

认证测试工程师 基础级 CTFL 大纲

版本: EN4.0_CN1.3

发布日期: 2024 年 4 月 11 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



若您对此文档有任何问题, 欢迎您扫码添加【官方微信号】反馈。

版权声明

版权声明© 国际软件测试认证委员会（以下简称 ISTQB®）。

ISTQB®是国际软件测试认证委员会的注册商标。

2023 年版°的基础级 v4.0 教学大纲的版权作者：Renzo Cerquozzi、Wim Decoutere、Klaudia Dussa-Zieger、Jean-François Riverin、Arnika Hryszko、Martin Klonk、Michaël Pilaeten、Meile Posthuma、Stuart Reid、Eric Riou du Cosquer（主席）、Adam Roman、Lucjan Stapp、Stephanie Ulrich（副主席）、Eshraka Zakaria。

2019 年版°的更新作者：Klaus Olsen（主席）、Meile Posthuma、Stephanie Ulrich。

2018 年版°更新作者：Klaus Olsen（主席）、Tauhida Parveen（副主席）、Rex Black（项目管理）、Debra Friedenber、Matthias Hamburg、Judy McKay、Meile Posthuma、Hans Schaefer、Radoslaw Smilgin、Mike Smith、Steve Toms、Stephanie Ulrich、Marie Walsh、Eshraka Zakaria。

2011 年版°更新作者：Thomas Müller(主席)、Debra Friedenber 和 ISTQB®基础级工作组。

2010 年版°更新作者：Thomas Müller(主席)、Armin Beer、Martin Klonk、Rahul Verma。

2007 年版°更新作者：Thomas Müller(主席)、Dorothy Graham、Debra Friedenber、Erik van Veenendaal。

2005 年版°作者：Thomas Müller(主席)、Rex Black、Sigrid Eldh、Dorothy Graham、Klaus Olsen、Maaret Pyhäjärvi、Geoff Thompson、Erik van Veenendaal。

版权所有。作者特此将版权转让给 ISTQB®。作者（作为当前版权持有者）和 ISTQB®（作为未来版权持有者）已同意以下使用条件：

- 作为非商业化用途的摘录和复制，在来源得到确认的情况下是可以的。对于培训机构，认可此大纲的作者和 ISTQB®是此大纲的来源和版权所有者，并且此类培训课程的任何广告只有在收到 ISTQB®认可的成员国委员会（中国是 CSTQB®）对培训材料的正式认证后才能提及大纲，任何经认证的培训机构都可以使用本大纲作为培训课程的基础。
- 如果认可作者和 ISTQB® 是此大纲的来源和版权所有者，任何个人或团体都可以使用本大纲作为文章和书籍的基础。
- 未经 ISTQB®书面批准，禁止以其他方式使用本大纲。
- ISTQB®认可的任何成员国委员会都可以翻译本大纲，前提是他们在教学大纲的翻译版本中复制上述版权声明。

修订记录

版本	日期	备注
CTFL v4.0	2023 年 4 月 21 日	CTFL v4.0 - 常规发布版本
CTFL v3.1.1	2021 年 7 月 1 日	CTFL v3.1.1 - 更新了版权所有和 Logo
CTFL v3.1	2019 年 11 月 11 日	CTFL v3.1 - 微小更新的维护发布
ISTQB® 2018	2018 年 4 月 27 日	CTFL v3.0 - 候选常规发布
ISTQB® 2011	2011 年 4 月 1 日	CTFL 大纲维护发布
ISTQB® 2010	2010 年 3 月 30 日	CTFL 大纲维护发布
ISTQB® 2007	2007 年 5 月 1 日	CTFL 大纲维护发布
ISTQB® 2005	2005 年 7 月 1 日	认证测试人员基础级大纲 V1.0
ASQF v2.2	2003 年 7 月	ASQF 基础级大纲 V2.2 “软件测试基础大纲”
ISEB v2.0	1999 年 2 月 25 日	ISEB 软件测试基础大纲 V2.0

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
EN4.0_CN1.2	2024年2月21日	修改：仪表盘->dashboard；test-first->测试先行
EN4.0_CN1.1	2024年1月4日	中文版部分内容修正。
EN4.0_CN1.0	2023年10月11日	CTFL 4.0 中文版常规发布版本
ISTQB®2018中文版	2019年8月25日	CTFL 3.0 维护版发布
ISTQB®2018中文版	2019年3月18日	CTFL 3.0 中文版常规发布
ISTQB®2011中文版 修订版2	2016年12月1日	ISTQB®2011 年版修订后维护发布
ISTQB®2011中文版 修订版1	2015年5月6日	ISTQB®2011 年版修订后维护发布

版本	日期	备注
ISTQB®2011中文版 V13	2011年12月15日	ISTQB®2011 年版修订后维护发布
ISTQB®2011中文版 v12	2011年7月1日	ISTQB®2011 版进行修订后维护发布
ISTQB®2010中文版 评审版 V01	2010年8月12日	ISTQB®2010 版进行修订后维护发布
ISTQB®2007中文版	2008年8月28日	ISTQB®2007 年版 中文版 常规发布
CSTQB®2007版	2007年7月1日	ISTQB®2005 年版 中文版本常规发布
CSTQB®2005版	2005年7月1日	在CSTQB®成立之前已在翻译大纲的相关内容

中国软件测试认证委员会

目录

版权声明.....	2
修订记录.....	3
目录.....	5
致谢.....	8
0. 大纲简介.....	11
0.1 本大纲的目的	11
0.2 软件测试领域认证测试工程师基础级	11
0.3 测试人员的职业发展路径	11
0.4 商业价值	12
0.5 可考核的学习目标和知识认知级别	12
0.6 基础级认证考试	13
0.7 课程认证	13
0.8 标准的处理	13
0.9 保持时效性	13
0.10 详细级别	13
0.11 大纲的结构	14
1. 测试基础 - 180 分钟	15
1.1 什么是测试?	16
1.1.1 测试目的.....	16
1.1.2 测试与调试.....	17
1.2 为什么需要测试?	17
1.2.1 测试对成功的贡献.....	17
1.2.2 测试和质量保证 (QA).....	18
1.2.3 错误、缺陷、失效和根本原因.....	18
1.3 测试原则	19
1.4 测试活动、测试件和测试角色	20
1.4.1 测试活动和任务.....	20
1.4.2 周境中的测试过程.....	21
1.4.3 测试件.....	21
1.4.4 测试依据和测试件之间的可追溯性.....	22
1.4.5 测试活动中的角色.....	22
1.5 测试中的基本技能和良好实践	23
1.5.1 测试所需的通用技能.....	23
1.5.2 完整团队方法.....	23
1.5.3 测试独立性.....	24
2. 软件开发生存周期中的测试 - 130 分钟	25
2.1 软件开发生存周期中的测试	26
2.1.1 软件开发生存周期对测试的影响.....	26
2.1.2 软件开发生存周期与良好的测试实践.....	26
2.1.3 测试是软件开发的驱动力.....	27
2.1.4 DevOps 与测试	27
2.1.5 左移的方法.....	28
2.1.6 回顾与过程改进.....	29
2.2 测试级别和测试类型	29

2.2.1	测试级别.....	29
2.2.2	测试类型.....	30
2.2.3	确认测试和回归测试.....	31
2.3	维护测试.....	32
3.	静态测试 — 80 分钟.....	33
3.1	静态测试基础.....	34
3.1.1	静态测试可检查的工作产品.....	34
3.1.2	静态测试的价值.....	34
3.1.3	静态测试和动态测试的差异.....	35
3.2	反馈和评审过程.....	36
3.2.1	利益相关方早期和频繁反馈的好处.....	36
3.2.2	评审过程的活动.....	36
3.2.3	评审的角色和职责.....	37
3.2.4	评审类型.....	37
3.2.5	评审的成功因素.....	38
4.	测试分析和设计 - 390 分钟.....	39
4.1	测试技术概述.....	40
4.2	黑盒测试技术.....	40
4.2.1	等价类划分.....	40
4.2.2	边界值分析.....	41
4.2.3	判定表测试.....	42
4.2.4	状态转移测试.....	43
4.3	白盒测试技术.....	43
4.3.1	语句测试和语句覆盖.....	44
4.3.2	分支测试和分支覆盖.....	44
4.3.3	白盒测试的价值.....	45
4.4	基于经验的测试技术.....	45
4.4.1	错误猜测.....	45
4.4.2	探索性测试.....	46
4.4.3	基于检查表的测试.....	46
4.5	基于协作的测试方法.....	47
4.5.1	协作用户故事编写.....	47
4.5.2	验收准则.....	47
4.5.3	验收测试驱动开发 (ATDD).....	48
5.	管理测试活动 - 335 分钟.....	49
5.1	测试规划.....	50
5.1.1	测试计划的目的是内容.....	50
5.1.2	测试人员对迭代和发布规划的贡献.....	50
5.1.3	入口准则和出口准则.....	51
5.1.4	估算技术.....	51
5.1.5	测试用例优先级排序.....	52
5.1.6	测试金字塔.....	53
5.1.7	测试象限.....	53
5.2	风险管理.....	53
5.2.1	风险定义和风险属性.....	54
5.2.2	项目风险和产品风险.....	54
5.2.3	产品风险分析.....	55

5.2.4	产品风险控制.....	55
5.3	测试监测、测试控制和测试完成.....	56
5.3.1	测试中使用的度量.....	56
5.3.2	测试报告的目的、内容和受众.....	57
5.3.3	沟通测试状态.....	58
5.4	配置管理.....	58
5.5	缺陷管理.....	59
6.	测试工具 - 20 分钟.....	60
6.1	测试活动中的工具支持.....	61
6.2	测试自动化的收益和风险.....	61
7.	参考文献.....	63
8.	附录 A - 学习目标 / 知识认知等级.....	66
9.	附录 B - 商业价值与学习目标的可追溯性矩阵.....	68
10.	附录 C - 发布说明.....	74

致谢

本大纲由 ISTQB® 成员国大会于 2023 年 4 月 21 日正式发布。

本大纲由 ISTQB® 基础级工作组团队和敏捷工作组团队联合制作完成：Laura Albert、Renzo Cerquozzi（副主席）、Wim Decoutere、Klaudia Dussa-Zieger、Chintaka Indikadahena、Arnika Hryszko、Martin Klonk、Kenji Onishi、Michaël Pilaeten（联合主席）、Meile Posthuma、Gandhinee Rajkomar、Stuart Reid、Eric Riou du Cosquer（联合主席）、Jean-François Riverin、Adam Roman、Lucjan Stapp、Stephanie Ulrich（联合主席）、Eshraka Zakaria。

团队感谢 Stuart Reid、Patricia McQuaid 和 Leanne Howard 的技术评审以及评审团队和各成员国委员会的建议和意见。

以下人员参与了本大纲的评审、评论和投票：Adam Roman、Adam Scierski、Ágota Horváth、Ainsley Rood、Ale Rebon Portillo、Alessandro Collino、Alexander Alexandrov、Amanda Logue、Ana Ochoa、André Baumann、André Verschelling、Andreas Spillner、Anna Miazek、Armin Born、Arnd Pehl、Arne Becher、Attila Gyúri、Attila Kovács、Beata Karpinska、Benjamin Timmermans、Blair Mo、Carsten Weise、Chinthaka Indikadahena、Chris Van Bael、Ciaran O’Leary、Claude Zhang、Cristina Sobrero、Dandan Zheng、Dani Almog、Daniel Sæther、Daniel van der Zwan、Danilo Magli、Darvay Tamás Béla、Dawn Haynes、Dena Pauletti、Dénes Medzihradszky、Doris Dötzer、Dot Graham、Edward Weller、Erhardt Wunderlich、Eric Riou Du Cosquer、Florian Fieber、Fran O’Hara、François Vaillancourt、Frans Dijkman、Gabriele Haller、Gary Mogyorodi、Georg Sehl、Géza Bujdosó、Giancarlo Tomasig、Giorgio Pisani、Gustavo Márquez Sosa、Helmut Pichler、Hongbao Zhai、Horst Pohlmann、Ignacio Trejos、Ilia Kulakov、Ine Lutterman、Ingvar Nordström、Iosif Itkin、Jamie Mitchell、Jan Giesen、Jean-François Riverin、Joanna Kazun、Joanne Tremblay、Joëlle Genois、Johan Klintin、John Kurowski、Jörn Münzel、Judy McKay、Jürgen Beniermann、Karol Frühauf、Katalin Balla、Kevin Kooh、Klaudia Dussa-Zieger、Klaus Erlenbach、Klaus Olsen、Krisztián Miskó、Laura Albert、Liang Ren、Lijuan Wang、Lloyd Roden、Lucjan Stapp、Mahmoud Khalaili、Marek Majernik、Maria Clara Choucair、Mark Rutz、Markus Niehammer、Martin Klonk、Márton Siska、Matthew Gregg、Matthias Hamburg、Mattijs Kemmink、Maud Schlich、May Abu-Sbeit、Meile Posthuma、Mette Bruhn-Pedersen、Michal Tal、Michel Boies、Mike Smith、Miroslav Renda、Mohsen Ekssir、Monika Stocklein Olsen、Murian Song、Nicola De Rosa、Nikita Kalyani、Nishan Portoyan、Nitzan Goldenberg、Ole Chr. Hansen、Patricia McQuaid、Patricia Osorio、Paul Weymouth、

Pawel Kwasik、Peter Zimmerer、Petr Neugebauer、Piet de Roo、Radoslaw Smilgin、Ralf Bongard、Ralf Reissing、Randall Rice、Rik Marselis、Rogier Ammerlaan、Sabine Gschwandtner、Sabine Uhde、Salinda Wickramasinghe、Salvatore Reale、Sammy Kolluru、Samuel Ouko、Stephanie Ulrich、Stuart Reid、Surabhi Bellani、Szilard Szell、Tamás Gergely、Tamás Horváth、Tatiana Sergeeva、Tauhida Parveen、Thaer Mustafa、Thomas Eisbrenner、Thomas Harms、Thomas Heller、Tobias Letzkus、Tomas Rosenqvist、Werner Lieblang、Yaron Tsubery、Zhenlei Zuo and Zsolt Hargitai。

ISTQB®基础级（2018 年版）工作组成员：Klaus Olsen（主席）、Tauhida Parveen（副主席）、Rex Black（项目经理）、Eshraka Zakaria、Debra Friedenberg、Ebbe Munk、Hans Schaefer、Judy McKay、Marie Walsh、Meile Posthuma、Mike Smith、Radoslaw Smilgin、Stephanie Ulrich、Steve Toms、Corne Kruger、Dani Almog、Eric Riou du Cosquer、Igal Levi、Johan Klin、Kenji Onishi、Rashed Karim、Stevan Zivanovic、Sunny Kwon、Thomas Müller、Vipul Kocher、Yaron Tsubery 和给出建议的各成员国委员会。

本大纲的中文版本由 ISTQB® 成员国委员会 CSTQB® 负责并于 2023 年 10 月 11 日正式发布。

本大纲的中文版本由 CSTQB® 基础级本地化工作组团队制作完成，参加翻译和评审工作的成员有（按姓氏拼音排序）：柴阿峰、崔启亮（联合质量保证）、崔哲、翟宏宝、董昕、贺炘、李颖丽、刘海英、刘晓更、商莉、杨婷、张银萍、周震漪（组长，联合质量保证）、朱伟、祝国松、左振雷。

致谢企业（企业名按首字母排序）：

德中睿智互联网技术(上海)有限公司



广州赛宝认证中心服务有限公司



0. 大纲简介

0.1 本大纲的目的

本大纲是国际软件测试认证基础级大纲。国际软件测试认证委员会（以下简称 ISTQB®）提供标准的教学大纲，用途如下：

1. 各成员国委员会翻译成本地语言并授权给培训机构。成员国委员会可以根据特定的语言调整教学大纲，并修改参考文献列表以适应本地出版物。
2. 认证机构使用当地语言编写适合本教学大纲学习目标的考试题。
3. 培训机构编制课件并确定适当的教学方法。
4. 认证考试的考生准备认证考试（认证考试可以作为培训课程的一部分或独立进行）。

国际软件和系统工程界以此推动软件和系统测试专业的发展，并作为著书和写文章的基础。

0.2 软件测试领域认证测试工程师基础级

基础级资质认证针对所有从事软件测试的人员。包括测试员、测试分析师、测试工程师、测试顾问、测试经理、软件开发人员和开发团队成员等。基础级资质也适用于想要了解软件测试基础知识的各类人员，例如项目经理、质量经理、产品负责人、软件开发经理、业务分析师、IT 主管和管理顾问。持有基础级证书的人员将能够获得继续参加更高级别的软件测试认证资质。

0.3 测试人员的职业发展路径

ISTQB® 计划为处于职业生涯各个阶段的测试专业人员提供支持，提供广泛和深入的知识。获得 ISTQB® 基础级认证的个人也可能对 ISTQB® 核心高级（测试分析师、技术测试分析师和测试经理）以及之后的专家级（测试管理或测试过程改进）感兴趣。希望在敏捷环境中培养测试实践技能各类人员都可以考虑获得敏捷技术测试员或大规模敏捷测试领导力认证。专业模块深入研究了特定的测试方法和测试活动领域（例如，测试自动化、人工智能测试、基于模型的测试、移动应用测试），还与特定测试领域相关（例如，性能测试、易用性测试、验收测试、安全性测试等），或为某些行业领域（例如，汽车或游戏行业）的集群测试技术。请访问 www.istqb.org 获取 ISTQB® 认证测试工程师计划的最新信息。

