测试工程师认证
高级大纲

测试自动化工程师

2016 版本

中文版本 2018.08

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

英文版权声明

高级测试大纲（测试自动化工程师）子工作组：Bryan Bakker, Graham Bath, Armin Born, Mark Fewster, Jani Haukinen, Judy McKay, Andrew Pollner, Raluca Popescu, Ina Schieferdecker, 2016 年。

中文版权声明

版本历史

版本	日期	说明
初稿	2015 年 8 月 13 日	初稿
第二版	2015 年 11 月 5 日	LO 映射和重定位
第三版	2015 年 12 月 17 日	修订 LO
Beta 版	2016 年 1 月 11 日	编辑
Beta	2016 年 3 月 18 日	Beta 版发布
2016 版大纲	2016 年 10 月 21 日	GA 发布
中文版	2018 年 8 月 2 日	CSTQB®发布

目 录

致谢

此文档由国际软件测试认证委员会高级子工作组的核心团队编制。

在此，核心团队感谢提供建议和输入的审查小组和所有国际委员会成员。

同时，参与完成本模块测试自动化高级测试大纲编写的工作组人员有： Bryan Bakker, Graham Bath（高级工作组主席）, Armin Beer, Inga Birthe, Armin Born, Alessandro Collino, Massimo Di Carlo, Mark Fewster, Mieke Gevers, Jani Haukinen, Skule Johansen, Eli Margolin, Judy McKay （高级工作组副主席）, Kateryna Nesmyelova, Mahantesh (Monty) Pattan, Andrew Pollner （高级测试自动化组主席）, Raluca Popescu, Ioana Prundaru, Riccardo Rosci, Ina Schieferdecker, Gil Shekel, Chris Van Bael.

本大纲核心团队作者包括： Andrew Pollner（主席）, Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu, Ina Schieferdecker.

下列成员参与了评审、评论和大纲表决工作（按字母顺序）： Armin Beer, Tibor Csöndes, Massimo Di Carlo, Chen Geng, Cheryl George, Kari Kakkonen, Jen Leger, Singh Manku, Ana Paiva, Raluca Popescu, Meile Posthuma, Darshan Preet, Ioana Prundaru, Stephanie Ulrich, Erik van Veenendaal, Rahul Verma.

此文档正式由 ISTQB® 大会于2016年10月21日颁布。

参加本大纲翻译的 CSTQB®专家有（按姓氏拼音排序）：陈冬严、崔哲、郭鹏、刘海英、汪健、王轶昆（组长）、徐明辉、徐文叶、张银萍。

负责本大纲最终质量评审的 CSTQB®专家有（按姓氏拼音排序）：陈耿、周震漪（组长）。

0. 本大纲的简介

0.1 本大纲的目的

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

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

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

0.2 本大纲的范围

0.2.1 *范围内*

本文档描述了测试自动化工程师（TAE）在设计、开发和维护测试自动化解决方案中的任务。重点在于自动化动态功能测试的概念、方法、工具和过程，以及这些测试与测试管理、配置管理、缺陷管理、软件开发过程和质量保证之间的关系。

本文所描述的方法适用于软件生命周期中广泛使用的方法（例如，敏捷开发、顺序开发、增量开发、迭代开发）、软件系统的类型（如嵌入式、分布式、移动式）和测试类型（功能性和非功能性测试）。

0.2.2 *范围外*

以下几个方面超出了测试自动化大纲的范围:

- 测试管理，自动化创建测试规范和自动化测试生成；
- 测试自动化经理（TAM）在计划、监督和调整测试自动化解决方案的开发和演进方面的任务；
- 自动化非功能性测试（例如，性能测试）的内容；
- 静态分析（例如，脆弱性分析/ vulnerability analysis）的自动化和静态测试工具；
- 软件工程方法和编程的教学（例如，使用哪些标准，哪些技能可以实现自动化测试解决方案）；
- 软件技术的教学（例如，使用哪一种脚本技术实现测试自动化解决方案）；
- 选择软件测试产品和服务（例如，使用哪一种产品和服务实现测试自动化解决方案）。

0.3 高级测试自动化工程师认证

0.3.1 *期望*

高级认证是针对那些已经在基础级的知识和技能的基础上，在一个或多个特定领域希望发展更多专业能力的人员。这些为高级专业技术人员提供的模块覆盖了更广泛的测试主题。

测试自动化工程师应该是一个具备通用的测试知识，同时又在测试自动化的特定领域有深入了解的人员。所谓深入的了解，意指在测试自动化理论和实践中有足够的知识，能够影响一个企业或者项目，在设计、开发和维护针对功能测试的测试自动化解决方案时所选择的方向。

高级模块概述 [ISQB-AL 模块] 文档描述了该模块的商业价值。

0.3.2 *入口要求和更新要求*

高级认证的一般入口标准在 ISTQB 网站 [ISQB-Web] 的高级部分进行了描述。

除了这些一般的入口标准，考生必须持有 ISTQB 基础级证书 [ISTQB-CTFL]，才能参加高级测试自动化工程师资格考试。

0.3.3 *知识水平*

为清晰可见，在每章的开头描述了本课程的学习目标。根据分配的学习目标来检验教学大纲中的每一个主题。

在 ISTQB 网站 [ISQB-Web] 中描述了为学习目标所分配的认知级别（K-级别）。

0.3.4 *考试*

本高级证书的考试应以本课程大纲加上基础课程大纲 [ISQB-FL] 为基础。对考试问题的回答可能需要使用基于这些大纲的多个章节的内容。

在 ISTQB 网站 [ISQB-Web]，高级水平部分描述了考试的形式。在 ISTQB 网站上也有一些对参加考试有用的信息。

0.3.5 *授权*

ISTQB 成员理事会可授权其培训提供者遵循本教学大纲来编写课程材料。
ISTQB 网站 [ISTQBWeb]中高级水平部分描述了针对培训提供者的课程授权具体规则。

0.4 标准内容与有用的信息

教学大纲的标准部分是可检验的（通过考试）内容，如：

- 学习目标；
- 关键词。

教学大纲的其余内容是对学习目标有用的阐述。

0.5 详细程度

本教学大纲的详细程度保证了能够支持国际上一致的教学和考试。为达到此目标，教学大纲包括：

- 每个知识域的学习目标，描述了认知学习的结果和应达到的理念体系（这些都是标准的）；
- 教学的信息列表，包括需要讲解的关键概念，被接受的文献或标准的来源，以及必要时引用的其他信息（这些都是有用的信息）。

教学大纲的内容不是对测试自动化工程的整个知识领域的描述；它仅反映了在认可的高级培训课程中要涵盖的细节程度。

0.6 大纲结构

本大纲主要有八章。顶层标题显示了学习该章节需要的时间，例如：

3.	通用测试自动化架构	270 分钟

表明授课第 3 章中的内容应该使用 270 分钟的时间。

具体的学习目标列在每一章的开头。

0.7 术语、定义和缩略词

软件文献中使用的许多术语都可以互用。本高级课程大纲中的定义都可在 ISTQB [ISTQB 术语表]公布的软件测试中使用的标准术语表中找到。

以下用于文档中的缩写：

缩写	英文	中文
CLI	Command Line Interface	**命令行界面**
EMTE	Equivalent Manual Test Effort	**等效手工测试工作量**
gTAA	Generic Test Automation Architecture	**通用测试自动化架构**
GUI	Graphical User Interface	**图形用户界面**
SUT	System Under Test	**被测系统**
TAA	Test Automation Architecture	**测试自动化架构**（通用测试自动化架构的一个实例，其定义测试自动化解决方案的架构）
TAE	Test Automation Engineer	**测试自动化工程师**（负责设计测试自动化架构的人员，以及负责测试自动化解决方案的实施、维护和技术的演进）
TAF	Test Automation Framework	**测试自动化框架**（测试自动化要求的环境，包括测试用具和工件，例如测试库）

TAM	Test Automation Manager	**测试自动化经理**（负责规划和监督测试自动化解决方案开发与演进的人员）
TAS	Test Automation Solution	**测试自动化解决方案**（测试自动化架构的实现/实施，包括测试用具和工件，例如测试库）
UI	User Interface	**用户界面**

1. 测试自动化的介绍和目标（30 分钟）

关键字

API 测试（API testing）, CLI 测试（CLI testing），GUI 测试（GUI testing），被测系统（System Under Test），测试自动化架构（test automation architecture），测试自动化框架（test automation framework），测试自动化策略（test automation strategy），测试自动化（test automation），测试脚本（test script），测试件（testware）

测试自动化的介绍和目标的学习目标

1.1 测试自动化的目的

ALTA-E-1.1.1 (K2) 对测试自动化的目标、优点、缺点及局限的解释

1.2 测试自动化的成功因素

ALTA-E-1.2.1 (K2) 识别测试自动化项目在技术上的成功要素

1.1 测试自动化的目的

在软件测试中，测试自动化（包括自动化测试执行）是下列任务中的一项或多项：

- 使用特制的软件工具来控制和设置测试前置条件；
- 执行测试；
- 比较实际输出和预期输出。

一个好的实践是把用于测试的软件和被测系统（SUT）本身分开，这样可以最小化二者的冲突，当然也有例外，例如嵌入式系统测试软件就需要部署到被测系统（SUT）中。

我们希望测试自动化可以在不同版本的被测系统（SUT）及其环境中持续、反复的运行很多测试用例，但是测试自动化不仅仅是一个没有人为干预的持续运行测试套件的机制，它还包含设计测试件的过程，包括：

- 软件；
- 文档；
- 测试用例；
- 测试环境；
- 测试数据。

针对测试活动的必要测试件包括：

- 实现自动化测试用例；
- 监督并且控制自动化测试的执行；
- 解释、报告、记录自动化测试的结果。

测试自动化使用不同的途径与被测系统（SUT）交互：

- 通过被测系统（SUT）的类、模块和库的公共接口实施测试（API 测试）；
- 通过被测系统（SUT）的用户接口实施测试（例如，图形用户界面（GUI）测试或命令行界面（CLI）测试）；
- 通过某个协议或服务实施测试。

测试自动化的目标包括：

- 提高测试效率；
- 提供更广泛的功能覆盖；
- 降低测试总成本；
- 执行那些手工测试无法实施的测试；
- 缩短测试执行周期；
- 增加测试频率 / 缩短测试周期所需时间。

测试自动化的优点包括：

- 每次构建后可以执行更多的测试；

- 执行手工无法实施的测试（实时、远程、并行测试）；
- 可以实施更复杂的测试；
- 测试执行的速度更快；
- 测试不易受到操作员错误的影响；
- 更有效、更高效地使用测试资源；
- 更快反馈软件质量；
- 提高系统的可靠性（例如，重复性，一致性）；
- 改进测试的一致性。

测试自动化的缺点包括：

- 实施测试自动化会涉及到额外费用；
- 建立测试自动化解决方案（TAS）需要初始投入；
- 需要额外的技术；
- 团队需要具备开发能力和自动化的能力；
- 需要持续不断的维护测试自动化解决方案（TAS）；
- 可能会偏离测试目标，比如：专注于自动化测试用例而忽略执行测试；
- 测试会变得更复杂；
- 自动化可能引入额外的错误。

测试自动化的局限性包括：

- 不是所有的手工测试都可以被自动化的；
- 自动化仅能检查机器可判断的结果；
- 自动化只能检查能够被自动化测试准则（Oracle）验证的实际结果；
- 不能替代探索性测试。

1.2 测试自动化成功的因素

下列成功因素适用于正在运作的测试自动化项目，因此重点是影响项目长期成功的因素。影响处在试点阶段（不算正在运行的项目）的测试自动化项目成功的因素在这里不作考虑（下文有专门章节阐述）。

测试自动化的主要成功因素包括以下内容：

测试自动化架构（TAA）

测试自动化架构（TAA）与一个软件产品的架构非常接近。应该明确架构支持的功能和非功能需求，通常这也是最重要的需求。

测试自动化架构（TAA）通常要求针对可维护性、性能和易学性进行设计（更详细情况和其它非功能特性方面信息可参考 ISO/IEC 25000:2014 标准）。引入对被测系统（SUT）架构有足够理解的软件工程师能为这样的设计提供帮助。

被测系统（SUT）的易测试性

被测系统（SUT）在设计时就要考虑具备易测试性，能支持测试自动化。在图形用户界面（GUI）测试的情况下，这意味着被测系统（SUT）应该尽可能地将图形用户界面（GUI）交互逻辑和数据与图

形接口的外观渲染分离。在 API 测试的情况下，这就需要公开更多的类、模块或命令行接口，以便于对它们进行测试。
应该首先从被测系统（SUT）中具备易测试性的部分开始。一般情况下，测试自动化成功的一个关键因素在于能够较容易的实现测试自动化脚本。围绕这个目标，并且为了提供一个成功实施的证明。测试自动化工程师（TAE）应该在被测系统（SUT）中识别出容易进行自动化测试的模块和组件，并从那里开始自动化。

测试自动化策略

针对被测系统（SUT）的可维护性和一致性的测试自动化策略是实用的和一致的。
有可能无法将测试自动化策略以同样的方式应用于被测系统（SUT）的新旧部分。在创建自动化策略时，就要考虑将其应用于代码不同部分的成本、收益和风险。
应该考虑用自动化测试用例对测试用户界面和 API 都进行测试，来检查结果的一致性。

测试自动化框架（TAF）

一个易于使用、很好的文档化和可维护的测试自动化框架（TAF）支持一致的自动化测试途径。
为了创建一个易用的和可维护的测试自动化框架（TAF），必须做到以下几点：

- 实现报告功能：测试报告应该提供关于被测系统（SUT）的质量信息（通过/失败/错误/未执行/中止，统计分析等等）。报告还应该为测试人员、测试经理、开发人员、项目经理及其他利益相关者提供信息，以帮助利益相关者获得质量的概述；
- 能易于故障排除：除了测试执行和日志记录，测试自动化框架（TAF）应该要提供一个简单的方法为失败的测试用例排除故障，测试用例失败可能是由于：
  - 在被测系统（SUT）内发现的失效
  - 在测试自动化解决方案（TAS）中发现的失效
  - 测试本身或者测试环境的问题
- 合理的处理测试环境：测试工具依赖于测试环境的一致性，在测试自动化中专门的测试环境是必需的。如果没有对测试环境和测试数据的控制，测试设置可能不满足测试执行的要求，并且很可能导致错误的执行结果；
- 自动化测试用例的文档化：测试自动化的目标必须是明确的。例如，应用程序的哪些部分要进行测试，测试到什么程度，需要测试哪些属性（功能性和非功能性）。这些都必须进行明确的描述和文档化；
- 自动化测试的可追溯性：测试自动化框架（TAF）应支持自动化测试工程师追溯至测试用例中的每个步骤；
- 使其易维护：理想情况下，自动化测试用例应该易于维护，使得维护工作不占用测试自动化的主要工作量。另外维护工作应该与被测系统（SUT）的变更规模成比例。为此，用例必须易分析、易修改以及易扩展，此外自动化的测试件应该具有较高的复用率，这样可以减少后续修改维护的工作量；
- 持续更新自动化测试：当新的或更改的需求导致测试或整个测试套件失败时，不要禁用失败的测试用例，而应该修复他们；
- 制定部署计划：确保测试脚本可以很容易地部署、修改和重新部署；

