认证测试工程师

高级大纲

敏捷技术测试（ ATT）

版本 1.1

国际软件测试认证委员会

版权声明

修订历史

版本	日期	备注
大纲 0.1版	2017年1月11日	独立部分
大纲 0.2 版	2017年5月24日	纳入工作组对 0.1 版的评审意见
大纲 0.3 版		纳入工作组对 0.2 版的评审意见
大纲 0.7 版		纳入对 0.3 版的 Alpha 版评审意见
大纲 0.71 版		工作组对 0.7 版进行了更新
大纲 0.9 版	2017年12月30日	Alpha 版候选
大纲 0.91 版	2018年1月6日	Alpha版发布
大纲 0.98 版	2019年1月12日	Beta 版候选
大纲 0.99 版		Beta版发布
大纲 2018 版		GA版
大纲 2019 版	2019年2月12日	文案略有改动 增加了几个学习目标（K1，K2级） 增加了版权信息 增加了所有章节的学习时间
大纲 2019 版	2019年3月14日	为适应官方简称“ATT”，更改了一些学习目标 修改了『致谢』章节
大纲 2019 版	2019年5月6日	一些小改动
大纲 2019 版	2019年6月7日	微小变化
大纲 2019 版	2019年7月8日	修订变更
大纲 2019 版	2019年7月10日	准备接受 Beta 版评审
大纲 2019 版	2019年9月14日	Beta 版评审后的改进和更正
大纲 2019 版	2019年11月14日	发布日期
大纲 2019 版	2019年12月9日	更改了 ISTQB® 标志 改变了K级培训时长（第0.6 章节）

目录

致谢

本文档由国际软件测试认证委员会敏捷工作组的一个团队编写，这个团队由Rex Black（主席）、Michael Pilaeten（副主席兼代理主席）和RenzoCerquozzi（产品所有者）领导。

高级敏捷团队感谢评审小组和各成员国委员会的建议和投入。

在高级敏捷技术测试工程师课程大纲完成时，工作组有以下成员：MichaelPilaeten（代理主席）、Renzo Cerquozzi（产品所有者）、AlonLinetzki（营销工作组）、Leanne Howard（词汇工作组）和KlausSkafte（考试工作组）。

作者：Leo van der Aalst，RenzoCerquozzi，BertrandCornanguer，IstvanForgacs，JaniHaukinen，NoamKfir，SammyKolluru，AlonLinetzki，TiloLinz，MichaelPilaeten，Marie Walsh。

内部评审员：MichaelArefi，VojtěchBarta，Renzo Cerquozzi， Graham Bath，Laurent Bouhier， Anders Claesson，Alessandro Collino，David Janota，David Evans，Leanne Howard，Matthias Hamburg，Kari Kakkonen，Tor KjetilMoseng，MeilePosthuma，Salvatore Reale，Marko Rytkönen,，Sarah Savoy，Klaus Skafte，Mike Smith，Chris van Bael。

例题的编写与评审：ArminBorn，RenzoCerquozzi，AlonLinetzki，TiloLinz，JamieMitchell，JaniHaukinen，Tobias Horn，ChrisvanBael，SuruchiVarshney。

本编写团队还感谢下列人士，他们来自各成员国委员会和敏捷专家团体，参加了对基础级敏捷扩展大纲的评审和投票，或提供了宝贵的意见和建议：Adam Roman，ArminBeer，BeataKarpinska，Chris Van Bael，ErwinEngelsma，GiancarloTomasig，GaryMogyorodi，IngvarNordström，JanaGierloff，JörnMünzel，Jurian van de Laar，KariKakkonen，LaurentBouhier，MarkoRytkönen，MartinKlonk，MatthiasHamburg，MeilePosthuma，PaulWeymouth，R. Green，RichardSeidl，RikMarselis，StephanieUlrich，Stephanie van Dijck，TalPe’er，TiloLinz，VeronicaSeghieri和 WimDecoutere。

特别感谢 ISTQB® 秘书长GalitZucker的指导和支持。

本文档于2019 年 10月18日由 ISTQB® 大会正式批准发布。

高级-敏捷技术测试工程师大纲中文翻译参与者（按姓氏拼音排序）：
胡迎、春罗放

高级-敏捷技术测试工程师大纲QA评审参与者（按姓氏拼音排序）：
翟宏宝（QA评审组长）、梁静、陶显锋、王丽娟

致谢企业：上海企顺信息系统有限公司

0. 本课程简介

0.1本文档的目的

这一课程大纲针对“敏捷技术测试工程师”的高级阶段提供了国际软件测试认证的基础。ISTQB® 按如下方式提供本课程大纲：

- 提供给各成员国委员会，翻译成当地语言，并授权给培训提供者。各成员国委员会可根据其特定语言需要调整课程大纲，并增加参考资料，以便符合当地出版物的需要。
- 提供给认证机构，根据本课程大纲学习目标，设计当地语言的考试试题。
- 提供给培训提供者，制作课件并确定合适的教学方法。
- 提供给希望获得认证的考生，帮助他们准备认证考试（作为培训课程的一部分，或独立参加考试）。
- 提供给国际软件和系统工程组织，既可以推进软件和系统测试的专业技术，也可以将本大纲作为图书和文章的基础。

ISTQB® 允许其他实体将本课程大纲用于其他目的，条件是他们事先寻求并获得 ISTQB® 的书面许可。

0.2 概述

《高级敏捷技术测试工程师概述》文档 [ISTQB_ATT_OVIEW]包括以下信息：

- 课程大纲的业务成果
- 课程大纲摘要
- 课程大纲之间的关系
- 认知级别的描述（K 级）
- 附录

0.3考核的学习目标

学习目标不仅最终会在业务上转化为成果，而且还可以设计用于实现高级敏捷技术测试工程师认证的考试。一般来说，这个课程大纲的所有部分都可以在 K1 级考核。也就是说，希望参加认证的用户要识别、记忆和牢记一个术语或概念。K2、K3 和 K4 级的学习目标在相关章节的开头显示。

0.4 高级敏捷技术测试工程师证书考试

高级敏捷技术测试工程师证书考试将以本大纲为基础。对考试问题的回答可能需要使用基于本课程大纲不止一节的材料。除了简介和附录部分，课程大纲的所有章节都是考核内容。标准、书籍和其他 ISTQB® 大纲列为参考资料，但其内容不在此考试的考核范围内，除非是本课程大纲本身从这些标准、书籍和 ISTQB® 大纲中总结出来的内容。

考试格式为多选题。

考试既可以作为认证培训课程的一部分，也可以独立参加（例如，在考试中心或公开考试）。参加考试之前不一定要完成认证的培训课程。

0.5 授权

只要培训提供者的课程材料遵循本课程大纲，ISTQB® 成员国委员会可以授权这些培训提供者。

培训提供者应从执行认证的成员国委员会或机构获得认证指南。已经获得认证的课程视为符合本课程大纲，并且在课程中允许进行 ISTQB® 考试。

0.6 课程大纲的结构

本大纲包含四章考试内容。

每一章的顶层标题指定各章的总时间；在章级别以下没有指定时间。

对于认证的培训课程，课程大纲要求至少 16 小时的教学，分布在以下各章：

- 第一章：180 分钟　　需求工程
- 第二章：540 分钟　　敏捷测试
- 第三章：135 分钟　　测试自动化
- 第四章：105 分钟　　部署和交付

以下给出完成每个学习目标的最短时间：

K2：　15 分钟
K3：　60 分钟
K4：　75 分钟

1. 需求工程180分钟

关键词

验收准则（Acceptance criteria），史诗（Epic），用户故事（user story）

需求工程的学习目标

ATT-1.x （K1）关键词

1.1需求工程技术

ATT-1.1.1-1 （K4）使用需求工程技术分析用户故事和史诗（Epic）

ATT-1.1.1-2 （K2）描述需求工程技术及其如何帮助测试员

ATT-1.1.2-1 （K4）针对给定的用户故事，使用需求工程和测试技术创建并评估可测试的验收准则

ATT-1.1.2-2 （K2）描述启发技术

1.1 需求工程技术

通过应用需求工程技术，敏捷团队可以改进用户故事（参见 [AgileFoundationExt]）和 Epic（参见 [AgileFoundationExt]），丰富上下文，考量影响和依赖关系，并且可以发现差距，如缺少非功能需求。