0.4 商业价值

本节列出了获得基础级认证的测试人员所预期的 14 项商业价值。

基础级认证的测试人员能够：

- FL-B01 理解什么是测试以及为什么测试是有益的。
- FL-B02 理解软件测试的基本概念。
- FL-B03 根据测试环境确定要实施的测试方法和活动。
- FL-B04 评估和改进文档的质量。
- FL-B05 提高测试的有效性和效率
- FL-B06 将测试过程与软件开发生存周期保持一致。
- FL-B07 理解测试管理原则。
- FL-B08 编写和传达清晰易懂的缺陷报告。
- FL-B09 理解影响测试优先级和工作量的因素。
- FL-B010 将测试作为跨职能团队的一部分工作。
- FL-B011 了解与测试自动化相关的风险和收益。
- FL-B012 确定测试所需的基本技能。
- FL-B013 理解风险对测试的影响。
- FL-B014 有效报告测试进度和质量。

0.5 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，并用于生成认证测试员基础级考试题。

一般来说，本教学大纲的 1 至 6 章的所有内容均可在 K1 级别上考核。也就是说，考生可能会被要求识别（Recognize）、牢记（Remember）或回顾（Recall）起这六章中提到的关键词或概念。具体的学习目标级别会显示在每章的开头部分，并按照以下分类：

- K1 牢记（Remember）
- K2 理解（Understand）
- K3 应用（Apply）

附录 A 中给出了学习目标的详细内容和示例。即使学习目标中没有明确提及，所有列在章节标题下方作为关键词的术语也都应牢记（K1 级别）。

0.6 基础级认证考试

基础级认证考试将基于本教学大纲。考试题的答案可能需要使用本教学大纲中多个部分的内容。除大纲简介和附录外，大纲的所有部分都是可以考核的。标准和书籍作为参考资料列在第 7 章，但除了教学大纲本身所摘要的标准和书籍内容外，其他内容不会被考核。请参阅基础级考试结构和规则。

0.7 课程认证

ISTQB® 成员国委员会（ISTQB® 在中国唯一的成员国委员会：CSTQB®）可以对培训机构的课程材料是否遵循本教学大纲进行认证。培训机构应从委员会（在中国为 CSTQB®）或执行认证的机构获取认证指南。经认证的课程被认为符合本教学大纲，并且可以将 ISTQB® 考试作为课程的一部分。本教学大纲的认证指南遵循过程管理和合规工作组发布的通用认证指南。

0.8 标准的处理

基础级教学大纲中引用了一些标准（例如 IEEE 或 ISO 标准）。这些参考文献提供了框架（例如有关质量特性的 ISO 25010 参考文献），或者提供了附加信息来源（如果读者需要）。标准文件不作为本大纲的考核内容。有关标准的更多信息，请参阅第 7 章。

0.9 保持时效性

软件行业变化迅速，为了应对这些变化并让利益相关方能够获取最新的相关信息，ISTQB® 工作组在 www.istqb.org 网站上创建了链接，其中有些链接涉及支持文档和标准的变化。这些信息不作为基础级教学大纲考核内容。

0.10 详细级别

本教学大纲的详细程度为有助于国际课程和考试保持一致。为了达到这一目标，本大纲由下面几个部分组成：

- 描述基础级意图的一般教学目的
- 列出了学员必须能回顾的术语（关键词）列表
- 每个知识领域的学习目标，描述需要达到的认知学习成果
- 对重要概念的描述，包括来源参考，例如：已认可的文献或标准。

教学大纲内容并不是对软件测试的整个知识领域的描述，只是提供了基础级培训课程所应涵盖的详细程度。它侧重于所有软件项目都能应用到的测试概念和技术，而这些软件项目独立于所使用的软件开发生存周期（SDLC）。

0.11 大纲的结构

本大纲共有六章考核内容。每章的一级标题指定该章的培训时长。每章的各节标题不再提供时长安排。对于已认证的培训课程，教学大纲要求至少 1135 分钟（18 小时 55 分钟）的教学时长，分布在以下六个章节中：

- 第 1 章：软件测试基础（180 分钟）
 - 学员学习与测试相关的原则、了解为什么需要测试以及测试的目的是什么。
 - 学员理解测试过程、主要的测试活动和测试件。
 - 学员理解测试所需的基本技能。
- 第 2 章：贯穿软件开发生存周期中的测试（130 分钟）
 - 学员学习如何将测试融入不同的开发方法中
 - 学员学习“测试先行”方法以及 DevOps 的概念
 - 学员学习不同的测试级别、测试类型和维护测试
- 第 3 章：静态测试（80 分钟）
 - 学员学习静态测试基础知识、反馈和评审过程
- 第 4 章：测试分析和设计（390 分钟）
 - 学员学习如何应用黑盒、白盒和基于经验的测试技术，从各软件工作产品中导出测试用例
 - 学员学习基于协作的测试方法
- 第 5 章：管理测试活动（335 分钟）
 - 学员学习如何进行测试总体计划，以及如何估算测试工作量。
 - 学员理解风险如何影响测试范围。
 - 学员学习如何监测和控制测试活动。
 - 学员了解配置管理如何支持测试。
 - 学员学习如何以清晰易懂的方式报告缺陷。
- 第 6 章：测试工具（20 分钟）
 - 学员学习如何对工具进行分类，并了解测试自动化的风险和益处。

1. 测试基础 - 180 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), 测试完成 (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), 测试监测 (test monitoring), 测试对象 (test object), 测试目的 (test objective), 测试规划 (test planning), 测试规程 (test procedure), 测试结果 (test result), 测试 (testing), 测试件 (testware), 可追溯性 (traceability), 确认 (validation), 验证 (verification)

第一章的学习目标:

1.1 什么是测试?

- FL-1.1.1 (K1) 识别典型的测试目的。
- 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) 举例说明为什么需要测试。
- FL-1.2.2 (K1) 回顾测试和质量保证之间的关系。
- FL-1.2.3 (K2) 区分根本原因、错误、缺陷和失效。

1.3 测试原则

- FL-1.3.1 (K2) 解释测试的七项原则。

1.4 测试活动、测试件和测试角色

- FL-1.4.1 (K2) 总结不同的测试活动和任务。
- FL-1.4.2 (K2) 解释上下文对测试过程的影响。
- FL-1.4.3 (K2) 区分支持测试活动的测试件。
- FL-1.4.4 (K2) 解释维护可追溯性的价值。
- FL-1.4.5 (K2) 比较测试中的不同角色。

1.5 测试的基本技能和良好实践

- FL-1.5.1 (K2) 举例说明测试所需的通用技能。
- FL-1.5.2 (K1) 回顾“完整团队”方法的优点。
- FL-1.5.3 (K2) 区分测试独立性的优点和缺点。

1.1 什么是测试？

软件系统是我们日常生活中不可或缺的一部分。大多数人都曾遇到过软件无法按预期运行的经历。无法正常工作的软件可能会导致许多问题，包括金钱、时间或商业声誉的损失；在极端情况下甚至会造成人员伤亡。软件测试评估软件质量，有助于降低软件运行失败的风险。

软件测试是一系列发现缺陷和评估软件产品质量的活动。这些软件产品在被测试时称为测试对象。关于测试的常见误解是认为软件测试仅包含执行测试（即运行软件并检查测试结果）。其实，软件测试还包括其他活动，并且必须与软件开发生存周期保持一致（参阅第 2 章）。

关于测试的另一个常见误解，认为测试完全专注于验证测试对象。虽然测试涉及验证，即检查系统是否满足指定要求，但它还涉及确认，即检查系统在其操作环境中是否满足用户和其他利益相关方的要求。

测试可以是动态的或静态的。动态测试涉及软件的执行，而静态测试不涉及软件的执行。静态测试包括评审（参阅第 3 章）和静态分析。动态测试使用不同类型的测试技术和测试方法来生成测试用例（参阅第 4 章）。

测试不仅仅是技术活动。它还需要进行适当的计划、管理、估算、监测和控制（参阅第 5 章）。

测试人员使用工具（参阅第 6 章），但重要的是要记住，测试很大程度上是智力活动，要求测试人员具备专业知识、使用分析技能并运用批判性思维和系统思维（Myers, 2011 年；Roman, 2018 年）。

ISO/IEC/IEEE 29119-1 (GB/T 38634.1-2020) 标准提供了有关软件测试概念的更多信息。

1.1.1 测试目的

典型的测试目的为：

- 评估工作产品，例如需求、用户故事、设计和代码
- 触发失效并发现缺陷
- 确保所需的被测对象的覆盖率
- 降低软件质量不足的风险级别
- 验证是否已满足指定需求
- 验证测试对象是否符合合同、法律和监管要求
- 向利益相关方提供信息，使他们能够做出明智的决策
- 建立对被测对象质量的信心

- 确认被测对象是否完整并按利益相关方的预期工作

根据不同的情况，测试的目的可能会有所不同；其中包括被测试的工作产品、测试级别、风险、遵循的软件开发生存周期(SDLC)以及与业务环境相关的因素，例如公司结构、竞争考虑因素，或上市时间。

1.1.2 测试与调试

测试和调试是不同的活动。测试可以触发由软件缺陷引发的失效（通过动态测试），也可以直接发现测试对象中的缺陷（通过静态测试）。

当动态测试（参阅第 4 章）触发失效时，调试涉及查找失效原因（缺陷）、分析并消除这些原因。这种情况下，典型调试过程包括：

- 重现失效
- 诊断（找到根本原因）
- 修复失效的原因

随后的确认测试将检查修复是否解决了问题。确认测试最好是由执行初始测试的同一个人来完成。还可以执行后续的回归测试，以检查修复是否导致测试对象的其他部分出现失效（有关确认测试和回归测试的更多信息，请参阅第 2.2.3 节）。

静态测试是为了发现缺陷，而调试则是为了消除缺陷。因为静态测试是直接发现缺陷，并且不会导致失效，所以不需要重现或诊断（参阅第 3 章）。

1.2 为什么需要测试？

测试作为一种质量控制形式，有助于在设定的范围、时间、质量和预算限制内实现商定的目标。测试对成功的贡献不应仅限于测试团队的活动。任何利益相关方都可以利用他们的测试技能使项目更接近成功。测试组件、系统和相关文档有助于识别软件中的缺陷。

1.2.1 测试对成功的贡献

测试提供了经济高效的发现缺陷的方法。这些缺陷可以通过后续活动进行消除（通过调试——一种非测试活动），因此测试间接地提高测试对象的质量。

测试提供了在软件开发生存周期（SDLC）的各个阶段直接评估测试对象质量的方法。这些措施是大型项目管理活动的一部分，有助于做出进入软件开发生存周期（SDLC）下一阶段的决策，例如决定版本是否发布。

测试为用户提供了开发项目的间接代表，测试人员确保他们对用户需求的理解贯穿于整个开发生存周期。另一种方法是让用户代表参与开发项目，但由于成本高且缺乏合适的用户，通常是不好操作。

为了满足合同或法律要求，或遵守监管标准，也可能需要进行测试。

1.2.2 测试和质量保证（QA）

虽然人们经常混用术语“测试”和“质量保证”（QA），但测试和 QA 并不相同。测试是质量控制（QC）的一种形式。

质量控制（QC）是以产品为导向的纠正方法，侧重于支持实现适当质量级别的活动。测试是质量控制的主要形式，而其他方法包括形式化方法（模型检查和正确性证明）、模拟和原型设计。

质量保证（QA）是以过程为导向的预防性方法，侧重于过程的实施和改进。它的工作原理是，如果正确遵循良好的过程，就会产生良好的产品。质量保证（QA）适用于开发和测试过程，是项目中每个人的责任。

测试结果会被质量保证（QA）和质量控制（QC）使用。在质量控制（QC）中，测试结果用于修复缺陷，而在质量保证（QA）中，测试结果提供关于开发和测试过程执行情况的反馈。

1.2.3 错误、缺陷、失效和根本原因

人会犯错误（error、mistake），从而产生缺陷（defect、fault、bug），进而导致失效（failure）。人犯错的原因多种多样，例如时间压力、工作产品的复杂性、过程、基础设施或交互，或者仅仅是因为疲劳或缺乏足够的培训。

缺陷可以在文档（如需求规格说明或测试脚本）中找到、在源代码中或者在支持工件中（例如构建文件）找到。软件开发生存周期（SDLC）早期生成的工件中的缺陷，如果未被发现，通常会导致生存周期后期产生带有缺陷的工件。如果代码中的缺陷被执行，系统可能无法按照应有的方式执行，或者执行了不应该执行的操作，从而导致失效。某些缺陷在执行时总是会导致失效，而另一些缺陷只会在特定情况下导致失效，有些缺陷可能永远不会导致失效。

错误和缺陷并不是导致失效的唯一原因。失效也可能是由环境因素引起的，例如辐射或电磁场引起固件中的缺陷。

根本原因是问题发生的根本原因（例如，导致错误的情况）。通过根本原因分析来识别根本原因，通常在发生失效或识别出缺陷时进行根本原因分析。一般认为，通过解决根本原因，例如消除根本原因，可以防止类似失效或缺陷，或降低其出现的频率。

1.3 测试原则

多年来，人们已经提出了许多测试原则，提供了适用于所有测试的通用指南。本教学大纲描述了七个测试原则。

1. 测试显示了缺陷的存在，而不能说明缺陷不存在。测试可以表明测试对象中存在缺陷，但不能证明没有缺陷（Buxton 1970）。测试降低了测试对象中未发现缺陷的可能性，但即使没有发现缺陷，测试也无法证明测试对象的正确性。

2. 穷尽测试是不可能的。除非在小型的案例中，否则测试所有的情况是不可行的（Manna, 1978 年）。与其试图进行穷尽测试，不如使用测试技术（参阅第 4 章）、测试用例优先级（参阅第 5.1.5 节）和基于风险的测试（参阅第 5.2 节）来聚焦测试工作。

3. 早期测试可以节省时间和费用。在软件开发过程的早期移除的缺陷不会在后续的衍生工作产品中产生更多缺陷。由于在软件开发生存周期（SDLC）后期发生的故障较少，因此，可以降低质量成本（Boehm, 1981 年）。为了早期发现缺陷，应尽早开始静态测试（参阅第 3 章）和动态测试（参阅第 4 章）。

4. 缺陷的集群效应。通常大多数已发现的缺陷出现在少数系统组件中，或者少数系统组件是引起大多数操作失效的原因（Enders, 1975 年）。这一现象是帕累托原理的一个例证。预测的缺陷集群和在测试或运行过程中观察到的实际缺陷集群是基于风险的测试的重要输入（参阅第 5.2 节）。

5. 测试会无效。如果重复多次相同的测试，在检测新缺陷方面会变得越来越无效（Beizer, 1990 年）。为了克服这种影响，可能需要修改现有的测试和测试数据，并且可能需要编写新的测试。然而，在某些情况下，重复相同的测试可能会产生有益的结果，例如，在自动化回归测试中（参阅第 2.2.3 节）。

6. 测试活动依赖于测试周境。没有一种放之四海而皆准的测试方法。周境不同，测试的方式也不同（Kaner, 2011 年）。

7. 不存在缺陷的谬论。期望软件验证会确保系统的成功是一种谬论（即误解）。彻底测试所有指定的需求并修复发现的所有缺陷，仍然可能产生不能满足用户需求和期望的系统，不能帮助实现客户的业务目标，并且与其他竞争系统相比处于劣势。除验证外，还应进行确认（Boehm, 1981 年）。

1.4 测试活动、测试件和测试角色

测试依赖于周境，但在高级别上，会有一些共同的测试活动集，如果没有这些活动，测试目标将很难达成。这些测试活动集形成测试过程。测试过程可根据特定场景进行裁剪。哪些测试活动应包含在测试过程中，如何实施，何时开始，通常会依据具体情况，体现在测试规划中（见第 5.1 节）。

以下各节将从测试活动、任务、周境的影响、测试件、测试件与测试依据之间的双向可追溯性和测试角色等方面对测试过程进行一般性描述。

有关测试过程的更多信息可参阅 ISO/IEC/IEEE 29119-2（或 GB/T 38634.2-2020）标准。

1.4.1 测试活动和任务

测试过程通常由以下主要活动组成。尽管这些活动看似遵循逻辑顺序，但通常采用迭代或者并行方式实施。这些测试活动通常需要针对系统和项目进行裁剪。

测试规划包括定义测试目的，在整体环境的约束下选择可达到目的的最佳方法。测试规划将在第 5.1 节中进一步说明。

测试监测和控制。测试监测包括持续检查所有测试活动，并且将实际进度与计划进行比较。测试控制包括采取必要的行动来实现测试目的。第 5.3 节对测试监测和控制进一步说明。

测试分析包括分析测试依据以识别可测试的特征，定义相关测试条件的优先级，以及有关的风险和风险级别（参阅第 5.2 节）。对测试依据和测试对象进行评估，以识别它们可能包含的缺陷并评估其可测试性。测试分析通常由掌握测试技术的人员提供支持（参阅第 4 章）。测试分析根据可度量的覆盖准则回答“测试什么？”的问题。