- 根据需要淘汰测试：如果自动化测试脚本不再有用也不再需要他们，确保可以很容易地淘汰这些测试脚本；
- 监控和恢复被测系统（SUT）：在真实项目里，需要不断运行测试用例或测试用例集，这时必须持续监督被测系统（SUT）。如果被测系统（SUT）遇到致命错误（例如崩溃），测试自动化框架（TAF）必须具有可恢复的能力，然后跳过当前的用例并继续执行下个用例。

测试自动化代码可能很复杂，很难维护。经常会出现测试代码与被测系统（SUT）代码规模几乎相同的情况。这就是为什么测试代码的可维护性极其重要。测试代码的维护性高要求在于：可能是使用了不同的测试工具，不同类型的验证类型以及不同的测试件（例如测试输入数据、测试准则、测试报告等），这些都是需要维护的。

根据以上维护的注意事项，除了需要完成的重要事项，还有以下应该避免的事项：

- 不要创建接口敏感的代码（即，它会受到图形界面的变化或 API 非必要部分变化的影响）；
- 不要创建对数据变化敏感的测试自动化脚本或高度依赖特定数据值（例如，测试输入依赖于其他测试输出）的测试自动化脚本；
- 不要创建上下文敏感的自动化环境（例如，操作系统的日期和时间，操作系统本地化参数或其他应用程序的内容）。这种情况下，最好是在必要的情况下使用测试桩，这样环境就可以被控制。

测试自动化项目符合成功条件越多，该项目就越有可能成功。并非所有成功条件都是必需的，在实践中，几乎没有什么项目能满足所有的成功条件。在测试自动化项目启动之前，通过考虑已经具备的因素和不具备的因素给当前做法带来的风险，并结合项目背景来分析项目成功的机会是非常重要的。一旦测试自动化架构（TAA）设计完毕，那重点就在检查哪些工作被遗漏或还需要继续完善。

2. 测试自动化准备—165 分钟

关键词

易测试性（testability）,驱动器（driver）,侵入级别（level of intrusion）,桩（stub）,测试执行工具（test execution tool）,测试钩子（test hook）,测试自动化经理（test automation manager）

本章节的学习目标

2.1 被测系统（SUT）中影响测试自动化的因素

ALTA-E-2.1.1 (K4) 通过对被测系统进行分析，从而决定合适的自动化解决方案

2.2 工具评估和选择

ALTA-E-2.2.1 (K4) 基于一个给定项目，分析测试自动化工具并报告技术结果和建议

2.3 为易测试性和自动化而设计

ALTA-E-2.3.1 (K2) 理解适用于被测系统的“易测试性设计”和“为测试自动化而设计”的方法

2.1 被测系统（SUT）中影响测试自动化的因素

当评估被测系统（SUT）的上下文及其环境背景时，需要识别影响测试自动化的因素，从而确定合适的解决方案。这可能包含以下部分：

- 被测系统（SUT）接口
  自动化测试用例会调用被测系统的一些动作。为此，被测系统必须提供接口，通过这些接口可以控制被测系统。可以通过用户接口（UI）控制，也可以通过更底层的软件接口实现。除此之外，一些测试用例可以在通讯级别进行接口连接（例如使用 TCP/IP，USB 或者专有通讯接口）。

  将被测系统分解可以使得测试自动化在各个不同的测试级别上和被测系统（SUT）通过接口进行交互。在一个特定的测试级别上进行测试自动化是可能的（例如，组件测试级别和系统测试级别），但是被测系统必须提供足够的支持。例如，在组件测试级别，可能没有用于测试的用户界面，因此需要提供不同的（可能是定制的）软件接口（也称为测试钩子 test hook）来支持测试。

- 第三方软件
  被测系统经常不仅包括内部组织编写的软件，也包括由第三方提供的软件。在某些情况下，第三方软件也需要测试。此时如果测试自动化是合理的，那么它可能需要一个不同的测试自动化解决方案，例如使用 API。

- 侵入级别
  不同的自动化测试途径（使用不同的工具）有不同的侵入级别。被测系统中为了自动化测试所需改动的数量越多，侵入级别越高。使用专用软件接口的自动化测试侵入级别相对较高，而使用已有的用户界面（UI）元素的侵入级别则相对较低。使用被测系统的硬件元素（例如键盘、手动开关、触摸屏、通信接口）则具有更高的侵入级别。

  高侵入级别的测试方法有“报假警”的风险。所谓错误报警是指，为实现系统的易测试性，所实施的侵入措施，可能导致测试模拟器（测试自动化解决方案（TAS））运行失败，而如果在真实环境中，通常不会出现这种情况。使用高侵入级别的测试自动化方案，经常会是比较容易的解决方案。

- 不同的被测系统架构
  不同的被测系统架构可能需要不同的测试自动化解决方案。例如，运用 C++语言并采用 COM 技术编写的被测系统，与使用 Python 语言编写的被测系统，两者的测试自动化方案肯定是不同的。不同的架构可以使用同一套测试自动化策略，但是这需要能够支持这些架构的混合的策略。

- 被测系统的规模和复杂度
  在制定未来开发计划时，应当考虑当前被测系统的规模和复杂度。对于小型、简单的被测系统，复杂和特别灵活的测试自动化方法是不必要的。相反的，对于超大型、复杂的被测系统，使用小

型和简单的测试自动化方法是不合适的。尽管在初期，为复杂的被测系统设计小型且简单的测试自动化方法也是合适的，但这只能是临时策略（详情参见第 3 章）。

如果被测系统现在是可用的状态，上述提到的许多因素是已知的（例如规模、复杂度、可用的软件接口），但是许多时候测试自动化的开发应当在被测系统可用之前就开始了。此时测试自动化工程师（TAE）需要对未知的内容进行评估并且明确提出所需的软件接口。（详情参见章节 2.3）

即使被测系统尚不可见，也可以开始自动化测试计划。例如：

- 当需求（功能性或非功能性）已知时，可以从这些需求中选择适合进行自动化的内容并且确定测试这些内容的方法。可以从这些选择的内容开始进行测试自动化计划，包括识别自动化需求以及确定测试自动化策略；
- 当架构和技术设计正在开发时，可以着手设计用于支持测试的软件接口。

2.2 工具评估和选择

测试自动化经理（TAM）负责工具选择和评估，测试自动化工程师（TAE）将参与评估和甄选工作，并且向测试自动化经理（TAM）提供信息。工具评估和选择流程在 ISTQB 基础级中有所介绍，测试经理高级大纲[ISTQB-AL-TM]中描述了更多细节。

测试自动化工程师（TAE）将参与整个工具的评估和选择过程，在以下活动中应作出特别的贡献：

- 评估组织成熟度，识别能够由测试工具支持之处；
- 评估测试工具可以支持的合适目标；
- 识别和收集潜在合适的工具信息；
- 根据目标和项目的限制，分析工具信息；
- 基于可靠的业务用例估算成本收益率；
- 推荐合适的工具；
- 识别工具与被测系统组件的兼容性。

功能测试自动化工具经常不能满足自动化项目的所有期望目标。以下是一组这类问题的示例（并非所有类型）：

发现的问题	例子	可能的解决方案
工具接口无法与已有工具一同工作	• 测试管理工具已经更新并且连接接口已经改变； • 来自于售前的信息是错误的，并不是所有的数据可以转移到报告工具。	• 在任何更新之前注意发布说明，对于大型的迁移，在迁移到生产环境之前进行测试； • 获取使用真实被测系统的工具演示； • 寻求供应商和/或用户社区论坛的支持。

发现的问题	例子	可能的解决方案
一些被测系统（SUT）的依赖变更后，导致测试工具不支持	• 开发部门已经将 Java 更新到最新版本	• 同步升级开发/测试环境以及测试自动化工具
无法捕获图形用户界面（GUI）上的对象	• 对象可见，但测试自动化工具不能与它交互	• 在开发中，尝试只使用众所周知的技术或对象； • 购买测试自动化工具之前，先做一个试点项目； • 让开发人员定义对象的标准。
工具看起来很复杂	• 工具的功能很强大，但只有一部分有用	• 从工具栏中删除不需要的功能，找到限制功能集的方法； • 选择符合需求的许可证； • 找到更侧重所需功能的替代工具。
与其他系统冲突	• 安装其他软件后，测试自动化工具不再工作，反之亦然	• 在安装前阅读发布说明或技术需求文档； • 与供应商确认，对其他工具不会有影响； • 在用户社区论坛中获取帮助。
影响被测系统	• 在使用测试自动化工具期间或之后，被测系统反应不同（例如，更长的响应时间）	• 使用不需要改变被测系统的工具（例如，安装库）
访问代码	• 测试自动化工具会改变部分源代码	• 使用不需要改变源代码的工具（例如，安装库）
有限的资源（主要是在嵌入式环境）	• 测试环境的空闲资源有限或资源耗尽（例如，内存）	• 阅读发布说明，并与工具提供商讨论，确认环境不会导致问题； • 在用户社区论坛中获取帮助。
升级	• 升级不会迁移所有数据，或者会破坏现有的自动化测试脚本、数据或配置； • 升级需要不同的（更好的）环境。	• 在测试环境上进行升级测试，与供应商确认迁移可正常进行； • 阅读升级的先决条件并决定是否值得进行升级； • 从用户社区论坛寻求支持。
安全性	• 测试自动化工程师无法得到测试自动化工具需要的信息	• 测试自动化工程师需要获得访问权限
不同环境和平台之间的不兼容性	• 测试自动化工具不能在所有的环境/平台上工作	• 实现自动化测试，最大限度地提高工具的独立性，从而最大限度地减少使用多个工具的成本。

2.3 易测试性设计和自动化设计

被测系统（SUT）的易测试性指的是用于支持测试的软件接口（例如，用于控制和观测被测系统）的可用性，它应当和被测系统的其他特性同步设计和实施。虽然这部分工作可以由软件架构师完成（易测试性只是系统非功能需求中的一部分），但实际上这经常是由自动化测试工程师完成或者参与完成的。

易测试性设计包含以下几部分：

- 可观测性：被测系统需要提供接口以便于了解系统的内部实现。测试用例可以使用这些接口来进行检查。例如，验证期望行为是否和实际行为相同；
- 可控性：被测系统需要提供接口以便于用其在被测系统上执行操作。这可以是用户界面（UI）元素、函数调用、通讯元素（例如，TCP/IP 或 USB 协议）、电子信号（用于物理开关）等；
- 清晰定义的架构：易测试性设计的第三个重要部分是架构，它提供了清晰和易懂的接口，在所有测试级别上提供控制和可见性。

测试自动化工程师（TAE）需要考虑用哪种方式可以实现对被测系统的测试（包括自动化测试）是有效的（测试合适的区域并找到严重的缺陷）和高效的（不花费过多工时）。每当需要特定的软件接口时，测试自动化工程师（TAE）必须说明这些接口，并由开发人员实现。定义易测试性是很重要的，如有需要，在项目早期定义需要定义额外的软件接口，这有助于制定工作计划和预算。

一些支持测试的软件接口示例如下：

- 现代电子表格工具提供的强大的脚本功能；
- 运用桩或模拟来仿真那些还不可用的或价格太昂贵的软件和/或硬件（例如，电子金融交易、软件服务、专用服务器、电子板、机械部件），允许在没有这些特定接口的情况下对软件进行测试；
- 软件接口（或者桩和驱动器）可以用来测试错误条件。以带有内部硬盘驱动器（HDD）的设备为例，应当对控制该 HDD 的软件（也成为驱动程序）进行故障或 HDD 的磨损测试。等待 HDD 自己出现故障，再来进行测试是低效的或不可靠的。通过实现软件接口，模拟具有缺陷或较慢的 HDD 就可以验证驱动程序执行的正确性（例如，提供错误信息，重试）；
- 用户界面（UI）尚不可用时，可以使用替代的软件接口来测试被测系统（SUT）（通常认为这是更好的方法）。嵌入式软件往往需要监测设备中的温度，并当温度高于一定水平时，触发冷却功能。在没有硬件的情况下，使用软件接口来指定温度，可以完成该测试；
- 可以使用状态转换测试来评估被测系统（SUT）的状态行为。检查被测系统是否处于正确状态，一种方法是通过定制软件接口来查询。（当然这也包括风险，请参见章节 2.1 中的侵入级别）。

自动化设计应考虑：

- 应尽早建立与现有测试工具的兼容性；
- 测试工具兼容性的问题是至关重要的，因为它可能影响自动化测试重要功能的能力（例如，与网格控件的不兼容将阻止使用该控件的所有测试）；
- 解决方案可能需要开发程序代码并调用 API。

易测试性设计对于一个好的测试自动化做法是至关重要的，同时它也有助于手工测试执行。

3. 通用测试自动化架构—270 分钟

关键词

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），线性脚本（linear scripting），基于模型的测试（model-based testing），过程驱动脚本（process-driven scripting），结构化脚本（structured scripting），测试适配层（test adaptation layer），测试自动化架构（test automation architecture），测试自动化框架（test automation framework），测试自动化方案（test automation solution），测试定义层（test definition layer），测试执行层（test execution layer），测试生成层（test generation layer）

通用测试自动化架构的学习目标

3.1 通用测试自动化架构（gTAA）介绍

ALTA-E-3.1.1 (K2) 解释通用测试自动化架构（**通用测试自动化架构（gTAA）**）的结构

3.2 测试自动化架构（TAA）设计

ALTA-E-3.2.1 (K4) 为给定的项目设计合适的测试自动化架构（TAA）

ALTA-E-3.2.2 (K2) 解释每一个层次在测试自动化架构（TAA）中的角色

ALTA-E-3.2.3 (K2) 理解测试自动化架构（TAA）设计的注意事项

ALTA-E-3.2.4 (K4) 针对给定的测试自动化解决方案（TAS），分析其实施、使用和维护需求的各项要素

3.3 测试自动化解决方案（TAS）开发

ALTA-E-3.3.1 (K3) 应用通用的测试自动化架构（TAA）（即 gTAA）的组件来构建专用的测试自动化架构（TAA）

ALTA-E-3.3.2 (K2) 解释当确定组件的可重用性时需要考虑的因素

3.1 通用测试自动化架构（gTAA）介绍

测试自动化工程师（TAE）负责设计、开发、实施和维护测试自动化解决方案（TAS）。随着每个解决方案的开发，都会有类似的任务需要去完成，类似的问题需要去解答，类似的问题需要解决和确定优先级。在测试自动化中，这些重复出现的概念、步骤和方法构成通用测试自动化架构的基础，简称为 gTAA。

通用测试自动化架构（gTAA）给出了层级、组件及通用测试自动化架构（gTAA）的接口定义，然后重新定义上述内容，成为具体的 TAA。此具体的测试自动化架构（TAA）用来构建特定的测试自动化解决方案（TAS）。这允许通过以下方式，构建测试自动化解决方案的结构化和模块化方法：

- 定义测试自动化解决方案（TAS）的概念空间、层级、服务和接口，使测试自动化解决方案（TAS）可以通过内部以及外部开发的组件得以实现；
- 支持简化的组件，以便有效且高效的开发测试自动化；
- 在不同的产品线和产品家族，以及软件技术和软件工具之间，针对不同的测试自动化解决方案（TAS）或演化中的测试自动化解决方案（TAS），重用测试自动化组件；
- 简化测试自动化解决方案（TAS）的维护和升级；
- 定义测试自动化解决方案（TAS）用户的核心特征。