虽然本节讨论的大多数需求工程技术来自传统的开发方法，但它们在敏捷开发中也是有效的。

一般来说，在传统项目中，需求工程活动和技术是正规化的，按顺序进行，并由指定的人员负责，如业务分析员、功能分析员、技术设计师、企业设计师和过程分析员。相反，在敏捷项目中，需求工程技术通过一种不那么正式的方法在整个项目中和每次迭代中得到应用。这些需求工程技术由所有敏捷团队成员更频繁地执行，使用连续的反馈周期，而不仅仅由团队的专门业务分析师或产品所有者执行。

1.1.1 使用需求工程技术分析用户故事和史诗（Epic）

作为一名测试员，为了能够有助于清楚描述（并可能改进）用户故事、Epic 和其他敏捷需求，必须了解、理解、选择和使用支持这些需求的各种需求工程技术。

这些技术的示例包括故事板、故事地图、人物角色、图表和用例。

故事板：故事板（不要与敏捷任务板或敏捷用户故事板相混淆）提供系统的可视化表示形式。故事板帮助测试员：

- 看到用户故事背后的思想过程，以及整体的“故事”，提供上下文，使人们能够快速地看到系统的功能流程，并发现逻辑上存在的任何差距。
- 以可视化形式描述与系统公共区域（主题）相关的用户故事组，这些用户故事可以考虑包含在同一个迭代中，因为它们可能会触及同一段代码。
- 帮助建立故事地图，在产品待办列表（Backlog）中划分 Epic 和相关用户故事的优先级。
- 帮助确定用户故事和 Epic 的验收准则。
- 根据系统设计的可视化表示，帮助选择正确的测试方法。
- 与故事地图一起，帮助划分测试的优先顺序，并确定桩、驱动器和/或模拟（Stub、Driver 和 Mock）的需求。

故事地图：故事地图（或用户故事地图）是一种使用两个独立维度来排列用户故事的技术。地图的水平轴表示每个用户故事的优先级顺序，而垂直轴表示实现的复杂性。使用故事地图可以帮助测试员：

- 确定系统的最基本功能，快速执行冒烟测试。
- 通过确定功能的顺序，划分测试的优先级。
- 以可视化方式描述系统的范围。
- 确定每个用户故事的风险级别。

角色：角色用来定义虚构的人物或原型，说明典型用户将如何与系统交互。使用角色可以帮助测试员：

- 通过识别可能使用系统的不同类型的用户，发现用户故事中的差距。
- 通过在用户故事中将特定类型用户与其他用户相比较，识别他们使用系统的方式有什么不一致。
- 明确用户故事验收准则
- 在探索性测试中发现额外的测试路径
- 揭示测试条件，特别是与特定用户组相关的条件，从而帮助确保足够的用户组覆盖并对用户组之间的差异进行充分的测试。

图表：实体关系图、类图和其他 UML 图等各种图表可以显示数据的结构或数据流以及系统的功能属性或行为，可以用来发现系统功能上的差距。

用例：用例（图表和规格说明）[Foundation]可以帮助测试员：

- 确保用户故事是可测试的并且大小适当。
- 确定用户故事是否需要细化或分解。
- 揭示被遗忘的利益相关者。
- 确定在测试设计时应该考虑的接口和集成点。
- 查看 Epic 和用户故事之间的关系，以便确保 Epic 没有丢失任何用户故事。

1.1.2 使用需求工程和测试技术确定验收准则

需求工程是一个由以下步骤组成的过程：

- 启发：发现、理解、记录和评审用户对系统的需求和约束的过程。启发技术应当用来推导、评估和提高验收准则。
- 文档：清晰、准确地记录用户需求和约束的过程。用户故事和验收准则的记录级别应当达到团队敏捷宣言原则所要求的高度。归档的类型取决于团队和利益相关者采取的归档方法。验收准则可以使用自然语言、模型（例如状态转换图）或示例来记录。
- 协商和确认：对于每个用户故事，多个利益相关者可能有不同的见解或偏好。由于这些见解和偏好可能是不一致的，甚至是相互冲突的，因此每个利益相关者的验收准则也是如此。这些冲突中的每一个都应在所有受影响的利益相关者之间确定、协商和解决。每一个被遗忘或未解决的冲突都可能危及项目的成功。在此步骤结束时，每个用户故事的内容都由受影响的利益相关者验证，例如，验证为“准备就绪的定义”（Definition of Ready）。
- 管理：随着项目的进展，意见和情况可能会改变。即使验收准则得到适当的启发、记录、协商和验证，验收准则仍然可能发生变化。由于可能发生更改，应该使用良好的配置和变更管理流程来管理用户故事。

为了确定验收准则，测试员可以使用多种启发技术，包括：

- 定量问卷：使用从封闭式问题中提取的定量数据是在各个数据点之间进行清晰比较的一种很好的方法。这通常会提供数值数据，可以包括在验收准则的数值结论中。定量问卷可作为大量利益相关者的启发技术，特别是用于非功能性验收准则。
- 定性问卷：开放式问题是增加定量研究质量的一种非常有效的方法。开放式问题最好作为关键问题的后续。这可能产生额外的信息，为此必须创建新的用户故事，或必须用这些信息补充现有的用户故事。定性问卷可以作为少数利益相关者的启发技术 – 因为处理需要更多的时间 – 并且适合于功能性验收准则。
- 定性访谈：定性访谈比定量查询更灵活，主要用于获取关于背景、上下文和原因的信息。它不太可能返回硬数据，但可以从对用户故事上下文的响应中得出验收准则。定性访谈可以是任何类型问卷的后续，用于深化得出的验收准则。

在观察技术（例如学徒式）、创造力技术（例如六顶思考帽）和辅助技术（例如低保真原型）的范畴中，还存在多种其他的启发技术。测试员的技术工具集将影响启发的验收准则的质量。

INVEST 和 SMART [INVEST] 以及等价划分、边界值分析、判定表和状态转换测试[Foundation]等测试技术也可用于识别和评估验收准则。

2. 敏捷测试540分钟

关键词

测试驱动开发（Test-driven development），行为驱动开发（behavior-driven development），验收测试驱动开发（acceptance test-driven development），实例化需求（specification by example），测试章程（test charter）

敏捷测试的学习目标

ATT-2.x　(K1)关键词

2.2 敏捷开发和测试技术

ATT-2.1.1-1　(K3)在敏捷项目中为给定示例的上下文应用测试驱动开发（TDD）

ATT-2.1.1-2　(K2)了解单元测试的特点

ATT-2.1.1-3　(K2)理解助记词 FIRST的含义

ATT-2.1.2-1　(K3)在敏捷项目中为特定用户故事的上下文应用行为驱动开发（BDD）

ATT-2.1.2-2　(K2)了解如何管理用于制定方案的准则

ATT-2.1.3　(K4)分析敏捷项目中的产品待办列表，以便确定引入验收测试驱动开发（ATDD）的方法

2.3 基于经验的敏捷测试

ATT-2.2.1-1　(K4)针对使用其他方法（包括基于风险的测试）为敏捷项目中的特定场景创建的测试，使用测试自动化、基于经验的测试和黑盒测试分析对测试方法的创建。

ATT-2.2.1-2　(K2)解释关键系统和非关键系统之间的差异

ATT-2.2.2-1　(K4)分析用户故事和 Epic，以便创建测试章程

ATT-2.2.2-2　(K2)了解基于使用经验的技术

2.1 代码质量

ATT-2.3.1-1　(K2)了解在敏捷项目中重构测试用例的重要性

ATT-2.3.1-2　(K2)了解重构测试用例的实际任务列表

ATT-2.3.2-1　(K4)在代码评审过程中分析代码，从而识别缺陷和技术债务

ATT-2.3.2-2　(K2)了解静态代码分析

2.1 敏捷软件开发和测试技术

在软件开发中，缺陷可能是由于代码写得欠佳和无法满足客户需求而产生的。敏捷开发技术通过应用测试驱动开发（TDD）、行为驱动开发（BDD）和验收测试驱动开发（ATDD）的概念来处理这些问题。TDD是一种提高软件产品质量的技术，而 BDD 和 ATDD 有助于提高其使用质量（特征和功能）。