测试设计包含如何将测试条件转化成测试用例和其他测试件（例如，测试章程）。这项活动通常与识别覆盖项有关，可以作为具体选择哪些测试用例输入的指南。测试技术（参阅第 4 章）可用于支持此活动。测试设计还包括定义测试数据需求、设计测试环境以及确认所需的基础设施和工具。测试设计回答“如何测试？”的问题。

测试实施包括创建或获取测试执行所需的测试件（例如，测试数据）。测试用例可以被组织到测试规程中，并且经常被组装成测试套件使用。创建人工和自动化的测试脚本。为实现高效的测试执行，测试规程要按照优先级在测试执行进度表中排序。构建测试环境，并验证其设置的正确性。

测试执行包括根据测试执行进度表执行测试（测试运行）。测试执行既可以人工进行也可以自动进行。测试执行可以采取多种形式，包括持续测试或结对测试会话。将测试的实际结果与预期结果进行比

较。记录测试结果。分析导致异常发生的可能原因。该分析以观察到的失效为依据报告异常（参阅 5.5 节）。

测试完成活动通常发生在项目里程碑处（例如，发布、迭代结束、测试级别完成），针对任何未解决的缺陷、变更请求或产品待办事项列表。未来可能有用的测试件都会被识别并归档，或移交给适当的团队。测试环境被恢复到约定状态。对测试活动进行分析，以确定未来迭代、发布或项目的经验教训和改进（见 2.1.6 节）。创建测试完成报告并与利益相关方沟通。

1.4.2 周境中的测试过程

测试不是孤立进行的。测试活动是组织内执行的开发过程的组成部分。测试由利益相关方资助，其最终目标是协助利益相关方达成其业务需要。因此，测试的执行方式将取决于多种环境因素，包括：

- 利益相关方（需要、期望、需求、合作意愿等）。
- 团队成员（技能、知识、经验水平、工作效率、培训要求等）。
- 业务领域（测试对象的重要性、已识别的风险、市场要求、特定的法律法规等）。
- 技术因素（软件类型、产品架构、使用的技术等）。
- 项目约束（范围、时间、预算、资源等）。
- 组织因素（组织架构、现有政策、已在应用的实践等）。
- 软件开发生存周期（工程实践、开发方法等）。
- 工具（可用性、易用性、依从性等）。

这些因素将对测试相关的各种问题产生影响，包括：测试策略、使用的测试技术、测试自动化程度、需要的覆盖级别、测试文档的详细程度、报告等。

1.4.3 测试件

测试件是根据第 1.4.1 节中描述的测试活动所创建的输出工作产品。不同组织在生成、结构、命名、组织和管理工作产品方面存在显著差异。适当的配置管理（参阅第 5.4 节）可确保工作产品的一致性和完整性。以下是工作产品列表的部分内容：

- **测试规划工作产品**包括：测试计划、测试进度表、风险记录表以及入口和出口准则（参阅第 5.1 节）。风险记录表是包含风险、风险可能性、风险影响以及风险缓解信息的列表（参阅第 5.2 节）。测试进度表、风险记录表、入口和出口准则通常是测试计划的一部分。
- **测试监测和控制工作产品**包括：测试过程报告（参阅第 5.3.2 节）、文档化的控制指令（参阅第 5.3 节）和风险信息（参阅第 5.2 节）。

- **测试分析工作产品**包括：（按优先级排序）测试条件（例如，验收准则，参阅第 4.5.2 节），在测试依据中发现的有关缺陷的缺陷报告（如果该缺陷还没有被直接修复）。
- **测试设计工作产品**包括：（按优先级排序）测试用例、测试章程、覆盖项、测试数据需求和测试环境需求。
- **测试实施工作产品**包括：测试规程、自动化测试脚本、测试套件、测试数据、测试执行计划和测试环境要素。测试环境要素的实例包括：桩、驱动器、模拟器和服务虚拟化。
- **测试执行工作产品**包括：测试日志和缺陷报告（参阅第 5.5 节）
- **测试完成工作产品**包括：测试完成报告（参阅第 5.3.2 节）、后续项目或迭代的改进行动项、文档化的经验教训和变更请求（例如，产品待办事项）。

1.4.4 测试依据和测试件之间的可追溯性

为了实施有效的测试监测和控制，测试过程中建立和维护测试依据元素、与元素相关的测试件（例如测试条件、风险、测试用例）、测试结果和发现的缺陷之间的可追溯性非常重要。

准确的可追溯性可支持覆盖评估，测试依据中定义可衡量的覆盖准则非常有用。覆盖准则可以作为关键绩效指标，推动活动的进行，以展示测试目的达成的程度（参阅第 1.1.1 节）。例如：

- 测试用例对需求的可追溯性可以验证测试用例是否覆盖了需求。
- 测试结果对风险的可追溯性可用于评估测试对象中剩余风险的级别。

除了评估覆盖范围之外，良好的可追溯性还可以确定变更的影响、促进测试审计，并有助于满足 IT 治理准则。良好的可追溯性还可以通过包含测试依据元素的状态，使得测试进度报告和测试完成报告更易于理解。也有助于以可理解的方式向利益相关方传达测试技术方面的问题。可追溯性提供了针对业务目标评估产品质量、过程能力和项目进展的信息。

1.4.5 测试活动中的角色

在本大纲中，涵盖了测试活动中的两个主要角色：测试管理角色和测试角色。这两个角色所承担的活动和任务取决于项目和产品的周境、人员的技能以及组织等因素。

测试管理角色全面负责测试过程、测试团队以及测试活动的领导工作。测试管理角色主要关注测试规划、测试监测和控制以及测试完成活动。测试管理角色开展工作的方式因周境而异。例如，在敏捷软件开发中，一些测试管理任务可由敏捷团队处理。对于跨多个团队或整个组织的任务可由开发团队之外的测试经理执行。

测试角色对测试的工程（技术）方面负有整体责任。测试角色主要关注测试分析、测试设计、测试实施和测试执行等活动。

不同的人员可能在不同的时间点扮演这些角色。例如，测试管理角色可以由团队领导、测试经理、开发经理等担任。也可以一个人同时承担测试和测试管理的角色。

1.5 测试中的基本技能和良好实践

技能是指个人根据自身的知识、实践和才能做好某件事的能力。好的测试人员应该具备一些基本技能才能把他们的工作做好。好的测试人员应该是有效的团队合作者，并且能够在不同测试独立性级别上开展测试工作。

1.5.1 测试所需的通用技能

以下虽然是些通用的技能，但它们对测试人员来说尤其有意义：

- 测试知识（例如，通过使用测试技术提高测试的有效性）。
- 全面、细致、好奇心、注重细节、有条不紊（对识别缺陷，尤其对于发现难以发现的缺陷）。
- 良好的沟通技巧、积极的倾听、具有团队合作精神（与所有利益相关方有效互动，以可理解的方式向他人传递信息，并报告和讨论缺陷）。
- 分析性思维、批判性思维、创造力（用以提高测试的有效性）。
- 技术知识（提高测试效率，例如使用适当的测试工具）。
- 领域知识（能够理解并与最终用户/业务代表沟通）。

测试人员经常是坏消息的传递者。责怪坏消息的传递者是常见的人类特质。因此，沟通技巧对于测试人员来说至关重要。传达测试结果可能被视为对产品及其作者的批评。确认偏见（Confirmation bias）可能会使人们难以接受与当前信念不符的信息。尽管测试对项目的成功和产品质量有很大贡献，但有些人可能会将测试视为破坏性的活动。为了改善这种观点，应以建设性的方式沟通有关缺陷和失效的信息。

1.5.2 完整团队方法

能够在团队环境中有效地工作并积极为团队目标做出贡献，是测试人员重要的技能。完整团队方法（来自“极限编程”（参阅第 2.1 节）的实践）基于这项技能构建。

在完整团队方法中，所有团队成员都具有进行各种类型任务的知识能力，并且每个人都要对质量负责。团队成员共享相同的工作空间（物理或虚拟），共址工作有助于沟通和互动。完整团队方法可提升团队活力，增强团队内部的沟通和协作，并通过充分利用团队内不同的技能，产生协同效应，从而使项目获益。

为确保达到期望的质量水平，测试人员要与团队其他成员密切合作。这些合作包括：与业务代表协作，帮助他们创建适合的验收测试；与开发人员协作，达成测试策略和测试自动化方法的共识。测试人员能够向团队其他成员传授测试知识，从而影响产品的开发。

由于对周境的依赖，完整团队方法可能并不总是适用。例如，在某些状况下，如安全关键系统，可能需要更高水平的测试独立性。

1.5.3 测试独立性

因为作者和测试人员之间存在认知偏差（cognitive biases）（参阅 Salman, 1995），因此一定程度的独立性可以使测试人员更有效地发现缺陷。然而，独立性并不能取代熟悉性，例如，开发人员可以高效地发现自己代码中的许多缺陷。

工作产品可以交由作者进行测试（无独立性），或交由来自作者同一团队的同行进行测试（一定的独立性），或交由组织内非作者团队的测试人员进行测试（高独立性），或交由组织外部的测试人员进行测试（非常高的独立性）。对于大多数项目而言，通常最好使用多个独立性级别的测试。例如，开发人员进行组件测试和组件集成测试，测试团队进行系统测试和系统集成测试，业务代表进行验收测试。

测试独立性的主要优势在于，独立的测试人员与开发人员相比，由于背景、技术视角、以及认知偏向的差异，更可能识别出不同类型的失效和缺陷。此外，独立的测试人员可以验证、质疑或推翻利益相关方在系统规格说明和系统实施期间所做的假设。

尽管如此，测试独立性也存在一些缺点。独立的测试人员可能与开发团队脱离，可能导致缺乏协作、沟通出现问题或与开发团队形成对立关系。开发人员可能会丧失对质量的责任感。独立测试人员可能被视为瓶颈，或被指责为发布延迟负有责任。

2. 软件开发生存周期中的测试 - 130 分钟

关键词

验收测试 (acceptance testing), 黑盒测试 (black-box testing), 组件集成测试 (component integration testing), 组件测试 (component testing), 确认测试 (confirmation testing), 功能测试 (functional testing), 集成测试 (integration testing), 维护测试 (maintenance testing), 非功能测试 (non-functional testing), 回归测试 (regression testing), 左移 (shift-left), 系统集成测试 (system integration testing), 系统测试 (system testing), 测试级别 (test level), 测试对象 (test object), 测试类型 (test type), 白盒测试 (white-box testing)

第二章的学习目标:

2.1 软件开发生存周期中的测试

- FL-2.1.1 (K2) 解释所选择的软件开发生存周期对测试的影响。
- FL-2.1.2 (K1) 回顾适用于所有软件开发生存周期的良好测试实践。
- FL-2.1.3 (K1) 回顾开发中“测试先行”方法的示例。
- FL-2.1.4 (K2) 总结 DevOps 对测试产生的影响。
- FL-2.1.5 (K2) 解释左移的方法。
- FL-2.1.6 (K2) 解释如何使用回顾作为过程改进的机制。

2.2 测试级别和测试类型

- FL-2.2.1 (K2) 区分不同的测试级别。
- FL-2.2.2 (K2) 区分不同的测试类型。
- FL-2.2.3 (K2) 区分确认测试和回归测试。

2.3 维护测试

- FL-2.3.1 (K2) 总结维护测试及其触发因素。

2.1 软件开发生存周期中的测试

软件开发生存周期（SDLC）模型是对软件开发过程的抽象、概要表述。SDLC 模型定义了软件开发过程中不同开发阶段和活动类型之间的逻辑和时间关系。SDLC 模型的示例包括：顺序开发模型（例如瀑布模型、V 模型）、迭代开发模型（例如螺旋模型、原型模型）和增量开发模型（例如统一软件开发过程）。

软件开发过程中的活动也可以通过更详细的软件开发方法和敏捷实践来描述。例如：验收测试驱动开发（ATDD）、行为驱动开发（BDD）、领域驱动设计（DDD）、极限编程（XP）、特征驱动开发（FDD）、看板、精益管理、Scrum 和测试驱动开发（TDD）。

2.1.1 软件开发生存周期对测试的影响

测试必须适应软件开发生存周期才能成功。软件开发生存周期的选择会对以下几个方面产生影响：

- 测试活动的范围和时间安排（例如测试级别和测试类型）
- 测试文档的详细程度
- 测试技术和测试方法的选择
- 测试自动化程度
- 测试人员的角色和职责

在顺序开发模型中，测试人员通常在软件开发的初期阶段参与需求评审、测试分析和测试设计。可执行代码通常在后期阶段创建，因此通常无法在软件开发生存周期早期进行动态测试。

在某些迭代和增量开发模型中，每次迭代都会提供可工作的增量原型或产品。也就是说在每个迭代中，静态和动态测试都可以在所有测试级别上执行。频繁的增量交付需要快速反馈和全面回归测试。

敏捷软件开发假定在项目过程中都可能发生变化。因此，在敏捷项目中，更倾向于轻量级的工作产品文档和全面测试自动化，以便更容易进行回归测试。此外，大部分人工测试往往使用基于经验的测试技术（参阅第 4.4 节），不需要事前进行大量的测试分析和设计。

2.1.2 软件开发生存周期与良好的测试实践

无论选择哪种软件开发生存周期模型，良好的测试实践包括以下内容：

- 每个软件开发活动，都有相应的测试活动，以便对所有开发活动进行质量控制
- 不同测试级别（参阅第 2.2.1 章）具有特定且不同的测试目标，可以确保测试既全面又避免冗余

- 给定测试级别的测试分析和设计始于相应的软件开发生存周期开发阶段，以便测试能够遵循早期测试的原则（参阅第 1.3 节）
- 相关文档的初稿完成时，测试人员立即参与评审工作产品，以便早期测试和缺陷检测，从而支持测试左移方法（参阅第 2.1.5 节）

2.1.3 测试是软件开发的驱动力

测试驱动开发（TDD）、验收测试驱动开发（ATDD）和行为驱动开发（BDD）是类似的开发方法，将测试定义为指导开发的手段。这些方法都实现了早期测试的原则（参阅第 1.3 节）并遵循左移方法（参阅第 2.1.5 节），因为测试在编写代码之前定义。它们支持迭代开发模型。这些方法的具体特点如下：

测试驱动开发（TDD）：

- 通过测试用例来指导编码（而不是详尽的软件设计）（Beck 2003）。
- 先编写测试，后编写代码以满足测试，最后对测试和代码进行重构。

验收测试驱动开发（ATDD）（参阅第 4.5.3 节）：

- 作为系统设计过程的一部分，从验收准则中导出测试（Gärtner 2011）。
- 在部分应用程序开发前，编写测试，以满足测试的要求。

行为驱动开发（BDD）：

- 用简化的自然语言编写测试用例来表达应用程序的期望行为，利益相关方容易理解，通常使用 Given/When/Then 格式（Chelimsky 2010）。
- 自动将测试用例转化为可执行的测试。

对于上述所有方法，通常应用自动化测试，以确保将来改写或重构的代码质量。

2.1.4 DevOps 与测试

DevOps 是一种组织方法，旨在通过使开发（包括测试）和运维部门共同努力，实现一系列通用目标，从而实现协同效应。DevOps 要求组织内部进行文化转变，将开发和运维的职能同等看待，以弥合开发（包括测试）和运维之间的差距。DevOps 提倡团队的自主权、快速反馈、集成工具链以及持续集成（CI）和持续交付（CD）等技术实践。通过 DevOps 交付流水线，软件团队可以更快地构建、测试和发布高质量的代码（Kim 2016）。

从测试的角度来看，DevOps 的好处包括：

- 代码质量的快速反馈，并判断变更是否对现有代码产生不利影响。
- 持续集成（CI）通过鼓励开发人员提交高质量的代码，并辅以组件测试和静态分析，在测试中实现左移方法（参阅第 2.1.5 节）。
- 促进 CI/CD 自动化过程，有助于建立稳定的测试环境。
- 更加关注非功能性质量特性（例如性能、可靠性）。
- 交付流水线的自动化，减少人工重复测试的需求。
- 由于自动化回归测试的规模和范围，降低了回归风险。

然而，DevOps 也面临着某些风险和挑战，包括：

- 必须定义和建立 DevOps 交付流水线。
- 必须引入和维护 CI/CD 工具。
- 测试自动化需要额外资源，这些资源可能难以建立和维护。

尽管 DevOps 提供了高度自动化测试，但从用户的角度来说，仍然需要人工测试。

2.1.5 左移的方法

测试早期介入的原则（参阅第 1.3 节）有时被称为“左移”，这是软件开发生存周期中较早进行测试的方法。左移建议测试应该早期进行（例如，代码实现或组件集成前开始测试），但不能因此忽视软件开发生存周期的后期测试。

许多良好的实践可以说明如何实现测试“左移”，包括：

- 从测试的角度评审规格说明。对规格说明进行评审通常可以发现潜在的缺陷，例如规格说明表述模糊、不完整和不一致。
- 编码之前编写测试用例，在代码实现过程中通过测试用具（test harness）运行代码。
- 使用持续集成（CI）和持续交付（CD），提供快速反馈和自动化组件测试，可以在代码提交到代码库时运行源代码测试。
- 在动态测试之前或作为自动化过程的一部分对源代码进行静态分析。
- 在可能的情况下，从组件测试级别开始进行非功能性测试。这是左移形式之一，因为非功能性测试类型通常在系统完整且代表性的测试环境就绪后，在软件开发生存周期的后期执行。

