文档编号：SY0006_CTAL_TAE_EN2.0_CN1.0

认证测试工程师
高级自动化测试工程师
教学大纲

版本：EN2.0_CN1.0

发布日期：2025 年 7 月 1 日

国际软件测试认证委员会

ISTQB® Certified Tester Foundation Level

中文版的翻译、编辑和出版统一由 ISTQB®授权的 CSTQB®负责

版权声明

英文版权声明

中文版权声明

修订记录

版本	日期	备注
v2016	2016/10/21	CTAL-TAE 正式版发布
v2.0	2024/05/03	CTAL-TAE v2.0 正式版发布
EN2.0_CN1.0	2025/07/01	CTAL-TAE 教学大纲 v2.0 中文本地化发布

目录

致谢

本文档由 ISTQB® 大会于 2024 年 5 月 3 日正式发布。

本文档由国际软件测试认证委员会专家工作组的测试自动化工作组编写： Graham Bath（专家工作组主席）、Andrew Pollner（专家工作组副主席兼测试自动化工作组主席）、Péter Földházi、Patrick Quilter、Gergely Ágnecz、László Szikszai。 测试自动化工作组评审员包括 Armin Beer、Armin Born、Geza Bujdoso、Renzo Cerquozzi、Jan Giesen、Arnika Hryszko、Kari Kakkonen、Gary Mogyorodi、Chris van Bael、Carsten Weise、Marc-Florian Wendland。

技术审核： Gary Mogyorodi

以下人员参与了本教学大纲的审核、评论和投票：

Horváth Ágota, Laura Albert, Remigiusz Bednarczyk, Jürgen Beniermann, Armin Born, Alessandro Collino, Nicola De Rosa, Wim Decoutere, Ding Guofu, Istvan Forgacs, Elizabeta Fourneret, Sudhish Garg, Jan Giesen , Matthew Gregg, Tobias Horn, Mattijs Kemmink, Hardik Kori, Jayakrishnan Krishnankutty, Ashish Kulkarni, Vincenzo Marrazzo, Marton Matyas, Patricia McQuaid, Rajeev Menon, Ingvar Nordström, Arnd Pehl, Michaël Pilaeten, Daniel Polan, Nishan Portoyan, Meile Posthuma, Adam Roman, Pavel Sharikov, Péter Sótér , Lucjan Stapp, Richard Taylor, Giancarlo Tomasig, Chris Van Bael, Koen Van Belle, Johan Van Berkel, Carsten Weise, Marc-Florian Wendland, Ester Zabar.

ISTQB Working Group Advanced Level Test Automation Engineer (Edition 2016): Andrew Pollner (Chair), Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu, Ina Schieferdecker.

ISTQB 高级测试自动化工程师（2016 年版）工作组： Andrew Pollner (主席), Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu, Ina Schieferdecker。

ISTQB® CTAL-TAE 高级-自动化测试工程师大纲 V2.0 文档中文翻译、评审及 QA 评审参与者（按姓氏拼音排序）：陈耿（QA 评审）、陈崇来、江蘅君、潘鑫、张喆。

0. 简介

0.1 本大纲的目的

本大纲构成了国际软件测试认证委员会高级测试自动化工程师（CTAL-TAE）认证的基础。ISTQB® 提供此大纲的目的如下：

1. 各成员国认证委员会将大纲翻译成当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. 培训机构制作课件并确定合适的教学方法；
4. 认证考生准备认证考试（认证考试可以作为培训课程的一部分或独立进行）；
5. 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，并以此作为书籍和文章的基础。

0.2 软件测试中的测试自动化工程师

测试自动化工程师资格认证面向参与软件测试和测试自动化的任何人。包括测试工程师、测试分析师、测试自动化工程师、测试顾问、测试架构师、测试经理和软件开发人员等角色的人员。该资格证书也适用于希望对测试自动化有基本了解的任何人，例如项目经理、质量经理、软件开发经理、业务分析师、IT 总监和管理顾问。

测试自动化工程师大纲面向希望实施或改进测试自动化的测试工程师。其定义了能支持可持续的解决方案的方法和实践。

与测试自动化解决方案相关的其他指南和参考模型，是所选软件开发生命周期的软件工程标准、编程技术和格式标准。本大纲并不教授软件工程的知识。然而，测试自动化工程师应当具备软件工程方面的技能、经验和专业知识。

此外，测试自动化工程师需要了解专业化编程和文档化的标准和最佳实践，以便在开发测试自动化解决方案时加以运用。这些实践可以提高测试自动化解决方案的可维护性、可靠性和安全性。此类标准通常基于质量特性。

0.3 测试工程师和测试自动化工程师的职业道路

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®测试自动化工程师认证的个人也可能对测试自动化策略（CT-TAS）认证感兴趣。

获得 ISTQB®认证测试工程师-测试自动化工程师认证的个人，还可能对核心高级（测试分析师、技术测试分析师和测试经理）和之后的专家级别（测试管理或测试过程改进）感兴趣。任何寻求在敏捷环境领域培养测试实践技能的人，都可以考虑敏捷技术测试员或大规模敏捷测试领导力认证。专家领域提供了深入探讨具有特定测试方法和测试活动的领域，例如在测试自动化策略、性能测试、安全测试、人工智能测试和移动应用测试中，或在需要特定知识的领域（例如，汽车软件测试或游戏测试）。 请浏览 www.istqb.org 以获取 ISTQB 认证测试工程师计划的最新信息。

0.4 商业价值

本节列出了获得测试自动化工程师认证的候选人的预期商业价值。

获得测试自动化工程师认证的候选人能够…

TAE-B01	描述测试自动化的目的
TAE-B02	理解在软件开发生命周期中的测试自动化
TAE-B03	理解基础设施的配置以启用测试自动化
TAE-B04	学到选择正确工具和策略的评估过程
TAE-B05	理解构建模块化以及可扩展测试自动化解决方案的设计概念
TAE-B06	选择一种方法（包括试点），来规划软件开发生命周期内的测试自动化部署
TAE-B07	设计和开发满足技术需求的测试自动化解决方案（新的或修改的）
TAE-B08	考虑测试自动化和测试件维护的范围和方法
TAE-B09	理解自动化的测试如何集成到 CI/CD 流水线
TAE-B10	理解如何收集、分析和报告测试自动化相关的数据，以便为利益相关方提供信息
TAE-B11	验证测试自动化基础设施
TAE-B12	定义测试自动化的持续改进机会

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，并用于创建认证测试工程师-测试自动化工程师考试。

一般来说，除简介和附录以外，本大纲的所有内容都可以在 K2、K3 和 K4 级别进行检查。也就是说，可能会要求考生识别、牢记或回忆八章中提到的任一关键词或概念。每章的开头列出了具体的学习目标级别，并分类如下：

- K2：理解
- K3：应用
- K4：分析

附录 A 提供了学习目标的更多细节和示例。

即使学习目标中没有明确提及，也应牢记章节标题下方作为关键词列出的所有术语。

0.6 测试自动化工程师认证考试

测试自动化工程师认证考试将以本大纲为基础。在回答考试问题时，可能需要使用本大纲中多个章节的材料。除简介和附录外，教学大纲的所有章节均可作为考试内容。标准和书籍被列为参考资料，但除了教学大纲本身根据这些标准和书籍中总结的内容外，其内容不作为考试内容。

有关测试自动化工程师认证考试的更多详情，请参阅与大纲基础、高级和专家模块兼容的考试结构和规则 v1.1 版文档。

参加测试自动化工程师考试的入门标准是考生对软件测试和测试自动化感兴趣。不过，强烈建议考生同时具备以下条件：

- 至少具备软件开发和软件测试方面的基本背景，如六个月的软件测试工程师或软件开发人员工作经验。
- 参加符合 ISTQB 标准（由 ISTQB 成员委员会认可）的课程。

基本要求事项：应在参加 ISTQB® 测试自动化工程师认证考试前获得 ISTQB® 基础级证书。

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。经认证的课程被视为符合本教学大纲，并允许将 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

测试自动化工程师大纲中引用了一些标准（例如（IEEE 和 ISO）。 这些参考资料的目的是提供一个框架（如关于 ISO 25010 质量特性的参考资料）或提供附加信息来源（若读者需要）。 请注意，大纲使用标准文档作为参考。 标准文档不用于考试。有关标准的更多信息，请参阅第 9 章。

0.9 保持时效性

软件行业日新月异。为了应对这些变化，并为利益相关方提供获取相关和最新信息的途径，ISTQB 工作组在 www.istqb.org 网站上创建了指向支持文档和标准变更的链接。根据测试自动化工程师大纲，此信息不会作为考试内容。

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。为实现这一目标，教学大纲包括以下内容：

- 描述测试自动化工程专家意图的一般教学目标
- 学生必须能够回顾的术语（关键词）列表
- 每个知识领域的学习目标，描述要达到的认知学习成果
- 对关键概念的描述，包括参考来源，如公认的文献或标准

大纲内容不是对软件测试的整个知识领域的描述;它反映了测试自动化工程师培训课程中要涵盖的详细程度。其侧重于可应用到所有软件项目（包括遵循敏捷方法的项目）的测试概念和技术。 本大纲不包含任何与敏捷测试相关的具体学习目标，但确实讨论了如何将这些概念应用于敏捷项目和其他类型的项目。

0.11 本大纲的组织方式

共有八章考核内容。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 21 小时的教学，分布在以下八个章节中：

- 第 1 章：45 分钟--测试自动化简介和目标
  - 测试人员了解测试自动化的收益及其局限性

- 介绍不同软件开发生命周期模型下的测试自动化
- 测试工程师学习被测系统（SUT）架构如何影响测试工具的适用性

- 第 2 章：180 分钟 - 准备测试自动化
  - 通过可观察性、可控制性和明确定义的架构，来设计被测系统的易测试性。
  - 测试人员学习不同环境下的测试自动化
  - 评估合适的测试自动化解决方案所需的因素
  - 测试工程师将学习制定测试自动化建议所需的技术考虑因素
- 第 3 章：210 分钟--测试自动化架构
  - 介绍测试自动化架构及其组件，从而形成测试自动化解决方案。
  - 测试人员将了解各层及其在测试自动化框架中的应用
  - 介绍使用测试自动化工具的多种方法
  - 测试人员将学习如何将设计原则和设计模式应用于测试自动化
- 第 4 章：150 分钟--实现测试自动化
  - 将介绍如何有效规划和部署测试自动化试点项目
  - 测试人员将了解部署风险和缓解策略
  - 将介绍提高测试自动化代码可维护性的因素
- 第 5 章：90 分钟--测试自动化的实施和部署策略
  - 测试人员将了解 CI/CD 流水线和跨测试级别的自动化测试执行
  - 将介绍测试自动化组件的配置管理
  - 测试人员将了解应用于 API 和契约测试的依赖关系
- 第 6 章：150 分钟--测试自动化报告和指标
  - 测试人员将学习可从被测系统收集数据的地方，以及用于分析和报告的测试自动化
  - 从被测系统报告和测试自动化中分析数据，揭示故障原因
  - 使用测试报告和仪表板为利益相关方提供信息

- 第 7 章：135 分钟--验证测试自动化解决方案
  - 测试人员将学习如何检查和验证测试自动化组件和环境的正确运行。
  - 确保测试脚本和测试套件的正确执行
  - 测试人员将学习何时执行根本原因分析
  - 将介绍分析测试自动化代码质量的技术
- 第 8 章：210 分钟--持续改进
  - 将涵盖用于改进测试用例的其他数据分析领域
  - 测试人员将学习改进和升级测试自动化解决方案及其组件的方法
  - 确定巩固和效率化测试自动化的方法
  - 测试人员将学习测试自动化工具如何帮助满足测试支持和设置需求

1. 测试自动化简介和目标 - 45 分钟 (K2)

关键词

被测系统（system under test），测试自动化（test automation），测试自动化工程师（test automation engineer）