测试自动化解决方案（TAS）包括测试环境（及其工件）和测试套件（一组测试用例，包括测试数据）。测试自动化框架（TAF）可用于实现测试自动化解决方案（TAS）。测试自动化框架（TAF）有助于实现测试环境，并提供工具，测试用具（harnesses）或支持库。

建议测试自动化解决方案（TAS）的测试自动化架构（TAA）符合以下规则，以便于测试自动化解决方案（TAS）更易开发、演化和维护：

- 单一责任：每个测试自动化解决方案（TAS）组件必须具有单一责任，并且该责任必须完全封装在组件中。也就是说，测试自动化解决方案（TAS）的每个组件应该仅仅负责一件事情，例如，生成关键字或数据、创建测试场景、执行测试用例、记录结果、生成执行报告；
- 扩展（详见 B.Merer 的开放 / 封闭原则）：测试自动化解决方案（TAS）的每个组件必须对扩展开放，对修改封闭。这个原则的意思是，可以丰富组件的行为而不会破坏向后兼容的功能；
- 可替换（详见里氏替换原则）：测试自动化解决方案（TAS）的每个组件都可以在不影响测试自动化解决方案（TAS）的总体行为的情况下被替换。组件可以由一个或多个可替换组件代替，但是表现出的行为是相同的；
- 组件分离（详见 R.C.Martin 的接口分离原则）：相比一个通用的、多功能的组件，若干专用组件更好些。这样可以通过消除不必要的依赖关系来更容易地进行替换和维护；
- 依赖倒置：测试自动化解决方案（TAS）的组件应该依赖于抽象内容而不是低层细节。换句话说，组件不应该依赖于特定的测试自动化场景。

通常，以通用测试自动化架构（gTAA）为基础的测试自动化解决方案（TAS）是由一组工具，插件和/或组件来实现。必须要注意的是，通用测试自动化架构（gTAA）是厂商中立的：它不预先定义实现测试自动化解决方案（TAS）的具体方法、技术或工具。通用测试自动化架构（gTAA）可以通过任何的软件

工程方法去实现，例如结构化，面向对象的，面向服务的，模型驱动以及任何软件技术和工具。事实上，测试自动化解决方案（TAS）通常使用现成的工具来实现，但也需要额外的，对被测系统（SUT）特定的增加和/或者修改。

与测试自动化解决方案（TAS）有关的其他指南和参考模型，可以参见 SDLC（软件开发生命周期）、编程技术、格式化标准等相关的软件工程标准。软件工程不在本教学大纲范围内，但是作为一名测试自动化工程师（TAE）应该具备有软件工程方面的技能、经验和专业知识。

此外，一名测试自动化工程师（TAE）需要了解行业编码、文档标准以及最佳实践，以便于在开发测试自动化解决方案（TAS）时合理使用它们。这些做法可以提高测试自动化解决方案（TAS）的可维护性、可靠性和安全性。这些标准通常是领域相关的。常用标准包括：

- 用于 C 或 C++的 MISRA；
- C++的 JSF 编码标准；
- MathWorks Matlab /Simulink® 的 AUTOSAR 规则。

3.1.1 *通用测试自动化架构（gTAA）概述*

通用测试自动化架构（gTAA）的结构自上而下的划分为：

- 测试生成层；
- 测试定义层；
- 测试执行层；
- 测试适配层。

通用测试自动化架构（gTAA）包含以下内容（见图 1：通用测试自动化架构）：

- 测试生成层，支持手动的或自动化的测试用例设计。它提供了设计测试用例的手段；
- 测试定义层，支持测试套件和/或测试用例的定义和实现。测试定义层将测试定义从被测系统（SUT）和/或测试系统技术和工具中独立出来。它包含定义高层测试和低层测试的手段，这些手段由测试数据的处理、测试用例的处理、测试规程的处理和测试库组件或它们的组合的处理来实现；
- 测试执行层，支持测试用例的执行和测试日志的记录。它提供测试执行工具，自动执行选定的测试用例，并提供记录日志的和报告的组件；
- 测试适配层，针对被测系统（SUT）的不同组件或接口，提供必要的代码适配自动化测试。为了通过 API、协议、服务等方式与被测系统（SUT）连接，它提供了不同的适配模块；
- 通用测试自动化架构（gTAA）还具有与测试自动化相关的项目管理，配置管理以及测试管理的接口。例如，测试管理和测试适配层之间的接口可以选择并配置与所选测试配置有关的合适的适配器。

通用测试自动化架构（gTAA）层级及其组件之间的接口通常是专用的，此处不再赘述。

要明白在任意给定的测试自动化解决方案（TAS）中，这些层不是固定不变的。例如：

- 如果测试执行要自动化，则需要使用测试执行层和测试适配层。它们不需要分离，可以一起实现，例如在单元测试框架中就是如此；
- 如果测试定义要自动化，则需要测试定义层；
- 如果测试生成是自动化的，则需要测试生成层。

通常，我们选择自下而上地实施测试自动化解决方案（TAS），但是也可以使用其他方法，例如用于手动测试的自动测试生成。一般来说，建议以增量方法（例如，sprint）实施测试自动化解决方案（TAS），这样可以尽快使用测试自动化解决方案（TAS）并证明测试自动化解决方案（TAS）的增加的价值。另外，推荐将概念证明作为测试自动化项目的一部分。

所有的测试自动化项目都需要像软件开发项目一样去理解，建立和管理，并需要专门的项目管理。测试自动化框架（TAF）开发的项目管理（即对整个公司、产品系列或产品线的测试自动化支持）可以与测试自动化解决方案（TAS）的项目管理（即具体产品的测试自动化）分开。

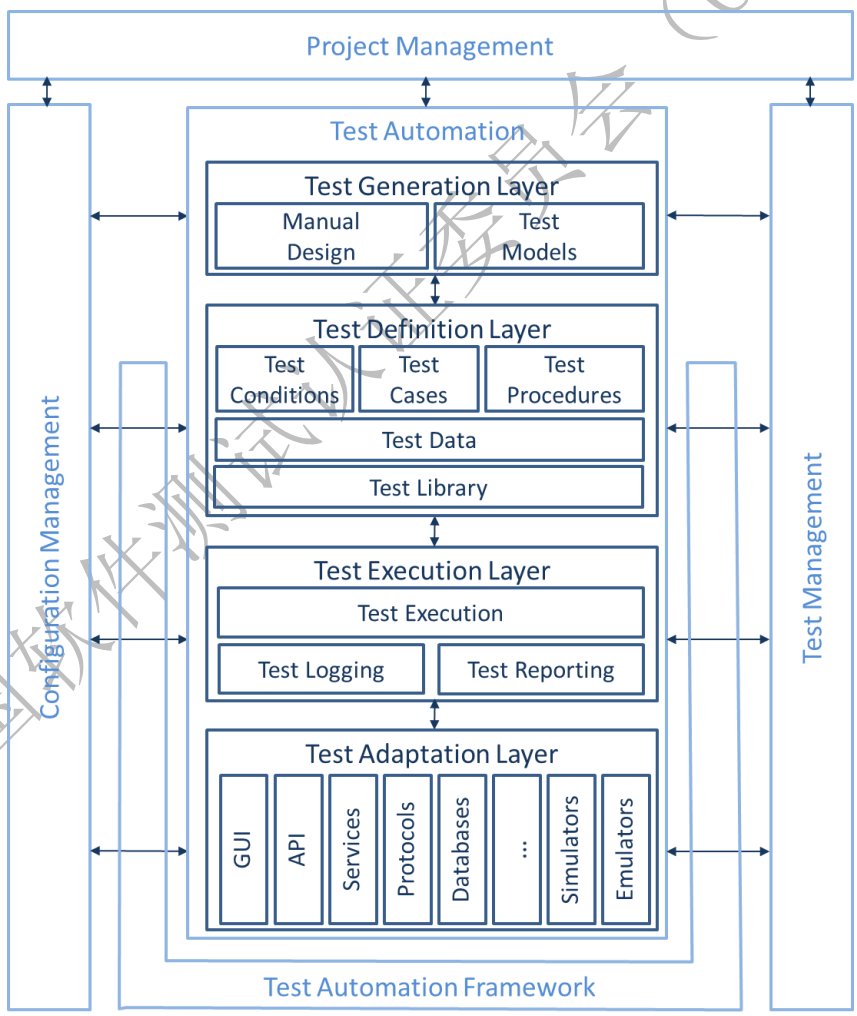

图 1：通用测试自动化架构

3.1.2 *测试生成层*

测试生成层包含以下工具支持：

- 手动的设计测试用例；
- 开发、捕获或导出测试数据；
- 从定义被测系统（SUT）和/或其环境的模型中自动生成测试用例（即自动化的基于模型的测试）。

测试生成层中的组件可用于：

- 编辑和浏览测试套件结构；
- 将测试用例与测试目标或被测系统（SUT）需求相关联；
- 测试设计的文档化。

对于自动测试生成，还可以包括以下功能：

- 能够对被测系统（SUT）及其环境和/或测试系统建模；
- 能够定义测试指令并配置/参数化测试生成的算法；
- 能够从生成的测试用例追溯回模型（元素）。

3.1.3 *测试定义层*

测试定义层包含以下工具支持：

- 描述测试用例（概要和/或详细）；
- 为详细测试用例定义测试数据；
- 为一条测试用例或一组测试用例制定相应的测试规程；
- 定义用于执行测试用例的测试脚本；
- 根据需要，提供对测试库的访问（例如关键字驱动的方法）。

测试定义层中的组件可用于：

- 划分 / 限制、参数化或实例化测试数据；
- 描述测试序列或完全独立的测试行为（包括控制语句和表达式），并对其进行参数化和/或分组；
- 测试数据、测试用例和/或测试程序的文档化。

3.1.4 *测试执行层*

测试执行层包含以下工具支持：

- 自动执行测试用例；
- 记录测试用例的执行日志；
- 报告测试结果。

测试执行层可以由提供以下功能的组件组成：

- 为测试执行而初始化和拆卸被测系统（SUT）；
- 初始化和拆卸测试套件（例如，包括测试数据的一组测试用例）；
- 配置并参数化测试初始化过程；

- 解释测试数据和测试用例，并将其转换为可执行的脚本；
- 操作测试系统和/或被测系统（SUT）来记录（或过滤）测试执行的日志和 / 或故障注入；
- 在测试执行期间分析被测系统（SUT）响应，以引导后续测试运行；
- 确认被测系统（SUT）在自动测试用例执行后的响应（比较预期结果和实际结果）来获得自动化测试执行的结果；
- 及时控制自动化测试的执行。

3.1.5 *测试适配层*

测试适配层包含以下工具支持：

- 控制测试用具（harness）；
- 与被测系统（SUT）的交互；
- 监控被测系统（SUT）；
- 模拟或仿真被测系统（SUT）环境。

测试适配层提供了以下功能：

- 在不依赖于具体技术的测试定义与被测系统（SUT）和测试设备所要求的特定技术之间作为中介；
- 应用不同的、与特定技术相关的适配器来与被测系统（SUT）进行交互；
- 将测试执行分布到多个测试设备或测试接口，或在本地执行测试。

3.1.6 *测试自动化解决方案（TAS）的配置管理*

通常，测试自动化解决方案（TAS）是在各种迭代或版本中进行开发的，并且需要与被测系统（SUT）的迭代或版本兼容。测试自动化解决方案（TAS）的配置管理可以包括：

- 测试模型；
- 测试定义/规范，包含测试数据、测试用例和库；
- 测试脚本；
- 测试执行引擎和辅助工具及组件；
- 用于被测系统（SUT）的测试适配器；
- 被测系统（SUT）环境的模拟器和仿真器；
- 测试结果和测试报告。

以上条目构成了测试件，他们必须为与被测系统（SUT）的版本匹配的正确的版本。在某些情况下，可能需要恢复到之前测试自动化解决方案（TAS）的版本，例如，假如需要使用较老版本的被测系统（SUT）去重现当时的现场。良好的配置管理能够实现这个功能。

3.1.7 *测试自动化解决方案（TAS）的项目管理*

所有的测试自动化项目都是软件项目，它和其他所有的软件项目一样需要进行项目管理。在开发测试自动化解决方案（TAS）时，测试自动化工程师（TAE）需要执行所建立的 SDLC 方法所有阶段的任务。而且，测试自动化工程师（TAE）必须明白，设计开发测试自动化解决方案（TAS）的环境时，其状态

信息（度量）应该能够易于被测试自动化解决方案（TAS）的项目管理人员提取，或者自动报告给测试自动化解决方案（TAS）的项目管理人员。

3.1.8 *测试自动化解决方案（TAS）对测试管理的支持*

测试自动化解决方案（TAS）必须支持被测系统（SUT）的测试管理。应该要简单地、或自动地将包括测试日志和测试结果的测试报告提供给被测系统（SUT）的测试管理（人员或系统）。

3.2 测试自动化架构（TAA）设计

3.2.1 *测试自动化架构（TAA）设计简介*

设计测试自动化架构（TAA）包含了多项主要的活动，可以根据测试自动化项目或组织的需求对这些活动进行排序。在后续章节将讨论这些活动。根据测试自动化架构（TAA）的复杂性，可能会需要更多或更少的活动。

获取定义合适的测试自动化架构（TAA）所需的需求

测试自动化方法的需求需要考虑以下几点：

- 测试过程中的哪个活动或者哪个阶段是应该自动化的，例如，测试管理、测试设计、测试生成或测试执行。值得关注的是测试自动化通过在测试设计和测试实施之间加入了测试生成，从而改变了基本的测试过程；
- 应该支持哪些测试级别，例如，组件级别、集成级别、系统级别；
- 应该支持哪些类型的测试，例如，功能测试、一致性测试、互操作性测试；
- 应该支持哪些测试角色，例如，测试执行者、测试分析师、测试架构师、测试经理；
- 应该支持哪些软件产品，软件产品线，软件产品系列，例如，对于已经实施的测试自动化解决方案（TAS），定义其范围和使用周期；
- 应该支持哪些被测系统（SUT）技术，例如，从被测系统（SUT）技术的兼容性角度来定义测试自动化解决方案（TAS）。

比较不同的设计或架构的方法

测试自动化工程师（TAE）在设计测试自动化架构（TAA）选择层时需要分析不同方法的利弊。包括但不仅限于以下内容：

- 测试生成层的注意事项：
  - 选择手动或自动测试生成；
  - 选择例如：基于需求、基于数据的、基于场景的或基于行为的测试生成；
  - 选择测试生成策略，例如，在使用基于数据方法时的典型分类树覆盖，在基于场景的方法中，使用正常的用例或异常的用例覆盖；在基于行为的方法中，使用转换/状态/路径覆盖；
  - 确定测试选择策略。在现实中完全组合测试生成是不可行的，因为它可能导致测试用例大爆炸。所以，应使用实用的覆盖标准、权重、风险评估等，指导测试生成和随后的测试选择。
- 测试定义层的注意事项：

- 选择数据驱动、关键字驱动、基于模式或模型驱动的测试定义；
- 选择测试定义的表达方式（例如，表格、基于状态的符号、随机的符号、数据流的符号、业务流程的符号、基于场景的符号等，通过使用电子表格，特定领域的测试语言，测试和测试控制符号（TTCN-3），UML 测试配置文件（UTP）等）；
- 选择风格指南和高质量测试定义的指南；
- 选择测试用例存储库。（电子表格、数据库、文件等）。