左移方法可能会在过程早期增加培训、工作量和成本，但可以节省过程后期的工作量和成本。

对于左移，重要的是让利益相关方相信并接受此种方法。

2.1.6 回顾与过程改进

回顾会议（也称为“项目总结会议/post-project meetings”和项目回顾）作为发布的里程碑，通常在项目或迭代结束后，按需召开。回顾会议的时间和组织方式取决于所采用的特定软件开发生存周期模型。回顾会议上，参与者（不仅限于测试人员，还包括开发人员、架构师、产品负责人、业务分析师等）讨论以下内容：

- 哪些工作是成功的，应予以保留？
- 哪些工作没成功，可以改进？
- 如何整合改进并保持未来成功？

应记录结果，通常作为测试完成报告的一部分（参阅第 5.3.2 节）。回顾对于成功实施持续改进至关重要，对任何建议的改进都要进行跟踪。

测试的典型收益包括：

- 增加测试的有效性/效率（例如，实施过程改进的建议）。
- 提高测试件的质量（例如，联合评审测试过程）。
- 团队凝聚力和学习能力（例如，提出问题，列出改进点）。
- 提高测试依据的质量（例如，处理和解决需求范围和质量方面的缺陷）。
- 改善开发和测试之间的合作（例如，定期评审和优化协作）。

2.2 测试级别和测试类型

测试级别是共同组织和管理的测试活动组。每个测试级别都是测试过程的一个实例，在给定的开发阶段，从单个组件到完整系统，或在适用情况下，乃至到系统的系统（systems of systems），执行软件相关的测试过程。

测试级别与软件开发生存周期内的其他活动相关。在顺序 SDLC 模型中，测试级别通常定义为：一个级别的出口准则是下一个级别的入口准则的一部分。在一些迭代开发模型中，这可能不适用。开发活动可能跨越多个测试级别。测试级别在时间上可能重叠。

测试类型，是与某种质量特性相关的测试活动的集合，这些测试活动中的大部分可以在每个测试级别进行。

2.2.1 测试级别

在本课程大纲中，将列举描述下列五个测试级别：

- **组件测试**（也称为单元测试），侧重于对单独组件的测试。组件测试通常需要一些特殊的支持，例如需要使用测试用具或者单元测试框架。组件测试通常由开发人员在他们的开发环境中进行。
- **组件集成测试**（也称为单元集成测试），侧重于对组件之间的接口及交互进行测试。组件集成测试重度依赖于集成策略方法，例如，自底向上集成，自顶向下集成或者大爆炸集成。
- **系统测试**，关注于对整个系统或产品的总体行为和能力，通常包含覆盖“端到端业务”的功能测试以及针对非功能质量特性的测试。对于一些非功能质量特性的测试，更倾向于一个完整系统，在具有代表性的测试环境中进行测试，例如，易用性测试。使用模拟的子系统也是可能的。系统测试可以由独立测试团队执行，并且与系统规格说明有关。
- **系统集成测试**，侧重于对被测系统与其他系统以及外部服务的接口的测试。系统集成测试需要合适的测试环境，最好是与运行环境类似的测试环境。
- **验收测试**，侧重于确认和展示部署准备情况，这意味着系统满足用户的业务要求。在理想情况下，验收测试应该由潜在用户执行。验收测试的主要形式有：用户验收测试（UAT）、运行验收测试、合同验收测试以及法规验收测试、Alpha 测试和 Beta 测试。

测试级别可以通过以下（非详尽）属性列表来区分，以避免测试活动的重叠：

- 测试对象
- 测试目的
- 测试依据
- 缺陷和失效
- 方法和职责

2.2.2 测试类型

在项目中可能会应用多种测试类型。本大纲主要涉及下列四种测试类型：

功能测试是用于评估组件或系统应该执行的功能的测试。功能是测试对象应该做的事情。功能测试的主要目的是检查功能完整性、功能正确性和功能适合性。

非功能测试是用于评估组件或系统除功能特性之外的其他属性。非功能测试是测试“系统表现得多么好”。非功能测试的主要目的是检查软件的非功能质量特性。在 ISO/IEC 25010 标准中列出了不同类型的非功能质量特性：

- 性能效率
- 兼容性

- 易用性
- 可靠性
- 信息安全性
- 维护性
- 可移植性

在开发生存周期的初期进行非功能测试有时是适当的（如，作为评审、组件测试或系统测试的一部分）。很多非功能测试是从功能测试派生而出的，他们使用同样的功能测试，在执行功能时测试非功能约束是否被满足，例如，通过检查执行完成某个功能所需要的时间，或者通过检查功能是否可以被移植到新平台上。如果较晚发现非功能缺陷，将严重威胁到项目的成功。非功能测试有时需要在特定的测试环境中进行，例如，执行易用性测试可能需要易用性测试实验室。

黑盒测试（参阅第 4.2 章节），是基于规格说明并根据测试对象外部的文档生成测试的测试技术。黑盒测试的主要目的是检查系统行为是否与规格说明描述一致。

白盒测试（参阅第 4.3 章节），是基于结构并根据系统的实施或系统的内部结构（如代码、结构、工作流和数据流）生成测试的测试技术。白盒测试的主要目标是通过测试将底层结构覆盖到可接受的水平。

上述四种测试类型都可应用到所有的测试级别，尽管每个测试级别的重点有所不同。可使用不同的测试技术为所有上述所提的测试类型导出测试条件和测试用例。

2.2.3 确认测试和回归测试

变更，通常指对组件和系统做出的改进，这种改进可以是增加新特征，或通过修改代码以移除缺陷进行修复。测试还应该包括确认测试和回归测试。

确认测试用于确认原有缺陷是否已经被成功修复的测试。根据风险的不同，测试人员可以对软件的缺陷修复版本进行不同的测试，包括：

- 执行先前由于存在缺陷而失败了的所有测试用例，以及，
- 增加新的测试，以覆盖由于修复缺陷引发的任何变更。

但是，当进行缺陷修复工作的时间和预算有限时，确认测试的范围可能被严格限定，即仅执行重现原有（由缺陷引起的）失效的步骤，以检查失效是否已经消失。

回归测试确认变更未造成任何不良后果，包括已经经过确认测试的修复。这些不良后果可能会影响进行更改的同一组件、同一系统中的其他组件，甚至其他关联的系统。回归测试可能不局限于测试对象

本身，还可以与环境相关。建议首先执行影响分析以优化回归测试的范围。影响分析显示软件的哪些部分可能受到影响。

回归测试套件会被多次运行，通常回归测试用例的数量会随着每次迭代或发布而有所增加，因此可以优先考虑自动化回归测试。这些测试的自动化应该在项目的早期开始。在使用持续集成（CI）时，比如 DevOps（参阅第 2.1.4 节），最好也包括自动化回归测试。根据情况，可能包括不同级别的回归测试。

如果缺陷修复和/或变更发生在某些测试级别上，对测试对象进行的确认测试和/或回归测试就需要在所有涉及的测试级别上进行。

2.3 维护测试

有多种不同类别的维护，可以是修正错误、应对环境变更、改进性能或改善维护性（详见 ISO/IEC 14764），因此，维护可以包括计划内的发布/部署和计划外的发布/部署（热修复 hot fix）。可以在变更实施之前进行影响分析，基于系统在其他领域潜在后果的分析，帮助决定是否应该实施变更。在生产环境中测试系统的变更，既包括评估更改实施的成功与否，也包括检查系统中保持不变的部分（通常是系统的大部分）是否存在可能的回归错误。

维护测试范围通常依赖于：

- 变更引起的风险程度
- 现有系统的规模
- 变更的大小

维护以及维护测试的触发因素可以有以下几类：

- 修改，如计划中的改进（如，基于发布版本），修正错误产生的变更，或者热修复。
- 运行环境的升级或者迁移，如从一个平台迁移至另一个平台，可能需要进行与新运行环境有关的测试，也可能需要进行与软件变更有关的测试；或者当数据从一个应用迁移至处于维护状态的另一个系统时，需要对数据迁移进行的测试。
- 退役，例如应用程序的生存周期即将结束。当系统退役时，如果需要长时间的数据保留，可能需要测试数据归档。如果在归档期间需要某些数据，则可能还需要测试归档后的数据恢复和检索过程。

3. 静态测试 -- 80 分钟

关键词

异常 (anomaly), 动态测试 (dynamic testing), 正式评审 (formal review), 非正式评审 (informal review), 审查 (inspection), 评审 (review), 静态分析 (static analysis), 静态测试 (static testing), 技术评审 (technical review), 走查 (walkthrough)

第 3 章的学习目标:

3.1 静态测试基础

- | | | |
|----------|------|-----------------------|
| FL-3.1.1 | (K1) | 认识可以通过不同静态测试技术检查的产品类型 |
| FL-3.1.2 | (K2) | 解释静态测试的价值 |
| FL-3.1.3 | (K2) | 比较静态测试与动态测试 |

3.2 反馈和评审过程

- | | | |
|----------|------|----------------------|
| FL-3.2.1 | (K1) | 识别与利益相关方早期反馈和频繁反馈的好处 |
| FL-3.2.2 | (K2) | 总结评审过程的活动 |
| FL-3.2.3 | (K1) | 回顾执行评审时主要角色承担的责任 |
| FL-3.2.4 | (K2) | 比较不同的评审类型 |
| FL-3.2.5 | (K1) | 回顾成功评审的因素 |

3.1 静态测试基础

与动态测试相比，静态测试不需要执行被测软件。通过人工检查（例如评审）或借助工具（例如静态分析），对代码、过程规格说明、系统架构规格说明或其他工作产品进行评估。测试目的包括提高质量、检测缺陷以及评估可读性、完整性、正确性、可测试性和一致性等特性。静态测试可用于验证和确认。

测试人员、业务代表和开发人员在实例映射（example mapping）、协作用户故事编写和待办列表（backlog）细化会议期间紧密合作，以确保用户故事和相关工作产品满足定义的准则，例如“就绪的定义”（参阅第 5.1.3 节）。可以应用评审技术来确保用户故事完整且易于理解，并包含可测试的验收准则。通过提出恰当的问题，测试人员能够深入探索、质疑并协助改进所提出的用户故事。

静态分析可以在动态测试之前识别问题，通常所需工作量相对较小，因为静态测试不需要测试用例，并且通常使用工具（参阅第 6 章）。静态分析通常被整合到持续集成框架中（参阅第 2.1.4 节）。虽然静态分析主要用于检测特定的代码缺陷，但也可用于评估维护性和信息安全性。拼写检查工具和可读性工具是静态分析工具的实例。

3.1.1 静态测试可检查的工作产品

几乎任何工作产品都可使用静态测试进行检查。例如，需求规格说明文档、源代码、测试计划、测试用例、产品代办项、测试章程、项目文档、合同和模型。

可以阅读和理解的任何工作产品都可以成为评审的对象。然而，对于静态分析来说，工作产品需要结构化，以便对其检查（例如，具有形式化语法的模型、代码或文本）。

不适合进行静态测试的工作产品包括难以人为解释的工作产品，以及不应该通过工具进行分析的工作产品（例如，由于法律原因，不应该对第三方可执行代码进行分析）。

3.1.2 静态测试的价值

静态测试可以在软件开发生存周期的早期阶段检测缺陷，从而实现早期测试的原则（参阅第 1.3 节）。静态测试还可以识别动态测试无法检测到的缺陷（例如，无法访问的代码、没有按预定实现的设计模式、不可执行的工作产品缺陷）。

静态测试提供了评估工作产品的质量和建立对工作产品信心的能力。通过验证文档化的需求，利益相关方还可以确保这些需求真正描述了他们的实际需要。由于静态测试可以在软件开发生存周期的早期

进行，因此可以在利益相关方之间建立共识。利益相关方之间的沟通也将得到改善。因此，建议更广泛的利益相关方参与静态测试。

尽管实施评审的成本可能很高，但项目总体成本通常比不进行评审低得多，因为实施评审的项目后期修复缺陷所需的时间和工作量较少。

与动态测试相比，静态分析可以更有效地检测代码缺陷，从而减少代码缺陷的数量，并且降低开发的总体工作量。

3.1.3 静态测试和动态测试的差异

静态测试和动态测试相辅相成。二者具有相似的目的，例如支持工作产品中缺陷的检测（参阅第 1.1.1 节），但也存在一些差异，例如：

- 静态测试和动态测试（通过失效分析）都可以检测错误，但是有些错误类型只能通过静态测试或动态测试来发现。
- 静态测试直接发现缺陷，而动态测试引发失效，并通过随后的分析确定相关缺陷。
- 静态测试可以更容易检测出在代码路径上的缺陷，这些缺陷在动态测试中很少被执行，或很难到达。
- 静态测试可用于不可执行的工作产品，而动态测试只能用于可执行的工作产品。
- 静态测试可用于测量不依赖于执行代码的质量特性（例如，维护性），而动态测试可用于测量依赖于执行代码的质量特性（例如，性能效率）。

通过静态测试更容易和/或更经济地发现的典型缺陷包括：

- 需求缺陷（例如，不一致、含糊不清、矛盾、遗漏、不准确、重复）。
- 设计缺陷（例如，低效的数据库结构、模块化程度低）。
- 特定类型的代码缺陷（例如，未定义值的变量、未声明的变量、无法访问或重复的代码、过度复杂的代码）。
- 与标准的偏差（例如，未遵守编码规范中的命名约定）。
- 错误的接口规格说明（例如，参数的数量、类型或顺序不匹配）。
- 特定类型的安全性漏洞（例如，缓冲区溢出）。
- 测试依据覆盖的差距或不准确（例如，验收准则的漏测）。

3.2 反馈和评审过程

3.2.1 利益相关方早期和频繁反馈的好处

早期和频繁反馈有助于早期沟通潜在的质量问题。如果在软件开发生存周期中，利益相关方很少参与，开发的产品可能不满足利益相关方的期望。如果不能交付利益相关方期望的结果，可能导致代价高昂的返工、错过期限、互相推诿，甚至导致整个项目失败。

在整个软件开发生存周期中，利益相关方的频繁反馈可以防止误解需求，确保早期理解和实施需求变更。有助于开发团队更好地理解构建的软件，专注于利益相关方的价值最大化，积极应对识别的风险。

3.2.2 评审过程的活动

ISO/IEC 20246 标准定义了通用的评审过程，该过程提供了结构化但灵活的框架，在这个框架内，特定的评审过程可以根据特定情形进行裁剪。如果需要更正式的评审，则需为不同活动设定多种任务。

许多工作产品的规模太大，无法通过一次评审全部覆盖。可以安排多次评审过程，以完成对整个工作产品的评审。

评审过程的活动包括：

- **规划。**在规划阶段应定义评审范围，包括目的、评审的工作产品、评估的质量特性、关注的重点领域、出口准则、支持信息（例如，标准、工作量和时间表）。
- **评审启动。**评审启动阶段的目标是确保所有参与者和相关事项都做好了开始评审的准备。确保各参与者都可以获取待评审的工作产品，理解各自的角色和职责，并获得执行评审所需的全部内容。
- **独立评审。**各评审员执行独立评审，以评估工作产品的质量，通过应用一种或多种评审技术（例如，基于检查表的评审、基于场景的评审）识别异常、提供建议、提出问题。ISO/IEC 20246 标准对不同的评审技术提供了更深入的研究。评审员记录所有已识别的异常、建议和问题。
- **交流和分析。**由于评审中识别的异常不一定是缺陷，所有异常都需要分析和讨论。每个异常，应该根据状态、所有权(ownership)和所需行动进行判定。通常通过评审会议完成，在评审会议期间，参与者还确定所评审的工作产品质量级别，以及需要采取的后续行动。可能需要后续的评审来完成行动。

- **修复和报告。**每个缺陷都应该创建一份缺陷报告，以便可以跟踪纠正措施。满足出口准则后，可以验收工作产品，报告评审结果。

3.2.3 评审的角色和职责

评审涉及多种利益相关方，他们可能担任多种角色。主要角色及职责包括：

- **经理：**决定评审内容，并提供资源，例如，评审所需的人员和时间。
- **作者：**创建和修复被评审的工作产品。
- **主持人（也称促进者）：**确保评审会议的有效进行，包括协调不同观点、进行时间管理，创造安全的评审环境，让每个人都能自由发言。
- **书记（也称为记录员）：**整理评审员发现的异常，记录评审信息，例如，评审会议上做出的决策和发现的新异常。
- **评审员：**执行评审。评审员可以是项目工作人员、专题相关专家或其他利益相关方。
- **评审组长：**全面负责评审，例如，决定评审人员、评审时间和评审地点。

更详细的其他可能角色，可参阅 ISO/IEC 20246 标准所述。

3.2.4 评审类型

评审有多种不同的类型，从非正式评审到正式评审。决定所需评审正式程度的因素包括：遵循的软件开发生存周期、开发过程的成熟度、评审的工作产品重要性和复杂性、法律或法规要求，以及审计跟踪的需要。相同的工作产品可以使用不同的评审类型进行评审，例如，先非正式评审，然后是更正式的评审。

选择正确的评审类型是实现评审目的的关键（参阅第 3.2.5 节）。评审类型的选择不仅基于评审目的，还应考虑项目要求、可用资源、工作产品类型和风险、业务领域和公司文化等因素。

常用的评审类型包括：

- **非正式评审** - 非正式评审不遵循规定的过程，也不需要正式的文档输出。主要目的是检测异常。
- **走查** - 走查由作者主持，可以达到多项目的，例如评估工作产品的质量、建立对工作产品的信心、培训评审人员、获得共识、产生新想法、激励和促进作者改进和检测异常。评审员可能在走查前进行独立评审，但这不是必需的。