第 1 章的学习目标：

1.1 测试自动化的目的

CTAL-TAE-1.1.1 (K2) 解释测试自动化的优缺点

1.2 软件开发生命周期中的测试自动化

CTAL-TAE-1.2.1 (K2) 解释如何在不同的软件开发生命周期模型中应用测试自动化

CTAL-TAE-1.2.2 (K2) 为被测统选择合适的测试自动化工具

1.1 测试自动化的目标

1.1.1 解释测试自动化的优缺点

测试自动化包括自动化测试执行和测试报告，是以下一项或多项活动：

- 使用专用软件工具控制和设置测试套件以执行测试。
- 以自动化的方式执行测试。
- 将实际结果与预期结果进行比较。

测试自动化提供能够与被测系统（SUT）交互的重要特性和功能。 测试自动化可覆盖广泛领域的软件。 解决方案涵盖多种软件（例如，有用户界面（UI）的被测系统、无用户界面的被测系统、移动应用程序、网络协议和连接）。

测试自动化有许多优势。 其：

- 与手动测试相比，每次构建可运行更多测试。
- 提供创建和执行无法手动执行的测试的能力（例如，实时响应、远程测试和并行测试）。
- 允许进行比人工测试更复杂的测试。
- 执行速度比人工测试更快。
- 较少受到人为错误的影响。
- 更有效及高效使用测试资源。
- 更快反馈被测系统的质量。
- 有助于提高系统可靠性（如可用性和可恢复性）。
- 提高测试周期内测试执行的一致性。

然而，测试自动化也有潜在的缺点，包括：

- 项目将涉及额外成本，因为可能需要聘请测试自动化工程师（TAE）、购买新硬件并开展培训。
- 建立测试自动化解决方案需要初始投资。
- 开发和维护测试自动化解决方案需要时间。

- 要求有明确的测试自动化目标，以确保成功。
- 测试僵化，对被测系统变化的适应性较差。
- 通过测试自动化引入额外缺陷。

测试自动化有其局限性，需要牢记：

- 并非所有手动测试都能实现自动化。
- 只能验证自动化测试程序被预置的功能。
- 测试自动化只能检查机器可解释的测试结果，这意味着某些质量特性可能在自动化中不可测试。测试自动化只能检查可由自动化测试结果参照物验证的测试结果。

1.2 软件开发生命周期中的测试自动化

1.2.1 解释如何在不同的软件开发生命周期模型中应用测试自动化

瀑布模型

瀑布模型是一种既线性又有顺序的软件开发生命周期模型。 这种模式有明确区分的不同阶段（即需求、设计、实施、验证和维护），每个阶段通常都有必须批准的文档。 测试自动化的实现通常与项目的实现阶段并行或在其后进行。 测试运行通常在验证阶段进行，因为在验证阶段之前，软件组件还没有准备好进行测试。

V 模型

V 模型是一种软件开发生命周期模型，其中的流程是按顺序执行的。 随着项目从高级别需求到低级别需求的确定，相应的测试和集成活动也随之确定，以确认这些需求。 这就是 CTFL 第 2.2 节所述的传统测试级别：组件、组件集成、系统、系统集成和验收。 为每个测试级别提供一个测试自动化框架（TAF）是可行的，也是值得推荐的。

敏捷软件开发

在敏捷软件开发中，测试自动化有无数的可能性。 与瀑布模型或 V 模型不同，在敏捷软件开发方法中，测试自动化工程师 和业务代表可以决定路线图、时间表和计划的测试交付。 在这种方法中，有代码审查、结对编程和频繁的自动化测试执行等最佳实践。 消除孤岛（即确保开发人员、测试人员和其他利益相关方一起工作）使团队能够以适当数量和深度的测试自动化覆盖所有测试级别，实现了

一个称为 "冲刺内(in-sprint)自动化 "的目标。 更多详情，请参阅 ISTQB CT-TAS 教学大纲第 3.2 节。

1.2.2 为特定的被测系统选择合适的测试自动化工具

要为特定项目确定最合适的测试工具，首先必须对被测系统进行分析。测试自动化工程师需要识别能作为工具选择基准的项目要求。

由于用户界面软件和网络服务等使用的测试自动化工具功能不同，因此了解项目希望在一段时间内实现的目标非常重要。 可以使用或选择的测试自动化工具和功能的数量没有限制，但必须始终考虑成本。 使用现成的商业工具或实施基于开源技术的定制解决方案都是一个复杂的过程。

下一个需要评估的问题是测试自动化团队的组成和经验。 如果测试人员几乎没有编程经验，那么使用低代码或无代码解决方案可能是一个可行的选择。

对于具有编程知识的技术测试人员来说，选择与被测系统语言相匹配的工具会有所帮助。 这样做的好处包括可以更有效地与开发人员一起调试测试自动化缺陷，并在团队之间对成员进行交叉培训。

2. 准备测试自动化 - 180 分钟（K4）

关键词

应用程序接口测试（API testing）、图形用户界面测试（GUI testing）、易测试性（testability）

第 2 章的学习目标：

2.1 理解实现测试自动化的基础架构配置

CTAL-TAE-2.1.1 (K2) 描述实现测试自动化的基础设施的配置需求

CTAL-TAE-2.1.2 (K2) 解释如何在不同环境中利用测试自动化

2.2 选择正确工具和策略的评估程序

CTAL-TAE-2.2.1 (K4) 分析被测系统，确定合适的测试自动化解决方案

CTAL-TAE-2.2.2 (K4) 说明工具评估的技术结论

2.1 了解实现测试自动化的基础架构配置

2.1.1 描述实现测试自动化的基础设施的配置需求

被测系统的易测试性（即支持测试的软件接口的可用性，如实现被测系统的控制和可观测性）应与被测系统其他功能的设计和实现同步进行。由于易测试性是系统的非功能性要求，这项工作一般由软件架构师来完成，通常还需要测试自动化工程师的参与，以确定可改进的具体领域。

为了提高被测系统的易测试性，可以采用不同的解决方案，这些解决方案具有不同的配置需求，例如：

- 易访问性标识符
  - 不同的开发框架可以自动生成这些标识符，开发人员也可以手动设置这些标识符。
- 系统环境变量
  - 可更改某些应用程序参数，以便通过管理员权限更容易地进行测试。
- 部署变量
  - 与系统变量类似，但可在开始部署前设置。

设计被测系统的易测试性包括以下几个方面：

- 可观测性： 被测系统需要提供能深入嗅探被测系统的接口。 测试用例可以使用这些接口来确定实际结果是否与预期结果一致。
- 可控性： 被测系统需要提供可用于对被测系统执行操作的接口。 这可以是用户界面元素、函数调用、通信元素（如传输控制协议/互联网协议（TCP/IP）和通用串行总线（USB）协议），或用于不同环境变量的物理或逻辑开关的电子信号。
- 架构透明： 架构文档需要提供清晰、易懂的组件和接口，以便在所有测试层面提供可观察性和可控性，并提高质量。

2.1.2 解释如何在不同环境中利用测试自动化

不同类型的自动化测试可以在不同的环境中执行。 这些环境因项目和方法而异，大多数项目都有一个或多个测试环境。 从技术角度看，这些环境可以通过容器、虚拟化软件或其他方法创建。

可以考虑的环境包括以下几种：

本地开发环境

本地开发环境是最初创建软件的地方，并且通过自动化来测试组件的功能适用性。 在本地开发环境中可以进行多种类型的测试，包括组件测试、图形用户界面（GUI）测试和应用程序编程接口（API）测试。 同样重要的是，通过在计算机上使用集成开发环境（IDE），可以执行白盒测试，尽早发现不良编码和质量问题。

构建环境

其主要目的是在 DevOps 生态系统中构建软件并执行测试，以检查构建结果的正确性。该环境可以是本地开发环境，也可以是持续集成/持续交付（CI/CD）代理，在该代理中可以执行低级别测试（即组件测试和组件集成测试）和静态分析，而无需实际部署到任何其他环境。

集成环境

在进行底层测试和静态分析后，下一个阶段是系统集成环境。 在这里，被测系统的候选发布版本与其他能测试的系统完全集成。在这个环境中，可以执行全自动化测试套件，包括用户界面测试或应用程序接口测试。 在这种环境下，没有白盒测试，只有黑盒测试（即系统集成和/或验收测试）。值得注意的是，这是应当加入监控的第一个环境，以了解使用被测系统期间后台发生的情况，从而有效调查缺陷/故障。

预生产环境

预生产环境主要用于评估非功能性质量特性（如性能效率）。虽然非功能测试可在任何环境下进行，但由于试生产环境与生产环境尽可能相似，因此更受关注。 通常，用户验收测试可由业务利益相关方执行，以验证最终产品，如有必要，也可在此执行现有的自动化测试套件。此环境也会受到监控。

生产/运行环境

生产环境可用于实时评估当用户与已部署的系统进行交互时的功能性和非功能性质量特性。 这样的系统具备监控和某些生产环境下进行测试的最佳实践（如金丝雀发布 canary release、蓝/绿部署和 A/B 测试）。

2.2 选择正确工具和策略的评估程序

2.2.1 分析被测系统来确定合适的测试自动化解决方案

每个被测系统可能互不相同，但仍可以通过分析各种因素和特征，以获得成功的测试自动化解决方案（TAS）。在调查被测系统的过程中，测试自动化工程师需要考虑其范围和给定能力来收集需求。从技术角度看，不同类型的应用程序（如网络服务、移动终端和网络）需要不同类型的测试自动化。建议与其他利益相关方（如手工测试人员、业务利益相关方和业务分析师）合作进行调查，以尽可能多地识别风险及其缓解措施，从而为未来提供有益的测试自动化解决方案。

测试自动化方法和架构的需求应考虑以下方面：

- 哪些测试过程活动（如测试管理、测试设计、测试生成和测试执行）应实现自动化。
- 应支持哪些测试级别。
- 应支持哪些测试类型。
- 应支持哪些测试角色和技能组合。
- 应支持哪些软件产品、产品线和系列（例如，确定所实施的测试自动化解决方案的跨度和生命周期）。
- 哪类被测系统需要与测试自动化解决方案兼容。
- 测试数据的可用性及其质量。
- 模拟无法达成的情况（如涉及第三方应用程序）的可能手段和途径。

2.2.2 说明工具评估的技术结论

在对被测系统进行分析并从所有利益相关方处收集需求后，很可能会有一些满足这些需求的测试自动化工具可供考虑。可能没有单一工具能满足所有已确定的要求，利益相关方应认识到这种可能性。

在对照表中记录 1.有关可能工具的发现，2. 这样的发现对各种直接和间接需求的反映，是很有帮助的。对照表的目的是让利益相关方根据具体要求了解工具之间的差异。对照表在列中列出工具，在行中列出需求。单元格中包含每个工具在每个需求方面的属性信息和优先级信息。

一般来说，应评估测试自动化工具，以确定其是否符合上一节（2.2.1）中确定的要求。评估和比较工具时应考虑的需求包括：

- 工具和集成开发环境（IDE）工具的编程语言/技术。
- 配置工具的能力，是否支持不同的测试环境、运行配置以及使用动态或静态设置值。
- 在工具内管理测试数据的能力。 测试数据管理可能被集成在用于版本控制的中央仓库。
- 对于不同的测试类型，可能需要选择不同的测试自动化工具。
- 提供报告功能的能力。这对于与项目的测试报告需求保持一致非常重要。
- 与项目或组织中使用的其他工具集成的能力，如 CI/CD、任务跟踪、测试管理、 报告或其他工具。
- 扩展整体测试架构的能力，评估工具的可扩展性、可维护性、可修改性、兼容性和可靠性。

该比较表是确定用于被测系统测试自动化的建议工具或工具集的良好源头。

决定使用哪种工具的过程可能各不相同，但提案应向适当的利益相关方展示，以获得批准。

3. 测试自动化架构 - 210 分钟（K3）

关键词

行为驱动开发（behavior-driven development）、捕获/回放（capture/playback）、数据驱动测试（data-driven testing）、通用测试自动化架构（generic test automation architecture）、关键字驱动测试（keyword-driven testing）、线性化脚本（ linear scripting）、基于模型的测试（model-based testing）、结构化脚本（structured scripting）、测试适配层（test adaptation layer）、测试自动化框架（test automation framework）、测试自动化解决方案（test automation solution）、测试用具（test harness）、测试脚本（test script）、测试件（testware）、测试步骤（test step）、测试驱动开发（test-driven development）

第 3 章的学习目标：

3.1 测试自动化中使用的设计概念

CTAL-TAE-3.1.1 (K2) 解释测试自动化架构的主要能力
CTAL-TAE-3.1.2 (K2) 解释如何设计测试自动化解决方案
CTAL-TAE-3.1.3 (K3) 在测试自动化框架中应用分层策略
CTAL-TAE-3.1.4 (K3) 不同方法实现自动化测试用例的应用
CTAL-TAE-3.1.5 (K3) 在测试自动化中应用设计原则和设计模式

