则检查测试结果和日志。
- 根据测试结果和日志评估组件或系统的质量级别。
- 确定是否需要做更多的测试(例如，如果原本旨在达某个级别的产品风险覆盖率的测试失败了，则需要编写和执行额外的测试)

在测试进度报告中向利益相关方通报测试进度，包括偏离计划的情况和帮助决定结束测试的信息。测试监督与控制将在第 5.3 节中进一步讲述。

测试分析

在测试分析过程中，分析测试依据以识别可测试特征和定义相关的测试条件。换句话说，测试分析根据可测量的覆盖标准来确定“测试什么”。

测试分析主要包括以下活动:

- 分析所考虑测试级别的测试依据，例如：
  - 需求规格说明，如业务需求、功能需求、系统需求、用户故事、史诗、用例或类似的工作产品，其中指定了所需的功能和非功能的组件或系统行为
  - 设计和实现信息，如系统或软件架构图或文档、设计规格说明、调用流图、模型图(例如 UML 或实体关系图 ERD)、接口规格说明或类似的工作产品，其中指定了组件或系统的结构
  - 组件或系统本身的实现，包括代码、数据库元数据和查询以及接口
  - 风险分析报告，其考虑了组件或系统的功能、非功能及结构问题
- 评估测试依据和测试项，以识别各种类型的缺陷，例如:
  - 歧义
  - 遗漏
  - 不一致
  - 不准确
  - 矛盾

- 多余的表述
- 识别被测特征和特征集
- 根据对测试依据的分析，并考虑到功能、非功能和结构特征、其他业务和技术因素以及风险级别，界定每个特征的测试条件并确定其优先次序
- 在测试依据的每个元素与相关测试条件之间获取双向可追溯性(参见第1.4.3节和第1.4.4节)

在测试分析过程中，使用黑盒测试技术、白盒测试技术、基于经验的测试技术(参见第 4 章)，可以减少遗漏重要测试条件的可能性，并且能更精确和准确的定义测试条件。

在某些情况下，测试分析得到的测试条件，这些条件将作为测试章程中的测试目标。在某些基于经验的测试中，测试章程是典型的工作产品(参见第 4.4.2 节)。当这些测试目标可以追溯到测试依据时，就可以测量基于经验的测试中实现的覆盖率。

在测试分析过程中识别缺陷是一个重要的潜在收益，特别是在没有使用其他评审过程和/或测试过程与评审过程之间间隔很短的情况下。这样的测试分析活动不仅要验证需求是否一致、是否表达正确以及是否完整，还需验证需求是否正确满足客户、用户和其他利益相关方的需求。例如，行为驱动开发(BDD)和验收测试驱动开发(ATDD)等技术，它们都需要在编码之前，从用户故事和验收标准中生成测试条件和测试用例，这些技术同样要在用户故事和验收标准中验证、确认和发现缺陷(参见ISTQB-CTFL-AT)。

测试设计

在测试设计时，将测试条件细化成概要测试用例、概要测试用例集和其他测试件。因此，测试分析回答了“测试什么？”的问题，而测试设计是回答了“如何测试？”的问题。

测试设计包括以下的主要活动:

- 设计测试用例和测试用例集，并确定其优先级
- 识别所需的测试数据以支持测试条件和测试用例
- 设计测试环境并识别所需的基础设施和工具
- 撷取测试依据、测试条件和测试用例之间的双向追溯性(参见第 1.4.4 节)

在测试设计过程中，将测试条件细化为测试用例和测试用例集，常常会涉及到使用测试技术(参见第 4 章)。

与测试分析一样，测试设计也可能在测试依据中识别出相似类型的缺陷。与测试分析一样，测试设计过程中识别出缺陷也是其重要的潜在效益。

测试实施

在测试实施期间，创建和/或完成测试执行所需的测试件，包括将测试用例排序为测试规程。所以，测试设计回答了“如何测试”的问题，而测试实施则回答了“我们现在是否已经有了运行测试所需的一切条件?”

测试实施包括以下主要活动:

- 开发并确定测试规程的优先级，如有可能，并创建自动化测试脚本
- 根据测试规程和(如果有的话)自动测试脚本来创建测试套件。
- 在测试执行进度表中，以促进有效测试执行的方式安排测试套件(参见第 5.2.4 节)
- 构建测试环境(包括可能的，测试工具、服务虚拟化、模拟器和其他基础设施项目)，并验证一切所需都已正确设置

- 准备测试数据并确保在测试环境中正确地加载
- 确认并更新测试依据、测试条件、测试用例、测试规程和测试套件之间的双向可追溯性(参见第 1.4.4 节)

测试设计和测试实施任务通常是结合在一起的。

在探索性测试和其他基于经验的测试类型中，测试设计和实施可能作为测试执行的一部分得到执行和记录。探索性测试可以基于测试章程(作为测试分析工作产品的一部分)，在探索性测试时直接进行设计和实施(参见第 4.4.2 节)。

测试执行

在测试执行期间，测试套件按照测试执行进度表运行。

测试执行包括以下主要活动:

- 记录测试项或测试对象、测试工具及测试件的 ID 和版本
- 手工或者使用测试执行工具执行测试
- 将实际结果与预期结果进行比较
- 分析异常现象以确定它们可能发生的原因(例如，出现失效可能是由于代码中的缺陷，但也可能出现假阳性结果（误报）(参见第 1.2.3 节))
- 根据实际观察到的失效报告缺陷(参见第 5.6 节)
- 记录测试执行的结果(例如通过、失败、阻塞)
- 作为对异常现象采取行动的结果，或作为计划要测试的一部分(例如，执行修正后的测试、确认测试和/或回归测试)，重复测试活动
- 确认并更新测试依据、测试条件、测试用例、测试规程和测试结果之间的双向可追溯性

测试结束

测试结束活动从已完成的测试活动中收集数据，以强化经验、测试件，以及任何其他相关信息。测试结束活动出现在项目里程碑点，例如：当软件系统发布时、当测试项目完成（或取消）时、当敏捷项目的迭代完成时、当测试级别完成时，或当维护版本完成时。

测试结束包括以下主要活动：

- 检查是否所有的缺陷报告已关闭，为测试执行结束时仍未解决的缺陷，是否已创建需求变更或产品待办事项
- 创建测试总结报告，并将信息传达给利益相关方
- 最后确定并归档测试环境、测试数据、测试基础设施及其他相关测试件，以便以后重复使用
- 将测试件移交给维护部门、其他项目团队和/或其他可以从使用测试件中获益的利益相关方
- 从已完成的测试活动中，分析所获得的经验教训来确定以后迭代、版本和项目所需的变更
- 使用收集到的信息来改进测试过程的成熟度

1.4.3. 测试工作产品

测试工作产品作为测试过程中的一部分被创建。正如不同的组织在实施测试过程的方式时存在明显差异一样，在测试过程中创建的工作产品的类型也存在明显差异，这些工作产品的组织和管理

方式，以及这些工作产品的名称也都存在明显差异。本大纲遵循上述测试过程以及本大纲和 ISTQB® 术语表中描述的工作产品。ISO 标准(ISO/IEC/IEEE 29119-3)也可以作为测试工作产品的指南。

本节所描述的很多测试工作产品都是可以使用测试管理工具和缺陷管理工具来获取和管理的(参见第 6 章)。

测试计划工作产品

测试计划的工作产品通常包括一个或多个测试计划。测试计划包括关于测试依据与其他测试工作产品之间可追溯性的相关信息(参见下面内容及第 1.4.4 节)，以及测试监督与控制期间使用的出口标准(或“完成的定义”)。测试计划描述参见第 5.2 节。

测试监督与控制工作产品

测试监督与控制的工作产品通常包括各种类型的测试报告，包括测试进度报告(持续和/或定期生成的)和测试总结报告(在各种已结束里程碑上生成的)。所有的测试报告都应该提供在截止日期前与测试受众相关的细节，包括一旦获得测试执行结果后的总结。

测试监督与控制的工作产品还应该解决项目管理所关心的问题，例如任务完成度、资源的分配和使用，以及工作量。

测试监督与控制，及在这些活动中创建的工作产品，将会在本课程大纲第 5.3 节做进一步解释。

测试分析工作产品

测试分析工作产品包括已定义的和按优先级排序的测试条件，理想情况下每一个测试条件都可以双向追溯到它所覆盖的测试依据的特定元素。对于探索性测试，测试分析可能涉及到创建测试章程。测试分析还可能发现和报告在测试依据上的缺陷。

测试设计工作产品

测试设计生成测试用例和测试用例集，以覆盖测试分析中定义的测试条件。通常设计概要测试用例是一种很好的做法，概要测试用例里没有具体的输入数据和预期结果的值。这样的概要测试用例可以在使用不同具体数据的多个测试周期中重复使用，同时仍能够充分记录测试用例的范围。理想情况下，每个测试用例都可以双向追溯到其覆盖的测试条件。

测试设计的工作产品包括：

- 设计和/或识别必要的测试数据
- 设计测试环境
- 识别基础设施和工具

记录的这些结果的范围差别会很大。

测试实施工作产品

测试实施工作产品包括:

- 测试规程以及这些测试规程的顺序
- 测试套件

- 测试执行进度表

理想情况下，一旦测试实施活动完成，就可以基于测试用例和测试条件，通过测试规程和测试依据的特定元素之间的双向可追溯性，来证明测试计划中确定的覆盖率准则是否实现。

在某些情况下，测试实施涉及使用工具创建工作产品，例如服务虚拟化和自动化测试脚本。

测试实施还可能需要创建和验证测试数据和测试环境。数据和/或环境验证结果的文档完整性可能会有很大差异。

测试数据为测试用例的输入和预期结果指定具体的值。这些具体的值，以及使用具体值的明确指示，将概要测试用例转换为可执行的详细测试用例。当在测试对象的不同版本上执行时，相同的概要测试用例可能使用不同的测试数据。使用测试结果参照物来识别与具体的测试数据相关的明确的预期结果。

在探索性测试中，可以在测试执行的过程中创建某些测试设计和测试实施的工作产品，尽管探索性测试文档化的范围(以及它们对测试依据中的特定元素的可追溯性)可能会差异很大。

测试分析中定义的测试条件可以在测试实施中进一步细化。

测试执行工作产品

测试执行工作产品包括:

- 记录各单个测试用例或测试规程状态的文档(例如准备运行、通过、失败、阻塞、故意跳过等)
- 缺陷报告(参见第 5.6 节)
- 测试过程中包含测试项、测试对象、测试工具及测试件的文档

理想情况下，一旦完成了测试执行，可以通过与测试规程相关的双向可追溯性来确定和报告测试依据中每个元素的状态。例如，我们可以说哪些需求通过了所有计划的测试，哪些需求的测试失败了，和/或有与该需求相关的缺陷，以及哪些需求的测试已经计划好了但在等待运行。这样就可以验证是否满足了覆盖标准，以及确认利益相关方是否可以理解这些测试结果报告。

测试结束工作产品

测试结束的工作产品包括测试总结报告、改进后续项目或迭代的行动项、变更请求或产品待办事项，以及最终的测试件。

1.4.4. 测试依据和测试工作产品之间的可追溯性

如第 1.4.3 节描述的，测试工作产品以及这些工作产品的名称差别很大。无论差异如何，为了实施有效的测试监督与控制，如上所述，在测试依据的每个元素和与该元素相关联的各种测试工作产品之间建立和维护整个测试过程的可追溯性是很重要的。除了评估测试的覆盖率外，良好的可追溯性支持：

- 分析变更的影响
- 测试的可审计
- 符合 IT 管理标准
- 提高测试进度报告和测试总结报告的易理解性，包括测试依据中元素的状态。(例如通过测试的需求、未通过测试的需求以及有待完成测试的需求)

- 将测试的技术方面与利益相关方关联起来，使利益相关方能够理解
- 根据业务目标，提供评估产品质量、过程能力和项目进展的相关信息

某些测试管理工具提供了与本节中描述的部分或全部测试工作产品相匹配的测试工作产品模型。有些组织建立了自己的管理系统来管理工作产品，并提供他们所需的可追溯性信息。

1.5. 测试心理学

软件开发，包括软件测试，都涉及到人的参与。因此，人的心理对软件测试有着重要的影响。

1.5.1. 心理学和测试

不论是在静态测试时识别缺陷，例如需求评审或用户故事细化，又或者是在动态测试执行时识别失效，都可能会被看作是对产品及其作者的批评。人类心理学中有个叫做“确认偏见”的原理，会让人难以接受与目前所持有的信仰相悖的信息。例如，由于开发人员希望他们的代码是正确的，但是他们存在确认偏见，这使得他们很难接受代码是错误的。除了确认偏见之外，其他的认知偏见都可能使得人们难以理解或接受测试产生的信息。而且，指责坏消息的传递者是人类的共同特征，而测试产生的信息往往包含了坏消息。

虽然测试对项目进展和产品质量有很大的贡献(参见第 1.1 节和第 1.2 节)，但是由于这些心理因素，还是会有人认为测试是一种破坏性的活动。为了减少这些观念所带来的影响，应该以建设性的方式去沟通关于缺陷和失效的相关信息。这样就可以缓解测试员和分析人员、产品负责人、设计人员和开发人员之间的紧张关系。这同时适用于静态测试和动态测试。

测试员和测试经理需要有良好的沟通技巧，才能够有效地沟通缺陷、失效、测试结果、测试进度和风险，并与同事建立积极的关系。良好沟通方式的例子包括：

- 从合作而不是争斗的方式开始项目。提醒项目的每位成员，大家的共同目标是追求更高质量的系统。
- 强调测试的好处。例如，对于作者而言，缺陷信息可以帮助他们改进他们的工作产品和他们的技能。对于组织而言，测试过程中发现并修复缺陷可以节省时间和金钱，并降低产品质量的总体风险。
- 以中性的、以事实为中心的方式去沟通测试结果和其他发现，而不要去指责引入该缺陷项的人员。编写客观且实际的缺陷报告和评审发现的问题。
- 尽量理解其他成员的感受，以及他们为什么对信息反应消极的原因。
- 确认其他成员已经理解了你的描述，反之亦然。

前面讨论了典型的测试目标(参见第 1.1 节)。清楚地定义正确的测试目标具有重要的心理暗示作用。大多数人倾向于将他们的计划和行动与团队、管理人员和其他利益相关方设定的目标保持一致。同样重要的是，测试员要以最低限度的个人偏见去坚持这些目标。

1.5.2. 测试和开发的思维方式

开发人员和测试员通常有不同的想法。开发的主要目标是设计并构建一个产品。如前文所述，测试的目标包括验证和确认产品，在发布之前发现缺陷等。因为这些不同的目标，就需要不同的思

维方式。将这些不同的思维方式结合在一起，有助于提高产品的质量。

思维方式反映了个人的假设以及作出决策和解决问题的首选方法。测试员的思维方式应该包括好奇心、职业的悲观主义、批判性的眼光、对细节的关注，以及良好和积极的沟通和人际关系的动机。随着测试员经验越来越丰富，测试员的思维方式也在不断成长和成熟。

开发人员的思维方式中可能会包含一些测试员思维方式中的元素，但成功的开发人员通常对设计和研究解决方案更感兴趣，而不是去考虑这些解决方案中可能存在的问题。此外，确认偏见使他们很难会意识到自己犯下的错误。

有了正确的思维方式，开发人员就可以自己测试自己的代码。不同的软件开发生存周期模型，通常有不同的组织测试员和测试活动的方式。由独立的测试员进行的一些测试活动可以提高缺陷检测的有效性，这对大型、复杂或安全关键系统尤其重要。因为测试员与作者有不同的认知偏见，所以独立的测试员带来的视角不同于工作产品的作者（例如，业务分析师、产品负责人、设计师和开发员）。

2. 软件开发生存周期中的测试（100 分钟）

关键词

验收测试（acceptance testing）、alpha 测试（alpha testing）、beta 测试（beta testing）、变更相关的测试（change-related testing）、商业现货软件（COTS）（commercial off-the-shelf）、组件集成测试（component integration testing）、组件测试（component testing）、确认测试（confirmation testing）、合同验收测试（contractual acceptance testing）、功能测试（functional testing）、影响分析（impact analysis）、集成测试（integration testing）、维护测试（maintenance testing）、非功能测试（non-functional testing）、运行验收测试（operational acceptance testing）、回归测试（regression testing）、法规验收测试（regulatory acceptance testing）、顺序开发模型（sequential development model）、系统集成测试（system integration testing）、系统测试（system testing）、测试依据（test basis）、测试用例（test case）、测试环境（test environment）、测试级别（test level）、测试对象（test object）、测试目标（test objective）、测试类型（test type）、用户验收测试（user acceptance testing）、白盒测试（white-box testing）