- 测试执行层的注意事项：
  - 选择测试执行工具；
  - 选择解释（通过使用虚拟机）或编译的方法来实施测试过程 – 该选择通常取决于所使用的测试执行工具；
  - 选择实现测试程序的实现技术（指令性，例如 C;功能性技术，例如 Haskell 或 Erlang;面向对象的技术，如 C++，C＃，Java；脚本技术，如 Python 或 Ruby，或工具特定技术） - 使用哪个选择，取决于所使用的测试执行工具；
  - 选择辅助库以简化测试执行（例如，测试设备库，编码库/解码库等）。
- 测试适配层的注意事项：
  - 选择被测系统（SUT）的测试接口；
  - 选择工具来激发和观察测试接口；
  - 选择在测试执行期间监视被测系统（SUT）的工具；
  - 选择跟踪测试执行的工具（例如，包括测试执行的时间）。

确定哪些领域可以从抽象化技术中获益

测试自动化架构（TAA）中的抽象化可以实现技术独立性，从而使得相同的测试套件可用于不同的测试环境和不同的目标技术。增强了测试工件的可移植性。此外，保证了供应商的中立性，避免了测试自动化解决方案（TAS）的锁定效应。抽象化技术还能够提高新的或正在升级过程中的被测系统（SUT）的可维护性和适应性。此外，抽象化有助于使非技术人员更容易理解测试自动化架构（TAA）（以及测试自动化解决方案（TAS）的实例化），因为测试套件可以文档化（包括图形化手段），并有更详细的解释和说明，从而提高可读性和可理解性。

测试自动化工程师（TAE）需要与利益相关者讨论，在软件开发、质量保证和测试过程中，在测试自动化解决方案（TAS）的哪个区域使用哪种级别的抽象化。例如，测试适配层和/或测试执行层的哪些接口需要被外部化（对外开放）、形式化定义、并在整个测试自动化架构（TAA）使用周期保持稳定？同时也需要讨论，是否使用抽象测试定义，或者测试自动化架构（TAA）是否仅使用只有测试脚本的测试执行层。同样也需要知道，测试生成是否通过使用测试模型和基于模型的测试方法进行抽象化的。在测试自动化架构（TAA）整体的功能性，可维护性和可扩展性方面，测试自动化工程师（TAE）还需要懂得如何平衡实现测试自动化架构（TAA）的复杂度。在测试自动化架构（TAA）中决定使用哪种抽象化技术时，需要考虑这些平衡。

测试自动化架构（TAA）中使用抽象化技术的地方越多，进一步演化或过渡到新方法或新技术时就会越灵活，但这样的初始投资会很巨大（例如，更复杂的测试自动化架构和工具，更高的技能要求，更长的学习曲线），也会使盈亏平衡的时间延迟，但是从长远来看是会得到回报的。当然也可能导致测试自动化解决方案（TAS）的性能降低。

虽然考虑详细的 ROI（投资回报率）是测试自动化经理（TAM）的工作职责，但测试自动化工程师（TAE）需要通过对不同的测试自动化架构和方法进行技术评估和比较，为 ROI 的分析提供时间、成本、工作强度和收益的相关数据。

了解被测系统（SUT）技术以及被测系统（SUT）如何与测试自动化解决方案（TAS）之间进行交互

访问被测系统（SUT）的测试接口是所有自动测试执行的核心。访问级别可以是：

- 软件级别，例如，被测系统（SUT）和测试软件链接在一起；
- API 级别，例如，测试自动化解决方案（TAS）调用（远程）应用程序编程接口提供的功能/操作/方法；
- 协议级别，例如，测试自动化解决方案（TAS）通过 HTTP，TCP 等与被测系统（SUT）进行交互；
- 服务级别，例如，测试自动化解决方案（TAS）通过 Web 服务，RESTful 服务等与被测系统（SUT）服务进行交互。

此外，一旦测试自动化解决方案（TAS）和被测系统（SUT）被 API、协议或服务分开，测试自动化工程师（TAE）就需要决定用于测试自动化解决方案（TAS）和被测系统（SUT）之间交互的范式。这些范式包括以下内容：

- 事件驱动的范式，通过事件总线上的事件交换来驱动交互；
- 客户端-服务器范式，通过服务请求者向服务提供商发起的服务调用来驱动交互；
- 端到端范式，通过来自任一端发起的服务调用来驱动交互。

通常，范式的选择取决于被测系统（SUT）架构，并可能对被测系统（SUT）架构产生影响。需要仔细分析和设计被测系统（SUT）和测试自动化架构（TAA）之间的互连，以便选择两个系统之间未来安全（future-safe）的架构。

了解被测系统（SUT）环境

被测系统（SUT）可以是独立软件，或者是作为其他软件（例如，综合系统）、硬件（例如，嵌入式系统）或环境组成部分（例如，信息物理系统 cyber-physical systems）的一部分进行工作的软件。测试自动化解决方案（TAS）模拟或仿真被测系统（SUT）环境是作为自动化测试设置的一部分。

测试环境实例和使用实例包括以下几个方面：

- 被测系统（SUT）和测试自动化解决方案（TAS）在一台计算机上 – 对于测试一个软件应用程序是有益的；
- 被测系统（SUT）和测试自动化解决方案（TAS）的分布在独立的，网络互联的计算机上 – 对于测试服务器软件是有益的；
- 增加附加的测试设备去触发和观察被测系统（SUT）的技术接口 – 对于测试诸如机顶盒软件是有益的；
- 通过网络化的测试设备模拟被测系统（SUT）的操作环境 – 对于测试网络路由器的软件是有益的；
- 模拟被测系统（SUT）物理环境的模拟器 – 对于测试嵌入式控制单元的软件是有益的。

对于给定的测试件体系结构，其时间和复杂度

虽然测试自动化解决方案（TAS）项目的工作量估算是测试自动化经理（TAM）的责任，但测试自动化工程师(TAE)要良好估算测试自动化架构(TAA)设计的时间和复杂性，以便支持测试自动化经理(TAM)的工作。估算方法和实例包括：

- 基于类比的估算，例如，功能点、三点估算法、（宽带）德尔菲估算法和专家估算；
- 使用任务分解法（WBS）进行估算，例如，在管理软件或项目模板中可以找到的任务；
- 参数估算，例如，构造性成本模型（COCOMO）；
- 基于规模的估算，例如，功能点分析法、故事点分析或用例（Use Case）分析；
- 团队估算，例如，计划扑克。

给定测试件结构实现的易操作性

除了测试自动化解决方案（TAS）的功能、与被测系统（SUT）的兼容性、长期稳定性和可进化能力、工作量要求和投资回报率（ROI）考虑之外，测试自动化工程师（TAE）还应特别的负责处理测试自动化解决方案（TAS）易用性的问题。包括但不限于：

- 面向测试人员的设计；
- 测试自动化解决方案（TAS）的易操作性；
- 测试自动化解决方案（TAS）在软件开发、质量保证和项目管理中对其他角色的支持；
- 在测试自动化解决方案（TAS）中，或使用测试自动化解决方案（TAS）有效组织、引导和搜索；
- 对于测试自动化解决方案（TAS）有用的文档、手册和帮助文本；
- 由测试自动化解决方案（TAS）提供的和有关测试自动化解决方案（TAS）的实践报告；
- 通过迭代设计方法来接受和处理测试自动化解决方案（TAS）的反馈和经验总结。

3.2.2 *自动化测试用例的方法*

测试用例需要翻译成针对被测系统（SUT）执行的动作序列，这些动作序列可以在测试过程中被记录或/和以测试脚本来实现。除了动作之外，自动化测试用例还应该定义与被测系统（SUT）交互的测试数据，并包括验证步骤，以验证被测系统（SUT）达到了预期结果。可以使用许多方法来创建动作序列：

1. 测试自动化工程师（TAE）将测试用例直接转化成可执行的自动化测试脚本。这个选项是最不推荐的，因为它缺少抽象过程并增加维护的工作量；
2. 测试自动化工程师（TAE）设计测试规程，并将其转换为自动化测试脚本。虽然这样具有抽象性，但是不能自动生成测试脚本；
3. 测试自动化工程师（TAE）使用工具将测试规程转化为自动化测试脚本。这样就兼具了抽象性和自动化脚本生成；
4. 测试自动化工程师（TAE）使用工具，直接从模型中自动生成测试规程并转化成测试脚本。这种方法的自动化程度最高。

请注意，这些方法在很大程度上取决于项目的实际情况。当然，如果从使用一个自动化程度不高的方法开始，也许效率会更高些，因为这样通常会更容易实现，也可以在短期内体现出额外的价值，但这也是一个可维护性较差的方案。

建立自动化测试用例较普遍的方法包括：

- 捕获/回放的方法，可用于选项 1；
- 结构化脚本的方法、数据驱动的方法和关键字驱动的方法，可用于选项 2 或 3；

- 基于模型的测试（包括过程驱动的方法），可用于选项 4。

接下来就解释说明一下这些方法的基本概念和各自的优缺点。

捕获/回放方法

基本概念

在捕获/回放方法中，在执行测试规程中定义的动作序列时，工具用来捕获与被测系统（SUT）间的交互。工具不仅捕获输入；工具也可以记录输出，以便后续检查。在事件回放期间，可以手工或者自动地检查输出：

- 手动的：测试人员必须观察被测系统（SUT）的异常输出；
- 完整性：捕获过程中所有的系统输出都要被记录，且可以在被测系统（SUT）上重现；
- 精确性：捕获时所有的系统输出都需要被记录，在被测系统（SUT）上都必须能准确的复现记录细节；
- 检查点：针对特定的数值，只在某些特定点（检查点）上，检查被选系统的输出是否满足定义的值。

优势

捕获/回放的方法可用于被测系统（SUT）的图形用户界面（GUI）或 API 级别上的测试。（常用于测试自动化的）初始阶段，它很容易安装和使用。

劣势

捕获/回放的脚本不易维护和改进，因为捕获被测系统（SUT）的操作强依赖于捕获时所用的被测系统（SUT）的版本。例如，在图形用户界面（GUI）级别上记录时，图形用户界面（GUI）布局的变化可能会影响到测试脚本，即使仅仅是图形用户界面（GUI）元素（在显示屏）的位置改变。因此，捕获/回放的方法很容易受到各种变化的影响。

而且，只能在被测系统（SUT）可用之后才可开始实现测试用例（脚本）。

线性脚本

基本概念

与所有的脚本技术一样，线性脚本也是从手动测试过程开始的。注意，虽然这些（知识）可能没有整理在文档内 -- 但总有测试分析师（TA）掌握关于运行什么样的测试以及如何运行这些测试的知识。

当手工执行每个测试用例时，测试工具会记录动作的序列，并在某些情况下捕获从被测系统（SUT）到屏幕的可见输出，这将导致每个测试规程都在一个（通常很大）脚本里。可以编辑（已经）记录的脚本以提高可读性（例如，通过对一些关键点添加注释来解释发生的事情），或使用该工具的脚本语言增加更多的检查。

可以通过工具回放脚本，让工具重复测试人员在录制脚本时做的相同操作。虽然这个方法可以用来自动化图形用户界面（GUI）测试，但对于需要大量自动化的测试来说，这种方法并不适用，而且需要有很多的软件（脚本）版本，这是因为被测系统（SUT）一旦有所变化，就需要高额的维护成本（被测系统（SUT）中的每个改变，都需要对已记录的脚本进行大量修改）。

优势

线性脚本的优势就是在开始自动化之前几乎不需要做什么准备工作。一旦学会使用这个工具，就只是录制手动测试并回放它（尽管其中的录制部分可能需要与测试工具进行额外的交互，需要比较实际输出与预期输出，验证软件当前是在正常工作的）。虽然编程技能不是必需的，但通常是会有帮助的。

劣势

线性脚本的缺点很多。给定的测试规程实施自动化所需的工作量主要取决于执行它所需的规模（步骤数或动作数量）。因此，要自动化第一千个测试规程与自动化第一百个测试规程的工作量几乎类似。换句话说，降低新建自动化测试成本的空间并不大。

此外，如果有第二个脚本执行与前一个脚本类似的测试，虽然具有不同的输入值，但是该脚本包含了与前一个脚本相同的指令序列，只是指令中包含的信息/数据（可理解为指令参数或参量）不同而已。如果有几个测试（以及对应的脚本）包含相同的指令序列，每当软件的变化影响到这些脚本时，所有这些受影响的脚本都需要维护。

因为脚本是编程语言，而不是自然语言，非程序员可能会很难理解它们。一些测试工具使用（工具独有）专用的语言，所以就需要时间学习这种语言以便熟练使用它。

录制的脚本中只包含通用的注释（如果有注释的话）。特别是长脚本需要详细的注释，以解释测试的每个步骤都发生了什么。注释将提高可维护性。当测试包含很多步骤时，脚本（包含许多指令）规模很快就变得非常大。

这些脚本是非模块化的，难以维护的。线性脚本并不遵循常用的软件可重用性和模块化的范例，它们通常与工具紧密耦合。

结构化脚本

基本概念

结构化脚本技术和线性脚本技术之间的主要区别是引入了一个脚本库。脚本库中包括可重复使用的脚本，这些脚本可以在多个测试中通用，完成一系列的操作。结构化的脚本最好的例子就是接口，例如，被测系统（SUT）的接口操作。

优势

这种方法的优势在于明显减少了因修改引起的维护工作量，并且可以降低自动化新测试时的成本（因为可以使用已有脚本，而不必从头开始创建它们）。

结构化脚本的优点主要是通过脚本的重用来实现的。无需创建线性脚本所需的脚本数量，就可以自动化更多的测试。这直接影响了构建和维护的成本。第二次测试及随后的测试将不会花费太多自动化的成本，因为可以重用为第一个测试而创建的脚本。

劣势

最初创建共享脚本所需的工作量可以看作是一个缺点，但只要合理应用，这个初始的投资应该会产生很大的回报/收益。在结构化脚本中简单的录制操作是不够的，创建所有脚本都将需要编程技能。脚本库必须妥善管理，脚本应该文档化，技术测试分析师（测试自动化架构（TAA））应该很容易就能找到所需的脚本（因此，一个合理的命名规则会很有帮助）。

数据驱动测试

主要概念

数据驱动的脚本技术是基于结构化脚本技术。最重要的区别是如何处理测试输入。（数据驱动测试是）从脚本中提取输入并放入一个或多个单独的文件（通常称为数据文件）中。

这意味着主测试脚本可以被其他很多的测试重用（而不仅仅是单个测试）。通常这种“可重用”的主测试脚本被称为“控制”脚本。控制脚本包含执行测试所需的指令序列，但从数据文件中读取输入数据。一个控制测试（此应为控制脚本）可以用于多个测试，但通常不足以覆盖广泛的自动化测试。因此，需要若干个控制脚本，但这只是自动化测试数量的一部分。

优势

使用数据驱动的脚本技术，可以大大减少增加新的自动化测试的成本。这个技术可以针对一个有用测试的很多变体，在特定区域进行更深入的测试，并可增加测试覆盖率。

通过数据文件“描述”测试，意味着测试分析师（TA）只要简单地通过选定一个或多个数据文件就可以指定“自动化”测试。这使得测试分析师（TA）可以更自由地指定自动化测试，而不需要过多依赖于技术测试分析师（TTA），而 TTA 往往可能是个稀缺资源。