3.1 测试自动化中使用的设计概念

3.1.1 解释测试自动化架构的主要能力

通用测试自动化架构（gTAA）

通用测试自动化架构是一种高层设计概念，它提供了测试自动化与测试自动化所连接的系统（即被测系统、项目管理、测试管理和配置管理）之间通信的抽象视图（见图 1）。 它还提供了设计测试自动化架构（TAA）时必须涵盖的能力。

通用测试自动化架构（gTAA）的接口描述如下：

- 被测系统接口描述了被测系统与测试自动化框架之间的连接（参见第 3.1.3 节中有关测试自动化框架的内容）。
- 项目管理接口描述测试自动化开发进度。
- 测试管理接口描述测试用例定义和自动化测试用例的映射。
- 配置管理接口描述 CI/CD 流水线环境和测试件。

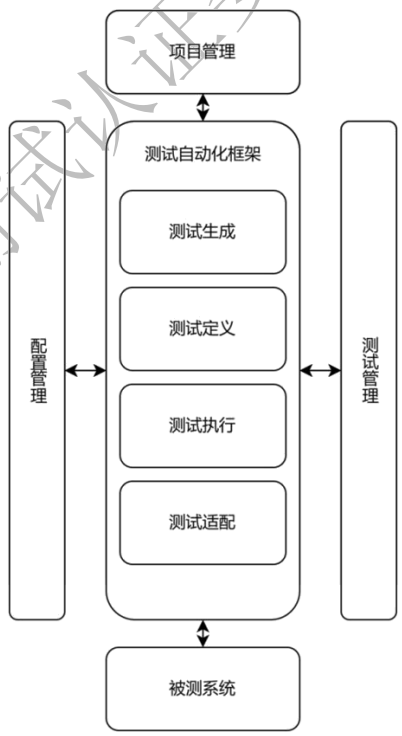

Figure 1: gTTA 图

测试自动化工具和库提供的能力

应根据给定项目的需要，从可用工具中识别和选择测试自动化的核心能力。

测试生成层： 支持根据测试模型自动设计测试用例。在生成过程中，可利用基于模型的测试工具（参见 ISTQB CT-MBT 教学大纲）。 测试生成是一种可选功能。

测试定义层： 支持测试用例和/或测试套件的定义和实现，测试用例和/或测试套件可选择从测试模型中导出。 它将测试定义与被测系统和/或测试工具分开。其包含定义高层和低层测试的手段，这些测试在测试数据、测试用例和测试库组件或其组合中进行处理。

测试执行层： 支持测试执行和测试记录。其提供了一个自动运行选定测试的测试执行工具，以及一个测试记录和测试报告组件。

测试适配层： 提供必要的功能，以适配被测系统（SUT）的各种组件或接口的自动化测试。其提供各种适配器，以便通过应用程序编程接口、协议和服务连接到被测系统。

3.1.2 解释如何设计测试自动化解决方案

测试自动化解决方案（TAS）是通过理解被测系统的功能、非功能和技术要求，以及实施解决方案所需的现有或必要工具来定义的。测试自动化解决方案使用商业或开源工具实施，可能需要额外的被测系统专用适配器。

测试自动化架构定义了整个测试自动化解决方案的技术设计。其应涉及：

- 选择测试自动化工具和工具专用库。
- 开发插件和/或组件。
- 确定连接和接口要求（如防火墙、数据库、统一资源定位器（URL）/连接、 辅助测试用的替身（mock）/桩（stub）、消息队列和协议）。
- 连接测试管理和缺陷管理工具。
- 利用版本控制系统和仓库。

3.1.3在测试自动化框架中应用分层策略

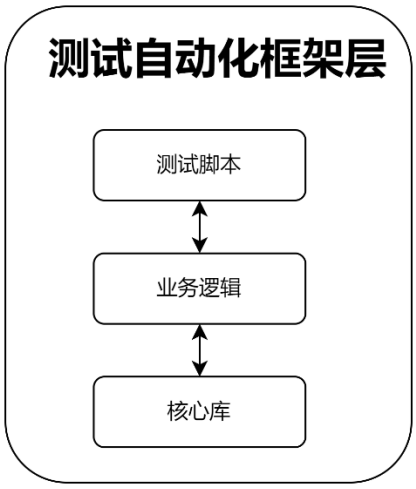

Figure 2: 测试自动化框架层

测试自动化框架

测试自动化框架是测试自动化解决方案的基础。它通常包括测试用具（也称为测试运行器）、测试库、测试脚本和测试套件。

测试自动化框架层

测试自动化框架层定义了具有相似用途的类别（如测试用例、测试报告、测试记录、加密和测试用具）的明确边界。为每个单一目的引入一个层，会使设计变得复杂。因此，建议尽量减少测试自动化框架层的数量。

测试脚本

其目的是提供被测系统的测试用例库以及测试套件的标注信息。它调用业务逻辑层的服务，可能涉及测试步骤、用户流程或 API 调用。不过，测试脚本不应直接调用核心库。

业务逻辑

所有依赖于被测系统的库都存储在这一层。这些库将继承核心库的类文件或使用核心库提供的外观（facades）（参见第 3.1.5 节关于继承和外观的内容）。业务逻辑层用于设置测试自动化框架以针对被测系统和附加配置运行。

核心库

独立于任何被测系统的所有库都存储在这一层。这些核心库可在共享相同开发技术栈的任何类型项目中重复使用。

扩展测试自动化

下面的示例（图 3）展示了核心库如何为多个测试自动化框架提供可重复使用的基础。在项目#1 中，有两个测试自动化框架建立在核心库之上，而另一个项目则利用已有的核心库来建立其测试自动化框架，用于测试应用程序#3。一个测试自动化工程师为项目#1 构建测试自动化框架，而另一个测试自动化工程师则为项目#2 构建测试自动化框架。

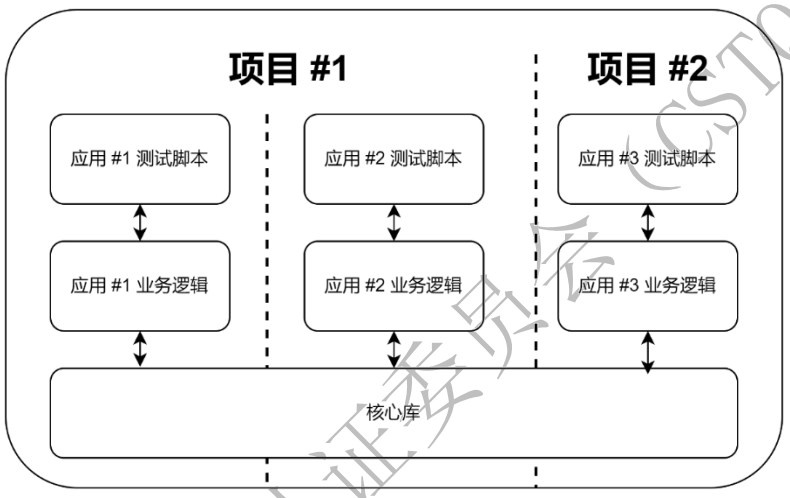

Figure 3: 将核心库应用于多个测试自动化框架

3.1.4 不同方法实现自动化测试用例的应用

团队可以选择多种开发方法来产生自动化测试用例。其中包括交互式脚本语言或编译式编程语言。不同的方法提供了不同的自动化优势，可在不同的情况下加以利用。 虽然测试驱动开发（TDD）和行为驱动开发（BDD）都是开发方法，但如果遵循得当，就能实现自动化测试用例开发。

捕获/回放

捕获/回放是一种在手动执行一系列操作的同时捕获与被测系统的交互的方法。这些工具会在捕获过程中生成测试脚本，依赖于所使用的工具，测试自动化代码可能是可修改的。不公开代码的工具有时被称为无代码测试自动化，而公开代码的工具则被称为低代码测试自动化。

优点：

- 初期易于构建和使用。

缺点：

- 难以维护、扩展和进化。
- 捕获测试用例时，被测系统必须可用。
- 仅适用于小范围和很少变化的被测系统。
- 捕获的被测系统执行在很大程度上取决于捕获时的被测系统版本。
- 记录每个单独的测试用例而不是重复使用现有的构建块相当耗时。

线性化脚本

线性化脚本是一种编程活动，不需要测试自动化工程师自定义测试库，用于编写和执行测试脚本。测试自动化工程师可以利用捕获/回放工具记录的任何测试脚本，然后对其进行修改。

优点：

- 易于构建和开始编写测试脚本。
- 与捕获/回放相比，测试脚本更容易修改。

缺点：

- 难以维护、扩展和演化。
- 捕获测试用例时被测系统必须可用。
- 只适用于小范围和很少变化的被测系统。
- 与捕获/回放相比，需要一定的编程知识。

结构化脚本

在测试库中引入可重复使用的元素、测试步骤和/或用户旅程。 在这种方法中，创建和维护测试脚本需要编程知识。

优点：

- 易于维护、扩展、移植、适配和演化。
- 业务逻辑可与测试脚本分离。

缺点：

- 需要编程知识。

- 测试自动化框架开发及定义测试件的初始投入相当耗费时间。

测试驱动开发

在实现被测系统的新功能之前定义测试用例，是开发流程的一部分。测试驱动开发方法是测试、编码和重构，又称红灯、绿灯和重构。开发人员确定并创建一个会失败的测试用例（红灯）。然后，他/她开发出满足该测试用例的功能（绿灯）。然后对代码进行重构，以优化代码并遵守代码整洁原则。下一个测试和下一个功能增量的过程将继续进行。

优点：

- 简化组件级测试用例开发。
- 改进代码质量和代码结构。
- 提高易测试性。
- 更容易实现所需的代码覆盖率。
- 减少缺陷向更高测试级别的传播。
- 改善开发人员、业务代表和测试人员之间的沟通。
- 通过测试驱动开发，未使用图形用户界面测试和应用程序接口测试验证的用户故事可快速达到出口准则。

缺点：

- 最初需要更多时间适应测试驱动开发。
- 不正确遵循测试驱动开发可能导致对代码质量的错误信心。

数据驱动测试

数据驱动测试（DDT）以结构化脚本方法为基础。测试脚本提供测试数据（如 .csv 文件、.xlsx 文件和数据库转储）。这样就可以使用不同的测试数据多次运行相同的测试脚本。

优点：

- 允许通过数据源快速、轻松地扩展测试用例。
- 可大幅降低添加新自动化测试的成本。
- 测试分析师可通过填充一个或多个描述测试的测试数据文件来编写自动化测试。 这样，测试分析师就能更自由地编写自动化测试，减少对技术测试分析师的依赖。

缺点：

- 可能需要适当的测试数据管理。

关键词驱动测试

关键词驱动测试（KDT）测试用例是由关键词和关键词操作的测试数据组成的测试步骤列表或表格。 关键词是从用户角度定义的。 这种技术通常建立在数据驱动测试的基础上。

优点：

- 测试分析师和业务分析师可以按照关键词驱动测试方法参与创建自动化测试用例。
- 关键词驱动测试也可用于独立于测试自动化的手动测试（参见 ISO/IEC/IEEE 29119-5 关于关键词驱动测试的标准）。

缺点：

- 实现和维护关键字是测试自动化工程师需要涵盖的一项复杂任务，当测试范围扩大时，这可能会变得具有挑战性。
- 对于较小的系统而言，这将造成巨大的工作量。

行为驱动开发

行为驱动开发利用自然语言格式（即 "给定"、"何时 "和 "然后"）制定验收标准，这些标准可用作自动化测试用例并存储在功能特性文件中。然后，行为驱动开发工具就能理解这些语言并执行测试用例。

优点：

- 改善开发人员、业务代表和测试人员之间的沟通。
- 自动化行为驱动开发场景可充当测试用例，确保规约的覆盖范围。
- 可利用行为驱动开发在测试金字塔的不同层次上生成多种测试类型。

缺点：

- 额外的测试用例（通常是负面测试条件和边缘案例）仍需由团队定义，通常由测试分析师或测试自动化工程师定义。
- 许多团队认为行为驱动开发只是一种用自然语言编写测试用例的方法，而没有让业务代表和开发人员参与到整个方法中来。