软件开发生存命周期中的测试的学习目标

2.1 软件开发生存周期模型

FL-2.1.1 （K2）阐述在软件开发生存周期中开发活动和测试活动之间的关系

FL-2.1.2 （K1）解释为什么软件开发生存周期模型必须适应项目周境和产品特性的原因

2.2 测试级别

FL-2.2.1 （K2）从目标、测试依据、测试对象、典型缺陷和失效，以及方法和职责的视角比较不同的测试级别

2.3 测试类型

FL-2.3.1 （K2）比较功能测试、非功能测试和白盒测试

FL-2.3.2 （K1）认识功能测试、非功能测试和白盒测试都可以发生在任何测试级别

FL-2.3.3 （K2）比较确认测试和回归测试的目的

2.4 维护测试

FL-2.4.1 （K2）总结维护测试的触发因素

FL-2.4.2 （K2）描述影响分析在维护测试中的作用

2.1. 软件开发生存周期模型

软件开发生存周期模型描述了软件开发项目中每个阶段要开展的活动类型，以及这些活动是如何在逻辑上和时间上相互关联的。有许多不同的软件开发生存周期模型，每个模型都要求使用不同的测试方法。

2.1.1. 软件开发和软件测试

为了能够进行适当的测试活动，熟悉常见的软件开发生存周期模型是测试员的重要职责。

在任何软件开发生存周期模型中，良好的测试都有以下几个特点：

- 每个开发活动会有对应的测试活动
- 每个测试级别会有对应的特定的测试目标
- 相应的开发活动期间，对特定的测试级别进行测试分析和设计
- 测试员参与讨论，以明确和改善需求和设计，并在初稿完成时立即参与工作产品（例如需求、设计、用户故事等）的评审工作

无论选择哪种软件开发生存周期模型，测试活动都应在生存周期的早期阶段开始，以符合测试的尽早介入原则。

常见的软件开发生存周期模型，本大纲分类如下：

- 顺序开发模型
- 迭代和增量开发模型

顺序开发模型将软件开发过程描述为线性的、顺序的活动流。它是指开发过程中的任何阶段都应该在完成前一阶段的基础上进行。从理论上讲，阶段之间没有重叠，但在实践中，都会受益于来自下一阶段的早期反馈。

在瀑布模型中，开发活动（例如需求分析、设计、编码、测试）是一个接一个完成的。在该模型中，只有在完成所有其他开发活动之后，才会进行测试活动。

与瀑布模型不同，V-模型将测试过程集成到了整个开发过程中，满足了尽早测试的原则。此外，V-模型还包括与每个相应开发阶段相对应的测试级别，这进一步支持了尽早测试（关于测试级别的讨论请参见第 2.2 节）。在该模型中，与每个测试级别相关联的测试执行是按顺序方式进行的，但在某些情况下会发生重叠。

顺序开发模型交付的软件包含了完整的特征集，但通常需要几个月或几年的时间才能交付给利益相关方和用户。

增量开发包括建立需求、设计、构建和测试一个系统的各个部分，这意味着软件的特征在不断增加。这些特征增量的大小各不相同，有些方法的增量很大，而有些方法的增量很小。特征增量的大小可以小到是对用户界面或新的查询选项的修改。

迭代开发发生在多个特征在一系列周期中一起被指定、设计、构建和测试的时候。迭代可能涉及早期迭代中发生功能变更，以及更改项目范围。每次迭代都会交付工作软件，这是整个特征集不断增长的子集，直到交付最终软件或停止开发。

示例包括：

- 统一软件开发过程（RationalUnifiedProcess）：每次迭代往往相对较长（例如，两到三个月），并且特征增量相对较大，例如两组或三组相关特征。
- 迭代增量框架（Scrum）： 每次迭代往往相对较短（例如，几个小时、几天或几周），并且特征增量相对较小，例如一些增强特征和/或两个或三个新特征。
- 看板（Kanban）：用或不用固定长度进行迭代，可以在完成时交付单个增强或特征，也可以将特征组合在一起，即刻发布。
- 螺旋式（Spiral）：包括创造实验增量，其中一些可能会被大量返工，甚至在后续的开发工作中被放弃。

在开发过程中，使用这些方法开发的组件或系统通常会涉及到测试级别的重叠和迭代。理想情况下，在每个特征交付之前，需在多个测试级别上进行测试。在某些情况下，团队使用持续交付和持续部署，这两者都涉及到将多个测试级别自动化，并作为其交付管道的一部分。许多使用这些方法的开发工作还包括自组织团队的概念，它改变测试工作的组织方式以及测试员和开发人员之间的关系。

这些方法形成了一个不断增长的系统，这个系统可以基于功能特征、基于迭代或者以更传统的主版本发布方式，交付给最终用户。无论软件增量是否发布给最终用户，随着系统的发展，回归测试变得越来越重要。

与顺序模型相比，迭代和增量模型可以在几周甚至几天内交付可用的软件，但只能在几个月甚至几年的时间内交付完整的需求产品集。

有关敏捷开发周境中软件测试的更多信息，请参见 ISTQB-CTFL-AT，Black 2017，Crispin 2008 和 Gregory 2015。

2.1.2. 周境中的软件开发生存周期模型

软件开发生存周期模型必须根据项目周境和产品特性来选择和调整。应该根据项目目标、正在开发产品的类型、业务优先级（例如上市时机）以及已识别的产品和项目风险，选择和调整合适的软件开发生存周期模型。例如，小型内部管理系统的开发和测试，应该与安全关键系统（如，汽车刹车控制系统）的开发和测试不同。另一个例子是，在某些情况下，组织和企业文化问题可能会阻碍团队成员之间的交流，从而阻碍迭代开发。

根据项目周境，可能需要合并或重组测试级别和/或测试活动。例如，为了将商业现货（COTS）软件产品集成到更大的系统中，采购者可以在系统集成测试级别（例如，集成到基础设施和其他系统中）和在验收测试级别（功能和非功能，以及用户验收测试和运行验收测试）上进行互操作性测试。关于测试级别的讨论，参见第 2.2 节；关于测试类型的讨论，参见第 2.3 节。

此外，软件开发生存周期模型本身可能会组合起来。例如，V-模型可用于后端系统及其集成的开发和测试，而敏捷开发模型可用于开发和测试前端用户界面（UI）和功能。可以在项目的早期使用原型，在试验阶段完成后，就采用增量开发模型。

物联网（IoT）系统由许多不同的对象组成，例如设备、产品和服务，通常会给每个对象应用单独的软件开发生存周期模型，这对物联网系统版本的开发提出了特殊的挑战。此外，在这些对象的软件开发生存周期中更加强调它们被投入使用后的后期阶段（例如，运行、更新和退役阶段）。

软件开发模型必须适应项目和产品特性的原因可以是：

- 系统的产品风险不同（复杂或简单项目）

- 许多业务单元可以是项目或程序的一部分（顺序开发和敏捷开发的结合）
- 一个产品交付市场的时间短（合并测试级别和/或在测试级别中结合不同的测试类型）

2.2. 测试级别

测试级别是一组可以共同组织和管理的测试活动。每个测试级别都是测试过程的一个实例，在特定的开发级别上由第 1.4 节中描述的活动组成，从单元或组件到完整的系统，又或是其他适用的综合系统。测试级别与软件开发生存周期内的其他活动相关。本课程大纲中使用的测试级别有：

- 组件测试
- 集成测试
- 系统测试
- 验收测试

测试级别具有以下属性：

- 具体目标
- 测试依据，基于此导出测试用例
- 测试对象（即正在测试的内容）
- 典型的缺陷和失效
- 特定的方法和职责

每个测试级别都需要合适的测试环境。例如，在验收测试中，类似生产的测试环境是非常理想的，而在组件测试中，开发人员通常使用他们自己的开发环境。

2.2.1. 组件测试

组件测试的目标

组件测试（也称为单元或模块测试）关注在可单独测试的组件。组件测试的目标包括：

- 降低风险
- 验证组件的功能和非功能行为是否符合设计和规定
- 建立对组件质量的信心
- 发现组件中的缺陷
- 防止缺陷遗漏到更高的测试级别

在某些情况下，特别是在增量和迭代开发模型中（如敏捷），代码持续发生着变化，组件的自动化回归测试，在对于变更没有破坏现有组件方面所建立的信心起着关键的作用。

组件测试通常独立于系统其它部分的测试，具体取决于软件开发生存周期模型和该系统，这可能需要模拟对象、服务虚拟化、用具、桩和驱动。组件测试可以包括功能（例如，计算的正确性）、非功能特性（例如，查找内存泄漏）和结构特性（例如，判定测试）。

测试依据

组件测试中可用作测试依据的典型工作产品包括：

- 详细设计
- 代码
- 数据模型
- 组件规格说明

测试对象

组件测试的典型测试对象包括：

- 组件、单元或模块
- 代码和数据结构
- 类
- 数据库模块

典型的缺陷和失效

组件测试发现的典型缺陷和失效包括：

- 功能不正确（例如，不符合设计规格说明中的描述）
- 数据流问题
- 代码和逻辑不正确

组件测试通常没有进行正式的缺陷管理，缺陷通常在发现后立即修复。但是，当开发人员报告缺陷时，这为根本原因分析和过程改进提供了重要信息。

特定的方法和职责

组件测试通常由编写代码的开发人员开展，组件测试是需要访问到被测软件的代码。开发人员可以将组件开发与发现和修复缺陷交替进行。开发人员经常在编写组件代码后编写并执行测试。但是，尤其在敏捷开发中，编写自动化组件测试用例可能先于编写应用程序代码。

例如，考虑测试驱动开发（TDD）。测试驱动开发是高度迭代的，基于开发自动化测试用例、构建和集成小段代码，然后执行组件测试、纠正所有问题并重构代码的循环。该过程将持续到完全构建好组件且通过了所有组件测试。测试驱动开发是测试优先方法的一个例子。虽然测试驱动开发起源于极限编程（XP），但它已经扩展到其他形式的敏捷以及顺序生存周期（参见 ISTQB-CTFL-AT）。

2.2.2. 集成测试

集成测试的目标

集成测试侧重于组件或系统之间的交互。集成测试的目标包括：

- 减少风险
- 验证接口的功能和非功能行为是否符合设计和规定
- 建立对接口质量的信心
- 发现缺陷（可能存在于接口本身，也可能存在于组件或系统内部）
- 防止缺陷遗漏到更高的测试级别

与组件测试一样，在某些情况下，自动化集成的回归测试可以增强信心，因为即使产品有变更也不会破坏已有的接口、组件或系统。

本大纲中描述了两种不同级别的集成测试，可以对不同大小的测试对象进行测试，如下所示：

- 组件集成测试，侧重于集成组件之间的交互和接口。在组件测试之后执行组件集成测试，并且组件集成测试通常是自动化的。在迭代和增量开发中，组件集成测试通常是持续集成过程的一部分。
- 系统集成测试，侧重于系统、软件包和微服务之间的交互和接口。系统集成测试还可以涵盖与外部组织（例如，web 服务）的交互和接口。在这种情况下，开发组织无法控制外部接口，这可能会给测试带来各种挑战（例如，外部组织代码中阻碍测试的缺陷得以解决、安排测试环境等）。系统集成测试可以在系统测试之后进行，也可以和正在进行的系统测试活动并行进行（不论是顺序开发还是迭代和增量开发）。

测试依据

集成测试中可用作测试依据的典型工作产品包括：

- 软件和系统设计文档
- 序列图
- 接口和通信协议规范；
- 用例
- 组件或系统级别的架构
- 工作流
- 外部接口定义

测试对象

集成测试的典型测试对象包括：

- 子系统
- 数据库
- 基础结构
- 接口
- API
- 微服务

典型的缺陷和失效

组件集成测试中典型缺陷和失效包括：

- 数据不正确、数据丢失或数据编码不正确
- 接口调用的顺序或时序不正确
- 接口不匹配
- 组件间通信失效
- 组件间未处理或处理不当的通信失效
- 关于组件间传递的数据含义、数据单位或数据边界的假设不正确

系统集成测试中典型缺陷和失效包括：

- 系统间的消息结构不一致
- 数据不正确、数据丢失或数据编码不正确
- 接口不匹配
- 系统间的通信失效
- 系统间未处理或处理不当的通信失效
- 关于系统间传递的数据定义、数据单元或数据边界的假设不正确
- 未遵守强制性安全规定

特定的方法和职责

组件集成测试和系统集成测试应该集中在集成本身。例如，如果将模块 A 与模块 B 集成，则应侧重于模块之间的通信，而不是单个模块的功能，因为在组件测试期间应该已经覆盖了单个模块的功能。如果将系统 X 与系统 Y 集成，测试应该侧重于系统之间的通信，而不是单个系统的功能，因为在系统测试中应该已经覆盖了单个系统的功能。功能测试、非功能测试和结构测试类型都适用于组件集成测试和系统集成测试。

组件集成测试通常由开发人员负责。系统集成测试通常由测试员负责。理想情况下，开展系统集成测试的测试员应该理解系统架构，并且能够影响到集成计划。

如果在组件或系统构建之前就已经计划了集成测试和集成策略，则组件或系统可以按照最有效测试所需的顺序构建。系统集成策略可以基于系统架构（例如，自上而下和自下而上）、功能任务、事务处理序列或系统或组件的一些其他方面。为了简化缺陷隔离并尽早发现缺陷，集成通常应该是增量式的（即每次增加少量的组件或系统）而不是“大爆炸”式的（即一次性集成所有的组件或系统）。对最复杂的接口进行风险分析有助于针对性的进行集成测试。

集成的范围越大，将缺陷定位到指定的组件或系统就越困难，这可能会增加风险并增加了排错的时间。在持续集成中，软件中的每个组件逐步集成（即功能集成）成为了普遍做法，这就是持续集成的一个原因。这种持续集成通常包括自动回归测试，且在理想情况下是在多个测试级别进行的。

2.2.3. 系统测试

系统测试的目标

系统测试侧重于整个系统或产品的行为和能力，通常会考虑系统可开展的端到端任务和开展这些任务时所展现的非功能行为。系统测试的目标包括：

- 减少风险

- 验证系统的功能和非功能行为是否按照设计和指定的要求进行
- 确认已完成系统成且系统按预期工作
- 建立对整个系统质量的信心
- 发现缺陷
- 防止缺陷遗漏到更高的测试级别或生产

对于某些系统，验证数据质量也可能是一个系统测试的目标。在某些情况下，也会与组件测试和集成测试一样，自动化系统回归测试可以增强信心，即便在代码变更时，也不会破坏现有的特性或端到端的功能。系统测试通常会提供信息给利益相关方用来决策系统是否发布。系统测试也可能是为了满足法律或法规要求或符合标准。

理想情况下，测试环境应与最终目标或生产环境相对应。

测试依据

系统测试中，可以作为测试依据的典型工作产品包括：

- 系统和软件需求规格说明（功能和非功能）
- 风险分析报告
- 用例
- 史诗和用户故事
- 系统行为模型
- 状态图
- 系统和用户手册

测试对象

系统测试的典型测试对象包括：

- 应用程序
- 硬件/软件系统
- 操作系统
- 被测系统（SUT）
- 系统配置和配置数据

典型的缺陷和失效

系统测试的典型缺陷和失效包括：

- 计算不正确
- 不正确的或非预期的系统功能或非功能行为
- 系统内的控制流和/或数据流不正确
- 不能正确完整地开展端到端功能任务
- 系统在系统环境中不能正常工作；
- 系统不能按照系统和用户手册中的说明进行工作

特定的方法和职责

系统测试应侧重整个系统的端到端行为，包括功能和非功能。系统测试应针对被测系统的各个方面使用最合适的技术（参见第 4 章）。例如，可以创建决策表来验证功能行为是否满足业务规则的要求。

系统测试通常由高度依赖规范说明的独立测试员进行测试。规格说明中的缺陷（例如，缺少用户故事、业务需求表述错误等）会导致对预期系统行为缺乏理解或产生分歧。这些情况会导致假阳性结果（误报）和假阴性结果（漏报），误报会导致浪费时间，漏报导致缺陷检测有效性的降低。测试员早期参与用户故事细化或静态测试活动，例如评审，有助于减少此类问题的发生。

2.2.4. 验收测试

验收测试的目标

与系统测试类似，验收测试通常侧重于整个系统或产品的行为和功能。验收测试的目标包括：

- 建立对整个系统质量的信心
- 确认系统是否完整且系统将按预期工作
- 验证系统的功能和非功能行为符合规范

验收测试可以提供信息，用以评估系统是否可以部署并交付给客户（最终用户）使用。虽然在验收测试期间可能会发现缺陷，但发现缺陷通常不是其目标。在验收测试过程中发现大量的缺陷，在某些情况下可能会被认为是一个重要的项目风险。验收测试也可能是为了满足法律或法规要求或标准。

验收测试的常见形式包括：

- 用户验收测试
- 运行验收测试
- 合同和法规验收测试
- Alpha 和 beta 测试

每个形式将在下面四个小节分别进行介绍

用户验收测试（UAT）

系统的用户验收测试通常侧重于验证系统是否适合在真实用户的环境或模拟运行环境中运行。其主要目标是建立信心，让用户能够以最低的难度、成本和风险使用这个系统去满足他们的要求、满足需求和开展业务流程。