劣势

这种方法的缺点是需要管理数据文件，并确保它们可以被测试自动化解决方案（TAS）读取，但这个缺点可以妥善得到解决。

此外，可能会缺失或忽略重要的逆向测试用例，逆向测试是测试规程和测试数据的组合。在主要针对测试数据的方法中，可能会忽略“逆向测试规程”。

关键字驱动测试

主要概念

关键字驱动的脚本技术基于数据驱动的脚本技术，它们之间有两个主要区别：（1）数据文件现在被称为“测试定义”文件或类似名字；（2）只有一个控制脚本。

测试定义文件包含测试描述，与等效的数据文件相比，测试分析师（TA）更易于理解该文件，它通常包含和数据文件一样的数据，但关键字文件（测试定义文件）还包含高级指令（“关键字”或“动作单词”）。

在选择关键字时，应该考虑选择那些对于测试分析师、被描述的测试、以及被测试的应用程序有意义的关键字。这些关键字大部分（但不完全）用于表示与系统的高层业务交互（例如“下单”）。

每个关键字表示与被测系统的一组详细交互。关键字序列（包括相关测试数据）用于定义测试用例。特殊关键字可用于验证步骤，或者关键字本身可以同时包含操作和验证步骤。

测试分析师（TA）的职责范围包括创建和维护关键字文件。这意味着，一旦实现了（针对关键字）的支持脚本，测试分析师（TA）可以简单通过在关键字文件中指定关键字（与数据驱动的脚本类似）来添加“自动化”测试。

优势

一旦编写了关键字的控制脚本和支持脚本，通过此脚本技术将大大减少添加新的自动化测试的成本。

通过关键字文件“描述”测试意味着测试分析师（TA）可以通过使用关键字和关联数据描述的测试来定义“自动化”测试。这使得测试分析师可以更自由地定义自动化测试，而无需过多依赖技术测试分析师（TTA 可能是个稀缺资源）。关键字驱动的方法优于数据驱动的方法，其原因是使用了关键字。每个关键字应该表示一系列详细的动作，能够产生有意义的结果。例如，“创建帐户”，“下单”，“检查订单状态”都是网络购物应用程序的所有可能的具体操作，每个都涉及一些详细的步骤。当一名测试分析师向另一名测试分析师介绍系统测试时，他们可能就这些抽象操作而不是详细步骤进行讨论。因此，关键字驱动方法的目的，就是实现这些抽象操作，并允许根据抽象操作定义测试，而无需参考详细步骤。

这些测试用例易于维护、阅读和编写，因为复杂性可以隐藏在关键字中（如果使用结构化脚本，复杂性会隐藏在库中）。关键字可以提供被测系统（SUT）接口复杂度的抽象表达。

劣势

实现关键字对于测试自动化工程师来说是一项重大任务，特别是在使用不支持此脚本技术的工具时。对于小型系统来说，实施该技术的开销可能远超收益。

需要注意确保实施正确的关键字，好的关键字可用于许多不同的测试中，而差的关键字可能只会使用一次或仅几次。

过程驱动型脚本

主要概念

过程驱动的方法基于关键字驱动的脚本技术，但使用的不同场景构成脚本，这里使用了体现被测系统（SUT）和其变体的用例（use cases），这些脚本将测试数据参数化，或组合到更高层次的测试定义中。

这样的测试定义更容易描述动作之间的逻辑关系，例如在特征测试中的“下单”之后应该是“检查订单状态”，或在健壮性测试中，在还没有“下单”就去“检查订单状态”。

优势

使用类似过程的、基于场景的测试用例能够从工作流视角来定义测试规程。过程驱动方法的目的是通过使用描述详细测试步骤的测试库来实现这些高级别的工作流（另见关键字驱动方法）。

劣势

技术测试分析师可能不太容易理解被测系统（SUT）的过程，在面向过程的脚本实现方面也是如此，尤其是如果工具不支持业务流程逻辑的情况下。

还需要确保使用正确的关键字实现正确的过程。好的过程将被其它过程引用，并产生许多相关的测试。而糟糕的过程则不会在关联性和错误检测能力等方面得到回报。

基于模型的测试

主要概念

基于模型的测试指自动化的生成测试用例，与自动化执行测试用例不同（自动化执行测试用例使用录制/回放，线性脚本，结构化脚本，数据驱动脚本或过程驱动脚本）。基于模型的测试使用正式（半正式）模型。这些模型与测试自动化架构（TAA）使用的脚本技术处于不同的抽象层次。可以使用不同的测试生成手段去为之前讨论的任意的脚本框架生成测试。

优势

基于模型的测试可以通过抽象让测试集中在需要被测试的本质内容上，比如被测试的业务逻辑、数据、场景、配置等。同时，也允许针对不同的目标系统和相应的技术生成测试，这样用于测试生成的模型由未来安全（future-safe）的测试件的呈现方式组成，在技术发展时易于重用和维护。

如果需求发生改变，那么只需要调整测试模型，就可以自动生成一套完整的测试用例。测试用例设计技术已包含在测试用例生成器中。

劣势

需要有建模的专业知识才能有效运行基于模型的测试方法。通过抽象被测系统（SUT）的接口、数据和/或行为进行建模的任务可能很困难。另外，建模和基于模型的测试工具还不是主流，而是正在成熟。基于模型的测试方法需要在测试过程中进行调整。例如，需要设立测试设计师的角色。此外，用于测试生成的模型构成了被测系统（SUT）质量保证的主要工件，并且它同样需要质量保证和维护。

3.2.3 *关于被测系统（SUT）的技术考虑*

在设计测试自动化架构（TAA）时应考虑被测系统（SUT）的技术方面。其中一些将在下面讨论，尽管这不是一个完整的列表，但可以作为重点考虑因素的样本。

被测系统（SUT）的接口

被测系统（SUT）具有内部接口（系统内部）和外部接口（与系统环境及其用户的接口或暴露在外的组件）。测试自动化架构（TAA）需要能够控制和/或观察可能受测试规程影响的被测系统（SUT）的所有接口（即接口需要具备可测性）。此外，还可能需要以不同的详细程度去记录被测系统（SUT）和测试自动化解决方案（TAS）之间的交互，通常包括时间戳（时间记录）。

在项目启动时（或在敏捷环境中是持续的），在架构定义期间需要关注测试重点，以验证所需的测试接口是否有效，或检查被测系统（SUT）所需的设备是否可测（易测试性设计）。

被测系统（SUT）数据

被测系统（SUT）使用配置数据来控制其实例化、配置、管理等。此外，被测系统（SUT）处理过程中还要使用用户数据。被测系统（SUT）还可以使用来自其他系统的外部数据去完成任务。根据被测系统（SUT）的测试规程，所有这些类型的数据都是可定义的、可配置的并且能够由测试自动化架构（TAA）实例化。在测试自动化架构（TAA）的设计中确定了如何处理被测系统（SUT）数据的具体方法，根据该方法，数据可以作为参数、测试数据表、测试数据库、实际数据等处理。

被测系统（SUT）配置

被测系统（SUT）可以部署在不同的配置中，例如，在不同的操作系统上、在不同的目标设备上、或者使用不同的语言设置。根据测试规程，不同的被测系统（SUT）的配置可能需要由测试自动化架构（TAA）来解决。测试规程可能需要不同测试设置（在实验室中）或测试自动化架构（TAA）的虚拟测试设置（在云中）与给定的被测系统（SUT）配置相结合。也可能需要为选定的被测系统（SUT）某些特性添加所选被测系统（SUT）组件的模拟器和/或仿真器。

被测系统（SUT）标准和法律规定

除了被测系统（SUT）的技术方面，测试自动化架构（TAA）的设计还要遵守法律和/或标准的要求，以便测试自动化架构（TAA）的设计合规。例如，要考虑到测试数据的私密性要求，或者影响到测试自动化架构（TAA）的记录和报告功能的保密性要求。

用于开发被测系统（SUT）的工具和工具环境

在被测系统（SUT）的开发过程中，不同的工具可用于被测系统（SUT）的需求工程、设计和建模、编码、集成和部署。测试自动化架构（TAA）连同它自己的工具应该考虑到被测系统（SUT）工具环境，以便实现工具的兼容性、可追溯性和/或重用性。

软件产品的测试接口

强烈建议不要在产品发布之前删除所有测试接口。大多数情况下，这些接口可以留在被测系统（SUT）中，不会导致最终产品出问题。如果保留这些接口，服务和支持工程师可以使用这些接口诊断问题以及测试维护版本。当然，需要验证这些测试接口不会带来安全隐患。如果需要，开发人员可以禁用这些测试接口，以确保它们在开发部门之外不能被使用。

3.2.4 *开发过程/质量保障过程的考虑*

在设计测试自动化架构（TAA）时，应考虑被测系统（SUT）的开发和质量保证过程的各个方面。其中一些将在下面讨论，尽管这不是一个完整的列表，但可以作为重要的样本。

测试执行的控制需求

根据测试自动化架构（TAA）所要求的自动化程度，可能需要测试自动化架构（TAA）的支持来完成交互式测试执行、批处理式测试执行或全自动测试执行。

报告要求

根据报告要求（包括报告类型和结构），测试自动化架构（TAA）需要能够以不同的格式和布局支持固定的、参数化的或定义的测试报告。

角色和访问权限

根据安全性要求，测试自动化架构（TAA）可能需要提供角色及系统的访问权限。

已建成的工具环境

被测系统（SUT）项目管理、测试管理、代码和测试库、缺陷跟踪、事件管理、风险分析等，都可以由工具支持，这些工具与现有的工具构成了工具环境。测试自动化架构（TAA）也同样需要工具或工具集的支持，并且该工具或工具集需要与已建成的工具环境无缝集成。另外，测试脚本也应该遵循与被测系统（SUT）代码相同的存储和版本管理要求，以便二者的修改流程能遵循相同的过程。

3.3 测试自动化解决方案（TAS）开发

3.3.1 *测试自动化解决方案（TAS）开发过程介绍*

测试自动化解决方案（TAS）的开发与其他软件开发项目相似，它遵循相同的程序和过程，包括开发人员和测试人员的同行评审。测试自动化解决方案（TAS）的特殊之处在于与被测系统（SUT）的兼容和同步，这些需求都要在测试自动化架构（TAA）设计（见第 3.2 节）和测试自动化解决方案（TAS）开发中予以考虑。此外，被测系统（SUT）会受测试策略的影响，例如，必须保证测试接口对测试自动化解决方案（TAS）可用。

本节使用软件开发生命周期（SDLC）来解释测试自动化解决方案（TAS）的开发过程及与被测系统（SUT）的流程方面相关的兼容性和同步性。这些方面对于其它开发过程同样重要，包括已选中的或已就位的被测系统（SUT）和/或测试自动化解决方案（TAS）开发 – 它们需要相应调整。

测试自动化解决方案（TAS）的基本 SDLC 如图 2 所示

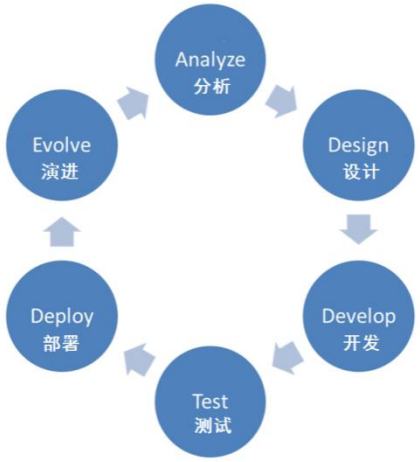

图 2： 测试自动化解决方案（TAS）的基本 SDLC

需要对测试自动化解决方案（TAS）的一系列需求进行分析和收集（见图 2）。我们需要按照测试自动化架构（TAA）的定义，指导测试自动化解决方案（TAS）的设计（见第 3.2 节）。然后通过软件工程的方法将设计转化成软件。请注意，测试自动化解决方案（TAS）还可能使用专用的硬件测试设备，这不在本大纲的考虑之列。与其他软件一样，测试自动化解决方案（TAS）也需要被测试，首先是测试自动化解决方案（TAS）的基本功能测试，然后是测试自动化解决方案（TAS）与被测系统（SUT）之间的互操作测试。在部署测试自动化解决方案（TAS）并开始使用之后，测试自动化解决方案（TAS）也还是需要不断演进、增加更多测试功能、修改测试、按照被测系统（SUT）的变更升级测试自动化解决方案（TAS）等。根据 SDLC，测试自动化解决方案（TAS）演化需要新一轮测试自动化解决方案（TAS）的开发。

同样要注意的，SDLC 中未显示测试自动化解决方案（TAS）的备份、归档和拆卸过程。与测试自动化解决方案（TAS）的开发过程一样，这些程序应该遵循组织中的既定方法。

3.3.2 *测试自动化解决方案（TAS）和被测系统（SUT）之间的兼容性*

过程兼容

被测系统(SUT)的测试应与其开发同步 – 并且，在测试自动化的情况下，与测试自动化解决方案(TAS)开发同步。因此，最好能够协调被测系统（SUT）开发过程、测试自动化解决方案（TAS）的开发过程以及测试过程。当被测系统（SUT）和测试自动化解决方案（TAS）开发在流程结构、流程管理和工具支持方面兼容时，可以获得很大的收益。

团队兼容

团队兼容性是测试自动化解决方案（TAS）与被测系统（SUT）开发兼容性的另一个方面。如果使用兼容的思维模式来处理和管理测试自动化解决方案（TAS）和被测系统（SUT）开发，两个团队都将从审查对方的需求、设计和/或开发工件、问题讨论、查找兼容的解决方案等方面获益。团队兼容性也有助于彼此的沟通和互动。

技术兼容

此外，应考虑测试自动化解决方案（TAS）和被测系统（SUT）之间技术的兼容性。从一开始就设计和实现测试自动化解决方案（TAS）和被测系统（SUT）之间的无缝交互是有益的。虽然有时在技术层面无法做到兼容(因为技术解决方案可能无法同时适用于测试自动化解决方案(TAS)和被测系统(SUT))，但可以通过使用适配器、封装器或其他的中介形式进行无缝交互。

工具兼容

需要考虑测试自动化解决方案（TAS）和被测系统（SUT）在管理工具、开发工具和质量保证工具方面的兼容性。例如，如果使用相同的需求管理和/或问题管理工具，就更容易协调信息交换以及测试自动化解决方案（TAS）和被测系统（SUT）的开发。

3.3.3 *在测试自动化解决方案（TAS）和被测系统（SUT）之间同步*

需求同步

在获取需求之后，将实现（开发）被测系统（SUT）和测试自动化解决方案（TAS）需求。测试自动化解决方案（TAS）的需求主要归纳为两个方面：（1）测试自动化解决方案（TAS）作为一个软件系统的需求，如测试设计、测试定义、测试结果分析等的测试自动化解决方案（TAS）特征需求；（2）通过测试自动化解决方案（TAS）对被测系统（SUT）进行测试的需求。这些是与被测系统（SUT）需求对应的测试需求，并反映了测试自动化解决方案（TAS）要测试的所有被测系统（SUT）的特征和属性。每当更新被测系统（SUT）或测试自动化解决方案（TAS）需求时，必须验证两者之间的一致性，并确认所有由测试自动化解决方案（TAS）测试的被测系统（SUT）需求都已经定义了测试需求。

开发阶段同步

为保证在测试被测系统（SUT）时测试自动化解决方案（TAS）已经就绪，需要协调两者的开发过程。当被测系统（SUT）和测试自动化解决方案（TAS）需求、设计、定义和实现同步时，效率最高。