2.1.1 测试驱动开发（TDD）

测试驱动开发是一种软件开发方法，它将设计、测试和编码结合在一个快速的迭代过程中。TDD采用了一种严格的测试优先方法，即在编码某种功能之前，创建一个表达代码预期功能的测试。在编写代码之前运行测试，检查测试会失败。一旦编写好代码，测试将再次执行，并应当会通过。

通常认为TDD 是第一个主要的测试优先编程方法，其他方法，如行为驱动开发（BDD）、验收测试驱动开发（ATDD）和实例化需求（SBE），都是从这种方法中派生出来的。

TDD为软件开发提供了一种渐进的方法，它将设计视为一个持续的过程。每次迭代对生产代码都构成了一个小的变化，这为开发人员轻微改进设计提供了机会。随着变更和深思熟虑的设计决策日积月累，将会诞生一个久经测试锤炼的、健壮的软件设计。

TDD从业人员使用多种做法和技术编写高质量的代码，避免技术债务的积累，包括：

- 单元测试和规范的测试优先方法
- 短迭代周期
- 即时和经常反馈
- 有效使用工具、源代码控制和持续集成
- 指导应用各种编程原则和设计模式

TDD从业人员编写单元测试以检查生产代码的预期行为。单元测试在特定条件下执行一个函数，并检查它是否产生预期结果。期望值是用断言来描述的，这些断言检查系统状态是否按预期变化，或者是否显示了特定的行为。例如，断言可以检查：

- 函数执行计算并返回预期结果
- 函数以特定的方式修改系统的状态
- 函数以特定的方式调用另一个函数

单元测试应该便于开发人员实现和维护。它们是自动化的，通常使用与生产代码相同的语言编写。单元测试应具有以下特点：

- 确定性 - 每次单元测试在相同的条件下运行，都应该产生相同的结果。
- 原子性 - 单元测试应该只测试与之相关的功能
- 孤立性 - 单元测试应该尽量只测试它最初打算使用的特定代码。单元测试不应相互依赖，应尽可能避免生产代码中的依赖。
- 快速性 - 单元测试应是小型快速的，以便可以提供即时反馈。应该可以在短时间内运行许多单元测试。

描述一个好的单元测试的另一个助记符是FIRST：Fast, Isolated, Repeatable, Self-Validating, Thorough（快速、孤立、可重复、自我验证、彻底）的首字母缩写。

有许多单元测试框架可用于许多不同的语言。它们的API、方法和术语不同，但都有一些共同的要素。无论语言或单元测试框架如何，单元测试通常遵循这三步模式：

- 安排/设置 - 准备执行环境。此步骤可以实例化对象，初始化系统状态，并根据需要注入数据。
- 操作 -执行正在测试的操作，然后检查操作产生的结果。此必需步骤可能只包含一行代码， 触发正在测试的操作。
- 断言 - 执行对预期输出或其他后置条件的实际检查。

迭代过程是TDD的基石。每次迭代的目的是对程序进行一个小的、集中的、仔细考虑的更改。按照颜色编码的惯例，迭代周期通常称为“红-绿-重构”周期：

- 红 - 编写一个失败的测试，描述一个未实现的期望。运行测试，确保它会失败。
- 绿 - 编写只满足测试描述的期望并使测试通过的生产代码。运行所有相关测试，确保全部通过。如果有任何失败，进行必要的更改，使所有测试通过。
- 重构 - 改进测试代码和生产代码的设计和结构，而不改变功能，同时确保所有相关测试继续通过。开发人员可以应用一系列重构来更改代码，而不改变行为，直到代码优化为止。在现代，大多数集成开发环境（IDE）和许多代码编辑器都可以自动和安全地应用重构技术。

2.1.2 行为驱动开发（BDD）

根据ISTQB®敏捷基础课程的大纲，行为驱动开发（BDD）是这样一种技术：开发人员、测试员和业务代表共同分析软件系统的需求，使用共享语言制定需求，并自动验证这些需求。

BDD深受两个不同规范的影响：测试驱动开发（TDD）和领域驱动设计（DDD）[Evans03]。它包含了这两个规范的许多核心原则，包括协作、设计、通用语言、自动化测试执行、短反馈周期等。TDD依赖于单元测试来验证实现细节，而BDD则依赖于可执行场景来验证行为等内容。BDD一般遵循规定的途径：

- 协作创建用户故事
- 将用户故事制定为可执行的场景和可验证的行为
- 实现行为，并通过执行场景来验证这些行为

应用BDD的团队从每个用户故事中提取一个或多个场景，然后将它们制定为自动化测试。一个场景表示特定条件下的单个行为。

场景通常基于用户故事的验收准则、示例和用例。编写BDD场景所使用的语言应该是所有团队成员（无论是技术成员还是非技术成员）都可以理解的。

因此，人们强烈倾向于使用自然语言（例如英语）来表达场景。此外，应用BDD的团队建立了一种通用语言[Evans03]，它基本上构成了对团队中的每个人都清楚和明确的术语，并且在任何地方都使用。

BDD方案通常由三个主要部分组成[Gherkin]：

- 条件（Given） - 描述行为触发之前的环境状态（先决条件）
- 如果（When） - 描述触发行为的动作
- 那么（Then） - 描述行为的预期结果

从用户故事中提取场景涉及：

- 识别用户故事指定的所有验收准则，并为每条准则都编写场景。有些准则可能需要多种场景。
- 识别功能用例和示例，并为每个用例和示例编写场景。许多用例和示例都是有条件的，因此识别每个条件至关重要。
- 在探索性测试时创建场景，帮助识别相关的现有行为、潜在的冲突行为、最小状态、替代流程等。
- 寻找重复使用的步骤或步骤组，避免重复工作。
- 识别需要随机数据或合成数据的区域。
- 识别需要模拟（mock）、桩（stub）或驱动器（driver）的步骤，以便保持隔离并避免执行集成或可能是昂贵的进程。
- 确保场景的原子性，不影响彼此的状态（孤立性）。
- 决定是根据“每个测试只检查一项”的原则将When节限制为一步，还是为了其他考虑（如测试执行速度）进行优化。

制定场景时，建议使用的准则包括：

- 场景应该从特定用户的角度描述系统支持的特定行为。
- 在描述步骤（Given, When, Then）时，场景应使用第三人的身份从用户的角度描述状态和交互。
- 场景应该是孤立的和原子的，这样它们就可以按任何顺序运行，而不会相互影响或相互依赖。Given 步骤应该使系统处于必要的状态，以便 When 步骤始终按预期执行。
- When 步骤应该描述用户执行的语义操作，而不是特定的技术操作，除非特别需要测试特定的操作。例如，“用户确认订单”（语义动作）通常优于“用户单击确认按钮”（技术动作），除非该按钮本身必须进行测试。
- Then 步骤应该描述具体的观察结果或状态，而不应指定通用的成功或错误状态。

2.1.3 验收测试驱动开发（ATDD）

验收测试驱动开发（ATDD）通过协调软件项目，根据客户的需求进行交付。验收测试是对系统的期望行为和功能的说明。用户故事代表了对客户有价值的功能。验收测试验证此功能是否正确实现。开发人员将此用户故事拆分为实现该功能所需的一组任务。开发人员可以使用TDD实现这些任务。当一个给定的任务完成后，开发人员将继续下一个任务，以此类推，直到用户故事完成，这是由验收测试成功执行来指示的。BDD和ATDD都是专注于客户，而TDD是专注于开发人员。

BDD 和 ATDD 相似，因为它们都产生了相同的结果：对要构建什么和如何正确构建的理解相同。BDD 是使用上述 Given/When/Then 语法来编写测试用例（例如，验收测试）的结构化方法。

ATDD 将测试意图与测试实现分离，测试意图是以人类可读的格式（如纯文本）表示，测试实现最好是自动化的。这种重要的分离意味着，面向业务的团队成员和其他非技术的团队成员，如产品所有者、分析师和测试员，可以主动地描述可测试的示例（或验收测试），从而在推动开发方面发挥积极的作用。不同角色和观点的利益相关者，如客户、开发人员和测试员，共同讨论潜在的用户故事（或其他形式的需求）以达成对要解决的问题的共同理解；相互提出关于功能的开放式问题；探索所需行为的具体、可测试的示例。