- 实施和维护自然语言测试步骤对测试自动化工程师来说是一项复杂的任务。
- 过于复杂的测试步骤会使调试工作变得困难和昂贵。

3.1.5 在测试自动化中应用设计原则和设计模式

测试自动化是一项软件开发活动。因此，设计原则和设计模式对于软件开发者来说是很重要的，他们对测试自动化工程师同样重要。

面向对象的编程原则

有四大面向对象编程原则：封装、抽象、继承和多态。

SOLID 原则

它是单一职责、开闭原则、里氏替换、接口隔离和依赖反转的首字母缩写词。这些原则提高了代码的可读性、可维护性和可扩展性。

设计模式

在众多设计模式中，有三种对测试自动化工程师最为重要。

外观模式隐藏了实现细节，只暴露了测试人员在测试用例中需要创建的内容，而单例模式通常用于确保只有一个驱动器与被测系统通信。

在页面对象模型中，创建了一个类文件，并称之为页面模型。每当被测系统的结构发生变化时，测试自动化工程师只需在一个地方进行更新，即页面模型内的定位器，而不是在每个测试用例中更新定位器。

流模型模式是对页面对象模型的扩展。它在页面对象模型上引入了一层额外的外观，该外观存储了与页面对象交互的所有用户操作。通过引入双外观设计，流模型模式提供了改进的抽象性和可维护性，因为测试步骤可以在多个测试脚本中复用。

4 实现测试自动化 - 150 分钟（K4）

关键词

风险（risk）、测试夹具（test fixture）

第 4 章的学习目标：

4.1 测试自动化开发

CTAL-TAE-4.1.1 (K3) 应用支持有效测试自动化试点和部署活动的指南

4.2 与测试自动化开发相关的风险

CTAL-TAE-4.2.1 (K4) 分析测试自动化的部署风险并规划缓解策略

4.3 测试自动化解决方案的可维护性

CTAL-TAE-4.3.1 (K2) 解释哪些因素支持和影响测试自动化解决方案的可维护性

4.1 测试自动化开发

4.1.1 应用支持有效测试自动化试点和部署活动的指南

确定测试自动化试点的验证范围非常重要。试点项目的实施时间并不长，但其结果可能会对项目的发展方向产生重大影响。

根据收集到的被测系统信息和项目要求，应评估以下内容，以制定优化测试自动化工作的指导原则：

- 将要采用的编程语言。
- 合适的现成商业工具/开源工具。
- 要覆盖的测试级别。
- 选定的测试用例。
- 测试用例开发方法。

根据上述要点，测试自动化工程师可以确定一些初步的方法。根据要求，可以创建几个不同的初始原型，以显示不同方法的优缺点。测试自动化工程师可以据此决定采用哪种方法。

确定时间线是满足日程和确保试点成功的重要一环。通常建议去定期检查试点项目的进展情况，以确定并降低任何风险。

在试点期间，还建议尝试将解决方案和已实施的代码集成到 CI/CD 中。这将能够尽早暴露问题，无论是在 被测系统、测试自动化解决方案 中，还是在组织内不同工具的整体集成中。

随着测试用例数量的增加，测试自动化工程师可以考虑改变最初的 CI/CD 设置，以不同的方式在不同的时间运行测试。

此外，在试点项目期间，还需要对其他非技术方面进行评估，例如：

- 团队成员的知识和经验。
- 团队结构。
- 许可和组织规则。
- 测试用例自动化过程中计划涵盖的测试类型和面向的测试级别。

试点项目完成后，测试自动化工程师和测试管理人员应评估项目的成败，并做出适当的决定。

4.2 与测试自动化开发相关的风险

4.2.1 分析测试自动化的部署风险并规划缓解策略

测试自动化框架与被测程序的接口需要作为架构设计的一部分加以考虑。然后再选择打包、测试记录和测试用具。

在试点实施过程中，需要考虑测试自动化代码的扩展和维护。这些都是试验评估阶段的关键因素，会严重影响最终决定。

从试点中可以发现不同的部署风险：

- 开放防火墙。
- 资源利用率（如 CPU 和 RAM）。

必须为部署风险做好准备，如防火墙问题、资源利用、网络连接和可靠性。这些与测试自动化没有严格的联系，但测试自动化工程师需要确保满足所有条件，以便在开发过程中提供可靠、有益的质量关卡。

使用真实设备进行移动测试自动化就是一个例子。移动设备必须接通电源、有足够的电池电量以便在测试期间工作、连接到网络并能访问被测系统。

技术部署风险可能包括：

- 打包
- 日志记录
- 测试结构
- 更新

打包

需要考虑打包问题，因为测试自动化的版本控制与被测系统的版本控制同样重要。测试件可能需要上传到一个存储库，以便在组织内部共享，无论是在内部还是在云中。

日志记录

测试日志记录提供了有关测试结果的大部分信息。有几种测试记录级别，出于各种原因，它们在测试自动化中都很有用：

- 致命：该级别用于记录可能导致测试执行中止的错误事件。
- 错误：当特定条件或交互失败，从而导致测试用例也失败时，使用此级别。
- 警告：当出现意外条件/操作，但没有破坏测试用例的流程时，使用此级别。
- 信息：该级别用于显示测试用例的基本信息和测试执行过程中发生的情况。
- 调试：该级别用于存储执行的具体细节，通常不作为基础日志出现，但在调查测试失败时非常有用。
- 跟踪：该级别与调试级别类似，但信息量更大。

测试结构

测试自动化解决方案中最重要的部分是测试用具和其中包含的测试夹具，它们是测试运行时必须具备的项目。测试夹具可自由控制测试环境和测试数据。可以为测试执行定义前置条件和后置条件，并以多种方式将测试用例组合成测试套件。在试点项目中，这些方面的评估也很重要。此外，测试夹具还能创建可重复和原子化的自动化测试。

更新

最常见的技术风险之一是测试用具（如代理）的自动更新和设备的版本变化。这些风险可以通过充足的电源、适当的网络连接和适当的设备配置计划来降低。

4.3 测试自动化解决方案的可维护性

4.3.1 解释哪些因素支持和影响测试自动化解决方案的可维护性

可维护性在很大程度上受到编程标准和测试自动化工程师对彼此期望的影响。一个黄金法则是遵循罗伯特-C-马丁（Robert C. Martin）提出的代码整洁原则（Robert C Martin，"Clean Code： A Handbook of Agile Software Craftsmanship"，2008 年）。

简而言之，代码整洁原则强调以下几点：

- 对类、方法和变量使用通用的命名规范，以获得有意义的名称。
- 使用符合逻辑和常用的项目结构。
- 避免硬编码。
- 避免为方法设置过多的输入参数。

- 避免冗长复杂的方法。
- 使用日志记录。
- 在有益和需要的地方使用设计模式。
- 注重易测试性。

命名规范（Naming Conventions)非常有助于明确给定变量的作用目标。 使用 "loginButton"（登录按钮）、"resetPasswordButton"（重置密码按钮）等易于理解的变量名有助于测试自动化工程师理解要使用哪个组件。

硬编码是在软件中嵌入数值而无法直接更改的过程。使用数据驱动测试可以避免硬编码，这样测试数据来自一个共同的来源，可以更容易地进行维护。硬编码可减少开发时间，但不建议使用，因为数据会经常变化，维护起来很费时间。对于那些不会经常变化的变量，也建议使用常量。这样做可以减少需要维护的源及位置。

此外，还强烈建议使用设计模式。如 3.1.5 所述，只要正确使用设计模式，就能实现结构合理、可适当维护的测试自动化代码。

为确保测试自动化代码的高质量，建议使用静态分析器。集成开发环境（IDE）中常用的代码格式化工具将提高测试自动化代码的可读性。

除了代码整洁原则，建议在版本控制中使用约定的分支结构和策略。为功能、版本和缺陷修复使用不同的分支有助于理解分支内容。

5 测试自动化的实现与部署策略 - 90 分钟 (K3)

关键词

契约测试（contract testing）

第 5 章学习目标：

5.1 集成到持续集成/持续交付(CI/CD)流水线

CTAL-TAE-5.1.1	(K3)	在 CI/CD 流水线中应用各测试级别的测试自动化
CTAL-TAE-5.1.2	(K2)	解释测试件的配置管理
CTAL-TAE-5.1.3	(K2)	解释应用程序编程接口（API）基础设施的测试自动化依赖性

5.1 集成到持续集成/持续交付(CI/CD)流水线

5.1.1 在流水线中应用各测试级别的测试自动化

测试自动化的一个主要好处是可以无人值守地运行实现的测试，使其非常适合在流水线中运行。这可以通过 CI/CD 流水线或定期执行测试的流水线来实现。

测试级别通常按以下方式集成：

- 构建过程中的测试自动化框架（TAF）/测试自动化解决方案（TAS）的配置测试可视为组件测试的一个变种。这些测试在测试自动化项目构建期间运行，用于检查测试脚本中所有使用的文件路径是否正确，文件是否真实存在并位于指定路径中。
- 组件测试是流水线构建步骤的一部分，它们针对单个组件（如类库和 Web 组件）执行。它们作为流水线的质量检查关卡，是持续集成流水线中的关键环节。
- 如果是针对底层组件或被测系统（SUT）的测试，组件集成测试可以作为持续集成流水线的一部分。在这种情况下，这些测试会与组件测试一起执行。
- 系统测试通常可以集成到持续部署流水线中，作为交付 SUT 的最后一道质量检查关卡。
- 不同系统组件之间的系统集成测试通常作为质量检查关卡集成到持续交付流水线中。这些系统集成测试确保分别开发的系统组件能够协同工作。

许多现代持续集成系统区分持续交付流水线的构建和部署阶段。在这些情况下，组件测试和组件集成测试属于第一个构建阶段。当第一阶段（即构建和测试）成功完成后，组件/被测系统(SUT)将被部署。

对于系统集成测试、系统测试和验收测试，有两种主要方法将其集成到此类流水线中：

1. 测试用例作为部署阶段的一部分，在组件部署后执行。这种方式的好处是可以根据测试结果判断部署是否失败并进行回滚。但是，如果需要重新运行测试，就必须进行重新部署。
2. 测试用例在独立的流水线中执行，由成功的部署触发。如果预期每次部署都需要运行不同的测试套件和各种自动化测试代码，这种方式会很有好处。在这种情况下，测试不会起到质量关卡的作用。因此，要回滚不成功的部署，还需要其他操作，通常是手动操作。

在这种情况下，可以使用一些简单的自动化测试脚本作为部署检查，以确保被测系统(SUT)已得到部署，但这些自动测试脚本无法全面验证功能的适用性。

流水线还可以用于其他测试自动化目的，比如：

- 定期运行不同的测试套件：可以在每天晚上运行回归测试套件（即夜间回归测试），这对于那些运行时间较长的测试套件特别有用，这样团队在第二天早上就能清楚地了解被测系统(SUT)的质量状况。
- 运行非功能测试：可以作为持续部署流水线的一部分，也可以独立运行，用于定期监控系统的某些非功能质量特征，如性能效率。

5.1.2 解释测试件的配置管理

配置管理是测试自动化不可或缺的一部分，因为自动化测试通常需要在多个测试环境和被测系统(SUT)的不同版本上执行。

测试自动化中的配置管理包括：

- 测试环境配置。
- 测试数据。
- 测试套件/测试用例。

测试环境配置

开发流水线中使用的每个测试环境可能具有不同的配置，例如不同的 URL 或访问凭证。测试环境配置通常与测试件一起存储。然而，当测试自动化用于多个项目或同一项目使用多个测试自动化框架（TAF）时，测试环境配置可以作为公共核心库的一部分或存放在共享仓库中。

测试数据

测试数据可能因测试环境或被测系统(SUT)的发布版本和功能集而异。与测试环境配置类似，测试数据通常与较小的测试自动化框架 TAFs 一起存储，但也可以使用测试数据管理系统。

测试套件/测试用例

一种常见做法是根据不同目的设置不同的测试用例套件，例如冒烟测试或回归测试。这些测试套件通常在不同的测试级别上执行，使用不同的流水线和测试环境。

被测系统(SUT)的每个发布版本都确定了一个功能集，包含用于评估该版本质量的测试用例和测试套件。测试件中有不同的方式来处理这种情况：

- 可以为每个发布版本或测试环境定义功能开关配置。每个功能都有相应的测试用例和测试套件。功能开关可以在测试件中用来识别在给定发布版本/测试环境上执行哪些测试套件。