缺陷追踪同步

缺陷可能与被测系统（SUT）、测试自动化解决方案（TAS）或需求/设计/定义有关。由于两个项目之间的关系，其中一个缺陷得到修正时，修正动作就可能会影响另一个。缺陷跟踪和确认测试必须同时在测试自动化解决方案（TAS）和被测系统（SUT）中得到处理。

被测系统（SUT）和测试自动化解决方案（TAS）演化同步

被测系统（SUT）和测试自动化解决方案（TAS）都可能发展，以适应新特征或禁用某些特征、修正缺陷或处理环境的变化（包括分别对被测系统（SUT）和测试自动化解决方案（TAS）的变更，因为一个是另一个的环境组件）。任意一个应用于被测系统（SUT）或测试自动化解决方案（TAS）的变更都可能会影响到另一个，所以应该同时处理针对被测系统（SUT）和测试自动化解决方案（TAS）的变更。

图 3 和图 4 展示了两种被测系统（SUT）和测试自动化解决方案（TAS）开发过程之间的同步方法。

图 3 所示的方法中，用于被测系统（SUT）和测试自动化解决方案（TAS）的两个 SDLC 主要在两个阶段同步：（1）测试自动化解决方案（TAS）的分析是基于被测系统（SUT）设计，被测系统（SUT）设计则是基于被测系统(SUT)的分析;(2)使用已部署的测试自动化解决方案(TAS)测试被测系统(SUT)。

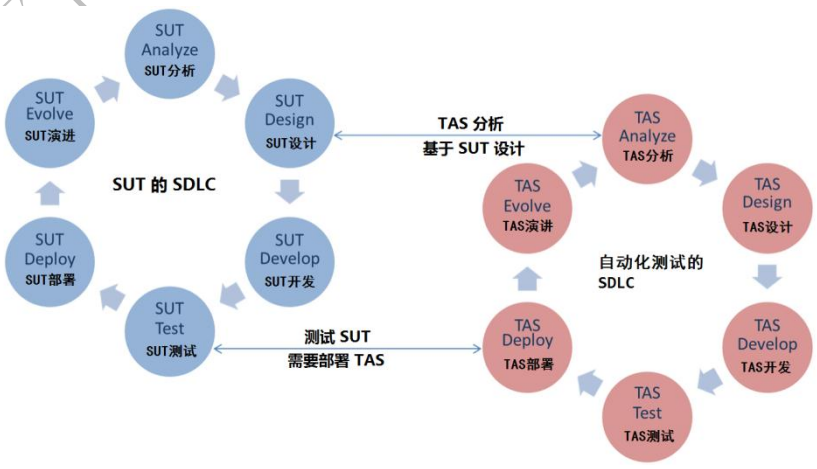

图 3：测试自动化解决方案（TAS）和被测系统（SUT）开发过程同步例子 1

图 4 显示了手动测试和自动测试的混合方法。每当在自动化测试之前使用手动测试，或者手动测试和自动化测试一起使用时，测试自动化解决方案（TAS）分析应同时以被测系统（SUT）的设计和手动测试为依据。这样，测试自动化解决方案（TAS）将与两者同步。第二个主要同步点如前所述：被测系统（SUT）测试需要用到已经部署的测试，在手动测试的情况下，测试被测系统（SUT）只需要遵循手动测试规程即可。

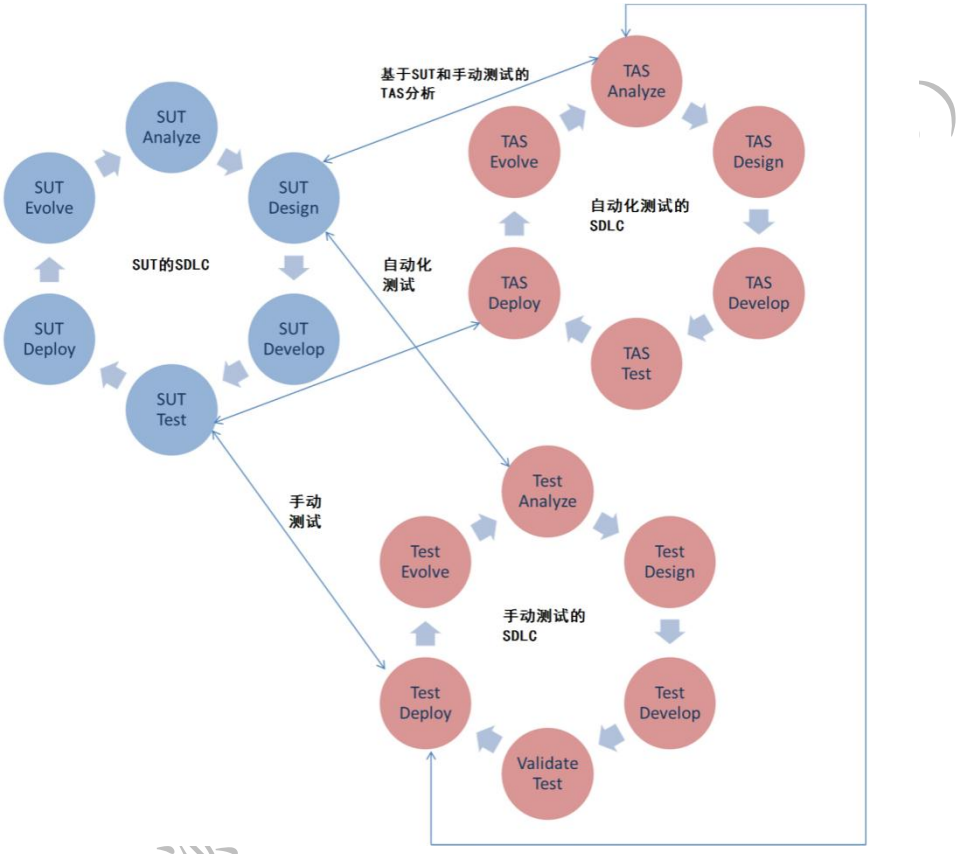

图 4： 测试自动化解决方案（TAS）和被测系统（SUT）开发过程同步例子 2

3.3.4 *在测试自动化解决方案（TAS）中创建重用*

重用测试自动化解决方案（TAS）是指在产品线、产品框架、产品域和/或项目系列之间重用测试自动化解决方案（TAS）工件（在其架构的任意级别）。测试自动化解决方案（TAS）的重用需求来自于产品的变化或者项目的变化需求。可以重用的测试自动化解决方案（TAS）工件包括：

- （部分）测试目标、测试场景、测试组件或测试数据的测试模型；
- （部分）测试用例，测试数据，测试规程或测试库本身；
- 测试引擎和/或测试报告框架；
- 被测系统（SUT）组件和/或接口的适配器。

如果测试自动化架构（TAA）的定义中，已经考虑到重用需求，那么测试自动化解决方案（TAS）可以通过以下方式提高重用的能力：

- 遵循测试自动化架构（TAA）或在需要时进行修改和更新；

- 文档化测试自动化解决方案（TAS）工件，使其易于理解，并可融入到新的上下文中；
- 保证任意测试自动化解决方案（TAS）工件的正确性，以便在新的上下文中，测试自动化解决方案（TAS）工件由于它的高质量而仍然可用。

请注意，虽然可重用设计是测试自动化架构（TAA）要重点关注的问题，但重用的维护和改进应该贯穿在整个测试自动化解决方案(TAS)的生命周期中。一定要持续关注和努力实现测试自动化解决方案(TAS)的重用，衡量并展示可重用设计带来的附加价值，并且告知众人：尽可能重用已有的测试自动化解决方案（TAS）。

3.3.5 *支持不同的目标系统*

测试自动化解决方案（TAS）支持各种目标系统指的是测试自动化解决方案（TAS）有测试软件产品不同配置的能力。不同的配置适用于以下任意一种：

- 被测系统（SUT）组件的互连数目；
- 运行被测系统（SUT）组件的环境（包括软件和硬件）；
- 用于实现被测系统（SUT）组件的技术、编程语言或操作系统；
- 被测系统（SUT）组件使用的函数库和程序包；
- 用于实现被测系统（SUT）组件的工具。

前四个方面影响测试自动化解决方案（TAS）的任意测试级别，最后一个方面主要适用于组件级别和集成级别的测试。

在定义测试自动化架构（TAA）时，就已确定测试自动化解决方案（TAS）测试不同软件产品配置的能力。然而对于软件产品的不同配置，测试自动化解决方案（TAS）必须有实现处理技术差异的能力，并且支持测试自动化解决方案（TAS）特征和组件的管理。

测试自动化解决方案（TAS）应对软件产品的多样性时，可以采用不同的方法：

- 可以使用测试自动化解决方案（TAS）和被测系统（SUT）的版本/配置管理来提供分别适合于测试自动化解决方案（TAS）和被测系统（SUT）的版本管理和配置管理；
- 将测试自动化解决方案（TAS）参数化，可使测试自动化解决方案（TAS）适应被测系统（SUT）的配置。

请注意，虽然测试自动化解决方案（TAS）的可变性设计是测试自动化架构（TAA）关注的主要问题，但其维护和改进应该贯穿整个测试自动化解决方案（TAS）的生命周期。需要持续关注和努力来修订、增加以及甚至删除某些可变性选项和形式。

4. 部署的风险和应急处理- 150 分钟

关键字

风险（risk），风险缓解（risk mitigation），风险评估（risk assessment），产品风险（product risk）

部署的风险和应急处理的学习目标

4.1 选择测试自动化方法和部署/推广的计划

ALTA-E-4.1.1 (K3) 应用有效的测试工具的试点和部署的准则

4.2 风险评估与缓解策略

ALTA-E-4.2.1 (K4) 分析可能导致测试自动化项目失败的部署风险，识别可能导致测试自动化项目失败的技术问题，制定缓解策略

4.3 测试自动化维护

ALTA-E-4.3.1 (K2) 了解哪些因素会支持和影响测试自动化解决方案（TAS）的可维护性

4.1 选择测试自动化方法和部署/推广的计划

实施和推广测试自动化解决方案（TAS）主要包含两个活动：试点和部署。这两个活动的具体步骤依赖于测试自动化解决方案（TAS）的类型和具体情况而有所不同。

关于试点，应至少考虑以下步骤：

- 确定合适的项目；
- 计划试点；
- 执行试点；
- 评估试点。

关于部署，应至少考虑以下步骤：

- 确定初始目标项目；
- 在选定项目中部署测试自动化解决方案（TAS）；
- 在预定期限后监测和评估测试自动化解决方案（TAS）；
- 向全组织或其它项目推广。

4.1.1 *试点项目*

工具的实施通常开始于某个试点项目。试点项目的目的是确保测试自动化解决方案（TAS）可以实现预期的收益。试点项目的目标包括：

- 掌握更多有关测试自动化解决方案（TAS）的详细信息；
- 了解测试自动化解决方案（TAS）与现有流程，程序和工具的契合度，明确可能需要的修改的地方。（通常会优选修改测试自动化解决方案（TAS），使其符合现有流程/程序，如果确实需要其它调整来“支持测试自动化解决方案（TAS）”，那么这种调整至少应该对现有流程来说是一种改进）；
- 设计自动化接口，以满足测试人员的需求；
- 确定使用、管理、存储和维护测试自动化解决方案（TAS）和测试资产的标准方法，包括与配置管理和变更管理的集成（例如，确定文件和测试的命名规范，创建库并定义测试套件的模块化规则）；
- 确定监视测试自动化运行的指标和测量方法，包括易用性，可维护性和可扩展性；
- 评估是否能以合理的成本实现收益。这将是使用测试自动化解决方案（TAS）时，重新调整预期的一次机会；
- 确定需要哪些技能，哪些当前已具备，还缺少哪些技能。

确定合适的项目

应按照如下准则仔细选择试点项目：

- 避免选择关键的项目。当测试自动化解决方案（TAS）的部署导致延误时，不应该对关键项目产生重大影响。在开始阶段测试自动化解决方案（TAS）的部署会花费不少时间。项目组应该意识到这一点；
- 不要选择简单的项目。简单项目不适合做试点项目，因为在简单项目上的部署成功并不意味着在非简单项目上也能取得成功，并且这样的试点为部署提供的信息较少；
- 将必要的利益相关方（包括管理层）都引入到试点项目的选择流程中；
- 试点项目的被测系统（SUT）应该是组织内其他项目的一个好的参考或榜样，例如，被测系统（SUT）应包含必须被自动化的具有代表性的图形用户界面（GUI）组件。

计划试点

应将试点项目视为规范的开发项目来运作：制定计划，预算经费和资源，报告进度，确定里程碑等。另外额外注意之处，是确保部署测试自动化解决方案（TAS）的人员（即：指定人员）有足够的精力用在此项工作上，尽管其他项目也需要他们。关于这一点，重要的是要得到管理层的承诺，特别是在共享的人力资源上，他们很可能无法全职投入部署工作。

如果测试自动化解决方案（TAS）不是由供应商提供，而是内部自主开发，那么相应的开发人员也需要参与部署活动。

执行试点

执行部署试点并注意以下几点：

- 测试自动化解决方案（TAS）是否提供了预期的功能（并由供应商承诺过）？如果没有，这需要尽快解决。如果测试自动化解决方案（TAS）是由内部开发，相应的开发人员需要提供任何缺失的功能来协助部署过程的顺利进行；
- 测试自动化解决方案（TAS）和现有流程是否相互支持？如果不是，则需要它们协调一致。

评估试点

召集所有利益相关者参与评估。

4.1.2 *部署*

只有当试点评估的结果是成功的，测试自动化解决方案（TAS）才可以被部署到部门/组织内的其他单位。推广部署应逐步实施并加以有效管理。成功的部署取决于以下几方面：

- 增量推广：以增量方式逐步在组织内其他部门部署推行。用这种方法，对新用户的支持工作就会分批以“波浪式前行”方式发生，而不是一次性全部到来。这样使测试自动化解决方案（TAS）的使用逐步增加。那么可能出现的瓶颈问题也会在成为实际问题前被确定和解决。如果需要，可以考虑添加许可证；
- 适应和改进流程以配合测试自动化解决方案（TAS）的使用：当不同的用户使用测试自动化解决方案（TAS）时，不同的流程也随之与测试自动化解决方案（TAS）相关联，并且需要调整适应测试自动化解决方案（TAS），或测试自动化解决方案（TAS）可能需要小的改变来适应流程；
- 为新用户提供培训和指导/辅导：新用户需要培训和指导去使用新的测试自动化解决方案（TAS）。确保这些工作到位。在实际使用测试自动化解决方案（TAS）之前，应向用户提供培训/研讨会；

- 定义使用指南：编写测试自动化解决方案（TAS）的使用指南，清单和常见问题解答。这样可以规避过多的支持问题；
- 收集实际使用中的信息：采用自动化方法来收集有关测试自动化解决方案（TAS）实际使用情况的信息。理想情况下，这些信息不仅包括使用测试自动化解决方案（TAS）的情况，还有测试自动化解决方案（TAS）（某些功能）的哪些部分被使用的信息。用这种方式，可以容易地监视测试自动化解决方案（TAS）的使用情况；
- 监测测试自动化解决方案（TAS）的使用，收益和成本：在一段时间内监控测试自动化解决方案（TAS）的使用情况，可以了解测试自动化解决方案（TAS）是否确实被使用。该信息还可用于重新计算业务案例（例如，节省了多少时间，阻止了多少问题）；
- 为指定测试自动化解决方案（TAS）的测试和开发团队提供支持；
- 收集所有团队的经验教训：与使用测试自动化解决方案（TAS）的不同团队进行评估/回顾会议。这样可以指出经验教训。团队会感觉他们的投入是有必要的，并希望进一步改善测试自动化解决方案（TAS）的使用；
- 确定和实施改进：根据团队的反馈和对测试自动化解决方案（TAS）的监测，确定并实施改进步骤，并且要与利益相关者明确沟通。