商定的示例（以及导致这些示例的讨论）本身就是有价值的产出，因此必须记住，虽然它们也可以作为验收测试自动化，但这样做并不总是必要或具有成本效益。这里有意地强调了有价值的示例而不是测试，是鼓励团队所有成员参与发现过程的一个重要技巧。

这说明了敏捷测试员在这种情况下的重要作用。在讨论过程中，测试员可能会考虑测试技术，如等价划分、边界值分析等。他们可以通过使用这种分析方法，鼓励向产品所有者询问在哪里可以找到关注的行为和边缘案例。其意图应始终是鼓励整个小组考虑并找到关键的示例。提供一个关注的输入组合，并询问该小组是否同意预期的产出，这是探索潜在的不确定性或不完全分析领域的好方法。

实例化需求（即 SBE）是指一组有用的模式，允许敏捷团队发现、讨论和确认业务利益相关者所需的重要结果以及实现这些结果所需的软件行为。该术语已广泛用于包括和扩展“验收测试驱动开发（ATDD）”这个术语的含义。

2.2 基于经验的敏捷测试

敏捷项目包含各种特征，如方法、迭代长度、适用的测试级别、项目和产品的风险级别、需求质量、团队成员的经验/专长水平、项目组织等，这些特征将影响敏捷项目中自动化测试、探索性测试和手工黑盒测试之间的平衡关系。

2.2.1 结合基于经验的技术和黑盒测试

在风险分析期间，会为单个系统特征和功能确定风险级别（例如，高、中、低）。接着，针对特定风险级别，在自动化测试、探索性测试和手工黑盒测试之间找到正确的组合和平衡。下面是一个描述这个想法的表格。在该表格中，风险级别垂直列出，三种测试方法水平列出。下列符号用于描述：

++ （强烈建议）
+ （建议）
o （中性）
- （不建议）
-- （不使用）

下表中的示例演示了可用于安全关键系统的不同测试技术（探索性、自动化和手工）的混合。这个表格也可应用于其他具体项目：

风险级别	自动化测试	探索性测试	黑盒测试
高	++	+	++
中	+	+	+
低	o	++	+

表1：任务和/或安全关键系统

表 1 的第一行指出，在这种情况下，除了探索性测试方法外，还将强烈建议将自动化测试和黑盒测试相结合。是否采用自动化的决定也会受到许多其他因素的影响。

以下示例演示了在非安全关键系统中使用不同测试技术的组合：

风险级别	自动化测试	探索性测试	黑盒测试
高	+	++	+
中	o	++	o
低	--	++	--

表2：非任务和/或安全关键系统

上面表 2 的最后一行指出，在这种情况下，强烈建议采用探索性测试方法，而其他方法可能不会使用。

在任何情况下（不管是不是安全关键系统），特定的混合关系将取决于项目的特征。

2.2.2 创建测试章程并解释其结果

在创建适当的测试章程之前，应该首先评估现有的 Epic 和用户故事。请参见第一章了解相关技术。

在分析 Epic 或用户故事以创建测试章程时，应考虑以下几点：

- 这个 Epic或用户故事中的用户是谁？
- Epic或用户故事的主要功能是什么？
- 用户可以执行哪些操作？（可从用户故事定义的验收准则列表中获取）
- 一旦特征或功能完成，用户故事的目标是否实现？或是否有其他测试任务影响工作完成标准（Definition of Done，DoD）？

测试章程的粒度很重要。它不应该分得过细，因为它必须围绕一个确定的问题（反应，回归）探索一个区域，或围绕一个用户故事或 Epic（前摄，揭示）探索一个区域。

它也不应该分得过粗略，因为它应该适合 60 到 120 分钟的时间框。运行探索性测试会话的目标是帮助我们针对一个现象、缺陷聚集区域等做出高质量的决策。这一探索的结果应该提供足够的信息来作出上述决定。

创建测试章程可以使用翻页板、电子表格、文档、现有的测试管理系统、角色、思维导图和“完整团队”方法。探索性测试员使用启发式方法来驱动他们在编写和执行探索性测试方面的创造力。启发式方法也可以用来创建测试章程，并在分析用户故事和 Epic 时创造性地思考。启发式方法的示例可以在[Whittaker09]和[Hendrickson13]中找到。

在探索性测试中发现的所有结果都应记录在案。探索性测试的结果应该提供对测试设计更好的见解，对测试产品的想法，以及对任何进一步改进的想法。在探索性测试中应记录的结果包括缺陷、想法、问题、改进建议等等。

记录探索性测试会话时可以使用工具，包括视频采集和日志记录工具、规划工具等。记录的文档应包括预期结果。在某些情况下，笔和纸就足够了，这取决于收集信息的数量。

在总结探索性测试过程时，汇报会议期间将收集和汇总信息，以便显示测试过程的进展、覆盖范围和效率状况。总结的信息可作为管理报告使用，也可用于任何级别和任何规模（单个团队、多个团队和大规模敏捷实现）的回顾性会议。然而，确定与探索性测试过程相关的准确测试度量可能相当具有挑战性。

2.3 代码质量方面的注意事项

技术债务的控制在敏捷项目中非常重要，特别是在整个发布过程中保持高水平的代码质量方面。为了实现这一目标，采用了各种技术。

2.3.1 重构

重构是一种以高效和受控的方式清理代码的方法，它澄清和简化现有代码和测试用例的设计，而不改变其行为。在敏捷项目中，迭代很短，这为所有团队成员创建了一个简短的反馈周期。短的迭代也给试图实现足够覆盖的测试员带来了挑战。由于迭代的性质，功能不断增长，特征随着时间的推移而增加和增强，在早期迭代中为特征编写的测试用例往往需要维护，甚至需要在以后的迭代中完全重新设计。通过使用渐进式的测试设计方法，对测试的更新和重构可以弥补特征变化带来的影响，并确保测试与产品的功能保持一致。

在理解用户故事并为每个用户故事编写验收准则后，可以分析当前迭代的功能对现有回归测试（手工和自动）的影响，并可能需要重构和/或增强测试。随着迭代一个接着一个进行，团队大量地维护和扩展代码；而如果没有连续的重构，这很难做到。

测试用例的重构可以如下执行：

- 识别：通过评审或因果分析来识别需要重构的现有测试。
- 分析：分析改变的测试对整体回归测试集的影响。
- 重构：在不改变可见行为的情况下，改变测试的内部结构，使它更易于理解且修改成本更低。
- 重新运行：重新运行测试，检查其结果，并在相关情况下记录缺陷。重构不应影响测试执行的结果。
- 评估：检查重新运行测试的结果，当测试通过团队定义和接受的质量阈值时，结束此阶段。

2.3.2 代码评审和静态代码分析：识别缺陷和技术债务

代码评审是由两个或两个以上的人（其中一个通常是作者）对代码进行系统性检查。静态代码分析是通过工具对代码进行系统性检查。在揭露和识别哪些问题影响代码质量时，两者都是有效和广泛的做法。代码评审和静态代码分析提供建设性反馈，帮助识别缺陷和管理技术债务。

资源限制、比预期更高的技术复杂性、快速变化的优先事项和技术限制等障碍可能会阻碍编写高质量代码的努力，并迫使程序员做出妥协，从而降低代码的质量，以便更直接获得结果。这些妥协可能会带来缺陷并引起技术债务。

技术债务是指由于现在选择一个劣质但更容易实施的解决方案，导致在未来实施一个更好的解决方案（包括消除潜在缺陷）需要更多的努力。技术债务往往是无意的，是由于微妙的妥协或随着软件的发展逐渐积累的微小或不被注意的变化。代码评审和静态代码分析有助于识别技术债务的不同原因，如复杂性增加、循环依赖、不同代码模块之间的冲突、代码覆盖率差、代码不安全等。其他类型的技术债务也可能发生在测试工件、基础设施和 CI 管道等方面。