- 测试件也可以与被测系统(SUT)使用相同的版本号一起发布。这样可以确保被测系统(SUT)版本与用于测试它的测试件精确匹配。这种发布通常通过配置管理系统使用标签或分支来实现。

5.1.3 解释应用程序编程接口（API）基础设施的测试自动化依赖关系

在执行应用程序编程接口（API）测试自动化时，了解以下依赖关系信息对于制定合适的策略至关重要：

- 应用程序编程接口（API）连接：理解可自动化测试的业务逻辑以及应用程序编程接口（API）之间的相互关系。

- 应用程序编程接口（API）文档：作为测试自动化的基准，包含所有相关信息（例如：参数、请求头和各种类型的请求-响应对象）。

集成自动化应用程序编程接口（API）测试可以由开发人员或测试自动化工程师（TAE）来完成。然而，随着左移测试的推行，建议支持并划分不同层次的测试。在 ISTQB CTFL 大纲中，提到了组件集成测试和系统集成测试，这可以通过一种称为契约测试的最佳实践来扩展。

契约测试

契约测试是一种集成测试，用于验证服务之间能否相互通信，以及服务间共享的数据是否符合特定的规则集。使用契约测试可以确保两个独立系统（例如两个微服务）之间的通信兼容性。它超越了模式验证的范畴，要求双方就允许的交互集达成共识，同时提供随时间演进的能力。它捕获各项服务之间的交互，将其存储在契约中，然后可用于验证双方是否都遵守该契约。它捕获服务之间交换的交互信息，将其存储在契约中，这可用来验证双方是否都遵守该契约。这种测试类型的主要优势之一是可以在软件开发生命周期（SDLC）的早期发现底层服务的缺陷，并且更容易识别这些缺陷的来源。

在消费者驱动的契约测试方法中，消费者设定其期望，确定提供者应如何响应来自该消费者的请求。在提供者驱动的契约测试方法中，提供者创建契约，展示其服务是如何运行的。

6 测试自动化报告和度量 - 150 分钟 (K4)

关键字

测量（measurement）、度量（metric）、测试日志（test log）、测试进度报告（test progress report）、测试步骤（test step）

第 6 章学习目标：

6.1 测试自动化数据的收集、分析和报告

CTAL-TAE-6.1.1	(K3)	从测试自动化解决方案（TAS）和被测系统(SUT)中收集数据的方法的应用
CTAL-TAE-6.1.2	(K4)	分析从测试自动化解决方案（TAS）和被测系统(SUT)获取的数据以更好地理解结果
CTAL-TAE-6.1.3	(K2)	解释测试进度报告如何构建和发布

6.1 测试自动化数据的收集、分析与报告

6.1.1 从测试自动化解决方案（TAS）和被测系统(SUT)中收集数据的方法的应用

数据可从以下来源收集：

- 被测系统（SUT）日志。
  - Web 端/移动端用户界面。
  - 应用程序编程接口（APIs）。
  - 应用程序。
  - Web 服务器。
  - 数据库服务器。
- 测试自动化框架（TAF）日志，用于提供审计追踪。
- 构建日志。
- 部署日志。
- 生产环境监控日志（参见 ISTQB CT-PT 大纲第 2.3 节）。
  - 用于进行趋势分析的生产环境性能监控。
  - 性能测试环境中的性能效率测试日志（例如：负载测试、压力测试和尖峰测试）。
- 截图和屏幕录制（来自自动化工具自带功能或第三方工具）。

由于测试自动化解决方案（TAS）的核心是自动化测试件，因此可以对其进行增强以记录其使用信息。对底层测试件的增强可被所有高层级的自动化测试脚本使用。例如，增强底层测试件以记录测试执行的开始和结束时间，这种功能可以应用于所有测试。

支持测量和测试报告生成的测试自动化功能特性

许多测试工具的脚本语言都支持测量和报告功能，这些功能可用于在执行单个测试和整个测试套件之前、期间和之后捕获和记录信息。

一系列测试运行的每一次测试报告都需要有分析功能，以考虑之前测试运行的测试结果，从而突出显示趋势，如测试成功率的变化。

测试自动化通常需要测试执行和测试验证的自动化，后者是通过比较实际结果中的特定元素与预期结果来实现的。这种比较最好使用测试工具的断言功能来完成。必须考虑比较结果所报告信息的级别。正确确定测试状态（即通过或失败）非常重要。如果是失败状态，则需要更多有关失败原因的信息（如屏幕截图）。

测试的实际结果和预期结果之间的差异并不总是清晰的，而工具支持在定义比较时可以提供很大帮助，忽略预期的差异，如日期和时间，同时突出显示任何意外的差异。

测试记录

测试日志是经常用来分析测试自动化解决方案（TAS）和被测系统（SUT）中潜在缺陷的来源。以下部分是按 TAS 和 SUT 分类的测试记录的例子。

测试自动化解决方案(TAS)日志记录

以下内容决定了是由测试自动化框架还是测试执行过程中负责记录信息：

- 当前正在测试执行的是哪个测试用例，包括开始时间和结束时间。
- 测试执行的状态，尽管失败可以在测试日志中轻易识别，但测试自动化框架（TAF）也应掌握这些信息并且通过仪表板报告。测试执行的状态可以是通过、失败或测试自动解决方案（TAS）失败。状态有时无法确定，因此对于一个组织来说，为它们制定明确且一致的状态定义是非常重要的。测试自动化解决方案（TAS）失败适用于缺陷不在被测系统中(SUT)的情况。
- 测试日志的低级别详细信息（例如：记录重要的测试步骤），包括时间信息。
- 测试用例在第三方工具帮助下能够识别被测系统(SUT)的动态信息（例如：内存泄漏）。当检测到失效时，实际结果和失效应该与执行的测试套件一起记录。
- 在可靠性测试或压力测试中需要多次执行测试循环时，应该有记录计数器以便轻松确定测试用例的执行次数。
- 当测试用例包含随机元素时（例如：状态转换测试中的随机参数或随机测试步骤），应记录随机数/选择。
- 测试用例执行的所有操作应以日志的形式记录下来，以便能够使用相同的测试步骤和相同的时间间隔来回放日志（或其部分）来重新运行测试。这对于重现已识别的故障以及捕获额外信息非常有用。此外，被测系统（SUT）也可以记录测试用例的操作信息，以便在重现客户识别的故障时使用。如果客户运行了一组测试，测试日志信息将被捕获，随后开发团队在排查缺陷时可以回放这些日志。

- 测试执行过程中可以保存屏幕截图，用于后续的根本原因分析。
- 每当测试套件触发失效时，测试自动化解决方案（TAS）应确保所有分析缺陷所需的信息都已保存或可用，同时还要包括与测试继续执行的相关信息（如适用）。测试自动化解决方案（TAS）应保存所有相关的崩溃转储和堆栈跟踪信息。此外，任何可能被覆盖的测试日志（例如：被测系统(SUT)上常用的循环缓冲区）都应另外存储以供后续分析。
- 使用不同颜色可以帮助区分不同类型的测试日志信息（例如：用红色标识缺陷，用绿色标识进度信息）。

被测系统(SUT)日志记录

将测试自动化结果与被测系统(SUT)日志进行关联分析，有助于确定被测系统(SUT)和测试自动化解决方案（TAS）中缺陷的根本原因。

- 当在被测系统(SUT)中发现缺陷时，应记录分析缺陷所需的所有必要信息，包括：日期和时间戳，缺陷的源代码位置，错误消息。
- 系统启动时，应将配置信息记录到文件中，主要包括：不同的软件/固件版本，被测系统(SUT)的配置信息，操作系统的配置信息。
- 使用测试自动化，可以方便地搜索被测系统(SUT)日志。测试自动化解决方案（TAS）在测试日志中识别的失效，应该能够无论是否使用额外工具都能便捷的在被测系统(SUT)的测试日志中找到，反之亦然。使用时间戳同步各种测试日志，有助于关联分析失败报告当时发生的情况。

与其他第三方工具的集成（例如：电子表格、XML 文档、Doc 文档、数据库和报告工具）

当自动化测试用例执行的信息需要用于其他跟踪和报告工具时（例如：更新可追溯性信息），应当能够以适合第三方工具的格式提供这些信息。这通常可以通过测试工具的现有功能（例如：测试报告的导出格式）来实现，或者通过创建与其他软件格式兼容的自定义报告来实现。

测试结果可视化

测试结果可以通过图表形式展示。考虑使用彩色图标（如红绿灯）来指示测试执行/测试自动化的整体状态，以便根据报告的信息做出决策。管理层特别关注测试结果的可视化摘要，这有助于他们进行决策。如果需要更多信息，他们还可以深入查看详细内容。

6.1.2 分析从测试自动化解决方案（TAS）和被测系统(SUT)获取的数据以更好地理解测试结果

测试执行完成后，分析测试结果很重要，需要识别被测系统(SUT)和测试自动化解决方案（TAS）中可能存在的失效。在这种分析中，从测试自动化解决方案（TAS）收集的数据是主要的，从被测系统（SUT）收集的数据是次要的。

- 分析测试环境数据以支持测试自动化的适当规模调整（例如：在云环境中）。
  - 集群和资源（例如：CPU 和内存）。
  - 单浏览器与多浏览器（即跨浏览器）测试执行。
- 比较之前测试执行的结果。
- 确定如何使用 Web 日志监控软件使用情况。

需要分析测试执行失效，因为存在潜在问题：

1. 检查之前的测试执行中是否发生过相同的失效。这可能是被测系统(SUT)或测试自动化解决方案（TAS）中的已知缺陷。测试自动化解决方案（TAS）可以设计为记录测试用例的历史测试结果，从而进一步帮助分析。
2. 如果是未知缺陷，需要识别测试用例及其测试内容。这测试用例可以是自解释的，或者是可以根据测试执行时记录的 ID 在测试管理系统中识别出来。
3. 找出测试用例中哪个测试步骤发生了失效。测试自动化解决方案（TAS）会记录这些信息。
4. 通过分析截图、应用程序编程接口、网络日志及其他任何能够显示被测系统状态的日志来验证是否与预期结果匹配。
5. 如果被测系统(SUT)的状态与预期不符，在缺陷管理系统中记录缺陷。确保包含所有必要的缺陷信息和证明其为缺陷的日志。

当出现失效时，被测系统(SUT)的实际结果仍然可能与预期结果匹配。在这种情况下，很可能是测试自动化解决方案（TAS）存在需要修复的缺陷，或者存在一个不可见的不匹配。

另一种可能发生的情况是测试环境在测试运行期间不可用或仅部分可用。在这种情况下，所有测试用例都可能失败，要么是因为相同的缺陷，要么是因为系统部分停机而出现看似真实的失效。要识别此类缺陷的根本原因，可以分析被测系统(SUT)日志，这些日志会显示测试运行时是否存在任何测试环境中断。

如果被测系统(SUT)为用户交互（即 UI 会话或 API 调用）实现了审计日志，这将有助于分析测试结果。系统通常会为每个交互添加一个唯一 ID，该 ID 在后续的调用和系统集成中保持一致。通过这种方式，只要知道请求/交互的唯一 ID，就可以观察和追溯系统的行为。这个唯一 ID 通常被称为关联 ID 或追踪 ID。测试自动化解决方案（TAS）可以记录这个 ID，以帮助分析测试结果。

6.1.3 解释如何构建和发布测试进度报告

测试日志详细记录了测试步骤、执行动作以及测试用例或测试套件的预期响应，然而，仅凭测试日志无法很好地概括整体测试结果。因此，需要具备测试报告功能。测试套件执行完成后，必须创建并发布一份简明的测试进度报告。可以使用报告生成器来完成这项工作。

测试进度报告的内容

测试进度报告必须包含测试结果、被测系统信息(SUT)和测试环境的文档，这些信息应以适合每个利益相关者的格式来记录。

需要知道哪些测试失败了以及失败的原因。为了使失效调查更容易，了解测试执行历史和报告人（即通常是创建或最后更新它的人）很重要。负责人需要调查失败的原因，报告相关缺陷，跟进缺陷修复并验证修复是否正确。