4.1.3 *在软件生命周期内部署测试自动化解决方案（TAS）*

测试自动化解决方案（TAS）的部署，在很大程度上依赖于测试自动化解决方案（TAS）将要测试的软件项目所处的开发阶段。

通常，一个新的测试自动化解决方案（TAS）或一个新版本的测试自动化解决方案（TAS）的部署，或者发生在项目的开始阶段，或者是项目到达一个里程碑，如代码冻结或冲刺的结束。这是因为部署测试自动化解决方案（TAS）涉及所有测试和修改，需要相当的时间和精力。同时，这也是避免由于测试自动化解决方案（TAS）不工作而导致测试自动化过程中断的好方法。然而，为修复测试自动化解决方案（TAS）的严重问题，或者更换测试自动化解决方案（TAS）运行环境的某个组件而实施的部署，则需要独立于被测系统（SUT）的开发阶段进行。

4.2 风险评估与缓解策略

技术问题可能导致产品或项目风险。典型的技术问题包括：

- 过度的抽象可能导致难以理解真正发生的行为（例如，使用关键字）；
- 数据驱动：数据表可能变得太大/复杂/繁琐；
- 测试自动化解决方案（TAS）对于某些操作系统库或组件的依赖性：测试自动化解决方案（TAS）使用了某些操作系统库或组件，但这些库或组件可能并不是在所有被测系统（SUT）的目标环境中都可用。

典型的部署项目风险包括：

- 人员配置问题：找到合适的人员维护代码库可能很困难；
- 新被测系统（SUT）可能造成测试自动化解决方案（TAS）操作不正确；
- 引入自动化的延迟；

- 由于被测系统（SUT）的更改而造成更新测试自动化解决方案（TAS）的延迟；
- 测试自动化解决方案（TAS）无法捕获要跟踪的（非标准）对象。

测试自动化解决方案（TAS）项目的潜在故障点包括：

- 迁移到不同的环境；
- 部署到目标环境；
- 开发部门新提交的功能。

下文所述的一些风险缓解策略，可用于处理上述风险。

无论是内部开发还是直接获取的解决方案，测试自动化解决方案（TAS）都具有自己的软件生命周期。像任何其他软件一样，测试自动化解决方案（TAS）需要进行版本控制并有文档记录其功能。否则，部署测试自动化解决方案（TAS）的不同部分，并使它们可以一起工作，可能在某些环境中将变得非常困难。

此外，必须清晰且易于遵循地，将部署步骤文档化保存。该部署过程与版本相关，因此也必须处于版本控制之下。

在部署测试自动化解决方案（TAS）时有两种不同的场景：

1. 首次部署
2. 维护部署 - 测试自动化解决方案（TAS）已经存在并需要维护

在首次部署测试自动化解决方案（TAS）之前，务必确保它可以在自己的环境中运行，不可随机更改，并且可以更新和管理测试用例。测试自动化解决方案（TAS）及其基础架构必须可维护。

在首次部署的情况下，需要以下基本步骤：

- 定义运行测试自动化解决方案（TAS）的基础架构；
- 创建测试自动化解决方案（TAS）的基础架构；
- 创建用于维护测试自动化解决方案（TAS）及其基础架构的流程；
- 创建用于维护测试自动化解决方案（TAS）所执行的测试套件的流程。

与首次部署相关的风险包括：

- 测试套件的总执行时间可能比测试周期中计划的执行时间长。在这种情况下，重要的是确保测试套件在下一个计划的测试周期开始之前获得足够的时间执行完毕；
- 测试环境的安装和配置问题（例如，数据库设置和初始加载，服务启动/停止）。一般来说，测试自动化解决方案（TAS）需要一种有效的方法，在测试环境中为已自动化的测试用例，设置所需的前提条件。

关于维护部署，还有其他注意事项。测试自动化解决方案（TAS）本身需要演进和发展，其更新必须部署到生产环境中。在将更新版本的测试自动化解决方案（TAS）部署到生产环境之前，需要像任何其他软件一样进行测试。因此，有必要检查新功能，验证测试套件可以在更新后的测试自动化解决方案（TAS）

上运行，可以发送报告，且不会出现性能或其他功能回归问题。在某些情况下，为适应新版本的测试自动化解决方案（TAS），可能需要升级整个测试套件。

维护部署时，需要执行以下步骤：

- 评估新版本的测试自动化解决方案（TAS）与旧版本之间的变化；
- 测试测试自动化解决方案（TAS）的新功能和回归测试；
- 检查测试套件是否需要适配新版本的测试自动化解决方案（TAS）。

测试自动化解决方案（TAS）的一次更新会引发以下风险和相应的缓解措施：

- 更改测试套件，使其在升级后的测试自动化解决方案（TAS）上运行：对测试套件进行必要的更改，并在部署到测试自动化解决方案（TAS）之前对其进行测试；
- 测试中使用的桩、驱动器和接口都需要更改以适配升级的测试自动化解决方案（TAS）：对测试工具进行必要的更改，并在部署到测试自动化解决方案（TAS）之前对其进行测试；
- 基础架构需要更改以适应升级的测试自动化解决方案（TAS）：对需要更改的基础架构组件进行评估，执行更改并使用升级的测试自动化解决方案（TAS）进行测试；
- 升级的测试自动化解决方案（TAS）有其他缺陷或性能问题：对风险与收益进行分析。如果发现的问题导致无法更新测试自动化解决方案（TAS），最好不要继续更新或等待下一个版本的测试自动化解决方案（TAS）。如果问题与收益相比可以忽略不计，测试自动化解决方案（TAS）仍然可以更新。确保发布说明中包含已发现的问题，通知测试自动化工程师和其他利益相关者，并尝试估计何时可以修复问题。

4.3 测试自动化维护

开发测试自动化解决方案不是一件简单的事情，它需要模块化、可扩展、可理解、可靠和可测试。而且，测试自动化解决方案也像其他软件系统一样，必须不断演进和发展。无论是由于内部变化还是运行环境的变化，维护都是构建测试自动化解决方案（TAS）过程中的一个重要环节。维护测试自动化解决方案（TAS）可能是为适应新的待测系统，或为增加支持新的软件环境，或为遵从新的法律法规，这都有助于确保测试自动化解决方案（TAS）的可靠和安全运行。它同时也优化了测试自动化解决方案（TAS）的使用寿命和性能。

4.3.1 *维护类型*

维护工作是针对现有运营中的测试自动化解决方案（TAS），并由系统的修改、迁移或退役所触发。此过程可以分为以下几类：

- 预防性维护 – 这类维护是为了使测试自动化解决方案（TAS）支持更多测试类型，在多个接口上测试，测试多个版本的被测系统（SUT）或支持在新的被测系统（SUT）运行测试自动化；
- 纠正性维护 – 这类维护是为了修复测试自动化解决方案（TAS）的故障。确保测试自动化解决方案（TAS）正常运行并降低使用风险的最佳方式是执行定期维护测试；

- 完善性维护 – 这类维护的目的是优化测试自动化解决方案（TAS），并修复非功能性问题。它可以维持测试自动化解决方案（TAS）的性能稳定，确保系统可用性，鲁棒性（健壮性）或可靠性；
- 适应性维护 - 随着市场上推出新的软件系统（操作系统，数据库管理系统，网络浏览器等），可能需要测试自动化解决方案（TAS）支持这些新系统。此外，测试自动化解决方案（TAS）可能需要遵守新的法律、法规或行业指定规范。在这种情况下，需要对测试自动化解决方案（TAS）进行更改使其适应变化。注意：一般情况下，为遵守法律法规规定了强制性维护，这类维护指定了具体规则和要求，有时候还包括审计要求。此外，随着集成工具的更新和新版本的创建，工具集成终端也需要维护和保持功能的可用。

4.3.2 *范围和方法*

维护可能影响测试自动化解决方案（TAS）的所有层级和组件。其影响范围取决于：

- 测试自动化解决方案（TAS）的规模和复杂度；
- 修改的力度；
- 修改带来的风险。

考虑到维护是针对运行中的测试自动化解决方案（TAS），所以很有必要进行影响分析来确认修改是如何影响系统的。根据影响的具体情况，逐步引入修改，并在每个步骤之后进行测试，以确保测试自动化解决方案（TAS）的连续运行。注意：如果测试自动化解决方案（TAS）的功能规范和其它相关文档已过时，维护可能会遇到麻烦。

由于时间效率是衡量测试自动化是否成功的主要因素，因此良好的实践方法对于维护测试自动化解决方案（TAS）至关重要，它们包括：

- 测试自动化解决方案（TAS）的部署过程和使用说明必须非常明晰并记录在案；
- 第三方的依赖关系，以及相关缺陷和已知问题必须记录在案；
- 测试自动化解决方案（TAS）必须是模块化设计的，因此它的每一部分都可以轻松更换；
- 测试自动化解决方案（TAS）必须运行在可替换的环境中，或环境中的组件可替换；
- 测试自动化解决方案（TAS）必须将测试脚本与测试自动化框架（TAF）本身分开；
- 测试自动化解决方案（TAS）必须与开发环境隔离开，这样确保对测试自动化解决方案（TAS）的更改不会对测试环境产生不利影响；
- 测试自动化解决方案（TAS），连同环境、测试套件和测试工具都必须进行配置管理。

第三方组件和其他库的维护注意事项如下：

- 通常情况下，测试自动化解决方案（TAS）将使用第三方组件来运行测试。测试自动化解决方案（TAS）也可能依赖于第三方库（例如，用户界面（UI）自动化库）。测试自动化解决方案（TAS）的所有第三方组件都必须进行文档化和配置管理；
- 有必要制定一个应对计划以防这些外部组件需要修改或修正。负责测试自动化解决方案（TAS）维护的人员需要了解相关联系人或在哪提交问题；
- 必须有关于使用第三方组件的许可证的文件，以便了解是否可以对其进行修改，以及什么人可以在什么程度上进行修改；

- 对于每个第三方组件，都需要获取有关更新和新版本的信息。保持第三方组件和库的及时更新，从长远看，是一项保证投资回报率的措施。

命名标准和其他惯例的注意事项包括：

- 制定命名标准和其他约定的基本原因就是：测试套件和测试自动化解决方案（TAS）本身必须易于阅读、理解、更改和维护。这样可以节省维护时间，同时最大限度地减少引入那些其实是很容易避免的回归或错误修复的风险；
- 当使用标准命名约定时，更容易培训新人加入测试自动化项目；
- 命名标准指的是变量和文件、测试场景、关键字和关键字参数。其他约定指的是测试执行的前提条件和后期操作、测试数据的内容、测试环境、测试执行状态以及执行日志和报告；
- 启动测试自动化项目时，所有标准和惯例都必须达成一致并记录在案。

文档相关的考虑要点：

- 需要有完好且最新的文档描述测试场景和测试自动化解决方案（TAS），这一点众所周知。但有两个问题与此有关：必须有人负责编写该文档和维护它；
- 虽然测试工具的代码可以自动生成文档或半自动生成文档，但所有设计、组件、与第三方的集成、依赖关系和部署过程都需要有人记录在案；
- 将文档编写作为开发过程的一部分是很好的实践经验。除非有相关文档记录或文档更新，否则一项任务不应被视为完成。

培训材料相关的考虑要点：

- 如果测试自动化解决方案（TAS）的文档写得好，可以作为测试自动化解决方案（TAS）培训材料的基础；
- 培训材料应综合测试自动化解决方案（TAS）的所有功能说明、设计和架构、部署和维护、用户手册、实践示例和练习以及使用技巧和窍门等；
- 培训材料的维护工作包括初始编写，定期审核。在实践中，一般由指定为测试自动化解决方案（TAS）培训师的团队成员来完成，并且最有可能是在被测系统（SUT）的生命周期迭代结束时（例如在冲刺结束时）进行。

5. 测试自动化报告与度量-165 分钟

关键词

自动化代码缺陷密度（automation code defect density），覆盖率（coverage），可追溯性矩阵（traceability matrix），等效的手动测试工作量（equivalent manual test effort），度量（metrics），测试日志（test logging），测试报告（test reporting）。

测试自动化报告和度量的学习目标

5.1 选择测试自动化解决方案（TAS）度量

ALTA-E-5.1.1 (K2) 可用于监控测试自动化策略和有效性的度量的分类。

5.2 测量实施

ALTA-E-5.2.1 (K3) 实现度量收集方法来支持技术和管理的要求。解释如何实现测试自动化的度量。

5.3 测试自动化解决方案（TAS）和被测系统（SUT）的测试日志

ALTA-E-5.3.1 (K4) 分析测试自动化解决方案（TAS）和被测系统（SUT）的测试日志。

5.4 测试自动化报告

ALTA-E-5.4.1 (K2) 解释如何构建和发布测试执行报告。

5.1 测试自动化解决方案（TAS）度量的选择

本节重点介绍可用于监控测试自动化策略以及测试自动化解决方案（TAS）的有效性和效率的度量。这些度量与用于监视被测系统（SUT）和被测系统（SUT）的（功能和非功能）测试的被测系统（SUT）相关度量是不同的。那些度量由项目的总体测试经理选定。测试自动化度量使得测试自动化经理（TAM）和测试自动化工程师（TAE）能够跟踪测试自动化目标的进度，并监控对测试自动化解决方案所做更改的影响。

测试自动化解决方案（TAS）度量可以分为两类：外部的度量和内部的度量。外部度量是衡量测试自动化解决方案（TAS）对其他活动（特别是测试活动）影响的度量。内部度量是衡量测试自动化解决方案（TAS）在实现目标方面有效性和效率的度量。

可测量的测试自动化解决方案（TAS）度量通常包括以下内容：

- 外部测试自动化解决方案（TAS）度量
  - 自动化收益；
  - 建立自动化测试的工作量；
  - 分析自动测试事件的工作量；
  - 维护自动化测试的工作量；
  - 失效与缺陷的比例；
  - 执行自动化测试的时间；
  - 自动化测试用例数目；
  - 通过和失败结果的数目；
  - 假失败和假通过结果的数量；
  - 代码覆盖。
- 内部测试自动化解决方案（TAS）指标
  - 工具脚本度量；
  - 自动化代码缺陷密度；
  - 测试自动化解决方案（TAS）组件的速度和效率。

上述度量详述如下：

自动化收益

测量和报告测试自动化解决方案（TAS）的益处尤为重要。这是因为成本（例如，在一段指定时间内参与的人数）很容易看到。在测试团队之外的人能够对整体的成本建立印象，但可能看不到获得的收益。

度量任何收益都取决于测试自动化解决方案（TAS）的目标。通常，这可能是节省时间或人力，增加执行的测试量（覆盖广度或覆盖深度，或执行频率），或其他一些优势，如增加可重复性，更高效的资源利用或更少的手动错误。这些可能的度量包括：

- 节省的人工测试工时数量；

- 减少执行回归测试的时间；
- 增加的测试执行周期次数；
- 增加的测试执行的数量或百分比；
- 自动化测试用例占整套测试用例的百分比（尽管自动化不能简单的与手动测试用例进行比较）；
- 覆盖上的增加（需求、功能、结构）；
- 由于测试自动化解决方案（TAS）而较早发现的缺陷数量（当早期发现的缺陷的平均收益是已知的，可以将其计算为预防成本的总和）；
- 通过手动测试不会发现而通过测试自动化解决方案(TAS)发现的缺陷数量(例如，可靠性缺陷)。