如果因有意的原因产生了技术债务（作为其他决定的结果，或作为妥协），或通过代码评审和静态代码分析确定了技术债务，那么应当努力减少技术债务。最好立即处理。如果不能立即处理，处理技术债务的任务应添加到产品待办列表（Backlog）。

一方面，需要额外的时间来分析和评审代码，另一方面，不断会产生技术债务，这两者之间的权衡几乎总是倾向于代码分析和评审。当有缺陷的代码和技术债务扩散时，在不损害系统其他部分的情况下纠正变得越来越困难、昂贵且耗时。代码分析和评审可以提高代码的质量，并可能减少总体时间支出。

除了帮助识别缺陷和管理技术债务外，代码分析和评审还提供了额外的好处：

- 培训和分享知识
- 提高代码的健壮性、可维护性和可读性
- 提供监督并保持统一的编码标准

代码评审

通过参与代码评审，测试员可以利用他们独特的视角，通过与程序员合作，在早期阶段识别潜在缺陷并避免技术债务，为代码质量做出有价值的贡献。测试员应该有能力阅读他们评审的代码中使用的编程语言，但有效地参与代码评审并不需要他们具有高超的编码技能。他们可以在许多方面发挥自己的专长，例如：询问有关代码行为的问题，建议可能没有考虑的用例，或者监视可能指示质量问题的代码度量。代码评审还提供了一个在程序员和测试员之间共享知识的机会。敏捷项目中代码评审的主要挑战之一是进行代码评审所需的短迭代和时间。重要的是要为代码评审做好计划，并在每一次迭代中都留出必要的时间来执行这些评审。

代码评审是由其他人（除作者外）执行或与其他人一起执行的手工活动，而且可能会借助工具的支持。通常，团队领导或团队中更有经验的程序员将执行代码评审，尽管他们也可以与其他团队成员一起完成。一般来说，测试员和其他非程序员的成员参与代码评审十分有益。

不同的代码评审方法在正式程度和严格程度上有所不同[Wiegers02]。越正式和严格的方法往往越彻底，但也越耗时。越不正式和严格的方法越不彻底，但可能要快得多。同行评审有不同类型，而敏捷团队通常在集成之前更倾向于频繁地执行更快的评审。

代码评审可以与评审人和作者/开发人员并排进行。这种模式在临时评审和结对编程中很常见，这有利于良好的沟通，鼓励更深入的分析和更好的知识共享。它还可以促进团队凝聚力和士气。

如果是分布式团队或团队更喜欢较为远程的方法，配置管理系统可以为代码评审过程提供便利。代码评审过程通常是部分自动化的，作为持续集成过程的一部分。这些过程可以支持在每一轮评审中由单个评审人进行代码评审，或者支持协作团队评审。

静态代码分析

在静态代码分析中，通过工具来分析代码并搜索特定的问题，而不执行代码。静态代码分析的结果可能指向代码中的明确问题，或提供需要进一步评估的间接指标。

许多开发工具，特别是集成开发环境（IDE），可以在编写代码时执行静态代码分析。这提供了即时反馈的好处，尽管它可能只能应用在持续集成过程中执行的一部分分析。

3. 测试自动化　135分钟

关键词

数据驱动测试（data-driven testing），关键词驱动测试（keyword-driven testing），测试规程（test procedure），测试方法（test approach）

测试过程的学习目标

ATT-3.x　(K1)关键词

3.1 测试自动化技术

ATT-3.1.1(K3)应用数据驱动和关键词驱动测试技术开发自动化测试脚本

ATT-3.1.2(K2)了解如何在敏捷环境中将测试自动化应用于给定的测试方法

ATT-3.1.3-1(K2)了解测试自动化

ATT-3.1.3-2(K2)了解各种测试方法之间的差异

3.2 自动化程度

ATT-3.2.1-1　(K2)了解应考虑哪些因素来确定足以跟上部署速度的测试自动化程度

ATT-3.2.1-2　(K2)了解敏捷环境中测试自动化的挑战

3.1 测试自动化技术

3.1.1 数据驱动测试

动机

数据驱动测试是一种测试自动化技术，对于测试步骤相同而测试数据输入具有不同组合的测试用例，它最大限度地减少了开发和维护所需的工作。数据驱动测试是一种成熟的技术，它并不只用于敏捷项目。因为它具有降低测试自动化开发和维护工作的能力，这种技术应该被认为是每个敏捷项目测试自动化策略的一部分。

概念

数据驱动测试的基本思想是将测试逻辑与测试数据分开。这样，测试规程会指定测试数据变量，这些变量引用单独存储在测试数据列表/表中的测试数据值。然后，可以使用不同的测试数据集重复执行测试规程。更多细节和示例可以在[AdvancedTestAutomationEngineer]中找到。

为敏捷团队带来的优势

- 敏捷团队可以快速适应产品每次迭代变化/增加的功能，因为更改/添加新的数据组合很简单，并且对现有的自动化几乎没有或根本没有影响。
- 敏捷团队通过添加、更改或删除测试数据表中的条目，可以轻松地扩大或缩小必要的测试覆盖率。这也有助于敏捷团队控制测试执行时间，以便满足在持续部署方面的约束。
- 该技术支持跨功能的理念，因为测试数据表比测试脚本更容易理解，从而使技术能力较弱的团队成员能更有效地参与其中。
- 由于测试数据表更容易理解，该技术可以帮助较少技术/非技术的团队成员和客户/用户对测试用例和验收准则进行早期反馈。
- 这种技术帮助敏捷团队更有效地完成测试自动化任务，因为它不仅降低了开发新测试的工作量，而且减少了对现有数据驱动测试的维护。

为敏捷团队带来的局限性

虽然在敏捷环境中使用这种技术没有具体的限制，但敏捷团队应该意识到使用这种方法的一般局限性。更多细节可在[AdvancedTestAutomationEngineer]中找到。

工具

- 测试数据的编辑、存储和管理通常是通过使用电子表格或文本文件来完成的。
- 大多数测试自动化工具/语言提供内置命令，读取电子表格或文本文件中的测试数据。

3.1.2 关键词驱动测试

动机

测试自动化的一个缺点是，自动化测试脚本比用自然语言描述的手工测试规程更难理解。应用关键词驱动测试自动化技术有助于提高自动化测试脚本的可读性、可理解性和可维护性。

概念

关键词驱动测试的基本思想是定义一组关键词，这些关键词取自产品的用例和/或客户的业务领域，并且用这些词汇来描述测试规程。因为它们是半自然语言，由此产生的自动化测试脚本比普通程序/脚本语言文本更容易理解。

可以在[AdvancedTestAutomationEngineer]中找到更多详细信息和示例。

在使用关键词编写测试规程时，可以采用多种不同的样式（参见[Linz14]中的第 6.4.2 章节『Keyword-driven Testing』）：

- 列表样式：测试规程是一个简单的关键词序列/列表；
- BDD样式：将测试规程（或场景）编写成一个与自然语言类似的句子，使用关键词作为句子的“可执行”部分（参见第2.1.2节『BDD』）；
- DSL样式：一种基于“特定领域语言”（domain-specific language，DSL）概念的形式语法（参见[Fowler/Parsons10]），定义了如何组合关键词（以及潜在的附加语言元素）。

为敏捷团队带来的优势

- 通过定义关键词或DSL，敏捷团队创建并规范其与领域相关的团队词汇表，帮助团队成员更清晰、更准确地沟通，因此有助于避免误解。
- 敏捷团队可以快速收集更多合格的客户/用户反馈。由关键词和/或用BDD/DSL 样式编写的测试规程比普通程序语言编写的测试代码更能被客户理解。非正式的验收准则可以形式化，而不会失去“自然语言的可读性”。这有助于了解业务逻辑本身，因此也有助于了解对验收准则的解释。
- 测试用例的创建方式归功于实时文档的创建和管理。
- 该技术因为用关键词编写测试规程不需要编程技能，从而可以融入更多的非技术成员，所以能帮助敏捷团队以跨功能的方式完成测试自动化任务。
- 相较于改变多个测试规程中的相同行为，为一个定义的关键词改变行为要少很多工作。这样可以大大减少维护工作，并腾出资源用于实施新测试。