测试报告也用于诊断测试自动化框架（TAF）组件的任何失败。

发布测试报告

测试报告应该发布给所有利益相关者。它可以上传到网站（云端或本地的），也可以发送到邮件列表或上传到其他工具，例如测试管理工具。个人可以通过电子邮件订阅报告或通过聊天机器人发布的聊天消息接收报告，这将有助于确保报告得到审核和分析。

另一种方法是识别被测系统（SUT）的问题部分，并保留测试报告历史记录，从而收集到高频回归的测试用例或测试套件的统计数据，用于趋势分析。

需要报告的利益相关者包括：

- 管理利益相关者。
  - 典型角色：解决方案或企业架构师、项目/交付经理、项目经理、测试经理或测试总监。
- 运营利益相关者。
  - 典型角色：产品负责人/经理、业务代表或业务分析师。

- 技术利益相关者
  - 典型角色：团队领导、Scrum 主管、网站管理员、数据库开发人员/管理员、测试组长、测试自动化工程师（TAE）、测试员或开发人员。

测试报告的内容和详细程度可能因接收者而异。技术利益相关者可能更关注底层细节，而管理层则更关注趋势，例如自上次测试运行以来增加了多少测试用例，通过/失败比率的变化以及测试自动化解决方案（TAS）和被测系统（SUT）的可靠性。运营利益相关方通常会更加重视与产品使用相关的指标。

仪表盘的创建

现代报告工具通过仪表盘、彩色图表、详细日志收集和自动化测试日志分析提供了多种报告选项。市场上有许多可用的工具可选择。

这些工具支持从流水线执行测试日志、项目管理工具和代码库等来源汇总数据。这些工具提供的可视化数据可帮助利益相关者了解趋势并做出相应决策。这些趋势可能包括缺陷聚类、特定测试环境中缺陷传播的增减、被测系统（SUT）性能退化以及构建的可靠性。

测试日志的人工智能/机器学习分析

近年来，一些测试自动化工具包含或基于机器学习（ML）算法。对测试日志中大量数据的自动化分析可以帮助测试自动化工程师减少以下工作所需的时间：

查找破坏了的定位器。

分析测试失败的原因（即判断是被测系统（SUT）还是测试自动化解决方案（TAS）中的缺陷）。

对测试报告中的常见缺陷进行分组 （参见 ISTQB CT-AI 大纲）。

7 验证测试自动化解决方案 - 135 分钟（K3）

关键字

静态分析（static analysis）

第 7 章的学习目标：

7.1 测试自动化基础设施的验证

CTAL-TAE-7.1.1	(K3)	计划验证测试自动化环境，包括测试工具搭建
CTAL-TAE-7.1.2	(K2)	解释给定自动化测试脚本和/或测试套件的正确行为
CTAL-TAE-7.1.3	(K2)	识别测试自动化产生不符合预期结果的位置
CTAL-TAE-7.1.4	(K2)	解释静态分析如何帮助提高测试自动化代码质量

7.1 测试自动化基础设施的验证

7.1.1 计划验证测试自动化环境，包括测试工具搭建

需要验证测试自动化环境和测试自动化解决方案（TAS）的所有其他组件是否按预期工作。这些检查通常在启动测试自动化之前进行。可以采取相应步骤来验证测试自动化环境的各个组件。下面将更详细地解释这些步骤。

测试工具的安装、搭建、配置和定制

测试自动化解决方案（TAS）由许多组件组成。每个组件都需要确保可靠和可重复的性能。测试自动化解决方案（TAS）的核心包括可执行组件、相应的功能库以及支持数据和配置文件。配置 TAS 的过程既包括使用自动安装脚本，也包括手动将文件放入相应文件夹。与操作系统和其他软件类似，测试工具也会定期推出服务补丁包，可能有可选或必选的插件以确保与任何给定的被测系统（SUT）环境兼容。

自动安装或从代码仓库复制具有其优势。它可以保证在不同的被测系统（SUT）上使用相同版本的测试自动化解决方案（TAS）和相同的测试自动化解决方案（TAS） 配置（在适用情况下）进行测试。可通过代码仓库对 TAS 进行升级。代码仓库的的使用和 TAS 升级到新版本的流程应与标准开发工具相同。

测试环境搭建/拆卸的可重复性

测试自动化解决方案（TAS）将在各种系统、服务器上实施，并支持 CI/CD 流水线。为确保 TAS 在每个测试环境中都能正常工作，有必要采用系统化的方法从任何给定的测试环境中加载和卸载 TAS。当 TAS 的构建和重建在单个测试环境中和多个测试环境中的运行方式没有明显差异时，就能成功实现这一目标。TAS 组件的配置管理确保了可靠地创建给定的配置。一旦配置完成，对构成 TAS 的各种组件进行记录，它能提供必要的知识，了解当 SUT 环境发生变化时，TAS 的哪些方面可能受到影响或需要更改。

与内部和外部系统/接口的连接性

一旦在给定的被测系统（SUT）环境中安装了测试自动化解决方案（TAS），在使用被测试系统（SUT）之前，应该进行一系列检查或前置条件验证，以确保与内部系统、外部系统的接口连接可用。例如，登录服务器，启动测试自动化工具，验证测试自动化工具是否可以访问被测系统，手动检查配置设置，并确保系统之间的测试日志和测试报告权限设置正确，都是一种良好的实践。建立测试自动化的前置条件对于确保 TAS 已正确安装和配置至关重要。

测试自动化框架(TAF) 组件测试

就像任何软件开发项目一样，TAF 组件也需要进行单独测试和验证。这包括功能和非功能测试（例如性能效率测试和资源利用率测试）。举例来说，在图形用户界面（GUI）系统中提供对象验证的组件，需要针对各种不同类型的对象进行测试，以确定对象验证功能的正确性。同样，测试日志和测试报告也应该能够准确反映测试自动化状态和被测系统（SUT）行为。非功能测试的例子包括评估 TAF 性能下降情况、监控系统资源利用情况（可能表明存在内存泄漏缺陷）以及 TAF 内部和外部组件缺乏互操作性。

7.1.2 解释给定自动化测试脚本和/或测试套件的正确行为

需要对自动化测试套件的完整性、一致性和正确行为进行测试。可采用不同类型的验证检查，以确保自动测试套件在任何给定时间都可用，或者确定其是否适合使用。

可以采取以下步骤来验证自动化测试套件。这些包括：

- 检查测试套件的组成。
- 验证针对测试自动化框架（TAF）新功能的测试。
- 考虑测试的可重复性。
- 考虑自动测试工具的侵入性 。

下面将详细解释每一个步骤。

检查测试套件的组成

检查完整性（如检查测试用例都有预期结果，检查测试数据是否存在），以及检查测试自动化框架 （TAF）和被测系统（SUT）的版本是否正确。

验证针对测试自动化框架（TAF）新功能的新测试

在测试用例中首次使用 TAF 的新功能时，应对其进行验证和密切监控，以确保该功能正常工作。

考虑测试的可重复性

当重复执行测试时，测试结果应始终相同。如果测试套件中的测试用例无法提供稳定的测试结果（例如，由于竞争条件），则应当将其从当前活动的自动测试套件中移出，单独进行分析以找出根本原因。否则，就会反复花费时间在这些测试运行上来分析失效。

考虑自动化测试工具的侵入性

测试自动化解决方案（TAS）通常与被测系统（SUT）紧密耦合。这样设计是为了在交互层面上有更好的兼容性。然而，这种紧密集成也可能导致不良后果。例如，当 TAS 位于 SUT 环境中时，SUT 的功能可能与手动进行测试时不同，这也潜在影响性能。

高度的入侵可能会在测试过程中出现在生产环境中并不存在的失效。如果这导致自动化测试失败，就会对 TAS 的信心急剧下降。开发人员可能会要求手动重现测试自动化发现的失败，以协助分析。

7.1.3 识别测试自动化产生不符合预期结果的位置

当测试脚本意外的失败或意外的通过时，必须进行根本原因分析。这包括检查测试日志、性能数据、测试脚本的初始化和清理过程。

执行一些隔离的测试也有助于问题分析。间歇性出现的失败更难分析。缺陷可能存在于测试用例、被测系统（SUT）、测试自动化框架（TAF）、硬件或网络中。监控系统资源可为找出根本原因提供线索。对测试用例、SUT 和 TAF 的测试日志文件进行分析，有助于识别缺陷的根本原因。可能还需要进行调试。为了帮助识别根本原因，可能需要测试分析师、业务分析师、开发人员或系统工程师的支持。

验证是否所有断言已到位。缺少断言可导致测试结果不确定。

7.1.4 解释静态分析如何帮助提高自动化测试代码质量

静态代码分析有助于发现程序代码中的漏洞和缺陷。这包括被测系统（SUT）或测试自动化框架（TAF）。

自动化扫描检查代码可以降低风险。这种方式提供了对 SUT 进行缺陷审查，确保代码质量标准得到满足和应用。这可视为一种主动缺陷检测技术，在 DevSecOps 实施中发挥着重要作用（即强调安全性的 DevOps）。这些扫描在软件开发生命周期（ SDLC） 的早期通过流水线进行，为开发团队提供即时反馈。

有关缺陷的输出结果通常被分为严重、高、中或低等级，这样开发团队就能确定修复缺陷的优先顺序。某些静态分析工具还能提供代码修复建议，以解决它们发现的缺陷。它会向开发团队展示有问题的代码行，并为开发人员提供可能的补救措施。此外，这些工具还通过以下方式帮助测试自动化工程师（TAE）： 衡量质量、建议需要添加注释的代码区域、改进代码设计以优化资源处理（例如，利用 try/catch 块和更好的循环结构），以及删除不良的库调用。

由于测试自动化工具使用编程语言，因此可能存在将质量不佳的测试自动化代码引入软件开发生命周期（SDLC）的风险。举个简单的例子，在自动化测试中使用用户名和密码是一种常见做法。可以想到，测试自动化工程师（TAE）可能会在一个或多个测试脚本中错误地以明文形式包含密码。

静态分析工具能使测试自动化代码质量受益。它们可以用于分析测试自动化代码中的安全违规问题，例如代码中的明文密码。静态分析工具支持多种编程语言，包括测试自动化软件使用的语言。因此，测试自动化工程师（TAE）势必将扫描代码的最佳实践扩展到测试自动化代码。即使测试自动化代码不一定与整体软件一起部署，但如果在伴随的自动化测试脚本中发现了密码，则明显存在潜在漏洞（参见 ISTQB CT-SEC 教学大纲）。

8 持续改进 - 210 分钟（K4）

关键字

模式验证（schema validation）、测试直方图（test histogram）

第 8 章的学习目标：

8.1 测试自动化的持续改进机会

TAE-8.1.1	(K3)	通过数据收集和分析发现改进测试用例的机会
TAE-8.1.2	(K4)	分析已部署的测试自动化解决方案的技术方面，并提供改进建议
TAE-8.1.3	(K3)	重构自动化测试件，使其与 SUT 更新保持一致
TAE-8.1.4	(K2)	总结使用测试自动化工具的机会

8.1 测试自动化的持续改进机会

8.1.1 通过数据收集和分析发现改进测试用例的机会

在考虑各种类型的数据时，可以通过以下方法改进数据收集和分析。

测试直方图

以直方图形式展示的测试数据可视化报告，帮助发现有关测试用例数据趋势中的潜在改进领域。由于许多持续集成/持续交付(CI/CD) 和测试报告工具都具备能力来显示不同的测试结果及其各自的测试数据（如异常日志、错误信息和屏幕截图），因此测试自动化工程师（TAE）可以决定可能的改进领域。测试直方图还能让 TAE 识别和选择那些脆弱的测试用例，并通过额外的改进或重新思考实际实现方式来重构它们。

人工智能

另一个最近的机会是利用人工智能（AI）来支持测试和测试自动化。例如，在用户界面（UI）测试用例中，数据还包括可被视为输入的 UI 定位器值。最新的前沿工具显示，它们有能力检测给定定位器是否发生变化。基于机器学习（ML） 和图像识别，它们可以识别新的选择器，并使用自愈算法修复测试用例，并在测试报告中包含已更改的定位器。这可以加快版本控制更改和代码维护等后续步骤。

模式验证

模式验证可应用于应用程序接口（API）数据分析（例如，来自目标端点的属性）和数据库分析（例如，软件字段的验证规则，允许的数据类型和值范围）。

通过模式验证，测试自动化解决方案（TAS）能够检查响应是否与实际业务规范相匹配。这种检查可用于确定服务响应中是否存在必须的响应元素，以及这些元素的对象类型是否与模式中定义的对象类型相匹配。如果模式发生破坏，解决方案会返回实际的验证结果，帮助 TAE 找出问题的根本原因。