运行验收测试（OAT）

操作或系统管理人员对系统的验收测试，通常是在（模拟的）生产环境中进行的。测试侧重于运行方面，可能包括：

- 测试备份和恢复

- 安装、卸载和升级
- 灾难恢复
- 用户管理
- 维护任务
- 数据加载和移植任务
- 检查安全漏洞
- 性能测试

运行验收测试的主要目标是建立信心，操作员或系统管理员能确保用户在运行环境中正常操作系统，即使在异常或困难的条件下也要确保系统可以正常工作。

合同和法规验收测试

合同验收测试是根据合同中生产定制软件的验收标准开展的。验收标准应在双方合同达成一致时确定，通常由用户或独立的测试员进行合同验收测试。

法规验收测试根据必须遵守的法规开展，例如政府、法律或安全法规。通常由用户或独立的测试员进行法规验收测试，有时监管机构也会参与见证或审计。

合同和法规验收测试的主要目标是建立信心，证明合同或法规要求已得到满足。

Alpha 和 beta 测试

Alpha 和 beta 测试通常被商业现货（COTS）软件的开发人员所采用，他们希望在软件产品投放市场之前从潜在或现有用户、客户和/或操作人员中获得反馈。Alpha 测试是在开发组织所在场地进行的测试，由潜在或现有客户、和/或操作人员或独立测试团队执行。Beta 测试是由潜在或现有的客户、和/或操作人员在他们本地执行。在完成 alpha 测试后，可以执行 Beta 测试，或之前没有执行过任何 alpha 测试的情况下执行 Beta 测试。

alpha 和 beta 测试的一个目标是在为潜在或现有客户和/或操作人员之间建立信心，即他们可以在正常的日常条件下，在操作环境中以最低的难度、成本和风险使用该系统实现目标。另外一个目标是发现与将要使用该系统的条件和环境有关的缺陷，尤其是在开发团队难以复制这些条件和环境的时候。

测试依据

验收测试中，可以作为测试依据的典型工作产品包括：

- 业务流程
- 用户或业务要求
- 法规、法律合同和标准
- 用例和/或用户故事
- 系统需求
- 系统或用户文档
- 安装规程
- 风险分析报告

此外，在运行验收测试中，作为衍生测试用例的测试依据，可以使用以下一个或多个工作产品：

- 备份和恢复规程
- 灾难恢复规程
- 非功能需求
- 操作文档
- 部署和安装说明
- 性能目标
- 数据库包
- 安全标准或法规

典型的测试对象

验收测试的典型测试对象包括：

- 被测系统
- 系统配置和配置数据
- 完整集成系统的业务流程
- 恢复系统和热站点（用于业务连续性和灾难恢复测试）
- 操作和维护流程
- 表格
- 报告
- 现有和转换的生产数据

典型的缺陷和失效

验收测试的典型缺陷包括：

- 系统工作流程不符合业务或用户要求
- 业务规则未正确实现
- 系统不满足合同或法规要求
- 非功能失效，例如安全漏洞、高负载下的性能效率不足或在被支持平台上的不正确操作

特定的方法和职责

验收测试通常是由客户、业务用户、产品负责人或系统操作员负责，也可能涉及其他利益相关方。

验收测试通常作为顺序开发生存周期中的最后一个测试级别，但也可能在其他时间发生，例如：

- 安装或集成商业现货（COTS）软件产品时，可能会对其进行验收测试；
- 在系统测试之前可能会对新功能增强进行验收测试。

在迭代开发中，项目团队可以在每次迭代期间和迭代结束时进行各种形式的验收测试，例如根据其验收标准验证新功能，以及确认新功能是否满足用户需求。此外，可能在每次迭代的最后，可能在每次迭代完成后或在一系列迭代之后，开展 alpha 测试和 beta 测试。也可能在每次迭代的最后，或者每次迭代完成或一系列迭代之后执行用户验收测试、运行验收测试、法规验收测试和合同验收测试。

2.3. 测试类型

测试类型是一组基于特定测试目标的测试活动，旨在测试软件系统或系统的一部分特定特性。这些目标可能包括:

- 评估功能质量特性，例如完整性、正确性和适当性
- 评估非功能质量特性，例如可靠性、性能效率、安全性、兼容性和易用性
- 评估组件或系统的结构或架构是否正确、完整并符合规定
- 评估变更的影响，例如确认缺陷已得到修复（确认测试）以及寻找因软件或环境变化而导致的不可预料的行为变化（回归测试）

2.3.1. 功能测试

系统的功能测试包括评估系统应该开展功能的测试。功能需求通常在工作产品中描述，例如业务需求规格说明、史诗、用户故事、用例或功能规格说明，或者它们可能没有文档记录。功能指的是系统应该做“什么”。

尽管每个测试级别的关注点不一样（参见第 2.2 节），但所有测试级别都应该执行功能测试（例如，组件测试以组件规格说明为基础）。

功能测试考虑软件的行为，因此可以通过使用黑盒技术获取组件或系统功能的测试条件和测试用例(参见第 4.2 节)。

功能测试的完整性可以通过功能覆盖来衡量。功能覆盖率是指测试执行某些功能的程度，并以所覆盖元素类型的百分比形式来表示。例如，使用测试和功能需求之间的可追溯性，可以计算出通过测试的需求的百分比，潜在地识别出覆盖的缺口。

功能测试设计和执行可能涉及特殊技能或知识，如软件解决的特定业务问题(例如，石油和天然气行业的地质建模软件)的知识。

2.3.2. 非功能测试

系统的非功能测试是用来评估系统和软件的特性，例如易用性、性能效率或安全性。有关软件产品质量特性的分类，参见 ISO 标准（ISO / IEC 25010）。非功能测试是测试系统在运行过程中“表现如何”。

与常见的误解相反，非功能测试可以并应该在所有的测试级别上开展，并且应尽早开展。对于项目的成功而言，在项目后期才发现非功能性缺陷是非常危险的。

可以使用黑盒技术（参见 4.2 节）生成非功能测试的测试条件和测试用例。例如，边界值分析可用于定义性能测试的压力条件。

非功能测试的完整性可以通过非功能覆盖来衡量。非功能覆盖是指通过测试执行某种类型的非功能元素所达到的程度，并且以所覆盖的元素类型的百分比形式来表示。例如，根据移动应用程序

的测试和支持的设备之间的可跟踪性，可以计算出通过兼容性测试的设备所占百分比，发现潜在的覆盖缺口。

设计和执行非功能测试可能会涉及特殊技能或知识，例如设计或技术的固有弱点（例如，与特定编程语言相关的安全漏洞）或特定用户基础（例如，医疗设施管理系统的用户角色）的知识。

有关非功能质量特性测试的更多详细信息，参见 ISTQB-CTAL-TA、ISTQB-CTAL-TTA、ISTQB-CTAL-SEC 和其他 ISTQB®相关模块内容。

2.3.3. 白盒测试

白盒测试基于系统内部结构或实现来生成测试。内部结构可能包括系统内的代码、架构、工作流和/或数据流(参见第 4.3 节)。

白盒测试的完整性可以通过结构覆盖来测量。结构覆盖是指某种类型的结构元素在测试中被使用的程度，并以所覆盖的元素类型的百分比形式来表示。

在组件测试级别，代码覆盖率基于已测试的组件代码的百分比。可以根据代码的不同方面（覆盖项）来度量，例如组件中测试的可执行语句的百分比，或者测试的判定结果的百分比。这些类型的覆盖统称为代码覆盖。在组件集成测试级别，白盒测试可能基于系统的架构，例如组件之间的接口，结构覆盖率可以根据测试所执行的接口的百分比来度量。

设计和执行白盒测试可能会涉及特殊技能或知识，例如构建代码的方式、数据的存储方式（例如，评估可能的数据库查询）以及如何使用覆盖工具并正确解释其结果。

2.3.4. 与变更相关的测试

当系统变更时，无论是修复缺陷，还是新增或修改功能，都应进行测试，以确认变更已修复缺陷或已正确实现功能，并且没有造成任何不可预见的不良后果。

- 确认测试：修复缺陷后，应该在软件的最新版本上，重新执行之前因该缺陷而导致失败的测试用例。为了覆盖修复缺陷所需的变化，也可以使用新的测试来测试软件。至少必须在新的软件版本上重新执行这些由缺陷引起失效的步骤。确认测试的目的是确认是否已成功修复原来的缺陷。
- 回归测试：部分代码所做的变更，无论是修复代码，还是其他类型的更改，都可能会意外地影响到除更改代码外的其他部分代码的行为，不管是在同一组件内，还是在同一系统的其他组件中，甚至在其他系统中。变更也可能包括环境的变化，例如操作系统或数据库管理系统的新版本。这种意外的副作用被称为回归。回归测试包括运行测试来检测这些意外的副作用。

确认测试和回归测试可以应用在所有的测试级别。

特别是在迭代和增量开发生存周期（例如，敏捷）中，新增特征、更改现有特征以及重构代码都会导致频繁更改代码，这也需要与变更相关的测试。随着系统不断发展，确认和回归测试非常重要。这尤其和物联网系统特别相关，在物联网中，会频繁更新或替换个别对象（例如，设备）。

由于要多次运行回归测试套件，并且回归测试套件通常变化缓慢，因此回归测试是自动化的有力候选者。应该在项目的早期就开始测试自动化（参见第 6 章）。

2.3.5. 测试类型和测试级别

可以在任何测试级别开展上述提到的任何测试类型。下面以银行应用程序为例，介绍功能测试、非功能测试、白盒测试以及与变更相关的测试在所有测试级别中的应用，从功能测试开始：

- 对于组件测试，根据组件是如何计算利息来进行测试设计。
- 对于组件集成测试，测试设计是基于如何将在用户界面捕获的账户信息传递到业务逻辑中。
- 对于系统测试，测试设计是基于帐户持有人如何在他们的支票帐户上申请信用额度。
- 对于系统集成测试，测试设计是基于系统如何使用外部微服务来检查帐户持有者的信用评分。
- 对于验收测试，测试设计是基于银行是如何处理批准或拒绝信贷申请。

以下是非功能测试的示例：

- 对于组件测试，性能测试的设计是为了评估开展复杂的总利息计算所需的 CPU 周期数。
- 对于组件集成测试，安全性测试的设计是针对从用户界面传到业务逻辑的数据所产生的缓冲区溢出漏洞。
- 对于系统测试，可移植性测试的设计是为了检查表示层是否在所有支持的浏览器和移动设备上工作。
- 对于系统集成测试，可靠性测试的设计是为了在信用评分微服务无法响应时，评估系统的健壮性。
- 对于验收测试，易用性测试的设计是为了评估银行信贷处理界面对残疾人的无障碍性。

以下是白盒测试的示例：

- 对于组件测试，测试的设计是为了对所有进行财务设计的组件实现完全的语句和判定覆盖（参见第 4.3 节）。
- 对于组件集成测试，测试的设计是为了测试浏览器界面中的每个屏幕如何将数据传递到下一个屏幕和业务逻辑。
- 对于系统测试，测试的设计是为了覆盖信用额度应用期间可能发生的网页序列。
- 对于系统集成测试，测试的设计是为了检查所有可能发送到信用评分微服务的查询类型。
- 对于验收测试，测试的设计是为了覆盖所有支持的财务数据文件结构和银行间转账的价值范围。

最后，以下是与变更相关的测试示例：

- 对于组件测试，为每个组件构建自动回归测试，并将其归入在持续集成框架内。
- 对于组件集成测试，测试的设计是为了确认当修复的代码已经集成到代码库时，与接口相关的缺陷已得到修复。
- 对于系统测试，如果工作流上的任何屏幕发生更改，则会重新执行指定的工作流的所有测试。
- 对于系统集成测试，每天重新执行与信用评分微服务交互的应用程序的测试，作为该微服务的持续部署的一部分。
- 对于验收测试，在验收测试中修复发现的缺陷后，将重新执行所有先前失败的测试。

虽然本节提供了每个级别的每种测试类型的示例，但对于所有软件而言，并不需要在每个级别上都运行每种测试类型。然而，在每个级别运行适用的测试类型很重要，尤其是测试类型出现的最早级别。

2.4. 维护测试

一旦部署到生产环境，就需要维护软件和系统。在已交付的软件和系统中，各种各样的变更几乎是不可避免的，比如修复运行使用中发现的缺陷，或添加新功能，或是删除或修改已经交付的功能。在组件或系统的生存周期中，还需要维护或改进其非功能性质量特性，特别是性能效率、易用性、可靠性、安全性和可移植性。

系统维护期间做任何变更时，应该执行维护测试，以评估更改的成功程度，并检查系统中未变更部分(通常是系统的大部分)可能出现的副作用(如回归)。

维护可包含计划发布和计划之外发布（热修复）。

根据维护版本的范围，维护版本可能需要在多个测试级别上进行多种测试类型的维护测试。维护测试的范围取决于：

- 变更的风险程度，例如，变更软件区域与其他组件或系统通信的程度
- 现有系统的规模
- 变更的大小

2.4.1. 维护的触发因素

导致软件维护，再到维护测试，其原因很多，包括计划的和计划外的变更。

维护的触发因素分类如下：

- 修改，例如计划的特性改善（例如，基于发布）、纠正和紧急变更，操作环境的更改（例如计划的操作系统或数据库升级）、商业现货（COTS）软件的升级以及缺陷和漏洞的补丁；
- 迁移，例如从一个平台到另一个平台，这可能需要对新环境以及更改的软件进行运行测试，或者当来自另一个应用程序的数据将迁移到正在维护的系统时进行数据转换测试；
- 退役，例如当应用程序的使用周期结束时。
  - 当应用程序或系统退役时，如果需要较长的数据保留期，可能需要测试数据迁移或归档。
  - 在长期保留归档之后，还可能需要测试保存/恢复规程。
  - 可能需要进行回归测试，以确保在使用中的任何功能仍然有效。

对于物联网系统，可以通过将全新或修改过的内容（例如硬件设备和软件服务）引入整个系统来触发维护测试。这类系统的维护测试特别强调在不同级别（例如，网络级别、应用级别）的集成测试和安全方面，特别是那些与个人数据有关方面。

2.4.2. 维护的影响分析

影响分析评估维护版本所做的变更，来识别预期的结果以及变更带来的预期和可能的副作用，并识别出系统中将受变更影响的区域。影响分析还可以帮助识别变更对现有测试的影响。可能受变更影响的任何现有测试在更新之后，需要对系统中的副作用和受影响区域进行回归测试。

根据系统其他区域的潜在结果，可以在变更之前进行影响分析来帮助确定是否应进行变更。

如果出现以下情况，影响分析会比较困难：

- 规格说明（例如，业务要求、用户故事、架构）已过期或缺失
- 测试用例未记录或已过期
- 测试和测试依据之间的双向可追溯性尚未得到维护
- 工具支持很弱或不存在
- 参与的人员没有领域和/或系统的知识
- 在开发过程中，对软件的维护性关注不足

3. 静态测试（135 分钟）

关键词

临时评审（ad hoc review）、基于检查表的评审（checklist-based review）、动态测试（dynamic testing）、正式评审（formal review）、非正式评审（informal review）、审查（inspection）、基于阅读视角的文档评审（perspective-based reading）、评审（review）、基于角色的评审（role-based review）、基于场景的评审（scenario-based review）、静态分析（static analysis）、静态测试（static testing）、技术评审（technical review）、走查（walkthrough）

静态技术的学习目标

3.1 静态测试基础

FL-3.1.1 （K1）认出可以通过不同静态测试技术检查的软件工作产品类型

FL-3.1.2 （K2）使用例子来描述静态测试的价值

FL-3.1.3 （K2）解释静态和动态测试技术之间的差异，考虑目标、被识别的缺陷类型、以及这些技术在软件生存周期中的作用

3.2 评审过程

FL-3.2.1 （K2）总结工作产品评审过程的活动

FL-3.2.2 （K1）认出正式评审中的不同角色和责任

FL-3.2.3 （K2）解释不同评审类型的差异：非正式评审、走查、技术评审、和审查

FL-3.2.4 （K3）应用评审技术在工作产品中发现缺陷

FL-3.2.5 （K2）解释影响评审成功的因素

3.1. 静态测试基础

与动态测试相比，静态测试依赖于软件工作产品的手动检查（即评审），或工具驱动的代码或其他软件工作产品的评估（即静态分析）。两种类型的静态测试都会评审正在测试的代码或其他软件工作产品，而无需实际执行所测试的代码或软件工作产品。

静态分析对于安全关键的计算机系统（例如，航空，医疗或核控制软件）很重要，但静态分析在其他环境中也变得重要和常见。例如，静态分析是安全测试的重要部分。静态分析通常也包含在自动软件构建和分布式工具中，例如，在敏捷开发，持续交付和持续部署中。