- **技术评审** - 技术评审由具有技术资质的评审员执行，并由主持人主持。技术评审的目的是就技术问题达成共识并做出决策，但也要检测异常、评估工作产品的质量、建立对工作产品的信心、产生新想法、激励和促进作者改进。
- **审查** - 由于审查是最正式的评审类型，因此需要遵循完整的通用过程（参阅第 3.2.2 节）。审查的主要目的是尽可能多的发现异常。其次是评估质量、建立对工作产品的信心，并激励和促使作者改进。收集和使用度量指标以改进软件开发生存周期，包括审查过程。在审查中，作者不能担任评审组长或记录员的角色。

3.2.5 评审的成功因素

评审的成功取决于多种因素，包括：

- 定义明确的目的和可度量的出口准则。对参与者的评价永远不应该成为评审目的。
- 选择适当的评审类型，以实现给定的目的，并适应工作产品的类型、评审参与者、项目要求和周境。
- 分段小块评审，以确保评审人员集中精力进行独立评审和/或参加评审会议（如有）。
- 向利益相关方和作者提供评审反馈，以便他们能够改进产品及其活动（参阅第 3.2.1 节）。
- 为评审参与者提供充足的评审准备时间。
- 管理层支持评审过程。
- 使评审成为组织文化的一部分，以促进学习和过程改进。
- 为所有参与者提供充分的培训，使他们知道如何履行自己的职责。
- 为评审会议提供便利。

4. 测试分析和设计 - 390 分钟

关键词

验收准则 (acceptance criteria), 验收测试驱动开发 (acceptance test-driven development), 黑盒测试技术 (black-box test technique), 边界值分析 (boundary value analysis), 分支覆盖 (branch coverage), 基于检查表的测试 (checklist-based testing), 基于协作的测试方法 (collaboration-based test approach), 覆盖 (coverage), 覆盖项 (coverage item), 判定表测试 (decision table testing), 等价类划分 (equivalence partitioning), 错误猜测 (error guessing), 基于经验的测试技术 (experience-based test technique), 探索性测试 (exploratory testing), 状态转移测试 (state transition testing), 语句覆盖率 (statement coverage), 测试技术 (test technique), 白盒测试技术 (white-box test technique)

第 4 章的学习目标:

4.1 测试技术概述

FL-4.1.1 (K2) 区分黑盒、白盒和基于经验的测试技术

4.2 黑盒测试技术

- FL-4.2.1 (K3) 使用等价类划分生成测试用例
- FL-4.2.2 (K3) 使用边界值分析生成测试用例
- FL-4.2.3 (K3) 使用判定表测试生成测试用例
- FL-4.2.4 (K3) 使用状态转移测试生成测试用例

4.3 白盒测试技术

- FL-4.3.1 (K2) 解释语句测试
- FL-4.3.2 (K2) 解释分支测试
- FL-4.3.3 (K2) 解释白盒测试的价值

4.4 基于经验的测试技术

- FL-4.4.1 (K2) 解释错误猜测法
- 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

4.5 基于协作的测试方法

- FL-4.5.1 (K2) 解释如何与开发人员和业务代表合作编写用户故事
- FL-4.5.2 (K2) 对编写验收准则的不同选项进行分类
- FL-4.5.3 (K3) 使用验收测试驱动开发 (ATDD) 生成测试用例

4.1 测试技术概述

测试技术支持测试人员开展测试分析（测试什么）和测试设计（如何测试）。测试技术有助于以系统的方式开发相对较小但充分的测试用例集。测试技术还可以在测试分析和设计期间帮助测试人员定义测试条件、识别覆盖项，并识别测试数据。有关测试技术及其相关度量的更多信息可以在 ISO/IEC/IEEE 29119-4（或 GB/T 38634.4-2020）标准以及（Beizer 1990, Craig 2002, Copeland 2004, Koomen 2006, Jorgensen 2014, Ammann 2016, Forgács 2019）中找到。

本大纲将测试技术分为黑盒、白盒和基于经验的测试技术。

黑盒测试技术（也称为基于规格说明的技术）是基于对测试对象的特定行为进行分析，而不考虑其内部结构。因此，测试用例不依赖于软件的实现方式。所以，如果具体实现改变了，但是所需要的行为保持不变，那么测试用例仍然是有用的。

白盒测试技术（也称为基于结构的技术）是基于对测试对象的内部结构和处理进行分析。由于测试用例依赖于软件的设计方式，因此白盒测试只能在测试对象的设计或实现之后创建。

基于经验的测试技术有效地利用测试人员的知识和经验进行测试用例的设计和实施。这些技术的有效性在很大程度上取决于测试人员的技能。基于经验的测试技术能检测出使用黑盒和白盒测试技术可能会遗漏的缺陷。因此，基于经验的测试技术是对黑盒和白盒测试技术的补充。

4.2 黑盒测试技术

以下各节中讨论的常用黑盒测试技术包括：

- 等价类划分
- 边界值分析
- 判定表测试
- 状态转移测试

4.2.1 等价类划分

等价类划分（EP）基于给定分区中的所有元素都会被测试对象以相同方式处理的预期将数据划分为多个分区（称为等价类）。此技术背后的原理是，如果测试用例测试了等价类中的一个值，并检测到缺陷，那么任何来自同一等价类的其他值的测试用例也应该能够检测到这个缺陷。因此，每个分区执行一个测试就足够了。

等价类划分可应用于与测试对象相关的任何数据元素，包括输入、输出、配置项、内部值、与时间相关的值和接口参数。等价类可以是连续的或离散的、有序的或无序的、有限的或无限的。等价类不能重叠，并且必须是非空集合。

对于简单的测试对象来说，等价类划分可能很容易，但在实践中，理解测试对象如何处理不同的值通常较为复杂。因此，应该谨慎进行划分。

包含有效值的分区称为有效等价类。包含无效值的分区称为无效等价类。有效值和无效值的定义可能因团队和组织而异。例如，有效值可能被解释为应该由测试对象处理的那些值，或者在规格说明中定义为其处理方式的值。无效值可能被解释为应该被测试对象忽略或拒绝的那些值，或者在测试对象规格说明中未定义其处理方式的值。

在等价类划分中，覆盖项是等价类划分。要使用这种技术实现 100%覆盖率，测试用例必须通过对所有已识别的等价类（包括无效等价类）至少覆盖每个分区一次。覆盖率是由被测试用例执行的等价类数量，除以已确定的等价类总数来度量的，并以百分比表示。

许多测试对象包括多组等价类分区（例如，具有多个输入参数的测试对象），这意味着一个测试用例将覆盖来自不同等价类组的等价类。在多组等价类情况下，最简单的覆盖准则被称为单选项覆盖（ECC – Each Choice Coverage, Ammann 2016）。单选项覆盖要求测试用例执行来自每个等价类组的每个等价类至少一次。单选项覆盖不考虑等价类的组合情况。

4.2.2 边界值分析

边界值分析（BVA）是基于执行等价类边界的技术。因此，边界值分析只能用于有序分区。分区的最小值和最大值是它的边界值。在边界值分析中，如果两个元素属于同一个分区，那么它们之间的所有元素也必须属于该分区。

边界值分析侧重于分区的边界值，因为开发人员更容易在这些边界值上出错。边界值分析发现的典型缺陷往往是位于实施边界上，这些边界在预期边界的上下相邻边界往往发生错位，或完全忽略了这些边界的处理。

本大纲涵盖了两个不同的边界值分析：二值边界值和三值边界值分析。它们在实现 100%覆盖所需的每个边界上的覆盖项方面有所不同。

在二值边界值分析中（Craig 2002, Myers 2011），每个边界值都有两个覆盖项：该边界值以及属于相邻分区的最近邻值。为了让二值边界值达到 100%的覆盖率，测试用例必须执行所有的覆盖项，即所

有已识别的边界值。覆盖率是由已被执行的边界值的数量，除以已识别的边界值总数度量，并以百分比表示。

在三值边界值分析中（Koomen 2006, O’ Regan 2019），每个边界值都有三个覆盖项：该边界值以及两侧的相邻值。因此，在三值边界值分析中，某些覆盖项可能不是边界值。为了让三值边界值分析达到 100% 的覆盖率，测试用例必须执行所有的覆盖项，即已识别的边界值及其相邻值。覆盖率是由已被执行的边界值及其相邻值的数量，除以已识别的边界值及其相邻值的总数度量，并以百分比表示。

三值边界值分析比二值边界值分析更严格，因为它可以检测出二值边界值分析所忽略的缺陷。例如，如果判定“if ($x \leq 10$) ...”错误地实现为“if ($x=10$) ...”，则按照二值边界值生成的测试用例（ $x=10$, $x=11$ ）无法检测出该缺陷。然而，由三值边界值生成的 $x=9$ ，可能会检测出该缺陷。

4.2.3 判定表测试

判定表用于测试系统需求的实现，这些需求指定了不同的条件组合产生不同的结果。判定表是记录复杂逻辑（如业务规则）的有效方法。

在创建判定表时，定义了系统的条件和导致的动作。这些组成了判定表的行。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唯一组合以及相关的动作。在有限判定表中，所有条件和动作的值（除了不相关或不可行的条件外，参阅下文）表示为布尔值（“真”或“假”）。另外，在扩展判定表中，某些或者所有条件和动作也可以采用多个值（例如，数字范围、等价类、离散值）。

条件的符号如下：“T”（true）表示满足条件；“F”（false）表示不满足条件；“-”表示条件的取值与动作结果不相关；“N/A”表示该条件对于给定的规则是不可行的。动作中的“X”表示动作应该发生；空白表示动作不应该发生。也可以使用其他符号。

完全的判定表有足够多的列来覆盖条件的每种组合。通过删除包含不可行的条件组合的列，能简化判定表。通过合并列，也能最小化判定表。在某些条件不影响结果的情况下，可将表中的某些列合并为一列。判定表最小化算法不在本大纲的范围内。

在判定表测试中，覆盖项是包含可行条件组合的列。为了让这种技术达到 100% 的覆盖率，测试用例必须执行所有这些列。覆盖率是由已被执行的列数量，除以可执行列的总数度量，并以百分比表示。

判定表测试的优点是提供了系统化的方法来识别所有的条件组合，否则其中的某些组合可能会被忽视。它也有助于发现需求中的任何漏洞或不一致。如果有很多条件，那么执行所有判定规则可能比较耗时，因为规则的数量会随着条件的数量呈指数增长。在这种情况下，为了减少需要执行的规则的数量，可以使用最小化判定表或基于风险的方法。

4.2.4 状态转移测试

状态转移图通过显示系统可能的状态和有效的状态转移来模拟系统的行为。转移由事件触发，该事件可能由守护条件进一步限定。假设转移是瞬时的，有时可能导致软件采取动作。常见的转移标签语法如下：“事件 [守护条件] / 动作”。如果守护条件和动作不存在或与测试人员无关，则可以被省略。

状态表是等价于状态转移图的一种模型。它的行表示状态，列表示事件（可能还包括守护条件）。状态表条目（单元格）表示转移，并包含目标状态以及结果动作（如果已定义）。与状态转移图相反，状态表通过空白单元格明确显示无效的转移。

基于状态转移图或状态表的测试用例通常表示为事件序列，这些事件会导致一系列的状态变化（和动作，如果需要）。一个测试用例可以并且通常会覆盖状态之间的若干次转移。

状态转移测试有多种覆盖准则，本大纲讨论其中的三种。

在**全状态覆盖**中，覆盖项是状态。为了达到 100% 的全状态覆盖率，测试用例必须确保所有状态都被访问到。覆盖率是由已被访问的状态数量，除以状态总数量，并以百分比表示。

在**有效转移覆盖**（也称为 0-switch 覆盖）中，覆盖项是单个有效转移。为了达到 100% 的覆盖率，测试用例必须执行所有的有效转移。覆盖率是由已执行的有效转移的数量，除以有效转移的总数量，并以百分比表示。

在**全转移覆盖**中，覆盖项是状态表中显示的所有转移。为了达到 100% 的全转移覆盖率，测试用例必须执行所有的有效转移，并尝试执行无效转移。在单个测试用例中仅测试一个无效转移有助于避免错误屏蔽，即一个缺陷阻止另一个缺陷被发现的情况。覆盖率是由已执行的测试用例所覆盖的执行有效转移和尝试无效转移的数量，除以有效和无效转移的总数量，并以百分比表示。

全状态覆盖弱于有效转移覆盖，因为它通常可以在不执行所有转移的情况下达到。有效转移覆盖是最广泛使用的覆盖准则。达到完整的有效转移覆盖可保证完整的全状态覆盖。达到完整的全转移覆盖可同时保证完整的全状态覆盖和完整的有效转移覆盖，并且应该是关键任务和安全关键软件的最低要求。

4.3 白盒测试技术

由于普及性和简单性，本节将重点介绍两种与代码相关的白盒测试技术：

- 语句测试
- 分支测试

安全关键、任务关键或高完整性环境中使用了更严格的技术，用于实现更全面的代码覆盖。白盒测试技术还能用于更高的测试级别（例如，API 测试），或使用与代码无关的覆盖（例如，神经网络测试中的神经元覆盖）。这些技术不在本大纲中讨论。

4.3.1 语句测试和语句覆盖

在语句测试中，覆盖项是可执行语句。目的是设计测试用例，通过执行代码中的语句，直到达到可接受的覆盖率级别。覆盖率是用测试用例执行的语句数除以代码中可执行语句的总数度量，并以百分比表示。

达到 100%语句覆盖率时，能确保代码中的所有可执行语句至少已被执行过一次。需要特别注意的是，带有缺陷的每条语句将被执行到，这可能导致失效，从而证明缺陷的存在。然而，使用测试用例执行语句不能在所有的情况下都检测到缺陷。例如，可能检测不到那些与数据相关的缺陷（例如：只有当分母被设置为 0 时才会发生的除以 0 的错误）。同样，100%的语句覆盖率不能确保所有的判定逻辑都被测试过，例如，它可能执行不到代码中的所有分支（参阅第 4.3.2 节）。

4.3.2 分支测试和分支覆盖

在控制流图中，分支指的是两个节点之间的控制转移，它显示了源代码语句在测试对象中执行时的可能顺序。每次控制转移可以是无条件的（如直线式代码）或有条件的（如判定结果）。

在分支测试中，覆盖项是分支。目的是设计测试用例，以执行代码中的分支，直到达成可接受的覆盖率级别为止。覆盖率是用测试用例执行的分支数除以分支总数度量，并以百分比表示。

当达到 100%的分支覆盖时，代码中的所有分支，不满足条件的以及满足条件的，都被测试用例执行。条件分支通常对应于“if... then”判定中的真（true）或假（false）结果、switch/case 语句的结果，或是退出或继续执行循环的判定。然而，使用测试用例来执行分支并不能在所有情况下都检测到缺陷。例如，可能无法检测到需要执行代码中特定路径的缺陷。

分支覆盖包含语句覆盖。任何一组测试用例，如果实现了 100%的分支覆盖率，也就实现了 100%的语句覆盖率（但反之不成立）。

4.3.3 白盒测试的价值

所有白盒技术都具有的基本优点是，在测试期间考虑到整个软件实现，这有助于在软件规格说明模糊、过时或不完整时检测缺陷。相应的弱点是如果软件没有实现一个或多个需求，白盒测试可能无法检测到由此产生的遗漏缺陷（Watson 1996）。

白盒技术可用于静态测试（例如，在代码试运行期间）。白盒技术非常适合于对尚未准备好执行的代码进行评审（Hetzel 1988），以及伪代码和其他可以用控制流图建模的高层逻辑或自顶向下的逻辑。

仅执行黑盒测试不能提供实际代码覆盖率的度量。白盒覆盖率的度量数据提供了客观的覆盖率度量，并提供了必要的信息，以允许生成额外的测试来提高覆盖率，从而增加对代码的信心。

4.4 基于经验的测试技术

以下各节将讨论常用的基于经验的测试技术：

- 错误猜测
- 探索性测试
- 基于检查表的测试

4.4.1 错误猜测

错误猜测是基于测试人员的知识来预测错误、缺陷和失效发生的技术，包括：

- 以前软件的运行方式。
- 开发人员常犯的错误类型以及由这些错误导致的缺陷类型。
- 其他类似应用软件产生的失效类型。

通常，错误、缺陷和失效可能与以下内容有关：输入（例如，正确的输入未被接受、参数错误或缺失）、输出（例如，错误的格式、错误的结果）、逻辑（例如，遗漏相关分支、错误的运算符）、计算（例如，不正确的操作数、错误的计算）、接口（例如，参数不匹配、不兼容的类型）或数据（例如，不正确的初始化、错误的类型）。

故障攻击是实现错误猜测的一种系统方法。这项技术要求测试人员创建或获取一份可能的错误、缺陷和失效列表，并设计测试来识别与缺陷相关的错误、暴露缺陷、或导致失效。这些列表可以基于经验、缺陷和失效数据，或者基于软件失败原因的常识来构建。