示例：一个应用程序编程接口（API）有六个强制性响应元素，这些元素在响应中必须是字符串。使用模式验证工具，就无需编写单独的断言来检查这些类型是否为字符串，它们的值是否不为空。模式验证将处理这些检查，从而使实现的测试自动化代码大大缩短，并提高检测后端服务缺陷的效率。

8.1.2 分析已部署的测试自动化解决方案（TAS）的技术方面并提出改进建议

除了使测试自动化解决方案（TAS）与被测系统（SUT）保持同步所需的持续维护任务外，还有许多改进 TAS 的机会。这些改进可带来一系列好处，包括更高的效率（例如，进一步减少人工干预）、更好的易用性、更多的功能以及更好的测试支持。至于如何改进 TAS 的决策，则取决于哪些功能能为项目带来最大价值。

可考虑改进测试自动化解决方案（TAS）的具体领域包括脚本编写、测试执行、验证、测试自动化架构（TAA）、测试自动化框架（TAF）、环境搭建和清理、文档化、TAS 功能以及 TAS 更新和升级。这些将在下面更详细地描述。

脚本编写

如 3.1.4 节所述，脚本编写技术从线性脚本编写方法到数据驱动测试方法，再到更复杂的关键字驱动测试方法，不一而足。对于所有新开发的自动化测试，升级当前的 TAS 脚本编写技术可能是合适的。这种技术可能会用于所有现有的自动化测试的改造中，或至少是那些涉及最大维护工作量的测试中。

测试自动化解决方案（TAS）改进的另一个领域可能集中在测试脚本的实现上。例如：

- 评估测试脚本/测试用例/测试步骤的重叠情况，以整合自动化测试。包含相似操作序列的测试用例不应反复实现这些测试步骤。应将这些测试步骤封装成函数并添加到库中，以便重复使用。这些库函数然后可以被不同的测试用例使用。这增加了测试件的可维护性。当测试步骤不完全相同但相似时，可能需要参数化。注意：这是关键字驱动测试中的典型方法。
- 为测试自动化解决方案（TAS）和被测系统（SUT）建立失效恢复程序。当测试套件在执行过程中发生失效时，TAS 应能从中恢复，并继续进行下一个的测试用例。当 SUT 出现失效时，TAS 需要在切实可行的情况下对 SUT 执行必要的恢复操作（如重新启动 SUT）。
- 评估等待机制，以确保使用最佳类型。有三种常见的等待机制：
  - 硬编码等待（即等待一定数量的毫秒数），考虑到软件响应时间的不可预测性，这可能是许多测试自动化缺陷的根本原因。
  - 通过轮询进行动态等待（例如，检查是否发生了特定的状态变化或操作）是更加灵活和高效：
    - TAS 只等待所需的时间，不会浪费测试时间。

- 当过程耗时超过预期时，轮询将一直等待到条件为真。记住要有超时机制，否则，如果出现缺陷，测试可能会永远等待下去。

- 一个更好的方法是订阅 SUT 的事件机制。这种方法比其他两种方法更可靠，但要求测试脚本语言需要支持事件订阅，SUT 需要向 TAS 提供这些事件。同样需要超时机制，否则如果出现缺陷，测试可能会永远等待。

测试执行

当自动回归测试套件因测试执行时间过长而无法完成时，可能需要在不同的测试环境中同时进行测试（如果可能的话）。当使用昂贵的系统进行测试时，所有测试都必须在单个目标系统上完成，这可能是一种限制。可能有必要将回归测试套件分成多个部分，每个部分在规定的时间内（例如，一个晚上）执行。进一步分析测试自动化覆盖率可能发现重复测试。消除重复可以减少测试执行时间，并进一步提高效率。在持续集成/持续发布(CI/CD)的情况下，一个好的做法是并行运行批处理作业，以优化测试执行时间。此外，在给定时间（如每天早上），安排自动化批处理作业去运行不同的流水线也是一种很好的做法，这样可以减少手动交互，加快开发流程。

验证

在创建新的验证功能之前，采用一套标准验证方法供所有自动测试使用。这将避免在多个测试中重复实现验证操作。当验证方法不完全相同但相似时，使用参数化将有助于允许一个函数被用于多种类型的对象。

测试自动化架构（TAA）

可能有必要更改测试自动化架构（TAA）以支持被测系统（SUT）可测试性的改进。这些更改可在 SUT 的架构和/或测试自动化解决方案（TAS）的 TAA 中进行。这可大大改进测试自动化，但可能需要对 SUT/TAS 进行重大变更和投资。例如，如果要更改 SUT 以提供用于测试的应用程序编程接口（API），那么测试自动化解决方案（TAS）也应进行相应的重构。在软件开发生命周期（SDLC）的后期添加这类功能代价会相当大；最好在测试自动化开始时和 SUT 的 SDLC 早期阶段就考虑到这一点。

测试自动化框架（TAF）

通常，测试自动化框架中使用的核心库会有新版本。有时这些是重大更新，最新版本不能立即在测试自动化框架的依赖列表中引用，因为这会破坏许多使用这些库的团队的测试。因此，最好先进行试点和影响分析。之后，可以创建一个采用计划。要么所有团队同时采用新版本的核心库，更新核心库层构建文件中的依赖关系，要么每个团队各自决定何时在其业务逻辑层中更新它。最终，一旦所有团队都准备好接受新版本的核心库，就可以更新核心库层中的依赖关系（见第 3.1.3 节）。

初始化和清理

在每个测试脚本或测试套件之前或之后重复的操作和配置应被移至初始化或拆卸方法中。这样，任何影响代码的更改都可以在一个地方更新，从而减少维护工作。例如，可以使用 Web 服务调用来满足 UI 测试的前置条件或后置条件（例如，用户注册、用户清理和配置文件设置）。

文档

这涵盖了所有形式的文档，从测试自动化文档（例如，测试自动化代码做什么，以及应该如何使用它），测试自动化解决方案（TAS）的用户文档，到 TAS 产生的测试报告和测试日志。

测试自动化解决方案（TAS）功能

添加 TAS 的功能和特性，如详细的测试报告、测试日志和与其他系统的集成。只添加将被使用的新功能。添加未使用的功能只会增加复杂性并降低可靠性和可维护性。

测试自动化框架（TAF）更新和升级

通过更新或升级到新版本的 TAF，可能带来以下好处：提供新功能供测试用例使用。风险在于更新 TAF，或者是升级现有测试工具，或者是引入新工具，都可能对现有测试用例产生不利影响。在推出新版本的测试工具之前，应先运行样本测试来验证最新版本的测试工具。样本测试应具有代表性，涵盖不同的 SUT、不同的测试类型和不同测试环境（如适用）的自动化测试。

8.1.3 重构自动化测试件以适应被测系统（SUT）的更新

在对现有被测系统（SUT）的一系列更改后，将需要更新测试自动化解决方案（TAS），包括测试自动化框架（TAF）和组件库。任何变更，无论多么微不足道，都可能对 TAS 的可靠性和性能产生广泛的不利影响。

识别测试环境组件的变化

评估需要进行哪些变更和改进。是否需要对测试件、定制功能库或操作系统进行更改？每一项都会对测试自动化解决方案（TAS）的运作产生影响。总体目标是确保自动化测试继续高效运行。应该逐步进行更改，采用最小可行产品的思维方式，以便通过有限的测试脚本运行来衡量对 TAS 的影响。一旦发现不存在副作用，就可以完全实施变更。完整的回归测试是验证变更是否对自动测试脚本产生不利影响的最后一步。在执行这些回归测试脚本的过程中，可能会发现失效。识别这些失效的根本原因（例如，通过测试报告、测试日志和测试数据分析），将为确保这些失效不是测试自动化改进活动造成的提供手段。

提高测试自动化解决方案（TAS）核心功能库的效率和有效性

随着 TAS 的成熟，会发现更高效地执行任务的新方法。这些新技术（例如，优化函数中的代码、使用更新的操作系统库）需要被纳入当前项目和未来项目使用的核心函数库中。

针对作用于同一控制类型的多个函数进行整合

在自动化测试运行期间，大部分操作是对图形用户界面（GUI）中控件的查询。这种查询旨在提供关于控件的信息（例如，可见/不可见、启用/未启用、尺寸和大小以及数据）。有了这些信息，自动化测试就可以从下拉列表中选择项目、在字段中输入数据以及从字段中读取值。有多种函数可以作用于控件以获取这些信息。有些函数是非常专用的，而其他函数则更具通用性。例如，可能有一个只适用于下拉列表的特定函数。另外，可能有一个函数可以通过将函数指定为其参数之一来与多个函数一起工作。因此，测试自动化工程师（TAE）能使用多个函数，而这些函数可以整合为更少的函数，从而达到相同的结果，并最小化维护工作量。

重构测试自动化架构（TAA）以适应被测系统（SUT）的变化

在测试自动化解决方案（TAS）的生命周期中，需要进行更改以适应被测系统（SUT）的变化。随着 SUT 的演变和成熟，底层的 TAA 也必须随之演变，以确保具备支持 SUT 的能力。在扩展功能时必须小心，不能以附加组件（或者打补丁）的方式实现这些功能，而是要在 TAA 中进行分析和更改。这将确保当新的 SUT 功能需要额外的测试脚本时，兼容的组件将到位，以适应这些新的自动化测试。

命名约定和标准化

随着变化的引入，新的测试自动化代码和函数库的命名约定需要与先前定义的标准保持一致（参见 4.3.1 节）。

对现有被测系统（SUT）脚本进行评估，以便修订或删除

变更和改进的过程还包括对现有测试脚本及其使用和持续价值进行评估。例如，对于某些既复杂又耗时的测试，将它们分解成几个更小的测试可能更可行且高效率。对于很少或根本不运行的测试则进行淘汰，将减少测试自动化解决方案（TAS）的复杂性，并使需要维护的内容更加清晰。

8.1.4 总结测试自动化工具的应用机会

除了实际测试外，测试自动化还有助于非特定测试活动，例如：

环境初始化和控制（测试自动化在环境管理中的应用）

某些测试脚本（例如，测试数据创建）可被用来在初始化方法中为新测试环境创建不同的测试数据。在需要根据不同的数据输入创建多个用户多个配置文件的情况下，团队可以使用自动测试脚本来调用 Web 服务端点以注册这些用户。测试脚本可以控制测试基础设施的初始化，也可在测试过程之后进行清理。这有助于节省时间并确保每个新的测试环境中都有适当的用户。例如，不同的测试日志和其他测试件能被自动地从测试环境中移除，使测试环境的维护和使用更加高效。

数据时效性（测试自动化在数据管理中的应用）

测试自动化可被用于操作测试环境中的测试数据。例如，在数据库中，可以检查和控制日期字段，使其从年份角度保持最新。

屏幕截图和视频生成

大多数现代用户界面（UI）测试自动化工具都内置了在特定条件下创建屏幕截图或视频并将其存储起来的功能。借助这些测试工具，团队就能为业务提供支持，为软件发布文档或营销目的创建实际使用的截图和视频。

9 参考文献

标准

测试自动化标准包括但不限于：

The Automatic Test Markup Language (ATML) by IEEE (Institute of Electrical and Electronics Engineers) consisting of 电气和电子工程师协会（IEEE）的自动测试标记语言（ATML）包括：

- IEEE Std 1671.1:测试描述
- IEEE Std 1671.2: 测试工具描述
- IEEE Std 1671.3: 被测单元描述
- IEEE Std 1671.4: 测试配置描述
- IEEE Std 1671.5: 测试适配器描述
- IEEE Std 1671.6: 测试站描述
- IEEE Std 1641: 信号和测试定义

IEEE Std 1636.1：测试结果

ISO/IEC/IEEE 29119-5 (2016) 软件和系统工程--软件测试--第 5 部分：关键词驱动测试

ISO/IEC 30130:2016 (E) 软件工程 - 软件测试工具的能力

欧洲电信标准协会（ETSI）和国际电信联盟（ITU）的测试和测试控制符号（TTCN-3）包括：

- ES 201 873-1: TTCN-3 核心语言
- ES 201 873-2: TTCN-3 表格表示格式 (TFT)
- ES 201 873-3: TTCN-3 图形表示格式（GFT）
- ES 201 873-4: TTCN-3 操作语义
- ES 201 873-5: TTCN-3 运行时接口 (TRI)