- 敏捷团队可以在整个测试金字塔中应用该技术（从而发挥其优势），因为在实现关键词来驱动测试对象时，可以利用任意接口，从接口级别、测试级别直到底层的 API级别（例如，通过API调用、REST调用、Soap调用等）。这意味着，因为应用广泛，该概念也适用于单元和集成测试用例，而不仅仅限于通过用户界面进行系统测试。但是，它会为单元测试添加不必要的抽象级别。

为敏捷团队带来的局限性

除了[AdvancedTestAutomationEngineer]中描述的这种方法的一般局限性外，敏捷团队还应该注意以下局限性和潜在的陷阱：

- 若要执行关键词驱动测试规程，必需适当的执行框架（例如，需要包括关键词解释器）。不建议从头开始重新创建框架。该团队将通过使用支持关键词驱动测试的现有框架或工具获得更高的速度。如果团队遵循BDD或DSL 样式，情况尤其如此。
- 以稳定、可维护的方式实现新关键词是困难的，这需要经验和良好的编程技能。关键词实现任务也与产品编码任务存在竞争关系。因此，最终测试自动化速度可能低于预期。
- 关键词集（命名、抽象级别）和/或DSL（语法规则）必须精心设计。否则，可理解性和/或可伸缩性将无法满足期望。
- 关键词的集合也必须正确管理。如果不这样做，关键词实现就不会得到成功，因为可能会发生同义词的多个实现，或者关键词没有或很少被测试用例使用。为了避免这些陷阱，团队应该让一个团队成员负责管理关键词词汇表。
- 团队应该意识到，应用关键词驱动测试需要一些前期投资（例如，定义关键词/域语言，选择适当的框架，实现第一组关键词），而且在项目开始时，测试自动化速度可能会降低。

工具

- 与数据驱动测试一样，一种常见但有限的方法是使用电子表格、文本或平面文件来编辑、存储和管理关键词驱动测试。
- 多种测试框架、测试执行工具和测试管理工具对关键字驱动测试提供内置支持；这些工具根据工具供应商或框架不同，命名为“关键词驱动测试”、“基于交互的测试”、“业务流程测试”或“行为驱动测试”。可用工具可在[ToolList]中找到。

3.1.3 将测试自动化应用于给定的测试方法

测试自动化不是测试的目标。测试自动化是一种策略，如果使用得当，可以通过提高测试效率、使测试有效针对某些类型的缺陷（例如性能和可靠性缺陷），或是通过早期发现缺陷，促进更大的策略性测试目标。这种策略十分适合当前测试领域的环境，因此正在蓬勃发展。

测试自动化通常采取许多不同的形式，并涉及许多不同的工具，这取决于团队所处的环境和需求。在大型项目中，通常不仅仅是有一个适合所有需求的解决方案，而且有多种相关的测试自动化策略。测试自动化的应用必须适合组织的测试策略和特定项目的测试方法。

测试自动化不仅仅是测试执行的自动化。测试自动化可以在测试环境配置、测试发布获取、测试数据管理和测试结果比较等诸多方面发挥重要作用。在规划和设计这些工具的使用时，应当考虑测试方法、已实现的敏捷生命周期的影响、这些测试自动化工具的能力以及其他各种工具与这些测试自动化工具的集成（例如，持续集成框架内的测试自动化）。

在某些情况下，测试自动化直接服务于迭代的目标，即构建新的功能。在其他情况下，测试自动化间接地支持这些目标；例如，通过减少与系统更改相关的回归风险。正如敏捷基础课程大纲中所讨论的，一些组织选择将这些支持性的测试自动化工作放入迭代团队之外的团队中。例如，在这些组织中，您可以看到由一个单独的团队创建和维护回归测试自动化框架，作为对多个迭代团队的服务。如果外部团队为迭代团队提供有用的服务，帮助这些迭代团队专注于他们的即时迭代目标，这种方法就可以成功。不过，依赖外部团队可能影响团队的任务，因为团队转移了对任务的（部分）控制。

以下示例说明了 ISTQB® 基础级测试工程师、高级测试经理和专家级测试经理课程大纲中所述的主要测试方法的测试自动化注意事项：

分析式：行为驱动开发（BDD）和验收测试驱动开发（ATDD）技术可以在敏捷环境中的分析方法中使用，例如，应用于测试自动化。BDD 和 ATDD 可用于在用户故事实现的同时（甚至之前）生成自动化测试。

基于模型：通过对功能行为进行基于模型的测试，有助于在用户故事实现期间自动创建测试，快速实现高效的测试。基于模型的测试还可以用于创建用户故事，因为模型可以用来测试需求，并帮助静态评审。模型通常用于测试可靠性和性能等非功能性行为，这是许多系统的重要特性，也是整个系统的突显特征。

有条不紊：由于敏捷项目包含短时间的多次迭代，自动化测试检查表（包含所有活动）可以作为有效执行稳定测试集的有条不紊的方法。

符合流程：对于必须符合外部定义的标准或条例的项目，这些标准或条例可以影响如何使用自动化测试或如何捕获自动化测试结果。例如，在 FDA 监管的（食品和药物管理局；高风险）项目上，自动化测试及其结果必须追溯到需求，并且结果必须包含足够的细节来证明测试通过。

反应性（或启发式）：反应性测试在敏捷测试中起着重要的确认作用，而大多数自动化测试主要起着验证作用。反应策略主要是手工的（例如探索性测试、错误猜测等），因为许多可以预先准备的测试将会实现自动化，自动化测试覆盖率的增加往往导致更大程度地采用遵循反应策略的手工测试。此外，余下的手工测试可以涵盖风险更大的领域。

执导式（或咨询式）：当测试覆盖范围由外部利益相关者指定并使用测试自动化时，测试团队响应请求的能力很重要。因此，遵循执导式测试策略的测试团队应该考虑在迭代中完成任务所需的时间和技能。

回归规避：在敏捷项目中，回归规避策略的一个主要特征是一组大型、稳定和不断增长的自动回归测试。足够的覆盖范围、可维护性和有效的结果分析是至关重要的，特别是随着回归测试数量的增加，这尤为重要。一种成功的回归规避方法不是专注于不断增长的回归测试集，而是专注于不断改进和重构创建的测试。

3.2 自动化程度

3.2.1 了解所需的测试自动化程度

自动化是敏捷项目中的一个重要元素，因为它不仅包括测试自动化，而且还包括部署过程的自动化（参见第 4 章）。持续部署是将新版本自动部署到生产环境中。而且，持续部署定期以较短的时间间隔进行。

由于持续部署是一个自动化过程，自动化测试必须足以保持所需的代码质量水平。仅仅执行自动单元测试不足以实现足够的测试覆盖。作为部署过程的一部分执行的自动化测试套件还必须包括集成和系统级测试。在实践中，可能仍然需要一些手工测试。

以下是敏捷环境下测试自动化面临的一些挑战：

- 测试套件数量：除了维护或固化迭代外，每次迭代实现额外的特征功能。为了使测试套件与产品的额外功能保持一致，敏捷团队必须在每次迭代中增强其测试套件。这意味着，在默认情况下，随着一次一次的迭代，测试套件中的测试用例数量不断增加。这需要仔细和深思熟虑地重构测试，以便增加覆盖范围，而不显著增加规模。不管怎样，维护、准备和执行整个测试套件所需的工作和时间都会随着时间的推移而增加。
- 测试开发时间：对于验证产品新功能或更改的功能，必须设计和实现应有的测试。这包括创建或更新必要的测试数据，并准备对测试环境的任何更新。测试的可维护性也会影响开发时间。
- 测试执行时间：不断增加的测试套件数量将导致执行测试所需的时间持续增加。
- 工作人员可用性：必须为每个部署配备创建、维护和执行测试套件所需的工作人员。要想在项目的整个过程都确保这一点，可能很难甚至不可能，特别是在假日、周末，或者如果在下班时进行部署。

解决这些挑战的一个策略是，通过对测试进行优先排序和/或通过风险分析，只选择、准备和执行测试的子集来最小化测试套件。这种策略的缺点是增加了风险，因为它意味着执行的测试数量减少。