3.1.1. 由静态测试检查的软件工作产品

几乎所有软件工作产品都可使用静态测试（评审和/或静态分析技术）进行检查，例如：

- 规格说明，包括业务需求、功能需求和安全性需求
- 史诗、用户故事和验收准则
- 架构和设计规格说明
- 代码
- 测试件，包括测试计划、测试用例、测试规程和自动化测试脚本
- 用户手册
- web 网页
- 合同、项目计划、进度表和预算计划
- 配置设置以及基础设施的设置
- 模型，如可用于基于模型测试的活动图（参见 ISTQB-CTFL-MBT 和 Kramer 2016）

评审可应用于任何工作产品，只要参与者能够阅读和理解这些工作产品。静态分析技术可有效地应用于具有规范结构（典型代表有代码或模型）的任何软件工作产品，可运用适当的静态分析工具。静态分析技术甚至可借助工具评估以自然语言编写的工作产品，如需求（例如，检查拼写、语法和可读性）。

3.1.2. 静态测试的好处

静态测试技术提供了多种好处。当在软件开发生存周期的早期应用静态测试，就能够在开展动态测试之前尽早地检测缺陷（例如，在需求或设计规格说明评审，待办事项列表的细化工作等）。早期发现的缺陷通常比软件开发生存周期后期发现的缺陷经济得多，特别是与软件部署和使用后发现的缺陷相比。使用静态测试技术来发现缺陷然后及时修复这些缺陷几乎总是比使用动态测试在测试对象中发现缺陷然后修复它们所花费的成本要低得多，特别是在考虑到更新其他软件工作产品以及开展确认和回归测试的额外成本。

静态测试的其他好处可能包括：

- 在动态测试执行之前，更高效地检测和修复缺陷
- 识别动态测试不易发现的缺陷

- 通过发现需求中的不一致、含糊不清、矛盾、遗漏、不准确和冗余来防止设计或编码缺陷
- 提高开发效率（例如，由于改进的设计、更易于维护的代码）
- 降低开发成本和时间
- 降低测试成本和时间
- 减少在软件开发生存周期后期或上线运营后失效，从而降低了软件在整个生存周期内的总的质量成本
- 在参与评审过程中改善团队成员之间的沟通

3.1.3. 静态测试和动态测试的差异

静态测试和动态测试可以具有相同的目标（参见第 1.1.1 节），例如提供对工作产品质量的评估并尽早识别缺陷。静态和动态测试通过发现不同类型的缺陷来相互补充。

静态测试和动态测试的一个主要区别在于静态测试直接发现软件工作产品中的缺陷，而动态测试是在运行软件时识别由缺陷引起的失效。缺陷可以在软件工作产品中存在很长时间而不会导致失效。缺陷所在的路径可能很少被执行或难以到达，因此设计并执行动态测试去发现这样的缺陷并不容易。静态测试可能付出更少的工作量就能找到缺陷。

另一个区别是静态测试可用于提高软件工作产品的一致性和内部质量，而动态测试通常侧重于外部可见行为。

与动态测试相比，通过静态测试可以更早期更经济的发现和修复一些典型的缺陷，包括：

- 需求缺陷（例如：不一致、含糊不清、矛盾、遗漏、不准确和冗余）
- 设计缺陷（例如：低效算法或数据库结构、高耦合、低内聚）
- 代码缺陷（例如：未定义值的变量、已声明但从未使用的变量、无法访问的代码、重复的代码）
- 与标准的偏差（例如：未完全遵循编码规范）
- 错误的接口规格说明（例如：主叫系统使用的测量单位与被叫系统不同）
- 安全漏洞（例如：对缓冲区溢出的敏感性）
- 测试依据的可追溯性或覆盖率的差距或不准确性（例如：针对验收准则缺少测试）

此外、大多数类型的维护性缺陷只能通过静态测试才能被发现（例如：没有很好的模块化、组件的可重用性较差、难以分析的代码以及很难保证修改代码而不引入新缺陷）。

3.2. 评审过程

评审类型是多样化的，从非正式到正式的评审。非正式评审的特点是既无需遵守既定的过程，也没有正式的文档化输出。正式评审的特点是团队参与、文档化评审结果以及开展评审的文档化过程。评审过程的正式程度与软件开发生存周期模型、开发过程的成熟度、需评审的软件工作产品的复杂性、任何法律或法规要求和/或审计跟踪等因素有关。

评审的关注点依赖于已达成一致的评审目标（例如：发现缺陷、增加理解、培训参与者如测试员和团队新成员，或对讨论和决定达成共识等）。

ISO 标准（ISO / IEC 20246）包含对软件工作产品评审过程的更深入描述，包括评审角色和评审技术。

3.2.1. 工作产品的评审过程

评审过程由以下主要活动构成：

计划

- 定义范围，包括评审目的、需评审的文档或部分文档以及需评估的质量特性
- 估算工作量和时间表
- 识别评审特性，例如评审类型以及相应的角色、活动和检查表
- 选择参与评审人员并分配角色
- 针对较正式的评审类型（例如：审查）制定入口和出口准则
- 核对入口准则是否满足（针对较正式的评审类型）

评审启动会

- 分发工作产品（以物理方式或通过电子方式）和其他材料，例如：事件日志表、检查表和相关工作产品
- 向评审参与者解释评审的范围、目标、过程、角色和工作产品
- 回答评审参与者可能对评审提出的任何问题

独立评审（即个人准备）

- 评审全部或部分工作产品
- 标注可能的缺陷、建议和问题

事件交流和分析

- 交流已识别的潜在缺陷（例如：在评审会议中）
- 分析潜在缺陷，并为这些缺陷分派责任人和状态
- 评估和记录质量特性
- 根据出口准则评估评审发现，以确定评审结论（拒绝；需重大变更；接受，但可能需要小修改）

修正和报告

- 当发现需要修改工作产品时而创建缺陷报告
- 修正在工作产品评审中发现的缺陷（通常由作者来进行）
- 与相关人员或团队交流缺陷（当工作产品中的发现关联到被评审的工作产品）
- 记录缺陷更新的状态（在正式评审中），可能包括对评审意见提交人的认同
- 收集度量数据（对较正式的评审类型）
- 核对出口准则是否满足（对较正式的评审类型）
- 接受满足出口准则的工作产品

因评审类型和正式程度不同，工作产品评审的结果会有所不同，参见第 3.2.3 节。

3.2.2. 正式评审的角色和职责

典型的正式评审主要有下面几种角色：

作者：

- 创建被评审的工作产品
- 修复工作产品评审过程中发现的缺陷（如果需要的话）

经理（管理者）：

- 负责制定评审计划
- 决定是否需要进行评审
- 分派人员、预算和时间
- 监督进行中的成本-效益
- 当产出不充分时，执行控制决策

促进者（通常称为主持人）：

- 保证评审会议的有效进行（当召开时）
- 需要时在评审的不同观点之间进行协调
- 通常是评审成功与否的关键人物

评审组长：

- 全面负责评审
- 决定哪些人员参加评审，并组织何时何地进行评审

评审员：

- 可能是专题相关专家、项目工作人员、对工作产品感兴趣的利益相关方，和/或具有特定技术或业务背景的人员
- 在评审中识别工作产品中的潜在缺陷
- 可能代表不同的视角（例如：测试员、开发人员、用户、操作员、业务分析师、易用性专家等）

书记（或记录员）：

- 在独立评审活动期间，收集发现的潜在缺陷
- 记录评审会议中的新发现的潜在缺陷、未解决的问题和决策（当评审会议召开时）

在有些评审类型中，一个人可能扮演多个角色，并且每个角色分工也可能因评审类型而异。此外，随着支持评审过程的工具的出现，特别是缺陷、未解决问题和决策记录，通常就不需要记录员。

更多角色的细节可参考 ISO 标准（ISO/IEC 20246）。

3.2.3. 评审类型

虽然评审可用于各种目的，但其主要目的之一是发现缺陷。所有评审类型都可以帮助检测缺陷，所选的评审类型应基于项目需求、可用资源、产品类型和风险、业务领域和公司文化，以及其他选择准则。

单一工作产品可能会使用一种以上的评审类型，如果使用了多种类型的评审，则顺序可能会有所不同。例如，可以在技术评审之前进行非正式评审，以确保工作产品已经为技术评审做好了准备。

下面描述的评审类型均可以作为同行评审，也就是说由有资格做同样工作的同事们来完成。

评审中发现的缺陷类型各不相同，尤其取决于评审的工作产品。（参见第 3.1.3 节，了解对不同工作产品的评审所发现的缺陷示例，以及 Gilb 1993，了解正式审查的有关信息）。

评审可以根据各种属性进行分类。以下列出了四种最常见的评审类型及其相关属性。

非正式评审（例如：伙伴检查、结对、结对评审）

- 主要目的：检测潜在缺陷
- 可能的附加目的：产生新的想法或解决方案，快速解决小问题
- 不基于正式的（文档化的）过程
- 可能不包含评审会议
- 可能由作者的同事（伙伴检查）或更多人参与评审

- 可能记录评审结果
- 有效性根据评审人员的不同而变化
- 使用检查表是可选的
- 敏捷开发中使用的非常普遍

走查

- 主要目的：发现缺陷、改进软件产品、考虑替代实施、评估与标准和规范的符合程度
- 可能的附加目的：交换关于技术或风格变化的想法、参与者的培训、达成共识
- 评审会议前的个人准备是可选的
- 评审会议通常由工作产品的作者主持
- 必须指定书记（记录员）
- 使用检查表是可选的
- 可能采用场景、演练或模拟的形式
- 生成潜在的缺陷记录和评审报告
- 在实践中可能从相当非正式到非常正式

技术评审

- 主要目的：获得共识、发现潜在缺陷
- 可能的进一步目的：评估质量和建立对工作产品的信心、产生新想法、激励和使作者能够改进未来的工作产品、考虑替代实施
- 评审员应该是作者的技术同行，以及同一学科或或其他学科的技术专家
- 需要在评审会议之前进行个人准备
- 评审会议是可选的，最好由经过培训的促进者（主持人）（通常不是作者）主持
- 必须指定记录员，最好不是作者
- 使用检查表是可选的
- 生成潜在的缺陷记录和评审报告

审查

- 主要目的：检测潜在缺陷，评估质量并建立对软件工作产品的信心，通过学习和根本原因分析防止未来出现类似缺陷
- 可能的进一步目的：激励和使作者能够改进未来的工作产品和软件开发过程、达成共识

- 遵循已定义的过程，基于规则和检查表，正式地记录输出
- 使用明确定义的角色，如第 3.2.2 节中规定的强制性角色，也可能包括专门的朗读者（在评审会议期间朗读工作产品的人经常会用自己的话来解释，即描述。）
- 需要在评审会议之前进行个人准备
- 评审参与者是作者的同行或与工作产品相关的其他学科专家
- 使用已规定的入口和出口准则
- 必须指定书记（记录员）
- 评审会议由经过培训的促进者（主持人）（不是作者）引导
- 作者不能担任评审组长、朗读者或书记（记录员）
- 通常生成潜在的缺陷记录和评审报告
- 收集度量并用于改进整个软件开发过程，包括审查过程

3.2.4. 评审技术的应用

在独立评审（即个人准备）活动期间可以应用许多评审技术来发现缺陷。这些技术可用于上述评审类型。这些技术的有效性可能因所使用的评审类型而异。下面列出了各种评审类型的不同独立评审技术的示例。

临时评审

在临时评审中，评审员很少或根本没有得到指导如何执行此任务。评审员经常按顺序阅读工作产品，在遇到问题时识别并记录事件。临时评审是一种常用的技术，几乎不需要准备。此技术高度依赖于评审员的技能，并可能导致不同评审员报告许多重复的事件。

基于检查表的评审

基于检查表的评审是一种系统性的技术，评审员基于评审开始时（例如，由促进者（主持人））分发的检查表来发现事件。评审检查表包含一组基于潜在缺陷的问题，这些问题可能来自经验。检查表应特定于所评审的工作产品类型，并应定期维护以涵盖以前评审中遗漏的事件类型。基于检查表的技术的主要优点是对典型缺陷类型的系统的覆盖。应注意不要简单地按照独立评审中的检查表，而是也要寻找检查表之外的缺陷。

场景和演练的评审

在基于场景的评审中，将为评审员提供有关如何通读工作产品的结构化指南。基于场景的评审支持评审员基于工作产品的预期使用，对工作产品开展“演练”（如果工作产品以适当的格式记录，例如用例）。与简单的检查表条目相比，这些场景为评审员提供了有关如何识别特定缺陷类型的更好

指导。与基于检查表的评审一样，为避免错过其他缺陷类型（例如，功能遗漏），评审员不应受限于文档化的场景。

基于视角

在基于视角的文档阅读评审中，类似于基于角色的评审，评审员在独立评审中采用不同的利益相关方观点。典型的利益相关方观点包括最终用户、市场人员，设计人员、测试员或操作员。使用不同的利益相关方视角可以更加深入地进行独立评审，同时可以减少评审员之间的重复问题。

此外，基于视角的文档阅读评审还要求评审员尝试使用被评审的工作产品来生成他们所需的衍生产品。例如，如果对需求规格开展基于阅读视角的文档评审，以查看是否包含所有必要信息，这时测试员将会尝试生成草拟的验收测试。此外，在基于视角的文档阅读评审中，希望使用检查表。

经验研究表明，通常情况下基于阅读视角的文档评审是评审需求和技术工作产品最有效的技术。关键的成功因素是基于风险适当地包括和权衡不同的利益相关方观点。有关基于阅读视角的文档评审的详细信息，参见 Shul 2000；有关不同评审技术的有效性，参见 Sauer 2000。

基于角色的评审

基于角色的评审技术，评审员可以从独立利益相关方角色的角度评估工作产品。典型角色包括特定的最终用户类型（有经验、没有经验、老人、小孩等），以及组织中的特定角色（用户管理员、系统管理员、性能测试员等）。同样的原则也适用于基于视角的阅读，因为角色是相似的。

3.2.5. 评审的成功因素

为了获得成功的评审，必须考虑适当的评审类型和使用的技术。此外，还有许多其他因素会影响评审结果。

组织层面的评审成功因素包括：

- 每次评审都有明确的目标，其在评审计划中得到定义，并将其作为可度量的出口准则
- 应用的评审类型要适用于要实现的目标，适用于软件工作产品的类型和级别，以及参与者
- 所使用的任何评审技术（例如基于检查表或基于角色的评审）要适用于在被评审的工作产品中有效的缺陷识别
- 使用的任何检查表要处理主要风险，并且检查表是最新的
- 大型文档以小块形式编写和评审，从而通过向作者提供早期和频繁的缺陷反馈来实施质量控制
- 参与者有足够的时间来准备
- 安排评审并得到足够的重视

- 管理者支持评审过程（例如，通过在项目进度表中考虑足够的时间进行评审活动）
- 评审与公司的质量和/或测试方针相结合。

与人相关的评审成功因素包括：

- 合适的人员参与以满足评审目标，例如，具有不同技能或观点的人员，他们可能将文档用作工作输入
- 测试员参加评审不但有利于提高评审质量，还可以通过评审了解产品，以使他们准备更有效测试，以及尽早准备这些测试
- 参与者将有充足的时间和精力关注细节
- 评审应分成小块任务进行，以便评审员在独立评审和/或评审会议期间不会失去注意力（当召开时）
- 对发现的缺陷持欢迎态度，并客观地处理缺陷
- 管理好评审会议，使参与者感受到他们的时间是有价值的
- 评审应该在信任的氛围中进行；结果不会用于评估参与者
- 参与者避免使用可能表示对其他参与者感到无聊、恼怒或敌意的肢体语言和行为
- 提供充分的培训，特别是对于较正式的评审类型，如审查
- 强调学习和过程改进的文化

（有关成功评审的进一步信息，请参阅 Gilb 1993，Wiegers 2002 和 van Veenendaal 2004）

4. 测试技术（330 分钟）

关键词

黑盒测试技术（black-box test technique）、边界值分析（boundary value analysis）、基于检查表的测试（checklist-based testing）、覆盖（coverage）、判定覆盖（decision coverage）、判定表测试（decision table testing）、错误推测（error guessing）、等价类划分（equivalence partitioning）、基于经验的测试技术（experience-based test technique）、探索性测试（exploratory testing）、状态转换测试（state transition testing）、语句覆盖（statement coverage）、测试技术（test technique）、用例测试（use case testing）、白盒测试技术（white-box test technique）