有关错误猜测和故障攻击的更多信息，请参阅（Whittaker 2002, Whittaker 2003, Andrews 2006）。

4.4.2 探索性测试

在探索性测试中，测试人员在了解有关测试对象的同时，设计、执行和评估测试。这种测试用于更深入地了解测试对象，使用重点测试进行深入探索，并为未测试的区域创建测试。

探索性测试有时使用基于会话的测试来构建。在基于会话的方法中，探索性测试在定义的时间盒内进行。测试人员使用包含测试目的的测试章程来指导测试。测试会话之后通常会汇报，包括测试人员和测试会话的测试结果感兴趣的利益相关方之间的讨论。在这种方法中，测试目的可被视为高级别的测试条件。在测试会话期间，识别并执行覆盖项。测试人员可以使用测试会话表格来记录所执行的步骤和发现的内容。

探索性测试在规格说明文档较少或不充分时，或测试的时间压力大的情况下很有帮助。探索性测试也有助于作为对其他更正式测试技术的补充。如果测试人员经验丰富，具备领域知识，并具备较高的基本技能（如分析能力、好奇心和创造力），探索性测试将更加有效。（参阅第 1.5.1 节）。

探索性测试可以结合其他测试技术使用（例如等价类划分）。关于探索性测试的更多信息，请参阅（Kaner 1999, Whittaker 2009, Hendrickson 2013）。

4.4.3 基于检查表的测试

在基于检查表的测试中，测试人员设计、实施和执行测试以覆盖检查表中的测试条件。检查表可以基于经验、对用户重要的知识，或者对软件失效原因和方式的理解来构建。检查表不应包含可以自动检查的项、更适合作为入口/出口准则的项或过于笼统的项（Brykczynski 1999）。

检查表的项通常以问题的形式表述。每个项应该可以单独和直接地进行检查。这些项可以涉及需求、图形界面属性、质量特性或其他形式的测试条件。可以创建检查表支持各种测试类型，包括功能和功能性测试（例如，易用性测试的 10 个启发式规则（Nielsen 1994））。

由于开发人员能够学会避免犯同样的错误，因此检查表的某些条目随着时间的推移可能逐渐不那么有效。可能还需要添加新条目，以反映新发现的高严重性缺陷。因此，应根据缺陷分析定期更新检查表。但是，需要注意避免检查表变得过长（Gawande 2009）。

在缺乏详细的测试用例的情况下，基于检查表的测试可以为测试提供指南和某种程度的一致性。如果检查表是高级别的，实际测试可能会出现一些可变性，从而可能获得更广泛的覆盖范围，同时保持较低的重复性。

4.5 基于协作的测试方法

上述每种技术（参阅第 4.2、4.3、4.4 节）在缺陷检测方面都有特定的目标。另一方面，基于协作的方法也侧重于通过协作和交流来避免缺陷。

4.5.1 协作用户故事编写

用户故事代表对系统或软件的用户或购买者有价值的特征。用户故事包括三个关键方面（Jeffries 2000），统称为“3C”：

- 卡片（Card）- 描述用户故事的媒介（例如索引卡、电子板中的条目）。
- 对话（Conversation）- 解释软件的使用方式（可以是文档化或口头形式）。
- 确认（Confirmation）- 验收准则（参阅第 4.5.2 节）。

用户故事最常见的格式是“作为一个【角色】，我希望【要实现的目标】，以便于我可以【为角色创造商业价值】”，其次是验收准则。

协作编写用户故事可以使用头脑风暴和思维导图等技术。通过协作，同时兼顾到业务、开发和测试三个视角，团队可以共同理解应该交付的内容和共享愿景。

好的用户故事应该是：独立的（Independent）、可协商的（Negotiable）、有价值的（Valuable）、可评估的（Estimable）、小巧的（Small）和可测试的（Testable）（INVEST）。如果利益相关方不知道如何对用户故事进行测试，可能说明用户故事不够清晰，或者没有反映出有价值的内容，或者利益相关方需要在测试方面得到帮助（Wake 2003）。

4.5.2 验收准则

用户故事的验收准则是实现用户故事必须满足的条件，以便被利益相关方接受。从这个角度来看，验收准则可以被视为测试应该覆盖的测试条件。验收准则通常是对话的结果（参阅第 4.5.1 节）。

验收准则用于：

- 定义用户故事的范围。
- 利益相关方之间达成共识。

- 描述正向和反向场景。
- 作为用户故事验收测试的依据（参阅第 4.5.3 节）。
- 实现准确规划和估算。

编写用户故事的验收准则有几种方式。最常见的两种格式是：

- 面向场景（例如，BDD 中使用的 Given/When/Then 格式，参阅第 2.1.3 节）
- 面向规则（例如，要点验证列表，或输入-输出（I/O）映射的表格形式）

大多数验收准则可以在两种格式中任选一种进行记录。然而，团队也可以使用自定义格式，只要验收准则明确定义且无歧义。

4.5.3 验收测试驱动开发 (ATDD)

ATDD 是测试先行（test-first）的一种方法（参阅第 2.1.3 节）。测试用例在实现用户故事之前创建。测试用例由具有不同视角的团队成員创建，例如客户、开发人员和测试人员（Adzic 2009）。测试用例可以人工或自动化执行。

第一步是进行规格说明讨论会，团队成员分析、讨论和编写用户故事及其验收准则（如果尚未定义）。在此过程中解决了用户故事中的不完整、模糊或缺陷。下一步是创建测试用例，可以由整个团队或测试人员单独完成。测试用例基于验收准则，并且视为软件运行方式的实例。这将有助于团队正确实现用户故事。

由于实例和测试是相同的，因此这些术语常常可以互换使用。在测试设计过程中，可以应用第 4.2、4.3 节和 4.4 节中描述的测试技术。

通常，第一批测试用例是正向测试，确认在没有异常或错误条件下的正确行为，并包括按照预期执行的活动序列。在完成正向测试用例后，团队应进行反向测试。最后，团队还应涵盖非功能质量特性（例如性能效率、易用性）。测试用例应以利益相关方可以理解的方式进行描述。通常，测试用例包含自然语言形式的句子，包括必要的前提条件（如果有），输入和后置条件。

测试用例必须覆盖用户故事的所有特性，并且不应超出故事范围。不过，验收准则可能详细描述用户故事中描述的一些问题。此外，任何两个测试用例都不应该描述用户故事的相同特性。

当以测试自动化框架支持的格式进行捕获时，开发人员可以在实现用户故事描述的特征时编写支持代码，从而自动化测试用例。验收测试就成为可执行的需求。

5. 管理测试活动 - 335 分钟

关键词

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), 风险分析 (risk analysis), 风险评估 (risk assessment), 风险控制 (risk control), 风险识别 (risk identification), 风险级别 (risk level), 风险管理 (risk management), 风险缓解 (risk mitigation), 风险监测 (risk monitoring), 基于风险的测试 (risk-based testing), 测试方法 (test approach), 测试完成报告 (test completion report), 测试控制 (test control), 测试监测 (test monitoring), 测试计划 (test plan), 测试规划 (test planning), 测试进度报告 (test progress report), 测试金字塔 (test pyramid), 测试象限 (testing quadrants)

第 5 章的学习目标:

5.1 测试规划

- FL-5.1.1 (K2) 举例说明测试计划的目的和内容。
- FL-5.1.2 (K1) 认识测试人员如何为迭代和发布规划增加价值。
- FL-5.1.3 (K2) 比较入口准则和出口准则。
- FL-5.1.4 (K3) 使用估算技术计算所需的测试工作量。
- FL-5.1.5 (K3) 应用测试用例优先级。
- FL-5.1.6 (K1) 回顾测试金字塔的概念。
- FL-5.1.7 (K2) 总结测试象限及其与测试级别和测试类型的关系。

5.2 风险管理

- FL-5.2.1 (K1) 利用风险可能性和风险影响识别风险级别。
- FL-5.2.2 (K2) 区分项目风险和产品风险。
- FL-5.2.3 (K2) 解释产品风险分析如何影响测试的充分性和范围。
- FL-5.2.4 (K2) 解释可以采取哪些措施分析产品风险。

5.3 测试监测、测试控制和测试完成

- FL-5.3.1 (K1) 回顾用于测试的度量。
- FL-5.3.2 (K2) 总结测试报告的目的、内容和受众。
- FL-5.3.3 (K2) 举例说明如何沟通测试状态。

5.4 配置管理

- FL-5.4.1 (K2) 总结配置管理如何支持测试。

5.5 缺陷管理

- FL-5.5.1 (K3) 准备缺陷报告。

5.1 测试规划

5.1.1 测试计划的目的是和内容

测试计划描述了测试项目的目的、资源和过程。测试计划的目的包括：

- 记录实现测试目的的方法和进度表。
- 有助于确保执行的测试活动符合既定标准。
- 作为与团队成员和其他利益相关方沟通的一种方式。
- 表明测试遵守现有的测试方针和测试策略（或解释为什么测试会偏离这些策略）。

测试规划引导测试人员思考，并促进测试人员面对与风险、进度、人员、工具、成本、工作量等相关的未来挑战。准备测试计划是思考实现测试项目目的所需工作量的过程。

测试计划的典型内容包括：

- 测试的周境（例如，范围、测试目的、约束条件、测试依据）。
- 测试项目的假设和限制。
- 利益相关方（例如，角色、职责、与测试、招聘和培训要求的相关性）。
- 沟通（例如，沟通的形式和频率、文档模板）。
- 风险记录（例如，产品风险、项目风险）。
- 测试方法（例如，测试级别、测试类型、测试技术、测试可交付成果、入口准则和出口准则、测试的独立性、要收集的度量、测试数据需求、测试环境需求、与组织测试方针和测试策略的偏差）。
- 预算和进度表。

有关测试计划及其内容的更多详细信息，请参阅 ISO/IEC/IEEE 29119-3（或 GB/T 38634.3-2020）标准。

5.1.2 测试人员对迭代和发布规划的贡献

在迭代软件开发生存周期（SDLC）中，通常会出现两种规划：发布规划和迭代规划。

发布规划着眼于产品的发布，定义和重新定义产品待办事项，并可能涉及将较大的用户故事细化为一组较小的用户故事。它还作为跨所有迭代中测试方法和测试计划的基础。参与发布规划的测试人员参与编写可测试的用户故事和验收准则（参阅第 4.5 节），参与项目和质量风险分析（参阅第 5.2 节），估算与用户故事相关的测试工作量（参阅第 5.1.4 节），确定测试方法，并为发布计划进行测试。

迭代规划着眼于完成单个迭代，并关注迭代待办事项。参与迭代规划的测试人员参与用户故事的详细风险分析，确定用户故事的可测试性，将用户故事分解为任务（尤其是测试任务），估算所有测试任务的测试工作量，并识别和细化测试对象的功能和非功能方面。

5.1.3 入口准则和出口准则

入口准则定义了开展特定活动的先决条件。如果不符合入口准则，这项活动很可能会变得更困难、耗时、成本高昂、风险更大。出口准则定义了完成测试活动必须达到的目标。应为每个测试级别定义入口准则和出口准则，并根据测试目的而有所不同。

典型的入口准则包括：资源的可用性（例如，人员、工具、环境、测试数据、预算、时间）、测试件的可用性（例如，测试依据、可测试的需求、用户故事、测试用例）和测试对象的初始质量水平（例如，所有冒烟测试都已通过）。

典型的出口准则包括：充分性度量（例如，已实现的覆盖级别、未解决的缺陷数量、缺陷密度、失败的测试用例数量）和完成标准（例如，已执行计划的测试、已执行的静态测试、已报告发现的所有缺陷、已自动化的所有回归测试）。

时间或预算不足也可以被视为有效的出口准则。即使不满足其他出口准则，如果利益相关方已经评审并接受了不做进一步测试就上线的风险，那么在这种情况下可以接受结束测试。

在敏捷软件开发中，出口准则通常称为完成的定义（DoD），定义团队发布项目的客观度量。入口准则包括了必须满足用户故事才能开始开发和/或测试活动，入口准则也称为就绪的定义（DoR）。

5.1.4 估算技术

测试工作量估算包括预测满足测试项目目的所需的测试相关工作量。要向利益相关方明确，估算是基于许多假设的，并且总是会出现估算误差。小任务的估算通常比大任务的估算更准确。因此，当估算大任务时，可以将其分解为一组较小的任务，然后再对其进行估算。

在本教学大纲中，描述了以下四种估算技术。

基于比率的估算。 在这种基于度量的技术中，数据是从组织内以前的项目中收集的，这使得可以得出类似项目的“标准”比率。组织自身项目的比率（例如，取自历史数据）通常是估算过程中能使用的最佳数据来源。可以使用这些标准比率来估算新项目的测试工作量。例如，如果在以前的项目中，开发与测试的工作量比为 3:2，而在当前项目中，预计开发工作量为 600 人日，那么测试工作量可以估算为 400 人日。

外推。这是基于度量的技术，在当前项目中尽早进行测量以收集数据。有了足够的观测结果，剩余工作所需的工作量可以通过外推这些数据（通常通过应用数学模型）来估算。这种方法非常适用于迭代的 SDLC。例如，团队可以将即将迭代的测试工作量外推为最近三次迭代的平均工作量。

宽带德尔菲。这是基于专家的迭代技术，专家进行基于经验的估算。每一位专家独立估算工作量。收集到的结果如果存在超出商定边界范围的偏差，专家们将讨论他们目前的估算。然后，要求每个专家根据反馈重新独立估算。不断重复此过程，直到达成共识。规划扑克是宽带德尔菲方法的一个变体，通常用于敏捷软件开发。在计划扑克中，通常使用代表工作量大小的数字卡片进行估算。

三点估算。在这种基于专家的技术中，专家做出三个估算：最乐观的估算（a）、最可能的估算（m）和最悲观的估算（b）。最终估算（E）是它们的加权算术平均值。在该技术最流行的版本中，估算值计算为 $E = (a + 4 \cdot m + b) / 6$ 。该技术的优点是允许专家计算测量误差： $SD = (b - a) / 6$ 。例如，如果估算值（人时）为：a=6, m=9 和 b=18，则最终估算值为 10 ± 2 人时（即 8 至 12 人时），因为 $E = (6 + 4 \cdot 9 + 18) / 6 = 10$ ， $SD = (18 - 6) / 6 = 2$ 。

可参阅（Kan 2003, Koomen 2006, Westfall 2009）了解更多的测试估算技术。

5.1.5 测试用例优先级排序

制定完测试用例和测试规程并组装到测试套件之后，测试套件就可以填写进测试执行进度表，该进度表定义了测试套件的运行顺序。在对测试用例进行优先级排序时，可以考虑不同的因素。最常用的测试用例优先级策略如下：

- **基于风险的优先级：**测试执行顺序基于风险分析的结果（参阅第 5.2.3 节）。优先执行覆盖最重要风险的测试用例。
- **基于覆盖范围的优先级：**测试执行顺序基于覆盖（例如，语句覆盖）。优先执行覆盖率最高的测试用例。另外一种优先级策略称为附加覆盖优先级，优先执行能达到最高覆盖的测试用例；后续每个测试用例都是达到最高附加覆盖率的测试用例。
- **基于需求的优先级：**测试执行的顺序基于测试用例对应需求的优先级。需求优先级由利益相关方定义。优先执行最重要需求相关的测试用例。

理想情况下，测试用例将根据它们的优先级排序运行，例如使用上述优先级策略之一。但是如果测试用例或正在测试的特征具有相关性，则这种做法可能不起作用。如果优先级较高的测试用例依赖于优先级较低的测试用例，则必须优先执行低优先级的测试用例。

测试执行的顺序还必须考虑资源的可用性。例如，可能仅在特定时间窗口内可用的测试工具、测试环境或人员。

5.1.6 测试金字塔

测试金字塔是表明不同测试可能具有不同颗粒度的模型。通过说明不同级别的测试自动化支持的不同目标，测试金字塔模型支持团队的测试自动化和测试工作量分配。各金字塔层表示测试集。层越高，测试颗粒度就越粗，测试独立性越弱和测试执行速度越慢。底层的测试规模小、独立性强、快速，只检查部分功能，因此通常需要大量测试来实现合理的覆盖率。顶层代表复杂的、高级别的、端到端的测试。这些高层的测试通常比底层的测试慢，并且它们通常检查大块的功能，因此通常只需要其中的几个测试就可以实现合理的覆盖。金字塔层的数量和命名可能不同。例如，最初的测试金字塔模型（Cohn 2009）定义了三层：“单元/组件测试”、“服务测试”和“UI 测试”。另一个流行的模型定义了单元（组件）测试、集成（组件集成）测试和端到端测试。也可以使用其他测试级别（参阅第 2.2.1 节）。

5.1.7 测试象限

Brian Marick（Marick 2003，Crispin 2008）定义的测试象限将敏捷软件开发中的测试级别与适当的测试类型、活动、测试技术和工作产品组合在一起。该模型可视化了这些内容，以确保所有适当的测试类型和测试级别都包含在 SDLC 中，解释了某些测试类型比其他测试类型与某些测试级别相关性更强，以此支持测试管理。该模型还为所有利益相关方（包括开发人员、测试人员和业务代表）提供了区分和描述测试类型的方法。