尽可能多的测试自动化会增加部署的频率和/或速度。跟上（甚至增加）部署频率和/或速度的另一种策略是尽可能多地实现自动化测试。

测试自动化是在任何项目中跟上或提高部署速度的有效工具，是持续部署的前提。为了获得适当数量的测试自动化来跟上部署速度，应分析和平衡以下优点和局限性：

优点

- 测试自动化可以对每个部署周期的测试覆盖范围都提供一个确定且可重复的级别的保证。
- 测试自动化可以减少测试执行时间，有助于提高部署速度。
- 测试自动化可以减少实现更高部署频率的限制。
- 持续部署有助于缩短上市时间和用户反馈周期。
- 测试自动化可以更频繁地发生，这使每个构建都可以执行所有测试，因而在CI中提供一个稳定的主线，软件会尽可能频繁地合并到主线中。

局限性

- 测试自动化需要测试开发和维护工作，这本身可能会延长开发的时间，从而降低部署频率。
- 系统级别的测试，特别是负载或性能测试（以及可能的其他非功能测试），即使是自动化的也可能需要很长时间才能执行。
- 由于许多因素，自动化测试可能会失败。通过的自动化测试用例可能不可靠（假阴性）。失败的自动化测试用例可能是由于与产品质量无关的错误或由于假阳性而失败。

发布频率应符合产品用户的需求。相较于较慢的频率，在太短的时间内交付过多的版本可能会使客户更为不快。因此，在一些情况下，技术上有可能进一步提高部署的频率，但从客户的角度来讲却并不会获得额外的好处。

4. 部署和交付　　105分钟

关键词

服务虚拟化（Service virtualization），持续测试（continuous testing）

部署和交付的学习目标

ATT-4.x　(K1)关键词

4.1 持续集成、持续测试和持续交付

ATT-4.1.1　　(K3)应用持续集成（CI）并总结其对测试活动的影响

ATT-4.1.2　　(K2)了解持续测试在持续交付和持续部署（CD）中的作用

4.2 虚拟化

ATT-4.2.1-1　　(K2)了解服务虚拟化的概念及其在敏捷项目中的作用

ATT-4.2.1-2　　(K2)了解服务虚拟化的优点

4.1 持续集成、持续测试和持续交付

4.1.1 持续集成及其对测试的影响

持续集成（CI）的目的是提供快速反馈，以便在代码中引入缺陷时尽快找到并修复它们。敏捷测试员应该帮助设计、实现和维护一个有效且高效的 CI 过程，不仅在创建和维护适合 CI 框架的自动化测试方面，而且在确定测试的优先顺序、必要环境、配置等诸多方面。

在 CI 的理想情况下，一旦代码构建完成，所有的自动化测试都会接着运行，以便验证软件继续按照定义的行为运行，并且没有因代码更改而损坏。然而，有两个相互矛盾的目标：

4.1 经常执行CI 过程以获得对代码的即时反馈
4.2 每次构建后尽可能彻底地验证代码

如前一章所述，如果在自动化测试的设计、实现和维护中不够仔细，会因执行所有自动化测试而使得每天完成多次 CI 过程花费过长的时间。即使有了仔细的测试自动化，在所有测试级别上完全执行所有自动化测试也会过度减缓 CI 过程。当在上述目标之间找到适当的平衡时，不同的组织有不同的优先事项，不同的项目需要不同的解决方案。例如，如果一个系统是稳定的，变更的频率较低，那么可能需要更少的 CI 周期。如果系统不断更新，那么最有可能需要更多的 CI 周期。

有一些解决方案支持这两个目标，它们是相辅相成的，可以并行使用。

第一种解决方案是，对测试进行优先排序，以便使用基于风险的测试方法始终执行基本和最重要的测试。

第二种解决方案是，使 CI 过程中的不同测试配置能够用于不同类型的 CI 周期。对于日间构建和测试过程，只执行基于预先优先级选择的基本测试。对于夜间 CI 过程，执行更多或可能所有不需要生产前环境的功能测试。在发布之前，在具有真实用户输入的生产前环境中进行更彻底的功能和非功能测试，包括与数据库、不同系统和/或平台的集成测试。

第三种解决方案是，通过减少用户界面（UI）测试的数量来加快测试执行。通常，完成单元/集成测试的执行没有时间问题，这些测试通常运行得很快。可能会出现这样的情况，即存在如此多的单元测试，以至于它们不能在 CI 过程中完全执行。真正的时间问题通常是因为使用端到端 UI 测试用例作为 CI 过程的一部分。解决办法是增加 API、命令行、数据层、服务层和其他非 UI 业务逻辑测试的数量，减少 UI 测试，在测试自动化金字塔中将自动化工作往底层推动。这不仅切实提高了测试的可维护性，而且也减少了测试执行时间。

当 CI 系统中的测试执行非常频繁，并且不可能运行所有测试用例时，可以使用第四种解决方案。开发人员或测试员根据对现有测试用例的代码修改以及对执行跟踪的了解，只选择和执行受更改影响的测试用例（也就是通过影响分析来选择测试）。由于在短周期内只修改了整个代码库的一小部分，所以需要执行的测试用例相对较少。然而，这样做可能会忽略大量的回归缺陷。

第五种解决方案是，将测试套件拆分为同等大小的块，并在多个环境（代理、构建场、云）上并行运行它们。这在使用 CI 的公司中非常普遍，因为他们本来就需要大量的构建服务器容量。

持续集成在多种技术平台上运行。随着开发和部署工具向云移动，CI 产品也随之移动。虽然大多数 CI 产品仍然设计为下载到本地环境中运行，但云已经缔造了一种新的产品 — 在团队可以快速开始使用的托管平台上提供 CI 服务。现在，团队可以避免因创建新环境、下载、安装和配置 CI 软件而消耗多余的成本和时间。通过移动到云端，团队可以轻松配置并开始工作。在实际应用时，云是一种灵活的方法，可以在必要时加快测试构建和测试执行。

目前的 CI 工具不仅支持持续集成，而且支持持续交付和持续部署。在一个不完全复制生产的环境中运行自动化测试可能会导致假阴性，但克隆生产环境可能会花费过多。解决这种矛盾的办法包括使用云测试环境，在必要的基础上复制生产环境；或者，创建测试环境，实现生产环境的一个缩小而现实的版本。

良好的 CI 系统需要能够在更复杂的环境和不同的平台上自动部署。测试员的任务是：计划要包括哪些测试用例；为了保持适当的覆盖范围，在某些或所有平台上为必须执行的测试用例划分优先级；并设计测试用例以在类似生产的环境中有效地验证软件。

4.1.2 持续测试在持续交付和部署（CD）中的作用

持续测试是一种方法，它涉及到早期测试、经常测试、随处测试和自动化的过程，以尽可能快地反馈与软件候选发布版本相关的业务风险。在持续测试中，对系统进行的修改会触发所需的测试（即那些更改所涉及的测试）自动执行，给开发人员及时提供反馈。持续测试可以应用于不同的情况（例如，在 IDE、CI、持续交付和持续部署等），但概念是相同的，即尽早自动执行测试。

持续交付需要持续集成。通过在构建阶段之后将所有代码更改部署到测试或生产前环境和/或类似生产的环境，持续交付扩展了持续集成。这种分阶段过程使得通过应用真实的用户输入和非功能测试（如负载、压力、性能和可移植性测试）来执行功能测试成为可能。由于嵌入的每一个更改都通过完全自动化交付到分阶段环境中，因此敏捷团队可以有信心，当达到工作完成标准（Definition of Done，DoD）时，系统可以通过按下按钮部署到生产中。

持续部署比持续交付更进一步，每一次更改都会自动部署到生产。持续部署的目的是将编写新代码的开发任务与让实际用户使用代码之间的时间间隔最小化。详情见 [Farley]。

4.2 服务虚拟化

服务虚拟化是指创建一个可共享的测试服务（虚拟服务）来模拟关联系统或服务的相关行为、数据和性能的过程。有了这种虚拟服务，即使实际服务仍在开发中或不可用，开发和测试团队也能够执行任务。

今天的开发团队和系统高度连接且相互依存，因此在开发周期中早期测试很难完成。通过使用服务虚拟化解耦团队和系统，团队可以在开发生命周期的早期使用更现实的用例和负载来测试他们的软件。