测试技术的学习目标

4.1 测试技术分类

FL-4.1.1 （K2）解释黑盒测试技术、白盒测试技术和基于经验的测试技术之间的特点、共性和差异

4.2 黑盒测试技术

FL-4.2.1 （K3）应用等价类划分从给定的需求生成测试用例

FL-4.2.2 （K3）应用边界值分析从给定的需求生成测试用例

FL-4.2.3 （K3）应用判定表测试从给定的需求生成测试用例

FL-4.2.4 （K3）应用状态转换测试从给定的需求生成测试用例

FL-4.2.5 （K2）解释如何从用例生成测试用例

4.3 白盒测试技术

FL-4.3.1 （K2）解释语句覆盖

FL-4.3.2 （K2）解释判定覆盖

FL-4.3.3 （K2）解释语句和判定覆盖的价值

4.4 基于经验的测试技术

FL-4.4.1 （K2）解释错误推测法

FL-4.4.2 （K2）解释探索性测试

FL-4.4.3 （K2）解释基于检查表的测试

4.1. 测试技术分类

本章节讨论的测试技术，其目的是帮助识别测试条件、测试用例和测试数据。

测试技术的选择基于很多因素，包括：

- 组件或系统的复杂性
- 法律法规标准
- 客户或合同的需求
- 风险级别和类型
- 可用的文档
- 测试员的知识和技能
- 可用的工具
- 时间和预算
- 软件开发生存周期模型
- 在组件或系统中期望发现的缺陷类型

有些技术更适用于特定的环境和测试级别，而有些则适用于所有的测试级别。在创建测试用例时，测试员通常使用测试技术的组合来实现测试工作的最佳结果。

在测试分析、测试设计和测试实施活动中使用测试技术的范围，可以从很不正式（很少甚至没有文档）到非常正式。适当的正式程度取决于测试周境，包括测试和开发过程的成熟度、时间限制、安全或合规要求、相关人员的知识和技能、以及所遵循的软件开发生存周期模型。

4.1.1. 测试技术分类和特点

本大纲将测试技术分为黑盒、白盒或基于经验的测试技术。

黑盒测试技术（也称为行为的或基于行为的技术）基于对适当测试依据的分析（例如：正式需求文档、规格说明、用例、用户故事或业务流程）。这些技术适用于功能和非功能测试。黑盒测试技术关注在测试对象的输入和输出，而不考虑其内部结构。

白盒测试技术（也称为结构的或基于结构的技术）基于对架构、详细设计、内部结构或测试对象代码的分析。与黑盒测试技术不同，白盒测试技术关注在测试对象的结构和处理过程。

基于经验的测试技术利用开发人员、测试员和用户的产品经验来设计、实施和执行测试。这类技术通常与黑盒和白盒测试技术相结合。

黑盒测试技术的共同特点包括：

- 测试条件、测试用例和测试数据的获取源自测试依据，可能包括软件需求、规格说明、用例和用户故事
- 测试用例可用于检查需求和需求实现之间的差距，以及需求本身的错误
- 覆盖度量是根据在测试依据中已测试的项和应用到测试依据的技术

白盒测试技术的共同特点包括：

- 测试条件、测试用例和测试数据的获取源自测试依据，可能包括代码、软件架构、详细设计或有关软件结构的任何信息资源
- 覆盖度量的依据是所选的结构（例如代码或接口）和应用于测试依据的技术中已测试的项

基于经验的测试技术的共同特征包括：

- 测试条件、测试用例和测试数据的获取源自测试依据，可能包括测试员、开发人员、用户和其他利益相关方的知识和经验。

知识和经验包括软件的预期使用、它的环境、可能的缺陷以及这些缺陷的分布。

国际标准（ISO/IEC/IEEE 29119-4）包含测试技术的描述和他们相关的覆盖度量（相关技术的更多信息可参见 Craig 2002 和 Copeland 2004）。

4.2. 黑盒测试技术

4.2.1. 等价类划分

等价类划分将数据划分为不同分区（也称为等价类），使得给定分区内所有成员期望被相同方式处理（参见 Kaner 2013 和 Jorgensen 2014）。等价分区有针对有效值或无效值的。

- 有效值是组件或系统应接受的值。包含有效值的等价分区称为“有效等价类”。
- 无效值是组件或系统应拒绝的值。包含无效值的等价分区称为“无效等价类”。
- 可以为与测试对象相关的任何数据元素识别分区，包括输入、输出、内部值、与时间相关的值（例如，事件之前或之后）和接口参数（例如，在集成测试期间被测试的集成组件）。
- 如果需要，可以将任何分区进一步划分到子分区。
- 每个值必须属于一个且只有一个等价类。
- 当测试用例中使用无效等价类，应单独测试，即不能与其他无效等价类组合，以保证没有失效屏蔽。当多个失效同时发生，但只能观察到一个失效时，其他失效被屏蔽了，这导致无法发现其他失效。

要使用此技术实现 100%覆盖率，测试用例必须通过使用每个等价类中至少一个值来覆盖所有已识别的等价类（包括无效等价类）。覆盖度量为至少有一个值已经测试过的等价类的数量除以所识别的等价类的总数，通常表示为百分比。等价类划分适用于所有测试级别。

4.2.2. 边界值分析

边界值分析是等价类划分的扩展，但仅适用于等价类是有序的、由数字或顺序数据组成。等价类的最小和最大值（或第一和最后的值）是其边界值（参看 Beizer 1990）。

例如，假设一个输入域接受单位整数值作为输入，使用（0-9）数码键限制输入，因此不可能输入非整数。包含的有效区间从 1 到 5（包含 1 和 5）。因此，存在三个等价类：无效（太低）；有效；无效（太高）。对于有效等价类，边界值是 1 和 5。对于无效（太高）分区，边界值

是 6。对于无效（太低）分区，只有一个边界值 0，因为这个分区仅有一个成员。

在上面的例子中，每个边界我们识别出两个边界值。无效（太低）和有效之间的边界给的测试值 0 和 1。有效和无效（太高）之间的边界给的测试值 5 和 6。作为这个技术的变体，每个边界可以识别出三个边界值：到边界之前、正好到边界、刚超过边界的值。在前面例子中，使用三点边界值，低端边界测试值是 0、1 和 2，以及高端边界测试值是 4、5 和 6 （参看 Jorgensen 2014）。

在区域的边界上的行为往往比在区域内的行为更容易出现错误。值得注意的是，不仅定义的边界，而且已实现的边界可能会被移动到它们预期位置的上方或下方，或可能会被完全忽略，或还可能会增加不需要的额外边界。边界值分析和测试是通过激发软件显示出与边界值所属区域预期不同的行为，边界值分析和测试将能发现几乎所有这类缺陷。

边界值分析可以应用于所有的测试级别。这个技术通常用来测试需要调用一系列数字的测试需求（包括日期和时间）。对区域的边界值覆盖度量是测试的边界值数量除以识别的边界测试值总数，通常用百分比表示。

4.2.3. 判定表测试

判定表是记录系统必须实现的复杂业务规则的一种很好的方法。当建立判定表时，测试员识别系统的条件（通常是输入）和导致的动作（通常是输出）。判定表的行，通常是条件在顶部，而动作在底部。判定表的每一列对应了一个判定规则，该规则定义了各种条件的一个唯一组合，其表示与该规则相关的动作的执行。条件值和动作通常表示为布尔值（“真”或“假”）或离散值（例如：红、绿、蓝），但也可以是数字或数字范围。这些不同类型的条件和动作可能一起出现在同一判定表中。

判定表的常见符号如下：

对条件：

- Y 表示条件是真（也可能显示为 T 或 1）
- N 表示条件是假（也可能显示为 F 或 0）
- - 表示条件值不关心（也可能显示为 N/A）

对动作：

- X 表示动作应发生（也可能显示为 Y 或 T 或 1）
- 空格 表示动作不应该发生（也可能显示为 - 或 N 或 F 或 0）

完全的判定表有足够多的列（测试用例）来覆盖条件的每个组合。通过删除不影响结果的列，测试用例的数量可以大大减少。例如通过移除不可能的条件组合。有关如何精简化判定表的进一步信息，参见 ISTQB-CTAL-TA。

判定表测试的最小覆盖标准通常是对判定表中每个判定规则至少有一个测试用例。这通常涉及覆盖所有条件的组合。覆盖度量是至少被一个测试用例测试过的判定规则的数量，除以判定规则总数，通常以百分比表示。

判定表测试的优点是可以帮助识别所有重要的条件组合，否则其中有些条件组合有可能被忽视。判定表测试也会帮助发现需求中的漏洞。判定表测试可能应用到所有基于条件组合决定软件行为的情况，可以在任何测试级别进行。

4.2.4. 状态转换测试

根据组件或系统当前条件或先前历史（例如自从系统初始化后发生过的事件），组件或系统可能会产生不同的响应。先前历史可以用状态的概念来总结。状态转换图不但显示可能的软件状态，同时包含了软件如何入口、出口，以及状态之间的转换的。转换是通过事件触发的（例如，用户输入一个值到输入域内）。事件导致转换。相同的事件从相同状态能导致两个或多个不同转换。状态改变可能导致软件采取动作（例如，输出计算或错误信息）。

状态转换表不但可显示状态之间所有有效转换和潜在的无效转换，而且可表示有效转换的事件和导致的动作。状态转换图通常仅仅显示有效转换，而不包括无效转换。

设计测试可以用来覆盖典型的状态序列、执行所有状态、执行每个转换、执行转换的特定序列，或测试无效的转换。

状态转换测试可以用于基于菜单的应用，其广泛使用在嵌入式软件行业。此技术也适用于有特定状态的业务场景的建模，或测试屏幕上显示的引导或“菜单”。状态的定义是抽象的，即可能表示几行代码或整个业务流程。

覆盖度量通常是识别出已测试状态或已测试转换的数量，除以测试对象已识别出的状态或转换的总数，一般以百分比表示。有关状态转换测试的覆盖准则的进一步信息可参见 ISTQB-CTAL-TA。

4.2.5. 用例测试

测试可以从用例中推导出来，用例是设计软件项交互的一种特殊方式，包含了软件功能的需求。用例关联了参与者（用户、外部硬件或其他组件或系统）和主体（用例所应用的组件或系统）。

每个用例规定了一个主体能与一个或多个参与者合作开展的一些行为（UML 2.5.1 2017）。一个用例能通过交互和活动以及前置条件、后置条件进行描述，如合适，还可以以自然语言进行描述。参与者和主体之间的交互可能导致主体的状态改变。交互可能以工作流、活动图，或业务流程模型的图示方式表示。

用例包含了其基本行为的可能变化，包括期望行为和错误处理（系统对程序、应用和通讯错误的反应和恢复，例如，导致一个错误信息）。设计测试来执行已定义的行为（基本的、异常的或替代的和错误处理）。覆盖度量是已测试的用例行为数除以用例行为的总数，通常以百分比表示。

有关用例测试覆盖准则的进一步信息，参见 ISTQB-CTAL-TA 高级测试分析师大纲。

4.3. 白盒测试技术

白盒测试是基于测试对象的内部结构。白盒测试技术能用在所有的测试级别，但本节讨论的两个代码相关的技术通常是用于组件测试级别。还有一些更高级的技术，可用于安全关键、任务关键，或高完整性环境以达到彻底的覆盖，但这些不在此处讨论。有关这些技术的进一步信息，参见 ISTQB-CTAL-TTA。

4.3.1. 语句测试和覆盖

语句测试执行代码中可执行语句。覆盖度量是被测试执行的语句数，除以测试对象中可执行语句总数，通常以百分比表示。

4.3.2. 判定测试和覆盖

判定测试执行代码中的判定，以及测试基于判定结果的可执行的代码。为此，测试用例跟随发生在判定点的控制流（例如，针对 IF 语句,一个对真（true）结果和一个对假（false）结果；针对 CASE 语句，测试用例将要求对所有可能结果，包括缺省（default）结果进行覆盖）。

覆盖度量是被测试执行的判定结果数，除以测试对象中判定结果的总数，通常以百分比表示。

4.3.3. 语句和判定测试的价值

当实现 100%语句覆盖时，它确保代码中的所有可执行语句至少已被测试过一次，但不能确保所有判定逻辑都已被测试过。在本大纲中讨论的两种白盒技术中，语句测试可能提供的覆盖低于判定测试。

当达到 100%的判定覆盖时，它会执行所有判定结果，包括测试取值为真（true）的结果和取值为假（false）的结果，即使没有清晰的假（false）语句（例如，IF 语句的代码中没有 else）。语句覆盖有助于发现那些未被其他测试执行到的代码中的缺陷。判定覆盖有助于发现那些在其他测试中没有对其所有真值（“真”/“假”）都执行到的代码（例如，判定语句）中的错误。

实现 100%的判定覆盖可保证 100%的语句覆盖（但反之不成立）。

4.4. 基于经验的测试技术

在应用基于经验的测试技术时，测试用例源自测试员的技能和直觉，以及他们在类似应用和技术方面的经验。这些技术有助于识别一些其它系统化技术无法轻易识别的测试用例。根据测试员的方法和经验，这些技术可以达到不同程度的覆盖和有效性。使用这些技术，可能会难以评估和度量覆盖。

以下各节将讨论常用的基于经验的技术。

4.4.1. 错误推测

错误推测法是一种基于测试员的知识来预估错误、缺陷和失效发生的技术，包括：

- 应用软件在过去是如何工作的
- 常犯的错误类型
- 在其它应用软件中发生的失效

错误推测技术的一个系统化方法是创建一个可能的错误、缺陷和失效的列表，并设计测试以发现失效以及导致它们出现的缺陷。可以根据经验或缺陷和失效的信息，也可根据软件故障原因的一般知识创建此错误、缺陷和失效列表。

4.4.2. 探索性测试

在探索性测试中，在测试执行期间动态地设计、执行、记录、和评估非正式的（非预定义的）测试。测试结果常用来进一步了解组件或系统，并对可能需要进一步测试的区域生成测试。

探索性测试有时通过基于会话的测试来构造活动。在基于会话的测试中，探索性测试在定义的时间盒内进行，测试员使用包含测试目标的测试章程来指导测试。测试员可能使用测试会话表格来记录随后步骤和发现。

探索性测试在规格说明文档较少或不充分，或测试的时间压力大的情况下是最有帮助的。探索性测试作为对其他更正式的测试技术的补充也是有用的。

探索性测试与应对性（reactive）测试策略高度相关（参见第 5.2.2 节）。探索性测试能与其它黑盒、白盒和基于经验的技术组合使用。

4.4.3. 基于检查表的测试

在基于检查表的测试中，测试员设计、实施和执行测试来覆盖检查表中的测试条件。作为分析的一部分，测试员可以创建新的检查表或扩充现有的检查表，但测试员也可能使用没有修改的现有检查表。可以基于检验、对用户重要内容的了解，或对软件失效的原因和方式的理解来构建检查表。

可产生用于支持各种测试类型的检查表，包括功能和非功能测试。在没有详细测试用例的情况下，基于检查表的测试能提供指南和在一定程度上保持一致性。由于它们是概要性的列表，因此在实际测试中往往会出现一些变化和衍生，可能导致更高的覆盖，但又保持了低重复性。

5. 测试管理（225 分钟）

关键词

配置管理（configuration management）、缺陷管理（defect management）、缺陷报告（defect report）、入口准则（entry criteria）、出口准则（exit criteria）、产品风险（product risk）、项目风险（project risk）、风险（risk）、风险级别（risk level）、基于风险的测试（risk-based testing）、测试方法（test approach）、测试控制（test control）、测试估算（test estimation）、测试经理（test manager）、测试监督（test monitoring）、测试计划（test plan）、测试规划（test planning）、测试进度报告（test progress report）、测试策略（test strategy）、测试总结报告（test summary report）、测试员/测试工程师（tester）

测试管理的学习目标

5.1 测试组织

FL-5.1.1（K2）解释独立测试的好处和缺点

FL-5.1.2（K1）识别测试经理和测试员的任务

5.2 测试计划和估算

FL-5.2.1（K2）总结测试计划的目的和内容

FL-5.2.2（K2）区分各种测试策略之间的差异

FL-5.2.3（K2）给出潜在的入口和出口准则的例子

FL-5.2.4（K3）应用优先级、技术和逻辑依赖关系的知识，为给定的一组测试用例安排测试执行进度

FL-5.2.5（K1）识别影响与测试相关工作量的因素

FL-5.2.6（K2）解释两种估算技术之间的差异：基于度量的技术和基于专家的技术

5.3 测试监督与控制

FL-5.3.1（K1）记忆用于测试的度量

FL-5.3.2（K2）总结测试报告的目的、内容和受众

5.4 配置管理

FL-5.4.1（K2）总结配置管理是如何支持测试的

5.5 风险和测试

FL-5.5.1（K1）通过使用可能性和影响程度定义风险级别

FL-5.5.2（K2）辨别项目和产品风险

FL-5.5.3（K2）通过例子描述产品风险分析是如何影响测试完整性和范围

5.6 缺陷管理

FL-5.6.1（K3）编写缺陷报告以覆盖测试过程中发现的缺陷

5.1. 测试组织

5.1.1. 独立测试

测试任务可以由具有特定测试角色的人员完成，也可以由其他角色的人员（例如，客户）完成。由于作者和测试员的认知取向不同，一定程度的独立性常常使测试员能更有效地发现缺陷（参见第 1.5 节）。但是，（测试员的）独立性却无法替代（开发人员的）熟悉性，开发人员可以在自己的代码中高效地发现许多缺陷。