在这个模型中，测试可以是面向业务的，也可以是面向技术的。测试还可以支持团队（即指导开发）或评价产品（即根据预期衡量其行为）。这两种观点的结合决定了四个象限：

- 象限 Q1（面向技术，支持团队）。此象限包含组件和组件集成测试。这些测试应该是自动化的，并包含在持续集成（CI）过程中。
- 象限 Q2（面向业务，支持团队）。此象限包含功能测试、实例、用户故事测试、用户体验原型、API 测试和模拟测试。这些测试检查验收准则，可以是人工的，也可以是自动的。
- 象限 Q3（面向业务，评价产品）。这个象限包含探索性测试、易用性测试和用户验收测试。这些测试是面向用户的，通常是人工测试。
- 象限 Q4（面向技术，评价产品）。此象限包含冒烟测试和非功能性测试（易用性测试除外）。这些测试通常是自动化的。

5.2 风险管理

组织面临许多内部和外部因素，使其无法确定是否以及何时能实现目的（ISO 31000）。风险管理使组织能够增加实现目的的可能性，提高产品质量，增加利益相关方的信心和信任。

风险管理的主要活动包括：

- 风险分析（包括风险识别和风险评估；参阅第 5.2.3 节）。
- 风险控制（包括风险缓解和风险监测；参阅第 5.2.4 节）。

基于风险分析和风险控制来选择、安排优先级和管理测试活动的测试方法称为基于风险的测试。

5.2.1 风险定义和风险属性

风险是指潜在的事件、危险、威胁或情况，其发生会造成不利影响。风险可由两个因素表征：

- 风险可能性 – 风险发生的概率（大于零且小于一）。
- 风险影响（危害）——风险发生的后果。

这两个因素表示了风险级别，这是衡量风险的指标。风险级别越高，其处理就越重要。

5.2.2 项目风险和产品风险

在软件测试中，人们通常关注两种类型的风险：项目风险和产品风险。

项目风险与项目的管理和控制有关。项目风险包括：

- 组织问题（例如，工作产品交付延迟、估算不准确、成本削减）。
- 人员问题（例如，技能不足、冲突、沟通问题、员工短缺）。
- 技术问题（例如，范围蔓延、工具支撑不足）。
- 供应商问题（例如，第三方交付失败、供应商公司破产）。

项目风险一旦发生，可能会对项目进度、预算或范围产生影响，从而影响项目实现其目的的能力。

产品风险与产品质量特性有关（如 ISO 25010 质量模型中所述）。产品风险的例子包括：功能缺失或错误、计算错误、运行时错误、架构不良、算法效率低、响应时间过长、用户体验差、安全漏洞。发生产品风险，可能会导致各种负面后果，包括：

- 用户不满。
- 收入、信任和声誉损失。
- 对第三方的损害。
- 维护成本高，服务台过载。
- 刑事处罚。
- 在极端情况下，造成身体损伤、受伤甚至死亡。

5.2.3 产品风险分析

从测试的角度来看，产品风险分析的目标是提供对产品风险的认识，以便以最小残留产品风险级别的方式集中安排测试工作量。理想情况下，产品风险分析自 SDLC 早期开始。

产品风险分析包括风险识别和风险评估。风险识别就是生成一份全面的风险清单。利益相关方可以使用各种技术和工具来识别风险，例如头脑风暴、研讨会、访谈或因果图。风险评估包括：对已识别的风险进行分类，确定其风险可能性、风险影响和级别，确定优先级，并提出处理方法。分类有助于提出缓解措施，通常同一类别的风险可以使用类似的方法来缓解。

风险评估可以使用定量或定性方法，也可以混合使用。在定量方法中，风险级别为风险可能性和风险影响的乘积。在定性方法中，可以使用风险矩阵来确定风险级别。

产品风险分析可能会影响测试的充分性和范围。其结果可以用于：

- 确定要进行的测试范围。
- 确定特定的测试级别，提供建议执行的测试类型。
- 确定采用的测试技术和达到的覆盖。
- 估算每项任务所需的测试工作量。
- 进行测试优先级排序，以尽早发现关键缺陷。
- 确定是否可以采用除测试之外的其他活动来降低风险。

5.2.4 产品风险控制

产品风险控制包括为应对已识别和评估的产品风险而采取的所有措施。产品风险控制由风险缓解和风险监测组成。风险缓解包括实施风险评估中提出的行动，以降低风险级别。风险监测的目的是确保缓解行动有效，获取进一步信息以改进风险评估，以及识别新出现的风险。

关于产品风险控制，风险分析后，可能有多种风险应对方案，例如通过测试缓解风险、接受风险、转移风险或应急计划（Veenendaal 2012）。可以通过测试缓解产品风险的活动如下：

- 选择具有适当经验和技能水平、适合特定风险类型的测试人员。
- 应用适当级别的测试独立性。
- 实施评审及静态分析。
- 应用适当的测试技术和覆盖级别。
- 针对受影响的质量特性应用适当的测试类型。
- 执行动态测试，包括回归测试。

5.3 测试监测、测试控制和测试完成

测试监测关注测试相关信息的收集。该信息可用于评估测试过程，并测量测试出口准则或与测试出口准则关联的测试活动是否已经满足，例如产品风险、需求、或验收准则是否达到了覆盖目标。

测试控制通过使用测试监测提供的信息，以控制指令的方式提供指导或必要的纠正措施，以达成最有效和最高效的测试。

控制指令的示例包括：

- 已识别的风险成为问题时，重新确定测试优先级。
- 重新评估由于返工而导致的测试项是否满足入口或出口准则。
- 调整测试进度表以应对测试环境交付延迟的问题。
- 在必要的时间和地点增加新资源。

测试完成从已完成的测试活动中收集数据，总结经验、整合测试件和收集任何其他相关信息。测试完成活动发生在项目里程碑处，如测试级别已完成、敏捷的迭代已结束、测试项目已完成（或取消）、软件系统已发布、或维护版本已完成。

5.3.1 测试中使用的度量

收集的测试度量用以显示相对于计划和预算的当前进度、测试对象当前质量、以及测试活动在达成期望目标或者迭代目标的有效性。测试监测收集各种度量以支持测试控制和测试完成。

常见的测试度量包括：

- 项目进度度量（如：任务完成情况、资源使用情况、测试工作量）。
- 测试进度度量（如：测试用例实施进度、测试环境准备进度、已执行/未执行测试用例数、通过/失败的测试用例数、测试执行时间）。
- 产品质量度量（如：可用性、响应时间、平均失效时间）。
- 缺陷度量（如：发现和修复的缺陷数和优先级、缺陷密度、缺陷检测率）。
- 风险度量（如：遗留风险级别）。
- 覆盖率度量（如：需求覆盖率、代码覆盖率）。
- 成本度量（如：测试成本、组织的质量成本）。

5.3.2 测试报告的目的、内容和受众

测试报告用于在测试过程中和测试结束后总结并沟通测试信息。测试进度报告支持对测试的持续控制，当出现由于偏离计划或环境变化而需要修改测试进度表、资源或测试计划时，测试进度报告应提供足够的信息。测试完成报告总结了测试的特定阶段（如：测试级别、测试周期、迭代），并能为后续的测试提供信息。

在测试监测和控制期间，测试团队为了让利益相关方随时了解情况，向他们提供测试进度报告。测试进度报告通常以固定频率（如：每天、每周等）生成，包括：

- 测试周期。
- 测试进度（如：进度提前或延迟），包括任何显著偏差。
- 测试的阻碍及解决方案。
- 测试度量（参阅 5.3.1 节示例）。
- 测试周期内的新风险和变化的风险。
- 为下一周期计划测试。

在项目、测试级别、测试类型完成，以及理想情况下出口准则得到满足时，测试完成期间准备测试完成报告。测试完成报告使用测试进度报告和其他数据。典型的测试完成报告包括：

- 测试总结。
- 基于原始测试计划（即：测试目的和出口准则）的测试和产品质量评估。
- 与测试计划的偏差（如：与计划的进度表、持续时间和工作量的差异）。
- 测试障碍和解决方案。
- 基于测试进度报告的测试度量。
- 未缓解的风险，未修复的缺陷。
- 与测试相关的经验教训。

在报告中，不同的受众需要不同的信息，因此影响报告的正式程度和频率。向同一团队的其他成员报告测试进度通常更频繁也不正式，而针对已完成项目的测试报告则应遵循固定的模板，并且只进行一次。

ISO/IEC/IEEE 29119-3 标准（GB/T 38634.3-2020）包括了用于测试进度报告（称为测试状态报告）和测试完成报告的模板和示例。

5.3.3 沟通测试状态

沟通测试状态的最佳方式各不相同，取决于测试管理的关注点、组织级测试策略、法规标准、或者在团队自组织情况下（参阅第 1.5.2 节），取决于团队本身。可能的选项包括：

- 与团队成员和其他利益相关方进行口头沟通。
- 仪表盘（如：CI/CD 仪表盘、任务板、燃尽图）。
- 电子沟通渠道（如：电子邮件、聊天工具）。
- 在线文档。
- 正式测试报告（参阅 5.3.2 节）。

可以使用上述中的一种或多种方式。对于分布式团队来说，由于地理距离或时区差异，直接面对面的沟通可能并不可行，因此更正式的沟通方式可能更合适。通常，不同利益相关方会对不同的信息感兴趣，因此沟通应进行相应的定制。

5.4 配置管理

在测试中，配置管理（CM）提供了识别、控制和跟踪工作产品的规程，如测试计划、测试策略、测试条件、测试用例、测试脚本、测试结果、测试日志和测试报告作为配置项。

对于复杂的配置项（如：测试环境），配置管理（CM）记录它所包含的项、项之间的关联关系和版本信息。如果该配置项被批准用于测试，它就成为了基线，只能通过正式的变更控制过程对其进行更改。

当创建新基线时，配置管理会记录已变更的配置项。使得通过恢复到先前的基线以重现测试结果成为可能。

为了恰当的支持测试工作，配置管理（CM）确保：

- 所有的配置项，包括测试项（测试对象的各个部分），均唯一标识、版本控制、变更跟踪，并与其他配置项建立关联，以确保在整个测试过程中的可追溯性。
- 所有标识的文档和软件项在测试文档中都被明确引用。

持续集成、持续交付、持续部署以及和其相关的测试通常作为自动化的 DevOps 流水线的一部分来实现（参阅第 2.1.4 节），其中通常包括自动化的配置管理（CM）。

5.5 缺陷管理

发现缺陷是测试的主要目标之一，建立缺陷管理过程是完全必需的。尽管这里指的是“缺陷”，但报告的异常既可能是真正的缺陷，也可能是其他（如：误报，变更请求）——这在处理缺陷报告的过程中会得到确认。在软件生存周期中的任何阶段都可能报告异常，其形式取决于软件生存周期。缺陷管理过程至少包括处理从发现到关闭各个异常的工作流程以及分类规则。工作流程通常包括记录报告的异常、分析和分类、决定适当的响应（如修复或保持原样）以及最终关闭缺陷报告的活动。所有利益相关方都必须遵守该过程。建议以类似的方式处理静态测试（尤其是静态分析）中的缺陷。

典型的缺陷报告包括以下目的：

- 向负责处理和解决缺陷报告的人员提供足够信息，以解决问题。
- 为跟踪工作产品质量提供途径。
- 为改进开发和测试过程提供思路。

动态测试期间记录的缺陷报告通常包括：

- 唯一标识符。
- 标题，简要总结所报告的异常情况。
- 发现异常的日期、提交的组织、作者及其角色。
- 测试对象和测试环境的标识。
- 缺陷的周境（如：运行的测试用例、执行的测试活动、软件生存周期阶段，以及其他相关信息，如使用的测试技术、检查表或测试数据）。
- 重现失效和解决方案的描述，包括可检测到异常的步骤、以及任何相关的测试日志、数据库转储、屏幕截图或者记录。
- 期望结果和实际结果。
- 缺陷针对利益相关方的利益或需求的严重程度（影响程度）。
- 修复的优先级。
- 缺陷的状态（如：打开、延迟、重复、待修复、待确认测试、重新打开、关闭、拒绝）。
- 引用（如：测试用例）。

使用缺陷管理工具时，其中的一些数据可能会自动包含在内（如：标识符、日期、作者和初始状态）。在 ISO/IEC/IEEE 29119-3 (GB/T 38634.3-2020) 标准中可以找到缺陷报告的模板和示例，该标准将缺陷报告 (Defect Reports) 称为事件报告 (Incident Reports)。

6. 测试工具 - 20 分钟

关键词

测试自动化 (test automation)

第 6 章的学习目标

6.1 测试活动中的工具支持

FL-6.1.1 (K2) 解释不同类型的测试工具如何支持测试。

6.2 测试自动化的收益和风险

FL-6.2.1 (K1) 回顾测试自动化的收益和风险。

中国软件测试认证委员会 (CSTQB®)

6.1 测试活动中的工具支持

测试工具支持并促进许多测试活动。示例包括但不限于：

- 管理工具 — 通过促进 SDLC、需求、测试、缺陷和配置的管理，提高测试过程的效率。
- 静态测试工具 — 支持测试人员执行评审和静态分析。
- 测试设计和实施工具 — 有助于生成测试用例、测试数据和测试规程。
- 测试执行和覆盖工具 — 有助于自动化测试执行和复盖率测量。
- 非功能性测试工具 — 允许测试人员执行难以执行或不可能人工执行的非功能性测试。
- DevOps 工具 — 支持 DevOps 交付流水线、工作流跟踪、自动化构建过程、CI/CD。
- 协作工具 — 促进沟通。
- 支持可扩展性和部署标准化的工具（例如，虚拟机、容器化工具）。
- 其他有助于测试的工具（例如，电子表格是测试活动环境中的测试工具）。

6.2 测试自动化的收益和风险

仅仅获得测试工具并不能保证成功，每种新工具都需要付出努力才能实现真正持久的收益（例如，工具的引入、维护和培训）。另外，工具存在风险，需要加以分析和缓解。

应用测试自动化的潜在收益包括：

- 减少重复性的人工活动以节省时间（例如，执行回归测试、重新输入相同的测试数据、比较预期结果与实际结果，以及按照编码规则进行检查）。
- 通过更高的一致性和可重复性来防止简单的人为错误（例如，始终从需求出发进行测试、测试数据以系统的方式创建、使用工具以相同的顺序以相同的频率执行测试）。
- 更客观的评估（例如，覆盖率），并提供人工无法实现的过于复杂的测量。
- 更容易得到测试的相关信息，以支持测试管理和测试报告（例如，有关测试进度、缺陷率和测试执行持续时间的统计数据、图表和聚合数据）。
- 减少测试执行时间，以提供更早的缺陷检测、更快的反馈和更早时间的上市。
- 让测试人员有更多时间设计新的、更深入、更有效的测试。

应用测试自动化的潜在风险包括：

- 对工具抱有不切实际的期望（包括功能性和易用性）。
- 对引入工具、维护测试脚本和更改现有人工测试过程所需的时间、成本和人力的估计不准确。
- 适合人工测试时，却使用测试工具。

- 过于依赖工具，例如忽视了必要的测试人员的批判性思维。
- 所依赖的工具供应商可能会倒闭、工具可能会报废、工具供应商将工具出售给其他供应商，或工具供应商提供较差的支持（例如，对询问的响应、版本升级和缺陷修复）。
- 使用可能被遗弃的开源软件，开源软件没有办法升级更新，或者如要进一步的开发其内部组件可能需要相当频繁的更新。
- 自动化工具与开发平台不兼容。
- 没有选择符合监管要求和/或安全标准的合适工具。

中国软件测试认证委员会 (CSTQB®)

7. 参考文献

标准

ISO/IEC/IEEE 29119-1 (2022) Software and systems engineering - Software testing - Part 1: General Concepts

ISO/IEC/IEEE 29119-2 (2021) Software and systems engineering - Software testing - Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2021) Software and systems engineering - Software testing - Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2021) Software and systems engineering - Software testing - Part 4: Test techniques

ISO/IEC 25010, (2011) Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models

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

ISO/IEC/IEEE 14764:2022 - Software engineering - Software life cycle processes - Maintenance

ISO 31000 (2018) Risk management - Principles and guidelines

书籍

Adzic, G. (2009) Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing, Neuri Limited

Ammann, P. and Offutt, J. (2016) Introduction to Software Testing (2e), Cambridge University Press

Andrews, M. and Whittaker, J. (2006) How to Break Web Software: Functional and Security Testing of Web Applications and Web Services, Addison-Wesley Professional

Beck, K. (2003) Test Driven Development: By Example, Addison-Wesley

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

Boehm, B. (1981) Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ

Buxton, J.N. and Randell B., eds (1970), Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27 - 31 October 1969, p. 16

Chelmsky, D. et al. (2010) The Rspec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends, The Pragmatic Bookshelf: Raleigh, NC

Cohn, M. (2009) Succeeding with Agile: Software Development Using Scrum, Addison-Wesley