虽然桩、驱动器和模拟在允许早期测试方面是有价值的，但手工创建的桩通常是无状态的，并对具有固定响应时间的请求提供简单的响应。虚拟服务可以支持有状态事务并保持动态元素的上下文环境（会话/客户 ID、日期和时间等），而且使用可变的响应时间来模仿消息流通过多个系统。

虚拟服务是借助服务虚拟化工具创建的，通常采用以下方式之一：

- 通过从数据中解释：XML文件、服务器日志中的历史数据或对数据进行简单采样的电子表格
- 通过监控网络流量：使用SUT（测试中的系统）将触发由服务虚拟化工具捕获和建模的依赖系统的相关行为。
- 通过从代理捕获：服务器端或内部逻辑在通信消息上可能看不到，这使得相关数据和消息从依赖服务器推送，以便重新创建为虚拟服务
- 如果上述任何一项都不适用，则需要根据适当的通信协议在项目团队内创建虚拟服务。

服务虚拟化的好处包括：

- 对正在开发的服务以并行方式进行开发和测试活动
- 服务和API的早期测试
- 早期测试数据设置
- 并行编译、持续集成和测试自动化
- 早期发现缺陷
- 减少共享资源的过度使用（COTS系统、主机、数据仓库）
- 通过减少对基础设施的投资来降低测试成本
- 启用SUT的早期非功能测试
- 简化测试环境管理
- 减少测试环境的维护工作（不需要维护中间件）
- 降低数据管理风险（这有助于遵守GDPR 的规定）

请注意，虚拟服务不需要包括实际服务的所有功能和数据，只需要测试 SUT 所需的部件。

引入服务虚拟化可能是一项复杂和潜在的昂贵工作。引入服务虚拟化工具应与向团队和组织引入任何新的测试工具相似对待。这一主题在 ISTQB® 基础课程大纲和 ISTQB® 高级测试经理课程大纲中讨论。

5. 参考资料

[AgileFoundationExt] 指《认证测试工程师基础级扩展大纲-敏捷测试工程师》2014版。

[Foundation]指《认证测试工程师基础级大纲》2018版。

[AdvancedTestAutomationEngineer]指《认证测试工程师高级测试自动化工程师大纲》2016版。

[ISO29119-5]即 ISO/IEC/IEEE29119-5，指《Software and systems engineering -- Software testing》的第5部分『Keyword-Driven Testing』。

书籍

[Adzic09] Adzic, Gojko. “Bridging the Communication Gap” Neuri Ltd, 2009

[Adzic11] Adzic, Gojko. “Specification by Example” Manning, 2011. [Beck02] Kent Beck, “Test Driven Development: By Example”, 2002

[Beck99] Beck, Kent. “Extreme Programming Explained” Addison Wesley, 1999.

[Carkenord08] Barbara A. Carkenord, “Seven Steps to Mastering Business Analysis”, J. Ross Publishing, 2008.

[Cohn 09] Mike Cohn: Succeeding with Agile: Software Development Using Scrum. Addison-Wesley Professional, 2009;

[Crispin08] Crispin, L. and Gregory, J. (2008) Agile Testing: A Practical Guide for Testers and Agile Teams.Crawfordsville : Addison-Wesley Professional

[Elfriede99] Dustin Elfriede, Automated Software Testing: Introduction, Management, and Performance: Introduction, Management, and Performance, Addison-Wesley Professional, 1999

[Evans03] Eric Evans, “Domain-Driven Design: Tackling Complexity in the Heart of Software”, 2003

[Fewster12] Mark Fewster, Dorothy Graham, Experiences of Test Automation: Case Studies of Software Test Automation, Addison-Wesley Professional, 2012

[Fowler/Parsons 10], Martin Fowler, Rebecca Parsons, Domain-Specific Languages, Addison-Wesley Signature, Series, 2010

[Hendrickson13] Elisabeth Hendrickson, “Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing”, Pragmatic Bookshelf, 2013

[Jeffries00] Ron Jeffries, Ann Anderson, and Chet Hendrickson, “Extreme Programming Installed,”
Addison-Wesley Professional, 2000

[Jorgensen13] Paul C. Jorgensen, Software Testing: A Craftsman’s Approach, Auerbach Publications; 4th edition, 2013

[Linz14] Tilo Linz, Testing in Scrum, A Guide for Software Quality Assurance in the Agile World, Rocky Nook, 2014

[Meszaros07] Gerard Meszaros, “xUnit Test Patterns: Refactoring Test Code”, Addison-Wesley, 1st Edition, Apress, 2007

[Michelsen12] John Michelsen and Jason English, “Service Virtualization: Reality is Overrated”, Apress, 1st Edition, 2012.

[Osherove09] Roy Osherove, “The Art of Unit Testing”, 2009

[Paskal15] Greg Paskal, “Test Automation in the Real World: Practical Lessons for Automated Testing”, MissionWares, 2015

[Smart15]; Smart, John Ferguson; "BDD in action", Manning, 2015

[Wein89] Weinberg, Gerald &Gause, Donald. “Exploring Requirements: Quality Before Design” Dorset House, 1989.

[Whittaker09] James A Whittaker, “Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design”, Addison-Wesley Professional, 2009

[Wiegers02] Karl Wiegers, “Peer Reviews in Software: A Practical Guide (Paperback)”, 2002.

敏捷术语

ISTQB® 词汇表中出现的关键词在每一章的开头位置列出。有关通用的敏捷术语，我们采用以下广为接受的互联网资源：https://www.agilealliance.org/agile101/agile-glossary/，其中包含了术语的定义。在定义相互冲突的情况下，优先使用 ISTQB® 词汇表中的定义。

其他参考资料

以下参考资料指向互联网和其他来源提供的资料。虽然这些参考资料在本课程大纲出版时进行了核对，但是如果这些参考资料不再提供，ISTQB® 不承担责任。

[Cohn09]The Forgotten Layer of the Test Automation Pyramid, https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid

[CyclomaticComplexity] https://en.wikipedia.org/wiki/Cyclomatic_complexity

[Farley]http://www.davefarley.net/?cat=5

[DSL]https://en.wikipedia.org/wiki/Domain-specific_language

[Fowler07]https://martinfowler.com/articles/mocksArentStubs.html

[Fowler04]Fowler, Martin. https://martinfowler.com/bliki/SpecificationByExample.html

[Fowler03]Martin Fowler, https://martinfowler.com/bliki/TechnicalDebt.html

[Gherkin]https://docs.cucumber.io/gherkin/

[INVEST]Bill Wake, “INVEST in Good Stories, and SMART Tasks”, http://xp123.com/articles/invest- in-good-stories-and-smart-tasks/

[IQBBA]International Qualifications Board for Business Analysts, http://www.iqbba.org/

[IREB]International Requirements Engineering Board, https://www.ireb.org/en

[IIBA] International Institute of Business Analysts (IIBA), https://www.iiba.org/

[Marick01] Marick, Brian, http://www.exampler.com/old-blog/2003/08/21/#agile-testing-project-1

[Marick03] Marick, Brian. http://www.exampler.com/old-blog/2003/08/22.1.html

[North06] Dan North, “Introducing BDD”, 博客, 2006

[TESTDOUBLES] WojciechBulaty, Bill Wake, “Stubbing, Mocking and Service Virtualization Differences for Test and Development Teams”, https://www.infoq.com/articles/stubbing-mocking- service-virtualization-differences

[ToolList] 测试工具审查，国际软件测试工具市场的信息平台， www.testtoolreview.de/en/

[xUnit]https://en.wikipedia.org/wiki/XUnit, https://en.wikipedia.org/wiki/Unit_testing

6. 附录

6.1 敏捷技术测试工程师专用术语词汇表

词汇术语	定义
角色	一个虚构的人物，代表某种类型的用户，以及他们将如何与系统交互。
故事板	对系统的一种可视化表示方式，在这个系统中，为了便于理解业务流程，在上下文中对用户故事进行形象的描述。
故事地图	一种在两个维度上排列用户故事的技术，水平轴表示它们的执行顺序，垂直轴表示所实现产品的复杂程度。