- ES 201 873-6: TTCN-3 控制接口 (TCI)
- ES 201 873-7: 结合 TTCN-3 使用 ASN.1
- ES 201 873-8: 结合 TTCN-3 使用 IDL
- ES 201 873-9: 结合 TTCN-3 使用 XML
- ES 201 873-10:TTCN-3 文档
- ES 202 781: 扩展：配置和部署支持
- ES 202 782: 扩展：TTCN-3 性能和实时测试
- ES 202 784: 扩展：高级参数化
- ES 202 785: 扩展：行为类型
- ES 202 786: 扩展：支持具有连续信号的接口
- ES 202 789: 扩展：扩展的 TRI

OMG（对象管理组）的 UML 测试配置文件（UTP） 规定了以下测试规范概念：

测试架构。

测试数据。

测试行为。

测试记录。

测试管理 。

ISTQB 文档

标识	参考文档
ISTQB-AL-TTA	ISTQB 认证测试工程师 ，高级大纲，技术测试分析师，4.0 版，2021 年 6 月，可以从 [ISTQB-Web] 查阅
ISTQB-FL	IISTQB 认证测试工程师，基础级大纲，4.0 版，2023 年 4 月，可以从 [ISTQB-Web] 查阅
ISTQB-MBT	ISTQB 认证测试工程师，基于模型的测试大纲，1.0 版，2015 年 10 月，可以从 [ISTQB-Web]查阅

ISTQB-PT	ISTQB 认证测试工程师，性能测试大纲，2018 年 12 月版，可以从 [ISTQB-Web] 查阅
ISTQB-SEC	ISTQB 认证测试工程师，安全性测试大纲，1.0 版，2016 年 3 月，可以从 [ISTQB-Web]查阅
ISTQB-TAS	ISTQB 认证测试工程师，《测试自动化策略大纲》，2024 年 2 月版，可以从 [ISTQB-Web]查阅
ISTQB-Glossary	ISTQB 术语表，可以从 [ISTQB-Web] 在线获取

书籍

Paul Baker, Zhen Ru Dai, Jens Grabowski, and Ina Schieferdecker, “Model-Driven Testing: Using the UML Testing Profile”, Springer 2008 edition, ISBN-10: 3540725628, ISBN-13: 978-3540725626

Efriede Dustin, Thom Garrett, Bernie Gauf, “Implementing Automated Software Testing: how to save time and lower costs while raising quality”, Addison-Wesley, 2009, ISBN 0-321-58051-6

Efriede Dustin, Jeff Rashka, John Paul, “Automated Software Testing: introduction, management, and performance”, Addison-Wesley, 1999, ISBN-10: 0201432870, ISBN-13: 9780201432879

Mark Fewster, Dorothy Graham, “Experiences of Test Automation: Case Studies of Software Test Automation”, Addison-Wesley, 2012

Mark Fewster, Dorothy Graham, “Software Test Automation: Effective use of test execution tools”, ACM Press Books, 1999, ISBN-10: 0201331403, ISBN-13: 9780201331400

Robert C Martin, “Clean Code: A Handbook of Agile Software Craftsmanship”, 2008, ISBN-10: 9780132350884

James D. McCaffrey, “.NET Test Automation Recipes: A Problem-Solution Approach”, APRESS, 2006 ISBN-13:978-1-59059-663-3, ISBN-10:1-59059-663-3

Daniel J. Mosley, Bruce A. Posey, “Just Enough Software Test Automation”, Prentice Hall, 2002, ISBN-10: 0130084689, ISBN-13: 9780130084682

Manikandan Sambamurthy, “Test Automation Engineering Handbook”, January 2023, ISBN: 9781804615492

Colin Willcock, Thomas Deiß, Stephan Tobies and Stefan Keil, “An Introduction to TTCN-3” Wiley, 2nd edition 2011, ISBN-10: 0470663065, ISBN-13: 978-0470663066

在线文章

Robert V. Binder, Suzanne Miller, “Five Keys to Effective Agile Test Automation for Government Programs” August 24, 2017, Software Engineering Institute, Carnegie Mellon University, https://resources.sei.cmu.edu/asset_files/Webinar/2017_018_101_503516.pdf

DoD CIO, Modern Software Practices “DevSecOps Fundamentals Guidebook: Activities & Tools”, Version 2.2, May 2023, https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsActivitesToolsGuidebookTables.pdf?ver=_Sylg1WJB9K0Jxb2XTvzDQ%3d%3d

Péter Földházi, “Tri-Layer Testing Architecture”, https://www.pnsqc.org/docs/PROP53522057-FoldhaziDraftFinal.pdf

Thomas Pestak, William Rowell, PhD, “Automated Software Testing Practices and Pitfalls”, September 2018, https://www.afit.edu/stat/statcoe_files/Automated%20Software%20Testing%20Practices%20and%20Pitfalls%20Rev%201.pdf

Andrew Pollner, Jim Simpson, Jim Wisnowski, “Automated Software Testing Implementation Guide for Managers and Practitioners”, October 2018, https://www.afit.edu/stat/statcoe_files/0214simp%202%20AST%20IG%20for%20Managers%20and%20Practitioners.pdf

The Selenium Project, “Page object models”, https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/

10 附录 A--学习目标/知识认知级别

以下定义的学习目标适用于本大纲。大纲中的每个专题都将根据其学习目标进行考察。

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头，如下所示。

等级 2：理解（K2）- 考生能选择与主题相关的陈述理由或解释，并能对测试概念进行总结、比较、分类和举例说明。

行为动词：分类、比较、区分、辨别、 解释、举例、演绎、总结

举例说明：

- “根据测试工具的目的和它们支持的测试活动对其进行分类。”
- “比较不同的测试级别。（可以用于寻找相同点、不同点，或两者兼有。）”
- “区分测试和调试。（寻找概念之间的差异。）”
- “辨别项目风险和产品风险。（可以通过概念区分，允许两个（或更多个）概念被单独分类。）”
- “解释环境对测试过程的影响。”
- “举例说明为什么需要进行测试。”
- “从给定的失败概况中推断缺陷的根本原因。”
- “总结工作产品评审流程中的各项活动。”

等级 3：应用(K3)- 考生能在遇到一个熟悉的任务时执行一个规程，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、实施、准备、使用.

举例说明：

- “应用边界值分析，从给定的需求中推导出测试用例。（应参考规程/技术/流程等。）”
- “实施指标收集方法，以支持技术和管理需求。”
- “准备移动应用程序的可安装性测试。”

- “使用可追溯性来监控测试进度，以确保测试目标、测试策略和测试计划的完整性和一致性。（在学习目标（LO）中，可用于要求考生能够使用某种技术或规程。类似于“应用”。）”

等级 4：分析（K4）- 考生能将与程序或技术相关的信息拆分成不同组成部分，以便更好地理解，并能区分事实和推论。典型的应用是分析文件、软件或项目情况，并提出适当的行动来解决问题或任务。

行为动词：分析、解构、概述、定优先级、选择。

举例说明：

- “分析给定的项目情况，以确定应采用哪种黑盒测试技术或基于经验的测试技术来实现特定目标。（只有与可衡量的分析目标相结合时，才可考察。格式应为“分析 xxxx 以达到 xxxx”（或类似格式）。）”
- “基于相关产品风险，对给定测试套件中的测试用例进行执行优先级排序。”
- “选择适当的测试级别和测试类型来验证给定的一组需求。（需要对选择进行分析。）”

参考文献：

(For the cognitive levels of learning objectives) 学习目标的认知水平

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

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

11 附录 B - 商业价值(成果)与学习目标的可追溯性矩阵

本节列出了“商业价值(成果)”与“测试自动化工程专家”的“学习目标”之间的可追溯性。

商业价值(成果)测试自动化工程专家			B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11	B12
第 1 章	测试自动化简介和目标													
1.1	描述测试自动化的目的													
1.1.1	解释测试自动化的优缺点	2	X											
1.2	软件开发生命周期中的测试自动化													
1.2.1	解释如何在不同的软件开发生命周期模型中应用测试自动化	2		X										
1.2.2	为被测系统选择合适的测试自动化工具	2		X										
第 2 章	准备测试自动化													
2.1	**理解实现测试自动化的基础架构配置**													
2.1.1	描述实现测试自动化的基础设施的配置需求	2			X									
2.1.2	解释如何在不同环境中利用测试自动化	2			X									
2.2	选择正确工具和策略的评估程序													
2.2.1	分析被测系统，确定合适的测试自动化解决方案	4				X								
2.2.2	说明工具评估的技术结论	4				X								
商业价值(成果)测试自动化工程专家			B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11	B12
第 3 章	测试自动化架构													
3.1	测试自动化中使用的设计概念													
3.1.1	解释测试自动化架构的主要能力	2					X							
3.1.2	解释如何设计测试自动化解决方案	2					X							
3.1.3	测试自动化框架分层的应用	3					X							
3.1.4	不同方法实现自动化测试用例的应用	3					X							

3.1.5	在测试自动化中应用设计原则和设计模式	3					X							
第 4 章	实现测试自动化													
4.1	测试自动化开发													
4.1.1	用支持有效测试自动化试点和部署活动的指南	3						X						
4.2	与测试自动化开发相关的风险													
4.2.1	分析测试自动化的部署风险并规划缓解策略	4							X					
4.3	测试自动化解决方案的可维护性													
4.3.1	解释哪些因素支持和影响测试自动化解决方案的可维护性	2								X				
第 5 章	测试自动化的实施与部署策略													
5.1	集成到持续集成/持续交付(CI/CD)流水线													
5.1.1	在 CI/CD 流水线中应用各测试级别的测试自动化	3									X			
5.1.2	解释测试件的配置管理	2									X			
商业价值(成果)测试自动化工程专家			B01	B02	B03	B04	B05	B06	B07	T08	B09	B10	B11	B12
5.1.3	解释应用程序编程接口（API）基础设施的测试自动化依赖性	2									X			
第 6 章	测试自动化报告和度量													
6.1	测试自动化数据的收集、分析与报告													
6.1.1	从测试自动化解决方案（TAS）和被测系统(SUT)中收集数据的方法的应用	3										X		
6.1.2	分析从测试自动化解决方案（TAS）和被测系统(SUT)获取的数据以更好地理解测试结果	4										X		
6.1.3	解释如何构建和发布测试进度报告	2										X		
第 7 章	验证测试自动化解决方案													
7.1	测试自动化基础设施的验证													
7.1.1	计划验证测试自动化环境，包括测试工具搭建	3											X	
7.1.2	解释给定自动化测试脚本和/或测试套件的正确行为	2											X	

7.1.3	识别测试自动化产生不符合预期结果的位置	2											X	
7.1.4	解释静态分析如何帮助提高自动化测试代码质量	2											X	
第 8 章	持续改进													
8.1	测试自动化的持续改进机会													
8.1.1	通过数据收集和分析发现改进测试用例的机会	3												X
商业价值(成果)测试自动化工程专家			B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11	B12
8.1.2	分析已部署的测试自动化解决方案（TAS）的技术方面并提供改进建议	4												X
8.1.3	重构自动化测试件，使其与 SUT 更新保持一致	3												X
8.1.4	总结测试自动化工具的机会	2												X

12 附录 C - 发布说明

ISTQB®测试自动化工程大纲 2024 是对 2016 版本的主要更新和重写。因此，本次更新没有提供按章节和小节详细列出的发布说明。内容已针对测试自动化工程师角色进行了显著增强，而策略相关的内容已移至新的 ISTQB®测试自动化策略大纲 2024。

13 附录 D--领域专用术语

术语名称	定义
开发安全运营	一种结合了开发、安全和运营，并使用高度自动化的方法论。
流模型模式	对工作域、其组件以及它们之间的相互连接的高层次视图。
用户旅程	从用户体验的角度展示用户如何与被测系统交互的一系列步骤。
页面对象模式	一种测试自动化设计模式，用于加强测试维护和减少代码重复。
版本控制	一种检入和存储特定版本源代码的过程。

14 索引

All terms are defined in the ISTQB® Glossary (http://glossary.istqb.org/).

所有术语在 ISTQB® 术语表中均有定义（http://glossary.istqb.org/）.