测试中的独立程度包括以下几类（独立性从低到高）：

- 没有独立的测试员；唯一可用的测试形式是开发人员测试他们自己的代码
- 开发团队或项目团队内的独立开发人员或测试员；这可能是开发人员测试他们同事的产品
- 组织内的独立测试团队或小组，向项目管理或企业执行管理层报告
- 来自业务组织或用户团体的独立测试员，或具有特定测试类型的专业知识的测试员，这些特定测试类型包括，例如易用性、安全性、性能、法规/合规性、或可移植性
- 组织外部的独立测试员，无论是工作现场（内包）或现场外（外包）

对于大多数类型的项目，通常最好有多个测试级别，其中一些级别由独立的测试员执行。开发人员应参与测试，特别是在较低级别，这样，开发人员能够控制他们自己工作的质量。

实现测试独立性的方式因软件开发生存周期模型而异。例如，在敏捷开发中，测试员可能是开发团队的一部分。在一些使用敏捷方法的组织中，这些测试员也可能被视为更大的独立测试团队的一部分。此外，在这类组织中，产品负责人可以开展验收测试，以在每次迭代结束时确认用户故事。

测试独立性的潜在好处包括：

- 独立的测试员与开发人员相比更可能会识别出不同类型的失效，因为他们有不同的背景、不一样的技术视角和具有不同的能力
- 独立测试员可以验证、质疑或反驳利益相关方在系统规格说明和实施过程中的假设
- 独立于供应商的测试员可以以一种正直和客观的方式报告正在测试的系统，而不会受到雇佣他们的公司的（政治）压力

测试独立性的潜在缺点包括：

- 与开发团队隔离，可能会导致与开发团队缺乏协作，会延迟向开发团队提供反馈，或与开发团队形成对抗关系
- 开发人员可能会失去对软件质量的责任感
- 独立测试员可能被视为瓶颈
- 独立测试员可能缺乏重要信息（例如，关于测试对象的信息）

许多公司能够认识到测试独立性的好处并避免其缺点。

5.1.2. 测试经理和测试员的任务

在本大纲中，涵盖了两个测试角色：测试经理和测试员。这两个角色开展的活动和任务取决于项目和产品背景、人员的技能以及组织。

测试经理的任务是总体负责测试过程和成功管理测试活动。测试管理角色可以是专业的测试经理或项目经理、开发经理或质量保证经理。在较大的项目或组织中，多个测试团队可以向测试经理、测试教练或测试协调员汇报，而每个团队由测试组长或主要核心测试员领导。

测试经理的典型任务可能包括：

- 制定或评审组织的测试方针和测试策略
- 通过考虑环境并理解测试目标和风险来规划测试活动。包括选择测试方法，估算测试的时间、工作量和成本，获取资源并定义测试级别和测试周期，以及规划缺陷管理
- 编写和更新测试计划
- 与项目经理、产品负责人和其他人员协调测试计划
- 与其他项目活动共享测试视角，如集成规划
- 启动针对测试的分析、设计、实施和执行并监督测试进度和结果，检查出口准则（或完成的定义）的状态，促使测试完成所有活动
- 根据收集的信息准备并提交测试进度报告和测试总结报告
- 根据测试结果和进度（有时记录在测试进度报告中，和/或在项目中已完成的其他测试的测试总结报告中）调整计划并采取所需的行动以进行测试控制
- 支持建立缺陷管理系统和针对测试件适当的配置管理
- 引入适当的度量来测量测试进度并评估测试和产品的质量
- 支持针对测试过程工具的选择和使用，包括工具选择预算的建议（以及可能的购买和/或支持），为试点项目分配时间和工作量，以及为使用工具方面提供持续支持
- 决定测试环境的实施
- 在组织内促进和支持测试员、测试团队和测试专业人员
- 培养测试员的技能和关心他们的职业发展（例如，通过培训计划、绩效评估、指导等）

测试经理的角色会因软件开发生存周期的不同而有所差异。例如，在敏捷开发中，上面提到的一些任务由敏捷团队处理，特别是那些与团队内部日常测试有关的任务，通常由团队内部的测试员完成。跨越多个团队或整个组织的一些任务、或与人事管理有关的，可以由开发团队以外的测试经理完成，他们有时被称为测试教练。有关管理测试过程的更多信息，参见 Black 2009。

测试员的典型任务可能包括：

- 评审并参与测试计划
- 分析、评审和评估需求、用户故事和验收准则、规格说明和模型的可测试性（即测试依据）
- 识别并记录测试条件，并获取测试用例、测试条件和测试依据之间的可追溯性
- 设计、建立和验证测试环境，通常与系统管理和网络管理协调
- 设计和实施测试用例和测试规程
- 准备并获取测试数据
- 创建详细的测试执行进度表

- 执行测试、评估结果并记录与预期结果之间的偏差
- 使用适当的工具来促进测试过程
- 根据需要自动化测试（可由开发人员或测试自动化专家支持）
- 评估非功能特性，例如性能效率、可靠性、易用性、安全性、兼容性和可移植性
- 评审他人开发的测试

这些角色的人员可以是从事测试分析、测试设计、特定测试类型或测试自动化的人员。根据与产品和项目相关的风险以及所选择的软件开发生存周期模型，不同的人员可以在不同的测试级别上担当测试员的角色。例如，在组件测试级别和组件集成测试级别，测试员的角色通常可以是开发人员。在验收测试级别，测试员的角色通常可以是业务分析师、领域专家和用户。在系统测试级别和系统集成测试级别，测试员的角色通常可以是独立的测试团队。在运行验收测试级别，测试员的角色通常由操作和/或系统管理员来完成。

5.2. 测试计划和估算

5.2.1. 测试计划的目的和内容

测试计划描述了开发和维护项目的测试活动。制定计划受组织层的测试方针和测试策略影响，也受开发生存周期和使用的方法（参见第 2.1 节）、测试范围、目标、风险、约束、重要性、可测试性和资源可用性的影响。

随着项目和测试计划的进展，测试计划中包含的信息也会越来越多，越来越详细。测试计划是一项持续的活动，贯穿整个产品的生存周期。（请注意，产品的生存周期可能超出项目范围，它还包括维护阶段。）应使用测试活动的反馈来识别不断变化的风险，以便调整计划。制定和记录规划可以在主测试计划中，也可以在单独的测试级别中，例如系统测试和验收测试，或者甚至可在单独的测试类型中，例如易用性测试和性能测试。测试计划活动可能包括以下内容，其中一些可能会记录在测试计划中：

- 确定测试的范围、目标和风险
- 定义整体测试方法
- 将测试活动集成并协调到软件生存周期活动中
- 确定要测试的内容，开展各种测试活动所需的人员和其他资源，以及如何开展测试活动
- 安排测试分析、设计、实施、执行和评估活动的进度表，可以按特定日期安排（例如，在顺序开发中）或以每次迭代的周境安排（例如，在迭代开发中）
- 选择测试监督和控制的度量
- 为测试活动编制预算
- 确定测试文档的详细程度和结构（例如，通过提供模板或示例文档）

测试计划的内容各不相同，可以超出上述主题。测试计划结构的样本以及测试计划的样本可以在 ISO 标准（ISO/IE /IEEE 29119-3）中找到。

5.2.2. 测试策略和测试方法

测试策略提供测试过程的一般描述，通常是在产品或组织级别上的。常见的测试策略类型包括：

- ***分析型***：此类测试策略基于对某些因素（例如，需求或风险）的分析。基于风险的测试是分析型方法的一个示例，这里是根据风险级别设计测试并确定测试的优先级。
- ***基于模型***：在这种类型的测试策略中，测试是基于产品的某些需求方面的模型设计的，例如功能、业务流程、内部结构、或非功能特性（例如，可靠性）。此类模型的示例包括业务过程模型、状态模型和可靠性增长模型。
- ***方法论型***：此类测试策略依赖于系统地使用一些预定义的测试集或测试条件，例如常见或可能缺陷类型的分类、重要质量特性列表或公司范围内的用于移动应用或网页的外观和感受标准。
- ***符合流程***（或符合标准）：此类测试策略是基于外部规则和标准来开展测试，包括分析、设计和实施测试，例如行业特定标准、过程文档、严格标识和使用的测试依据，或由组织制定或针对组织制定的任何过程或标准。
- ***指导型***（或咨询）：此类测试策略主要由利益相关方、业务领域专家或技术专家的建议、指导或指示驱动，这些利益相关方、业务领域专家或技术专家可能来自测试团队之外，甚至是在组织之外。
- ***规避回归型***：这种类型的测试策略的动机是希望避免现有功能的回归。此测试策略包括重用现有测试件（尤其是测试用例和测试数据）、回归测试的广泛自动化以及标准测试套件。
- ***应对型***：在这种类型的测试策略中，测试被动应对被测组件或系统以及测试执行期间发生的事件，而不是预先计划（如前面的策略所示）。测试经过设计和实施，并可能立即执行以响应从先前测试结果中获得的知识。探索性测试是应对型策略中常用的技术。

通常通过组合这些测试策略的类型来创建适当的测试策略。例如，基于风险的测试（分析型策略）可以与探索性测试（应对型策略）相结合；它们相互补充，并且在一起使用时可以实现更有效的测试。

虽然测试策略提供了测试过程的一般描述，而测试方法是为特定项目或版本定制的测试策略。测试方法是选择测试技术、测试级别和测试类型，以及定义入口准则和出口准则（或分别是就绪的定义和完成的定义）的起点。策略的定制是基于项目的复杂性和目标、正在开发的产品类型和产品风险分析相关的决策。所选方法取决于项目背景，并可能考虑诸如风险、安全、可用资源和技能、技术、系统性质（例如，定制与 COTS）、测试目标和法规等因素。

5.2.3. 入口准则和出口准则（就绪的定义和完成的定义）

为了有效控制软件和测试的质量，建议使用准则来定义特定测试活动何时开始以及活动何时完成。入口准则（敏捷开发中常称为就绪的定义）定义了进行特定测试活动的前置条件。如果不满足入口准则，则活动可能会更难、更耗时、成本更高、风险更大。出口准则（敏捷开发中常称为完成的定义）定义了为了声明测试级别或一组测试已完成所必须达到的条件。应为每个测试级别和测试类型定义入口和出口准则，并根据测试目标而有所不同。

典型的入口准则包括：

- 可测试的需求、用户故事、和/或模型的可用性（例如，遵循基于模型的测试策略时）
- 满足任何先前测试级别出口准则相关的测试项的可用性
- 测试环境的可用性
- 必要的测试工具的可用性
- 测试数据和其他必要资源的可用性

典型的出口准则包括：

- 已执行计划的测试
- 已实现规定的覆盖级别（例如，需求、用户故事、验收准则、风险、代码）
- 未解决的缺陷数量在规定的限度内
- 估计的遗留缺陷数量足够低
- 可靠性、性能效率、易用性、安全性和其他相关质量特性的评估级别已达到要求

即使没有满足出口准则，但由于预算已耗尽、预定的时间已到、和/或将产品推向市场的压力，测试活动被缩减是很常见的。如果项目的利益相关方和业务负责人已经评审并接受了无须进一步测试的情况下上线的风险，那么在这种情况下结束测试是可以接受的。

5.2.4. 测试执行进度表

一旦生成了各种测试用例和测试规程（某些测试规程可能会自动化）并集成到测试套件中，测试套件就可以安排在测试执行进度表中，该进度表定义了它们的运行顺序。测试执行进度表应考虑诸如优先级、依赖性、确认测试、回归测试和执行测试的最有效顺序等因素。

理想情况下，测试用例将基于其优先级次序运行，通常是首先执行具有最高优先级的测试用例。但是，如果测试用例具有依赖性或者测试的功能具有依赖性，则此方法可能不适用。如果具有较高优先级的测试用例依赖于具有较低优先级的测试用例，则必须首先执行此优先级较低的测试用例。同样，如果不同测试用例存在依赖关系，则无论其相对优先级如何，都必须对它们进行适当的顺序排序。确认测试和回归测试也必须根据变化后快速反馈的重要程度设置优先级，但这里同样需要考虑它们的依赖性。

在一些情况下，测试的不同顺序是可能的，伴随这些不同顺序会有不同的效率。在这种情况下，必须在测试执行效率与遵守优先级之间进行权衡。

5.2.5. 影响测试工作量的因素

测试工作量估算包括预估为满足特定项目、发布或迭代的测试目标所需的测试相关工作量。影响测试工作量的因素可能包括产品的特性、开发过程的特性、人员的特性、和测试结果，如下所示。

产品的特点

- 与产品相关的风险
- 测试依据的质量
- 产品规模
- 产品应用领域的复杂性
- 质量特性要求（例如，安全性、可靠性）
- 所需的测试文档详细程度
- 对法律和法规的合规性要求

开发过程的特点

- 组织的稳定性和成熟度
- 所使用的开发模型
- 测试方法
- 使用的工具
- 测试过程
- 时间压力

人的特点

- 参与人员的技能和经验，尤其是类似的项目和产品（例如，领域知识）
- 团队凝聚力和领导力

测试结果

- 发现缺陷的数量和严重程度
- 所需要的返工量

5.2.6. 测试估算方法

有许多估算方法用于确定适当测试所需的工作量。两种最常用的方法是：

- 基于度量的方法：根据先前类似项目的度量，或基于典型值来估算测试工作量
- 基于专家的方法：根据测试任务负责人或专家的经验来估算测试工作量

例如，在敏捷开发中，燃尽图是基于度量的方法的示例，当获取和确定剩余工作量并报告，然后与团队速度（敏捷项目中生产力的度量）一起使用，以确定团队在下一次迭代中可以完成的工作量；而计划扑克是基于专家的方法的一个例子，团队成员根据他们的经验估算出交付一个特征的工作量（ISTQB-CTFL-AT）。

在顺序开发模型的项目中，缺陷移除模型是基于度量方法的示例，其中获得和报告了缺陷的数量和移除它们所需的时间，然后为未来估算类似性质的项目提供了基础；而宽带德尔菲估算技术是基于专家的方法的一个例子，其中专家组根据他们的经验提供估算（ISTQB-CTAL-TM）。

5.3. 测试监督与控制

测试监督与控制的目的是收集信息并提供有关测试活动的反馈和可视化。可以手动或自动收集要监督的信息，这些信息可以用来评估测试进度并测量是否满足测试出口准则，或评估敏捷项目中定义的相关测试任务是否完成，例如满足产品风险、需求，或验收准则的覆盖目标。

测试控制描述了根据收集到和（可能）报告的信息和度量所采取的任何指导或纠正措施。其措施可能涵盖任何测试活动，并可能影响软件生存周期中的其他活动。

测试控制措施的示例包括：

- 当已识别的风险发生时（例如，软件交付延迟），重新确定测试优先级
- 由于测试环境或其他资源的可用或不可用而更改测试进度表
- 由于返工，重新评估测试项是否符合入口或出口准则

5.3.1. 测试中使用的度量

在测试活动期间和结束时收集度量，用以评估：

- 与计划的时间表和预算对应的进展
- 测试对象的当前质量
- 测试方法的充分性
- 与目标相关的测试活动的有效性

常见的测试度量包括：

- 在测试用例准备中，计划工作已完成的百分比（或计划测试用例已实施的百分比）
- 在测试环境准备中，计划工作已完成的百分比
- 测试用例执行（例如，测试用例运行/未运行、测试用例通过/失败，和/或测试条件通过/失败的数量）
- 缺陷信息（例如：缺陷密度、发现和修复的缺陷、失效率和确认测试结果）
- 需求、用户故事、验收准则、风险，或代码的测试覆盖
- 任务完成、资源分配和使用以及工作量
- 测试成本，包括与发现下一个缺陷的成本与收益，或执行下一个测试的收益相比的成本

5.3.2. 测试报告的目的、内容、和受众

测试报告的目的是在测试活动（例如，测试级别）期间和结束时，总结和传达测试活动信息。在测试活动期间准备的测试报告可能被称为测试进度报告，而在测试活动结束时准备的测试报告可能被称为测试总结报告。

在测试监督和控制期间，测试经理定期为利益相关方发布测试进度报告。除了测试进度报告和测试总结报告的常用内容之外，典型的测试进度报告还可能包括：

- 对照测试计划，测试活动和进度的状态
- 妨碍进度的因素
- 计划在下一个报告周期进行的测试
- 测试对象的质量

当达到出口准则时，测试经理会发布测试总结报告。此报告根据最新的测试进度报告和任何其他相关信息，提供所开展测试的总结。

典型的测试总结报告可能包括：

- 所开展测试的总结
- 测试周期内发生的情况的信息
- 计划的偏离，包括测试活动的进度、持续时间，或工作量偏差
- 对照出口准则或完成的定义，测试和产品质量的状态
- 已阻碍或继续阻碍进度的因素
- 缺陷、测试用例、测试覆盖、活动进度和资源消耗的度量（例如，如 5.3.1 中所述）
- 剩余风险（见 5.5 节）
- 生成的可重用的测试工作产品