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

- Crispin, L. and Gregory, J. (2008) Agile Testing: A Practical Guide for Testers and Agile Teams, Pearson Education: Boston MA
- Forgács, I., and Kovács, A. (2019) Practical Test Design: Selection of traditional and automated test design techniques, BCS, The Chartered Institute for IT
- Gawande A. (2009) The Checklist Manifesto: How to Get Things Right, New York, NY: Metropolitan Books
- Gärtner, M. (2011), ATDD by Example: A Practical Guide to Acceptance Test-Driven Development, Pearson Education: Boston MA
- Gilb, T., Graham, D. (1993) Software Inspection, Addison Wesley
- Hendrickson, E. (2013) Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing, The Pragmatic Programmers
- Hetzel, B. (1988) The Complete Guide to Software Testing, 2nd ed., John Wiley and Sons
- Jeffries, R., Anderson, A., Hendrickson, C. (2000) Extreme Programming Installed, Addison-Wesley Professional
- Jorgensen, P. (2014) Software Testing, A Craftsman's Approach (4e), CRC Press: Boca Raton FL
- Kan, S. (2003) Metrics and Models in Software Quality Engineering, 2nd ed., Addison-Wesley
- Kaner, C., Falk, J., and Nguyen, H.Q. (1999) Testing Computer Software, 2nd ed., Wiley
- Kaner, C., Bach, J., and Pettichord, B. (2011) Lessons Learned in Software Testing: A Context-Driven Approach, 1st ed., Wiley
- Kim, G., Humble, J., Debois, P. and Willis, J. (2016) The DevOps Handbook, Portland, OR
- Koomen, T., van der Aalst, L., Broekman, B. and Vroon, M. (2006) TMap Next for result-driven testing, UTN Publishers, The Netherlands
- Myers, G. (2011) The Art of Software Testing, (3e), John Wiley & Sons: New York NY
- O' Regan, G. (2019) Concise Guide to Software Testing, Springer Nature Switzerland
- Pressman, R.S. (2019) Software Engineering. A Practitioner's Approach, 9th ed., McGraw Hill
- Roman, A. (2018) Thinking-Driven Testing. The Most Reasonable Approach to Quality Control, Springer Nature Switzerland
- Van Veenendaal, E (ed.) (2012) Practical Risk-Based Testing, The PRISMA Approach, UTN Publishers: The Netherlands
- Watson, A.H., Wallace, D.R. and McCabe, T.J. (1996) Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, U.S. Dept. of Commerce, Technology Administration, NIST
- Westfall, L. (2009) The Certified Software Quality Engineer Handbook, ASQ Quality Press
- Whittaker, J. (2002) How to Break Software: A Practical Guide to Testing, Pearson
- Whittaker, J. (2009) Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design, Addison Wesley

Whittaker, J. and Thompson, H. (2003) How to Break Software Security, Addison Wesley

Wiegers, K. (2001) Peer Reviews in Software: A Practical Guide, Addison-Wesley Professional

文章和网页

Brykczynski, B. (1999) “A survey of software inspection checklists,” ACM SIGSOFT Software Engineering Notes, 24(1), pp. 82-89

Enders, A. (1975) “An Analysis of Errors and Their Causes in System Programs,” IEEE Transactions on Software Engineering 1(2), pp. 140-149

Manna, Z., Waldinger, R. (1978) “The logic of computer programming,” IEEE Transactions on Software Engineering 4(3), pp. 199-229

Marick, B. (2003) Exploration through Example, <http://www.exampler.com/old-blog/2003/08/21.1.html#agile-testing-project-1>

Nielsen, J. (1994) “Enhancing the explanatory power of usability heuristics,” Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, ACM Press, pp. 152 - 158

Salman, I. (2016) “Cognitive biases in software quality and testing,” Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16), ACM, pp. 823-826.

Wake, B. (2003) “INVEST in Good Stories, and SMART Tasks,” <https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/>

8. 附录 A - 学习目标 / 知识认知等级

下面定义的学习目标适用于本大纲。大纲中的每个主题都将根据其学习目标进行考试。学习目标根据与其对应的知识认知等级以相对应的行为动词开始，如下所示。

等级 1: 牢记 Remember (K1) - 考生要能牢记 remember、认识 recognize 和回顾 recall 一个术语或概念。

行为动词: 识别 identify、回顾 recall、牢记 remember、认识 recognize

举例说明:

- “识别典型的测试目的。”
- “回顾测试金字塔的概念。”
- “认识测试人员如何为迭代规划和发布规划提升价值。”

等级 2: 理解 Understand (K2) - 考生要能选择与主题相关陈述的原因或解释，并能对测试概念进行总结 summarize、比较 compare、分类 classify 和举例 give examples。

行为动词: 分类 classify、比较 compare、对比 contrast、区别 differentiate、区分 distinguish、举例说明 exemplify、解释 explain、举例 give examples、演绎 interpret、总结 summarize

举例说明:

- “对编写验收准则的不同选项进行分类。”
- “比较测试活动中的不同角色”（寻找相似性、差异性或两者）。
- “区分项目风险和产品风险”（可以通过概念区分）。
- “举例说明测试计划的目的和内容。”
- “解释环境对测试过程的影响。”
- “总结评审过程中的活动。”

等级 3: 应用 Apply (K3) - 考生能在遇到通常的任务时执行规程，或者选择正确的规程并应用于给定的环境中。

行为动词: 应用 apply、实施 implement、准备 prepare、使用 use。

举例说明:

- “应用测试用例优先级”（应指规程、技术、过程、算法等）。

- “准备缺陷报告。”
- “使用边界值分析导出测试用例。”

参考资料（针对学习目标的知识认知等级）：

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching

评估： A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon

中国软件测试认证委员会 (CSTQB®)

9. 附录 B - 商业价值与学习目标的可追溯性矩阵

本节列出了与业务价值相关的基础级学习目标的数量，以及基础级业务价值和基础级学习目标之间的可追溯性。

商业价值：基础级		FL-B01	FL-B02	FL-B03	FL-B04	FL-B05	FL-B06	FL-B07	FL-B08	FL-B09	FL-B010	FL-B011	FL-B012	FL-B013	FL-B014
B01	理解什么是测试以及为什么测试是有益的	6													
B02	理解软件测试的基本概念		22												
B03	根据测试环境确定要实施的测试方法和活动			6											
B04	评估和改进文档的质量				9										
B05	提高测试的有效性和效率					20									
B06	将测试过程与软件开发生存周期保持一致						6								
B07	理解测试管理原则							6							
B08	编写和传达清晰易懂的缺陷报告								1						
B09	理解影响测试优先级和工作量的因素									7					
B010	将测试作为跨职能团队的一部分工作										8				
B011	了解与测试自动化相关的风险和收益											1			
B012	确定测试所需的基本技能												5		
B013	理解风险对测试的影响													4	
B014	有效报告测试进度和质量														4

章/节/子节	学习目标	K-等级	商业价值													
			FL-B01	FL-B02	FL-B03	FL-B04	FL-B05	FL-B06	FL-B07	FL-B08	FL-B09	FL-B010	FL-B011	FL-B012	FL-B013	FL-B014
第 1 章	测试基础															
1.1	什么是测试?															
1.1.1	识别典型的测试目的	K1	X													
1.1.2	区分测试与调试的不同	K2		X												
1.2	为什么需要测试?															
1.2.1	举例说明为什么需要测试	K2	X													
1.2.2	回顾测试和质量保证之间的关系	K1		X												
1.2.3	区分根本原因、错误、缺陷和失效	K2		X												
1.3	测试原则															
1.3.1	解释测试的七项原则	K2		X												
1.4	测试活动、测试件和测试角色															
1.4.1	总结不同的测试活动和任务	K2			X											
1.4.2	解释上下文对测试过程的影响	K2			X			X								
1.4.3	区分支持测试活动的测试件	K2			X											
1.4.4	解释维护可追溯性的价值	K2				X	X									
1.4.5	比较测试中的不同角色	K2									X					
1.5	测试中的基本技能和良好实践															
1.5.1	举例说明测试所需的通用技能	K2											X			
1.5.2	回顾“完整团队”方法的优点	K1									X					
1.5.3	区分测试独立性的优点和缺点	K2			X											

章/节/子节	学习目标	K-等级	商业价值													
			FL-B01	FL-B02	FL-B03	FL-B04	FL-B05	FL-B06	FL-B07	FL-B08	FL-B09	FL-B010	FL-B011	FL-B012	FL-B013	FL-B014
第 2 章	测试开发生存周期中的测试															
2.1	软件开发生存周期中的测试															
2.1.1	解释所选择的软件开发生存周期对测试的影响	K2						X								
2.1.2	回顾适用于所有软件开发生存周期的良好测试实践	K1						X								
2.1.3	回顾开发中“测试先行”方法的示例	K1					X									
2.1.4	总结 DevOps 对测试产生的影响	K2					X	X			X	X				
2.1.5	解释左移的方法	K2					X	X								
2.1.6	解释如何使用回顾作为过程改进的机制	K2					X				X					
2.2	测试级别和测试类型															
2.2.1	区分不同的测试级别	K2		X	X											
2.2.2	区分不同的测试类型	K2		X												
2.2.3	区分确认测试和回归测试	K2		X												
2.3	维护测试															
2.3.1	总结维护测试及其触发因素	K2		X					X							
第 3 章	静态测试															
3.1	静态测试基础															
3.1.1	认识可以通过不同静态测试技术检查的产品类型	K1				X	X									
3.1.2	解释静态测试的价值	K2	X			X	X									
3.1.3	比较静态测试与动态测试	K2				X	X									
3.2	反馈和评审过程															

章/节/子节	学习目标	K-等级	商业价值													
			FL-B01	FL-B02	FL-B03	FL-B04	FL-B05	FL-B06	FL-B07	FL-B08	FL-B09	FL-B010	FL-B011	FL-B012	FL-B013	FL-B014
3.2.1	识别与利益相关方早期反馈和频繁反馈的好处	K1	X			X					X					
3.2.2	总结评审过程的活动	K2			X	X										
3.2.3	回顾执行评审时主要角色承担的责任	K1				X							X			
3.2.4	比较不同的评审类型	K2		X												
3.2.5	回顾成功评审的因素	K1					X						X			
第4章	测试分析和设计															
4.1	测试技术概述															
4.1.1	区分黑盒、白盒和基于经验的测试技术	K2		X												
4.2	黑盒测试技术															
4.2.1	使用等价类划分生成测试用例	K3					X									
4.2.2	使用边界值分析生成测试用例	K3					X									
4.2.3	使用判定表测试生成测试用例	K3					X									
4.2.4	使用状态转移测试生成测试用例	K3					X									
4.3	白盒测试技术															
4.3.1	解释语句测试	K2		X												
4.3.2	解释分支测试	K2		X												
4.3.3	解释白盒测试的价值	K2	X	X												
4.4	基于经验的测试技术															
4.4.1	解释错误猜测法	K2		X												
4.4.2	解释探索性测试	K2		X												

章/节/子节	学习目标	K-等级	商业价值													
			FL-B01	FL-B02	FL-B03	FL-B04	FL-B05	FL-B06	FL-B07	FL-B08	FL-B09	FL-B010	FL-B011	FL-B012	FL-B013	FL-B014
4.4.3	解释基于检查表的测试	K2		X												
4.5	基于协作的测试方法															
4.5.1	解释如何与开发人员和业务代表合作编写用户故事	K2				X						X				
4.5.2	对编写验收准则的不同选项进行分类	K2										X				
4.5.3	使用验收测试驱动开发（ATDD）生成测试用例	K3					X									
第5章	管理测试活动															
5.1	测试规划															
5.1.1	举例说明测试计划的目的是内容	K2		X						X						
5.1.2	认识测试人员如何为迭代和发布规划增加价值	K1	X									X		X		
5.1.3	比较入口准则和出口准则	K2				X		X								X
5.1.4	使用估算技术计算所需的测试工作量	K3							X		X					
5.1.5	应用测试用例优先级	K3							X		X					
5.1.6	回顾测试金字塔的概念	K1		X												
5.1.7	总结测试象限及其与测试级别和测试类型的关系	K2		X							X					
5.2	风险管理															
5.2.1	利用风险可能性和风险影响识别风险级别	K1							X						X	
5.2.2	区分项目风险和产品风险	K2		X											X	
5.2.3	解释产品风险分析如何影响测试的充分性和范围	K2					X				X				X	
5.2.4	解释可以采取哪些措施分析产品风险	K2		X			X								X	
5.3	测试监测、测试控制和测试完成															

章/节/子节	学习目标	K-等级	商业价值													
			FL-B01	FL-B02	FL-B03	FL-B04	FL-B05	FL-B06	FL-B07	FL-B08	FL-B09	FL-B010	FL-B011	FL-B012	FL-B013	FL-B014
5.3.1	回顾用于测试的度量	K1								X						X
5.3.2	总结测试报告的目的、内容和受众	K2					X			X						X
5.3.3	举例说明如何沟通测试状	K2											X			X
5.4	配置管理															
5.4.1	总结配置管理如何支持测试	K2					X		X							
5.5	缺陷管理															
5.5.1	准备缺陷报告	K3		X						X						
第6章	测试工具															
6.1	测试活动中的工具支持															
6.1.1	解释不同类型的测试工具如何支持测试	K2					X									
6.2	测试自动化的收益和风险															
6.2.1	回顾测试自动化的收益和风险	K1					X						X			

10. 附录 C - 发布说明

ISTQB®基础级大纲 v4.0 是基于基础级大纲（v3.1.1）和敏捷测试人员 2014 大纲基础上的重大更新。因此，没有针对每章节的详细发布说明。下面给出本大纲的主要变化汇总。此外，在一份单独的发布说明文件中，ISTQB®提供了基础级大纲 3.1.1 版、2014 版敏捷测试人员大纲中的学习目标（LO）与新基础级 V4.0 大纲中学习目标之间的可追溯性，给出了已被添加、更新或删除的那些学习目标（LO）。

在编写此大纲时（2022-2023 年）已有 100 多个国家的 100 多万人参加了 ISTQB®基础水平考试，全世界有 80 多万人是 ISTQB®认证测试工程师。期望他们都已经阅读了 ISTQB®的基础级大纲，能够通过考试，这使得基础级大纲可能成为有史以来阅读量最大的软件测试文档！这一重大更新是针对这一传承进行的，旨在提升数十万人对 ISTQB®向全世界测试社区所发布的（大纲等资料）质量水准的认可度。

在这个版本中，所有学习目标都经过了编辑后使其成为（不可再细分的）原子，并在学习目标和大纲部分之间创建了一对一的可追溯性，因此没有学习目标也就没有内容。目标是使这个版本更容易阅读、理解、学习和翻译，重点是提高实用性以及知识和技能之间的平衡。

此版本进行了以下重要更改：

- 整体大纲的缩减。大纲不是一本教科书，而是一份文件，用来概述软件测试基础课程的基本要素，包括应该涵盖的主题和级别。因此，特别是：
 - 在大多数情况下，实例会被排除在大纲文本之外。培训机构的任务是在培训期间提供实例和练习。
 - 遵循了“大纲写作检查表”，该检查表建议了每个 K 级学习目标（Los）的最大文本大小规模（K1=最多 10 行，K2=最多 15 行，K3=最多 25 行）。
- 与基础级大纲 FL v3.1.1 和敏捷测试员大纲 v2014 相比，减少了学习目标 LOs 的数量
 - 在基础级大纲 FL v3.1.1 (15) 和敏捷测试员大纲 AT2014 (6) 共有 21 个 K1 的学习目标，而在此版本中有 14 个 K1 学习目标。
 - 在基础级大纲 FL v3.1.1 (40) 和敏捷测试员大纲 AT2014 (13) 共有 53 个 K2 的学习目标，而在此版本中有 42 个 K2 学习目标。
 - 在基础级大纲 FL v3.1.1 (7) 和敏捷测试员大纲 AT2014 (8) 共有 15 个 K3 的学习目标，而在此版本中有 8 个 K3 学习目标。
- 提供了更多关于软件测试和相关主题的经典和/或受人尊敬的书籍和文章的参考资料
- 第 1 章（测试基础）中的主要变化
 - 扩大和改进了测试技能部分。
 - 增加了关于完整团队方法的章节（K1）。

- 关于测试独立性的章节从第 5 章移至第 1 章。
- 第 2 章（在软件开发生存周期中的测试）中的主要变化
 - 改写和改进第 2.1.1 章节和第 2.1.2 章节，修改了相应的学习目标 Los。
 - 更多地关注实践，如：测试先行的方法（K1）、测试左移（K2）、回顾（K2）。
 - 关于 DevOps 背景下测试的新章节（K2）。
 - 集成测试级别分为两个独立的测试级别：组件集成测试和系统集成测试。
- 在第 3 章（静态测试）中的主要变化
 - 删除了评审技术部分以及相关 K3 学习目标（应用评审技术）。
- 第 4 章（测试分析和设计）中的主要变更
 - 删除了用例测试（但仍存在于高级测试分析师教学大纲中）。
 - 更多地关注基于协作的测试方法：关于使用 ATDD 导出测试用例的新 K3 L0 和关于用户故事和验收准则的两个新 K2 L0。
 - 判定测试和覆盖被分支测试和覆盖所取代（首先，分支覆盖在实践中更常用；其次，不同的标准对判定的定义不同，而不是“分支”；第三，这解决了旧 FL2018 中一个微妙但严重的缺陷，该缺陷声称“100%的判定覆盖意味着 100%的语句覆盖”——这句话在程序中没有判定项时是不正确的）。
 - 关于白盒测试价值这节有所改进。
- 第 5 章（管理测试活动）中的主要变更
 - 删除了关于测试策略/测试方法的部分。
 - 用于估算测试工作量的估算技术上的新 K3 L0。
 - 更多地关注测试管理中众所周知的敏捷相关概念和工具：迭代和发布计划（K1）、测试金字塔（K1）和测试象限（K2）。
 - 关于风险管理的章节通过描述四项主要活动来更好地构建：风险识别、风险评估、风险缓解和风险监测。
- 第 6 章（测试工具）中的主要变更
 - 减少了一些关于测试自动化问题的内容，因为对于基础级来说过于超前——删除了关于工具选择、执行试点项目和将工具引入组织的部分。