测试报告的内容将根据项目、组织要求和软件开发生存周期而有所不同。例如，一个有许多利益相关方的复杂项目，或一个受管制的项目可能比一个快速软件更新需要更详细和更严格的报告。另一个例子，在敏捷开发中，测试进度报告可以包含在任务看板、缺陷摘要和燃尽图中，这可以在每日站会期间讨论（参见 ISTQB-CTFL-AT）。

除了根据项目周境裁剪定制的测试报告外，还应根据报告的受众裁剪定制测试报告。提交给技术受众或测试团队的测试报告中的信息类型和数量可能与执行总结报告中包含的信息类型和数量是不同的。在第一种情况下，有关缺陷类型和趋势的详细信息可能很重要。在后一种情况下，高级别报告（例如，按优先级的缺陷状态总结、预算、进度表、和通过/未通过/未测试的测试条件）可能更合适。

ISO 标准（ISO/IEC/IEEE 29119-3）涉及两种类型的测试报告，测试进度报告和测试完成报告（在本大纲中称为测试总结报告），并包含每种类型的结构和示例。

5.4.配置管理

配置管理的目的是在整个项目和产品生存周期期间，建立和维护组件或系统、测试件以及他们之间相互关系的完整性。

为了有效支持测试，配置管理可能涉及确保以下内容：

- 所有测试项都有唯一标识、版本控制、变更跟踪并彼此相互关联
- 所有测试件项都有唯一标识、版本控制、变更跟踪、彼此互相关联并且关联到测试项版本，以便在整个测试过程中保持可追溯性

- 所有已识别的文档和软件项在测试文档中明确引用

在测试计划期间，应该识别和实施配置管理过程和基础设施（工具）。

5.5. 风险和测试

5.5.1. 风险的定义

风险涉及未来可能发生负面后果的事件。风险级别取决于事件发生的可能性以及事件的影响程度（伤害）。

5.5.2. 产品和项目的风险

产品风险涉及工作产品（例如，规格说明、组件、系统或测试）可能无法满足其用户和/或利益相关方的合法需求的可能性。当产品风险与产品的特定质量特性（例如，功能性、可靠性、性能效率、易用性、安全性、兼容性、维护性和可移植性）相关联时，产品风险也称为质量风险。产品风险的例子包括：

- 软件可能无法根据规格说明执行其预期功能
- 软件可能无法根据用户、客户、和/或利益相关方的需要执行其预期功能
- 系统架构可能无法充分支持某些非功能性需求
- 在某些情况下，可能会不正确地执行特定计算
- 循环控制结构可能不正确地编码
- 对于高性能事务处理系统，响应时间可能不足
- 用户体验（UX）反馈可能无法满足产品预期

项目风险涉及的情况如果发生，可能会对项目实现目标的能力产生负面影响。项目风险的例子包括：

- 项目事件：
  - 延迟可能会出现在交付、任务完成，或满足出口准则或完成的定义时
  - 不精确的估算、将资金重新分配给更高优先级的项目，或整个组织的一般成本削减可能导致资金不足
  - 后期更改可能会导致大量的返工
- 组织事件：
  - 没有足够的技能和培训，员工不足
  - 人员事件可能会导致冲突和问题
  - 由于业务优先级冲突，用户、业务人员或领域专家可能无法支持
- 政治事件：
  - 测试员可能没有充分传达他们的需要和/或测试结果

- 开发人员和/或测试员可能无法跟进测试和评审中发现的信息（例如，没有改进开发和测试实践）
- 对测试可能存在不正确的态度或期望（例如，不了解测试期间发现缺陷的价值）

- 技术事件：
  - 可能没有很好地定义需求
  - 考虑到现有的限制，可能无法满足需求
  - 测试环境可能未按时准备就绪
  - 数据转换、迁移规划、和它们的工具支持可能延迟
  - 开发过程中的弱点可能会影响项目工作产品的一致性或质量，例如设计、编程、配置、测试数据、和测试用例
  - 较差的缺陷管理和类似问题可能导致累积缺陷和其他技术债务
- 供应商事件：
  - 第三方可能无法提供必要的产品或服务、或破产
  - 合同事件可能会给项目带来问题

项目风险可能会影响开发活动和测试活动。在某些情况下，项目经理负责处理所有项目风险，但测试经理有时也会负责与测试相关的项目风险。

5.5.3. 基于风险的测试和产品质量

风险用于关注测试期间所需的工作量。它用于决定何时何地开始测试，以及确定需要更多关注的区域。测试用于降低发生不良事件的可能性，或减少不良事件的影响。测试用作风险缓解活动，提供有关已识别风险的信息，并提供有关剩余（未解决）风险的信息。

基于风险的测试方法提供了降低产品风险水平的机会。它涉及产品风险分析，包括产品风险的识别以及每种风险的可能性和影响的评估。由此产生的产品风险信息用于指导测试计划、测试规格说明、测试用例的准备和执行，以及测试监督和控制。尽早分析产品风险有助于项目的成功。

在基于风险的方法中，产品风险分析的结果可用于：

- 决定要使用的测试技术
- 决定要开展的测试的特定级别和类型（例如，安全性测试、可达性测试）
- 决定测试的范围
- 确定测试优先级，以尽早发现重要缺陷
- 决定是否可以采用除测试之外的任何活动来降低风险（例如，为缺乏经验的设计员提供培训）

基于风险的测试利用项目利益相关方的集体知识和洞察力来进行产品风险分析。为确保产品失效的可能性最小化。风险管理活动提供了一种严格的方法：

- 分析（并定期重新评估）可能出现的错误（风险）
- 决定哪些风险的处理是很重要
- 实施行动以缓解这些风险

- 制定应急计划去处理可能成为实际事件的风险

此外，测试可能识别新风险，帮助确定应该缓解哪些风险，并降低风险的不确定性。

5.6. 缺陷管理

由于测试的目标之一是发现缺陷，因此应记录测试期间发现的缺陷。记录缺陷的方式可能会有所不同，具体取决于所测试的组件或系统的环境、测试级别和软件开发生存周期模型。应对所发现的任何缺陷进行调查，并从发现和分类到其解决进行跟踪（例如，缺陷的修正和缺陷修正后成功进行确认测试，缺陷延迟到后续版本再修改，接受缺陷作为永久产品限制等）。为了管理所有要解决的缺陷，组织应建立缺陷管理过程，其中包括工作流和分类规则。此过程必须与参与缺陷管理的所有人员达成一致，包括架构师、设计人员、开发人员、测试员和产品负责人。在某些组织中，缺陷记录和跟踪可能不一定很正式。

在缺陷管理过程中，某些报告可能会描述假阳性（误报）失效，而不是由于缺陷导致的实际失效。例如，当网络连接中断或超时，测试可能会失败。这类行为不是由测试对象中的缺陷引起的，但这是异常情况，需要进一步进行调查。测试员应尽量减少假阳性（误报）缺陷的数量。

在编码、静态分析、评审期间，或在动态测试过程中，或软件产品的使用中都可能会报告缺陷。可能会对代码或工作系统、或任何类型的文档（包括需求、用户故事和验收准则、开发文档、测试文档、用户手册或安装指南）中的事件报告为缺陷。为了获得有效和高效的缺陷管理过程，组织可以为缺陷的属性、分类和工作过程定义标准。

典型的缺陷报告包括以下目标：

- 向开发人员和其他利益相关方提供有关发生的任何不良事件的信息，使他们能够识别特定的影响，并通过最少的测试来再现和隔离问题，并根据需要纠正潜在的缺陷或以其他方式解决问题
- 为测试经理提供跟踪工作产品质量和对测试的影响（例如，如果报告了大量缺陷，测试员将花费大量时间报告缺陷而不是运行测试，并且需要更多的确认测试）
- 提供开发和测试过程改进的想法

动态测试期间提交的缺陷报告通常包括：

- 标识符
- 报告的缺陷的标题和简短摘要
- 缺陷报告的日期、发布组织和作者
- 识别测试项（正在测试的配置项）和环境
- 观察到缺陷所在的开发生存周期阶段
- 缺陷的描述，以便能再现和消除缺陷，包括日志、数据库转储、屏幕截图或记录（如果在测试执行期间发现的）
- 预期结果和实际结果
- 缺陷对利益相关方利益的影响范围或程度（严重性）

- 紧急/优先去修复
- 缺陷报告的状态（例如打开、延期、重复、等待修复、等待确认测试、重新打开、关闭）
- 结论、建议和批准
- 一般问题，例如由于缺陷而引起的变更影响到的其它区域
- 变更的历史记录，例如项目团队成员针对缺陷采取的一系列操作，以隔离、修理和确认缺陷已被修复
- 参考，包括发现问题的测试用例

当使用缺陷管理工具时，上述有些内容会自动包括和/或管理，例如，在缺陷管理工具的工作流中自动分配标识符、分配和更新缺陷报告状态等。在静态测试期间，特别是评审中发现的缺陷，通常以不同的方式记录，例如，在评审会议记录中。

可以在 ISO 标准（ISO/IEC/IEEE 29119-3）中找到缺陷报告内容的示例（其将缺陷报告称为事件报告）。

6. 测试的支持工具（40 分钟）

关键词

数据驱动测试（data-driven testing），关键词驱动测试（keyword-driven testing），测试自动化（test automation），测试执行工具（test execution tool），测试管理工具（test management tool）

测试工具的学习目标

6.1 测试工具的考虑

FL-6.1.1 （K2）根据目的以及支持的测试活动，对测试工具分类。

FL-6.1.2 （K1）识别测试自动化的益处和风险

FL-6.1.3 （K1）牢记测试执行和测试管理工具的特别注意事项

6.2 有效使用工具

FL-6.2.1 （K1）识别选择工具的主要原则

FL-6.2.2 （K1）记忆通过试点项目引入工具的目的

FL-6.2.3 （K1）识别组织中评估、实施、部署和持续支持测试工具的成功因素

6.1. 测试工具的考虑

测试工具可以用于支持一种或多种测试活动。这些工具包括：

- 直接用于测试的工具，如测试执行工具、测试数据准备工具
- 帮助管理需求、测试用例、测试规程、自动化测试脚本、测试结果、测试数据和缺陷等的工具，以及报告和监督测试执行的工具
- 用于分析和评估的工具
- 任何对测试有帮助的工具（从这个意义来说，电子表格也是测试工具）

6.1.1. 测试工具分类

根据实际情况，测试工具可以有如下一个或多个目的：

- 对于可自动重复执行的任务，或手动执行时需要消耗大量资源的任务（例如，测试执行、回归测试），可改进其测试活动的效率
- 在整个测试过程中，通过支持手动测试活动来提高测试活动效率（参见 1.4 节）
- 通过更多测试一致性和更高缺陷重现性，来改进测试活动质量
- 自动化无法通过手动执行的活动（例如：大规模性能测试）
- 增加测试的可靠性（例如：自动比较大数据或行为模拟）

工具可以按照多种规则进行分类，例如：目的、价格、许可证模式（例如：商业或开源）和使用的技术等。在本大纲中，是按照测试工具能支持的测试活动来进行分类。

有些工具只能支持或主要支持一种活动；有些工具可以支持多种活动，在这种情况下将它们分类到联系最紧密的那一类活动中。同一家供应商的测试工具，尤其是那些为了协同工作而设计的测试工具，可能会被集成到一个套件中。

某些类型的测试工具本身是植入式的，这意味着测试的实际结果可能会受到影响。比如，由于使用性能测试工具执行了额外的指令，可能导致实际的响应时间的不同；或者使用覆盖工具可能得到了不正确的代码覆盖率。使用植入式工具导致的结果也称为探测影响（探针效应）。

某些测试工具提供的支持可能更适合开发人员（比如在进行组件和集成测试时使用的工具）。在本节以下的分类中将这些工具用“(D)”来标记。

支持测试和测试件管理的工具

管理工具适用于整个软件开发生存周期中的所有测试活动。

支持测试和测试件管理的工具举例如下：

- 测试管理工具和应用生存周期管理工具（ALM）
- 需求管理工具（例如：与测试对象的可追溯性）
- 缺陷管理工具
- 配置管理工具
- 持续集成工具（D）

支持静态测试的工具

静态测试工具与第 3 章描述的活动与收益相关。工具举例如下：

- 静态分析工具(D)

支持测试设计和实施的工具

测试设计工具旨在帮助测试设计和实施时生成可用的工作产品，包括测试用例、测试规程和测试数据。工具举例如下：

- 基于模型的测试工具
- 测试数据准备工具

有时候，支持测试设计和实施的工具，也会支持测试执行和日志记录，或直接输出到其它支持测试执行和日志记录的工具。

支持测试执行和日志记录的工具

许多工具可以支持和提高测试执行和日志记录活动。工具举例如下：

- 测试执行工具（例如：执行回归测试）
- 覆盖工具（例如：需求覆盖、代码覆盖(D)）
- 测试用具(D)

支持性能测量和动态分析的工具

性能测量和动态分析工具对于支持性能和负载测试活动是必不可少的，因为这些活动无法有效地通过手工方式来完成。工具举例如下：

- 性能测试工具
- 动态分析工具(D)

支持专业测试要求的工具

除了支持一般测试过程的工具外，还有许多用于支持对非功能特性的专业测试工具。

6.1.2. 测试自动化的收益和风险

仅仅购买工具并不能保证测试自动化的成功。要让引入组织的新工具能获得真正持续的成效还需要大量投入。测试中使用工具具有潜在的收益和机会，但是同样存在风险。这对测试执行工具（通常用作测试自动化）来说尤其正确。

使用工具支持测试执行的潜在收益包括：

- 减少重复性的手工工作来节省时间（比如，执行回归测试、环境设置/拆除、重新输入相同测试数据，和代码规则检查）
- 更好的一致性和可重复性（比如，测试数据按照一致的方式产生，用工具按照相同的顺序

和频率执行测试，以及始终从需求出发进行测试）
- 更客观的评估（比如，静态测量、覆盖）
- 更容易得到测试的相关信息（比如，关于测试进展、缺陷发生率和性能的统计和图表）

使用工具支持测试的潜在风险：
- 对工具抱有不切实际的期望（包括功能性和易用性）
- 低估首次引入工具所需的时间、成本和工作量（包括培训和额外的专业知识）
- 低估从工具中获得较大和持续收益所需付出的时间和工作量（包括测试过程所需的变更和使用工具方法的持续改进）
- 低估对工具生成的测试工作产品进行维护所需的工作量
- 对工具过分依赖（替代测试设计或执行，或者对一些更适合手工测试的方面却使用自动测试工具）
- 忽视对测试工作产品的版本控制
- 忽视多个重要工具之间的关联和互操作性问题，例如：需求管理工具、配置管理工具、缺陷管理工具，和其他从不同供应商获得的工具
- 工具供应商破产、停止维护工具或将工具卖给其他供应商的风险
- 供应商对工具的支持、升级和缺陷修复支持不力
- 开源工具项目中止的风险
- 工具可能不再支持新平台或新技术的风险
- 工具所有权可能不清晰而带来的风险（例如：指导、升级等）

6.1.3. 测试执行和测试管理工具的特殊考虑

为了顺利和成功实施，在选择测试执行以及测试管理工具和将它们集成到组织内时，应考虑到许多方面。

测试执行工具

测试执行工具使用自动化的测试脚本执行测试对象。为了获得可观收益，经常需要为这类工具投入很多工作量。

- 捕获测试方法：通过记录（捕获）测试工程师的手动操作而生成的测试脚本，看起来似乎很吸引人，但是这种方法不适合大量的测试脚本。捕获的脚本只是一种线性表达，脚本内包含特定数据和操作。当发生意外事件时，这类脚本可能会很不稳定，并且需要随着系统用户界面的不断发展而需要不断进行维护。
- 数据驱动测试方法：这种测试方法是将测试输入和期望结果与脚本分离，通常可以存放在一个电子表格中，这样可以使用更通用测试脚本读取输入数据，从而用不同的数据执行相同的测试脚本。
- 关键字驱动测试方法：在这种测试方法中，通用脚本处理描述系统要执行操作的关键字（也称为行为字），然后调用这些关键字脚本来处理相关联的测试数据。

上面这些测试方法都需要有脚本语言方面的专业技术人员（测试工程师、开发人员或测试自动化专家）。当使用数据驱动或关键字驱动的测试方法时，不熟悉脚本语言的测试工程师也可以为这些预定义脚本创建测试数据和/或关键字。无论使用什么脚本技术，都需要比较每次测试的预期结果和

实际结果，可以是动态的（测试执行时）或者先存储，然后在后期（测试执行结束后）进行比较。

在 ISTQB-CTAL-TAE，Fewster 1999 和 Buwalda 2001 中给出了数据驱动和关键字驱动测试方法的更多细节和示例。

基于模型的测试（MBT）工具能够将功能规格说明以模型的方式呈现出来，比如活动图。该任务通常是通过系统设计人员来开展的。MBT 工具通过解释模型来生成测试用例规格说明，并且存储到测试管理工具和/或通过测试执行工具执行（参见 ISTQB-CTFL-MBT）。

测试管理工具

由于不同的原因，测试管理工具通常需要与其他工具或电子表格程序有接口，例如：

- 以便生成符合组织所需格式的有用信息
- 为了维护需求管理工具中针对需求的一致可追溯性
- 为了链接到配置管理工具中测试对象版本信息

当使用集成工具时（例如：应用生存周期管理），由于包含组织中其他团队使用的测试管理模块以及其他模块（例如：项目进度和预算信息），需要特别认真的考虑。

6.2. 工具的有效使用

6.2.1. 工具选择的主要原则

为组织选择工具所需要考虑的关键点有：

- 评估自己组织的成熟度和优缺点
- 识别引入工具能改进测试过程的机会
- 了解测试对象所使用的技术，以便选择与此技术相适合的工具
- 了解组织内已使用的构建和持续集成工具，以便确保工具的兼容与集成
- 根据明确的需求和客观的准则对工具进行评估
- 考虑工具是否提供免费试用期（以及多长时间）
- 评估供应商（包括培训、提供的支持及其他商业方面考量），或非商业性工具（例如：开源）的支持
- 针对工具使用的指导和培训，识别内部需求
- 评估培训需求时，需要考虑工作中将直接使用工具的人员的测试（以及测试自动化）技能
- 考虑各种许可证模式的优缺点（例如：商业或开源）
- 根据实际的情况估算成本-收益比（如果需要的话）

作为最后一步，应进行可行性研究（概念验证），以确定该工具是否在所测试的软件和当前基础设施内有效运行，或在必要时确定为有效使用该工具而需要对该基础设施进行的修改。

6.2.2. 组织引入工具的试点项目

完成工具选择和成功进行可行性研究（概念验证）后，将选择的工具引入组织通常从试点项目开始，试点项目有以下目的：

- 深入了解工具有关的知识，了解工具的优缺点
- 评估工具与现有过程以及实践的配合程度，确定哪些方面需要作修改
- 定义一套标准的方法来使用、管理、储存和维护工具及测试工作产品（比如，定义文件和测试的命名规则，选择编码标准，创建库和定义模块化测试套件）
- 评估在付出合理的成本后能否得到预期的收益
- 理解工具应该收集和报告的度量，并对工具进行配置，以保证度量的获取和报告

6.2.3. 工具的成功因素

在组织内成功地评估、实施、部署和持续支持工具的因素包括：

- 逐步在组织的其余部分推广工具
- 调整并改进过程来配合工具的使用
- 为工具使用者提供培训、辅导和指导
- 定义工具使用指南（例如：自动化的内部标准）
- 实施一种在实际使用中收集工具使用信息的方法
- 监督工具的使用和收益
- 为测试团队使用工具提供支持
- 在所有团队内收集经验和教训

同样重要的是保证工具在技术上和组织上集成到软件开发生存周期中，同时包括独立负责操作的组织和/或第三方供应商。

有关使用测试执行工具的经验和建议，可以参见 Graham 2012。

7. 参考文献

标准

ISO/IEC/IEEE 29119-1 (2013) Software and systems engineering - Software testing - Part 1: Conceptsand definitions

ISO/IEC/IEEE 29119-2 (2013) Software and systems engineering - Software testing Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2013) Software and systems engineering - Software testing - Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2015) Software and systems engineering - Software testing - Part 4: Testtechniques

ISO/IEC 25010, (2011) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models
GB/T 25000.10-2016 系统与软件工程系统与软件质量要求和评价016 (SQu
第统与部分：系统与软件质量模型（中文译者注）

ISO/IEC 20246: (2017) Software and systems engineering — Work product reviews

UML 2.5, Unified Modeling Language Reference Manual, http://www.omg.org/spec/UML/2.5.1/, 2017

ISTQB®文档

ISTQB Glossary
ISTQB® 软件测试专业术语中英文对照表（中文版已发布）

ISTQB® Foundation Level Overview 2018
ISTQB® 基础级概述2018

ISTQB-CTFL-MBT Foundation Level Model-Based Tester Extension Syllabus
ISTQB® 基础级扩展大纲基于模型测试工程师（中文版已发布）

ISTQB-CTFL-AT Foundation Level Agile Tester Extension Syllabus
ISTQB® 基础级扩展大纲敏捷测试工程师（中文版已发布）

ISTQB-CTAL-TA Advanced Level Test Analyst Syllabus
ISTQB® 高级测试分析师大纲（中文版已发布）

ISTQB-CTAL-TTA Advanced Level Technical Test Analyst Syllabus
ISTQB® 高级技术测试分析师大纲（中文版已发布）

ISTQB-CTAL-TM Advanced Level Test Manager Syllabus
ISTQB® 高级测试经理大纲（中文版已发布）

ISTQB-CTAL-SEC Advanced Level Security Tester Syllabus
ISTQB® 高级安全性测试工程师大纲

ISTQB-CTAL-TAE Advanced Level Test Automation Engineer Syllabus
ISTQB® 高级测试自动化工程师大纲（中文版已发布）

ISTQB-CTEL-TM Expert Level Test Management Syllabus
ISTQB® 专家级测试管理大纲

ISTQB-CTEL-ITP Expert Level Improving the Test Process Syllabus
ISTQB® 专家级改进测试过程大纲（中文版已发布）

书籍和论文

Beizer, B. (1990) Software Testing Techniques (2e), Van Nostrand Reinhold: Boston MA

Black, R. (2017) Agile Testing Foundations, BCS Learning & Development Ltd: Swindon UK

Black, R. (2009) Managing the Testing Process (3e), John Wiley & Sons: New York NY

Buwalda, H. et al. (2001) Integrated Test Design and Automation, Addison Wesley: Reading MA

Copeland, L. (2004) A Practitioner's Guide to Software Test Design, Artech House: Norwood MA

Craig, R. and Jaskiel, S. (2002) Systematic Software Testing, Artech House: Norwood MA
系统的软件测试，电子工业出版社，2003

Crispin, L. and Gregory, J. (2008) Agile Testing, Pearson Education: Boston MA
敏捷软件测试，清华大学出版社，2010

Fewster, M. and Graham, D. (1999) Software Test Automation, Addison Wesley: Harlow UK
软件测试自动化技术与实例详解，电子工业出版社，2000

Gilb, T. and Graham, D. (1993) Software Inspection, Addison Wesley: Reading MA

Graham, D. and Fewster, M. (2012) Experiences of Test Automation, Pearson Education: Boston MA
自动化测试最佳实践，机械工业出版社，2013

Gregory, J. and Crispin, L. (2015) More Agile Testing, Pearson Education: Boston MA
深入敏捷测试，清华大学出版社，2017

Jorgensen, P. (2014) Software Testing, A Craftsman's Approach (4e), CRC Press: Boca Raton FL
软件测试C Press: Bo（原书第ss:，机械工业出版社，2017

Kaner, C., Bach, J. and Pettichord, B. (2002) Lessons Learned in Software Testing, John Wiley & Sons:New York NY

Kaner, C., Padmanabhan, S. and Hoffman, D. (2013) The Domain Testing Workbook, Context-DrivenPress: New York NY

Kramer, A., Legeard, B. (2016) Model-Based Testing Essentials: Guide to the ISTQB® Certified ModelBased Tester: Foundation Level, John Wiley & Sons: New York NY

Myers, G. (2011) The Art of Software Testing, (3e), John Wiley & Sons: New York NY
软件测试的艺术（原书第3版），机械工业出版社，2012

Sauer, C. (2000) “The Effectiveness of Software Development Technical Reviews: A BehaviorallyMotivated Program of Research,” IEEE Transactions on Software Engineering, Volume 26, Issue 1, pp 1-

Shull, F., Rus, I., Basili, V. July 2000. “How Perspective-Based Reading can Improve RequirementInspections.” IEEE Computer, Volume 33, Issue 7, pp 73-79

van Veenendaal, E. (ed.) (2004) The Testing Practitioner (Chapters 8 - 10), UTN Publishers: TheNetherlands

Wiegers, K. (2002) Peer Reviews in Software, Pearson Education: Boston MA
软件同级评审，机械工业出版社，2003

Weinberg, G. (2008) Perfect Software and Other Illusions about Testing, Dorset House: New York NY
完美软件电子工业出版社，2009

其它资源（本大纲中未直接引用）

Black, R., van Veenendaal, E. and Graham, D. (2019) Foundations of Software Testing: ISTQB® Certification (4e), Cengage Learning: London UK

Hetzel, W. (1993) Complete Guide to Software Testing (2e), QED Information Sciences: Wellesley MA

软件测试基础教程：第件测试基础教程，2009

8. 附录 A——课程大纲背景

本文档的历史

本文档是国际软件测试认证委员会(ISTQB®) 认证测试工程师基础级大纲,是 ISTQB®（www.istqb.org）批准的第一级别（初级）的国际资质。

本文件由国际软件测试认证委员会(ISTQB®)任命的成员组成的工作组于 2019 年 6 月至 8 月编制,更新内容是在各成员国委员会使用了 2018 年基础级教学大纲后经过评审添加的。

先前的基础级 2018 教学大纲在 2014-2018 年期间，国际软件测试认证委员会（ISTQB®）指定了一些软件方面的专家组成一个工作组，来准备本文档。本基础级大纲 2018 版本经 ISTQB® 各成员国委员会代表初步初步评审后，再由国际软件测试团体的代表们评审。

基础级认证资质的目标

- 为测试作为一门必要且专业的软件工程分支获得认可；
- 为测试工程师职业发展提供一个标准化的框架；
- 使具有专业水准的合格测试工程师能被雇主、客户和同行认可，并且提升测试工程师的竞争力；
- 在软件工程原则下，促进一致和良好的测试实践；
- 确定与产业相关且有价值的测试主题；
- 使得软件供应商通过宣传测试工程师招聘政策，雇佣认证测试工程师，从而获得相对于对手更有竞争力的商业优势；
- 为测试工程师和对测试感兴趣的人们提供一个机会，来获得国际上认可的测试资质证书。

国际资质认证的目标

- 在不同国家之间进行测试技能方面的相互比较；
- 测试工程师在不同的国家之间进行交流更加容易；
- 在跨不同国家项目和国际项目上，不同国家和地区的测试工程师可以对测试问题有一个共同的理解；
- 增加全球范围内具有资质的测试工程师数量；
- 基于国际组织发起，将比某个国家的组织具有更大的影响力和更高价值；
- 通过本大纲和术语，开发有关测试的国际通用理解和知识体系，来提升所有参与者的测试知识水平；
- 促进更多的国家将测试工作职业化；
- 测试工程师可以用本国语言获得国际通用的资质证书；
- 在不同的国家之间共享测试知识和资源；
- 通过更多国家的参与，使得测试工程师和该资质得到国际认可。

本资质认证的入门要求

参加 ISTQB® 认证测试工程师基础级考试的入门要求是考生对软件测试有兴趣。然而，我们还是强烈建议考生同时具有：

- 在软件开发或软件测试领域具有基本的行业背景，比如有 6 个月的系统测试、用户验收测试，或者软件开发等的经验；
- 参加有 ISTQB® 授权（ISTQB® 认可的某个成员国委员会），并符合 ISTQB® 标准的学习课程。

软件测试领域基础级认证的背景和历史

独立的软件测试工程师认证开始于英国计算机协会 ISEB（信息系统考试委员会），其软件测试委员会（www.bcs.org.uk/iseb）成立于 1998 年。在 2002 年德国的 ASQF 开始实施德国的测试工程师资质认证计划（www.asqf.de）。本大纲基于英国的 ISEB 和德国的 ASQF 的大纲内容，对它们进行了一些更新以及增加了一些新的内容，并且将重点转向了为测试工程师提供更多实践帮助。

在本国际认证推出以前获得的软件测试基础级认证证书（比如从 ISEB、ASQF 或 ISTQB® 认可的成员国委员会）被视为和本国际认证一样有效。基础级认证证书不会过期失效，也不需要重新进行更新。获得证书的时间在证书上注明。

在每个参与国，具体的事务由 ISTQB® 认可的国家或地区软件测试委员会来管理。成员国委员会的具体职责由 ISTQB® 制定，由每个成员国委员会具体实施。成员国委员会的具体职责包括培训机构的授权和认证考试设置等。

9. 附录 B——学习目标/知识认知级别

下面定义的学习目标适合于本大纲，大纲中的每个主题都会根据其学习目标进行考试。

级别 1：牢记（Remember）（K1）

考生应识别、牢记和记忆术语或者概念的内容。

关键字：识别（identify）、牢记（remember）、回忆（retrieve）、记忆（recall）、认出（recognize）、认知（know）。

例子

能够识别“失效”的定义：

- “不能向最终用户或其他利益相关方提供服务”，或
- “组件或者系统的实际运行情况与期望的发布、服务或结果背离”。

级别 2：理解（Understand）（K2）

考生应能选择与大纲主题相关的陈述理由或解释，同时对测试概念能够进行总结、比较、分类并举例说明。

关键字：总结（summarize）、概括（generalize）、摘要（abstract）、归类（classify）、比较（compare）、映射（map）、对比（contrast）、举例说明（exemplify）、解释（interpret）、翻译（translate）、描述（represent）、推断（infer）、结论（conclude）、分类（categorize）、构建模块（construct models）、解释（explain）、区分（differentiate）、描述（describe）、辨别（distinguish）、给出（give）。

例子

请解释测试应该尽早进行设计的原因：

- 在缺陷移除成本低的时候就发现它们；
- 尽早发现那些最重要的缺陷。

请解释集成测试和系统测试之间的异同：

- 相同：需测试多个组件，可以包括非功能性的测试类型；
- 不同：集成测试关注于接口和交互，而系统测试则关注于从全系统角度进行测试，如端到端的流程。

级别 3：应用（Apply）（K3）

考生应能够选择测试概念或者技术的正确应用，并将其应用到给定的场景中。

关键字：实施（implement）、执行（execute）、运用（use）、遵循流程（follow a procedure）、使用流

程（apply a procedure）。

例子

- 可以确认有效和无效等价类的边界值。
- 能从给定的状态转换图选择测试用例，以覆盖所有的状态转换。

参考文献（针对学习目标的认知级别）

Anderson,L.W.and Krathwonhl,D.R.（eds）（2001）A Taxonomy for Learning,Teaching,and Assessing:A Revision of Bloom's Taxonomy of Educational Objectives,Allyn & Bacon：Boston MA

10. 附录 C——发布备注

ISTQB® 基础教学大纲 2018 V3.1 是在 2018 版基础上的一个小的更新，针对 2018 V3.1 版本说明已经创建，每个章节都有概述。此外，还发布了一个修改模式（含变化标识）的基础级教学大纲 2018 V3.1 供参考比较。

ISTQB® 基础级大纲 2018 版对 2011 版的重大更新和重写。为此，没有针对每个章和节的详细发布备注。然而，本处将给出修订原则的汇总。另外，有一份独立发布的备注文档，ISTQB® 给出了 2018 版本与 2011 版本学习目标之间的追溯信息，包括增加、更新或删除的学习目标。

从 2017 年开始超过 55 万人在 100 多个国家参加基础级考试，全球范围超过 50 万人获得认证测试工程师资质。如果期望他们都读过基础级大纲从而通过了考试，这将使得基础级大纲成为至今为止最多人读过的软件测试文档。

本次主要更新涉及继承以及提高其价值，使 ISTQB® 在全球测试团体内传达给接下来的 50 万人。

本版本中，所有学习目标被编辑成原子化，以便生成从每个学习目标到与学习目标相关的内容节（以及考试问题）的清晰追踪，以及从内容节（以及考试问题）返回到相关学习目标的清晰追踪。另外，通过采用其它 ISTQB® 大纲使用的经证明的启发式教学和规则，根据每一章覆盖的学习目标分析，每章的时间分配比 2011 版本更实际。

虽然这是基础级大纲，表达的是已经过时间检验的最佳实践和技术，我们的修订使得材料表述更现代化，特别在软件开发方法（例如：敏捷和持续交付）和技术（例如：物联网）的术语方面。我们已更新参考标准，它们的更新如下：

1. ISO/IEC/IEEE 29119 代替 IEEE Standard829

2. ISO/IEC 25010 代替 ISO 9126。

3. ISO/IEC 20246 代替 IEEE 1028。

另外，自从最近十年 ISTQB® 文件夹显著增长，我们增加了与其它 ISTQB® 大纲的广泛交叉参考。相关之处经仔细地评审以使得与所有大纲和 ISTQB® 术语一致。本版本的目标是更容易阅读、理解、学习和翻译，关注于提升有用的实践，以及知识和技能之间的平衡。

本版本所做修改的详细分析，可以参见基础级教学大纲 2018，参看 ISTQB® 认证测试工程师基础级版本说明 2018。

11. 索引

【以下空白】