我们注意到测试自动化通常可以节省手动测试工作量。节省下来的工作量可以用于其他类型的（手动）测试（例如，探索性测试）。这些附加测试发现的缺陷也可以被视为测试自动化解决方案（TAS）的间接收益，因为测试自动化才使得执行这些手动测试成为可能。没有测试自动化解决方案（TAS），这些测试将不会被执行，因而将不能发现这些额外缺陷。

建立自动化测试的工作量

实现测试自动化的工作量是与测试自动化相关的重要成本之一。这通常超过手动运行相同测试的成本，因此可能不利于推广测试自动化的使用。虽然实施特定自动化测试的成本在很大程度上取决于测试本身，但其他因素，如所使用的脚本技术，是否熟悉测试工具，环境以及测试自动化工程师的技术水平也会对其产生影响。

因为较大或较复杂的测试通常比那些较短或更简单的测试需要更长时间来实现测试自动化，所以计算测试自动化的构建成本可以基于平均构建时间。也可以通过特定的一组测试的平均成本进一步予以修正，例如针对相同功能的测试或在给定测试级别的测试。另一种方法是将测试自动化的构建工作量作为手动运行测试所需的工作量的一个比例（等效手动测试工作量，EMTE）来表达。例如，可能需要两倍手动测试工作量来自动化一个测试用例，或两倍等效手工测试工作量（EMTE）。

分析被测系统（SUT）故障的工作量

分析通过自动测试执行发现的被测系统（SUT）中的故障可能比分析手动执行的测试所发现的故障，要复杂得多，因为运行测试的测试人员通常知晓导致手动测试失败的事件。按照 3.1.4 中的设计级别和第 5.3 和 5.4 章的报告级别的描述要求进行操作可以缓解这个风险。该度量可以表示为每个失败的测试用例的平均值，也可以表示为等效手工测试工作量（EMTE）的一个因数。后者特别适用于复杂性和执行长度上变化显著的自动化测试情况。

可以获得的被测系统（SUT）和测试自动化解决方案（TAS）日志记录在分析故障时起着至关重要的作用。日志应提供足够的信息来提高分析的效率。重要的日志特性包括：

- 被测系统（SUT）日志记录和测试自动化解决方案（TAS）日志记录应该同步；
- 测试自动化解决方案（TAS）应该记录预期行为和实际行为；
- 测试自动化解决方案（TAS）应该记录要执行的操作。

另一方面，被测系统（SUT）应记录执行的所有操作（无论操作是手动还是自动测试的结果）。任何内部错误都应该被记录，并且确保能跟踪任何崩溃的转储和堆栈记录都可用。

维护自动化测试的工作量

将自动化测试与被测系统（SUT）保持同步更新所需的维护工作量会非常大，其成本最终可能超过测试自动化解决方案（TAS）带来的益处。这也是许多自动化工作失败的原因。因此，监控维护的工作量就非常重要，特别是当采取合适的措施来减少维护工作量时，或至少要防止维护工作量的无限增长。

维护工作量的度量可以表示为被测系统（SUT）的每个新版本进行维护的所有自动化测试的工作量的总和。它们也可以表示为每个更新的自动化测试的平均工作量或以等效手工测试工作量（EMTE）的因数表示。

另一个相关的度量是需要维护的测试的数量或百分比。

当知道（或可以推导出）自动化测试的维护工作量时，该信息可以在决定是否实现某些功能或修复某些缺陷方面发挥关键作用。当被测系统（SUT）变化时，必须考虑因软件变更而引起的测试用例维护工作量。

失效与缺陷的比例

自动化测试的一个常见问题是，许多测试可能因为软件中的同一个缺陷而失败。虽然测试的目的是突显出软件中的缺陷，但多个测试用例揭示同一个缺陷则是浪费。对于自动化测试尤其如此，因为分析每个失败的测试所需的工作量都会很大。度量特定缺陷导致的自动化测试用例失败的数量有助于判断多个测试由于同一缺陷而失败是否成为一个严重问题。解决方案在于自动测试的设计及其有选择的执行。

执行自动化测试的时间

一个简单的度量是执行自动化测试所需的时间。在测试自动化解决方案（TAS）的开始阶段，这可能并不重要，但随着自动化测试用例数量的增加，这个度量可能变得非常重要。

已自动化的测试用例数

该度量可用于显示测试自动化项目的进度。但是必须注意：自动测试用例的数量并没有显示太多有用的信息；例如，它并不表明测试覆盖增加。

通过和失败结果的数量

这是一个通用的度量，它可以追踪多少个自动化测试用例通过，以及有多少个未能达到期望的结果。必须分析失败原因，以确定其是由于被测系统（SUT）中的缺陷引起的，还是由于诸如环境问题或测试自动化解决方案（TAS）本身之类的外部问题引起的。

假失败和假通过的用例数量

如前文的几个度量所示，分析测试失败可能需要相当长的时间。如果这是一个误报，就更令人沮丧。当问题出现在测试自动化解决方案（TAS）或测试用例中而不是被测系统（SUT）时，会发生这种情况。务必要控制假警报的数量（潜在的浪费人力），应将其保持在较低水平。虚假失败会降低用户对测试自动化解决方案(TAS)的信心。相反，虚假通过的结果可能更危险。在虚假通过的案例中，被测系统(SUT)中是存在缺陷的，但该缺陷没有被自动化测试识别出来，所以自动化测试报告该测试用例通过。没有检测到潜在的缺陷可能是因为：结果验证不正确，使用无效的测试方法或测试用例的预期结果错误。

虚假警报可能由测试代码中的缺陷引起（参见度量“自动化代码缺陷密度”），但也可能是因被测系统（SUT）运行不稳定而造成的不可预测的行为（例如，超时）导致。由于测试钩子的侵入的级别不同，也可能引起误报。

代码覆盖

了解不同测试用例提供的被测系统（SUT）代码覆盖可以揭示有用的信息。这也可以在高级别中度量，例如回归测试套件的代码覆盖。并没有绝对的百分比数值能够表示覆盖率足够。没有一个绝对的百分比能表示特别合适的覆盖率。除了最简单的软件应用程序之外，100%的代码覆盖率几乎是无法实现的。然而，普遍认为覆盖越高越好，因为它降低了软件部署的总体风险。该度量也可以揭示被测系统（SUT）的活动。例如，如果代码覆盖率下降，这很可能意味着有功能添加到被测系统（SUT）中，但没有将相应的测试用例添加到自动化测试套件中。

工具脚本度量

有许多可用于监控自动化脚本开发的度量。大多数类似于被测系统（SUT）的源代码度量。代码行（LOC）和圈复杂度可用于标识过于庞大或复杂的脚本（这样的脚本可能需要重新设计）。

注释与可执行语句的比率可以用来表示脚本文档化和注释程度。不符合脚本编写标准的违规数量可以表明对脚本编写标准的遵守程度。

自动化代码缺陷密度

自动化代码与被测系统（SUT）的代码没有区别，因为它是软件，也会包含缺陷。自动化代码的重要程度与被测系统（SUT）代码相当。应该采用良好的编码实践和标准，并通过诸如代码缺陷密度等度量来监测其结果。使用配置管理系统可以较容易地收集这些信息。

测试自动化解决方案（TAS）组件的速度和效率

在同一环境中执行相同测试步骤所需的时间如果有差异，可能表示被测系统（SUT）中存在缺陷。如果被测系统（SUT）在相同的时间内没有执行相同的功能，则需要进行分析。这可能表明系统的可变性是不可接受的，并且可能会随着负载的增加而恶化。测试自动化解决方案（TAS）的执行不应妨碍被测系统（SUT）的性能。如果性能是被测系统（SUT）的关键要求，测试自动化解决方案（TAS）设计需要重点考虑这一点。

趋势度量

在上述这些度量指标中，趋势（即，度量随时间变化的方式）可能比特定时刻的度量数值更有报告价值。例如，已知维护的每个自动化测试的平均维护成本大于之前的两个被测系统（SUT）版本的平均维护成本，可能表明需要采取行动确认增加的原因，并采取措施扭转这个趋势。

度量的成本应尽可能低，这通常可以通过自动收集和自动报告来实现。

5.2 度量的实施

由于测试自动化策略都以自动化测试件作为自己的核心，因此可以增强自动化测试件以记录有关其使用的信息。如果将抽象技术与结构化的自动化测试件结合在一起，对底层自动化测试件的任何增强，都可用于所有较高级别的自动化测试脚本。例如，提升底层自动化测试件来记录执行测试的开始和结束时间就可应用于所有测试。

支持度量和报告生成的自动化功能

许多测试工具的脚本语言都支持通过能够记录单个的测试、测试集和整个测试套件的执行前、中、后的信息的功能以进行度量和报告。

对每个已经有多次执行的测试的报告需要有一个特性分析来考虑先前的测试运行结果，这样就能更好的突出趋势（如测试成功率的变化）。

自动化测试通常需要测试执行和测试验证的自动化。测试验证，是将测试输出中的特定元素与预定义的预期输出进行比较。这种比较最好由测试工具进行。还要考虑比较结果的报告信息级别。测试用例的结果状态（例如，通过或失败）要正确确认。如果是失败，将需要提供有关故障原因的更多信息（例如屏幕截图）。

区分测试的实际和预期结果之间的预期差异并不总是容易的，但是通过工具的支持能提供很大的帮助，例如忽略预期的差异（如日期和时间），同时突出显示任何非预期差异。

与其他第三方工具集成（电子表格、XML、文档、数据库、报表工具等）

当自动测试用例的执行信息将要用于其他工具（用于跟踪和报告，例如更新可追溯性矩阵）时，应该以适合于这些第三方工具的格式提供信息。通过测试工具的已经具备的功能（报告的导出格式），或创建定制的报告与其他程序要求的格式一致（如 Excel 要求的“.xls”、Word 要求的“.doc”、网页要求的“.html”）可以达到此目的。

结果的可视化（仪表板、图表、图形等）

测试结果应在图表中进行可视化。可以使用颜色来指示测试执行中的问题，例如，红绿灯，以指示测试执行/自动化的进度，以便根据报告的信息进行决策。管理层对可视化摘要特别感兴趣，可以一眼就看到测试结果；如果需要更多的信息，他们仍可以继续挖掘细节。

5.3 测试自动化解决方案（TAS）和被测系统（SUT）的日志

日志记录在测试自动化解决方案（TAS）中非常重要，包括测试自动化本身和被测系统（SUT）的日志记录。测试日志经常用于分析潜在问题。以下几节是按测试自动化解决方案（TAS）或被测系统（SUT）分类的测试日志示例。

测试自动化解决方案（TAS）日志（是测试自动化框架（TAF）的日志还是测试用例本身的记录信息不是很重要，日志内容取决于上下文）应包括以下内容：

- 目前正在执行哪个测试用例，包括开始时间和结束时间；
- 测试用例执行的状态，尽管在日志文件中可以轻松识别故障，但是框架本身也应该具有这些信息，并且应该通过仪表板进行报告。测试用例的执行状态可以是通过、失败或测试自动化解决方案（TAS）错误。测试自动化解决方案（TAS）错误的结果针对不是被测系统（SUT）出现故障的情况；
- 高层级的测试日志的详细信息（记录重要步骤），包括时间信息；
- 测试用例能够借助第三方工具识别的被测系统（SUT）的动态信息（例如内存泄漏）。在检测到事件时，正在执行的测试用例及这些动态测量的实际结果和失败信息应同时被记录下来；
- 在可靠性测试/压力测试（循环执行多次）的情况下，应该有一个计数器，以便可以轻松确定测试用例已执行多少次；
- 当测试用例具有随机的部分（例如，随机参数或状态机测试中的随机步骤）时，应记录随机数和随机的选择；
- 测试用例执行的所有操作都应该以这样的方式进行记录：日志文件（或其部分内容）是可以回放的，并且以完全相同的步骤和相同的时间重新执行。这可用于检查已识别故障的可重现性并捕获附加信息。测试用例的动作信息也可以记录在被测系统（SUT）本身，以便再现客户识别的问题时使用（客户运行场景，捕获日志信息，然后在对问题进行故障排除时由开发团队回放）；
- 在测试执行期间，可以保存屏幕截图和其他视觉捕获信息，以便在故障分析期间使用；
- 一旦测试用例遇到失效，测试自动化解决方案（TAS）应确保分析问题所需的所有信息都可用并已存储，有关继续测试的任何信息（如果适用）也应该如此。任何相关的崩溃转储和堆栈跟踪都应由测试自动化解决方案（TAS）保存到一个安全的位置。任何可能被覆盖的日志文件（循环缓冲区经常用于被测系统（SUT）上的日志文件）亦应被复制到这个安全的位置用于后续分析；
- 使用颜色有助于区分不同类型的记录信息（例如，错误用红色表示，进度信息用绿色表示）。

被测系统（SUT）日志：

- 当被测系统（SUT）发现一个问题时，应记录分析问题所需的所有必要信息：包括日期和时间戳，问题的来源位置，错误信息等；
- 被测系统（SUT）可以记录所有的用户交互（可直接通过用户界面，也可以通过网络接口等）。这样就可以正确地分析客户识别的问题，开发人员可以据此尝试重现问题；
- 系统启动时，应将配置信息记录到一个文件中，该文件由不同软件/固件版本，被测系统（SUT）配置，操作系统配置等组成。

所有不同的日志信息都应该易搜索。在测试自动化解决方案（TAS）的日志文件中识别的问题应该在被测系统（SUT）的日志文件中容易地识别，反之亦然（基于或不基于附加工具）。可以使用时间戳同步各种日志，这有助于关联报告错误时发生的情况。

5.4 测试自动化报告

测试日志能够提供有关测试用例和/或测试套件的执行步骤、操作和响应的详细信息。但是，单独的日志不能提供对整体执行结果的良好概述。为此，有必要具备报告功能。每次执行测试套件后，必须创建并发布简明的报告。可以采用可重复使用的报告生成器组件来完成这个任务。

报告内容

测试执行报告必须包含摘要，概述执行结果、被测试的系统以及运行测试的环境，以适用于每个利益干系人。

有必要知道哪些测试失败，以及失败的原因。为了使故障排除更容易，重要的是要知道测试执行的历史和负责人（通常是创建测试或最后更新它的人）。负责人需要调查失败的原因，报告与之相关的问题，对问题的修复进行跟踪，并检查修复是否已正确实施。

报告也用于诊断测试自动化框架（TAF）组件的任何故障（参见第 7 章）。

发布报告

这份报告需要发布给所有对执行结果感兴趣的人。可以把报告上传到网站上，发送到邮件列表或上传到另一个工具，如测试管理工具。从实践的角度来看，如果对执行结果感兴趣的人能够通过便利的订阅、或通过电子邮件接收报告，他们就很有可能仔细阅读报告并对其进行分析。

还有一个选择是，识别被测系统（SUT）的有问题的部分，保留报告的历史，以便能够收集经常回归的测试用例或测试套件的统计信息。

6. 将手工测试转换到自动化测试环境-120 分钟

关键词

确认测试（confirmation testing），回归测试（regression testing）

将手工测试转换到自动化测试环境的学习目标

6.1 自动化的准则

ALTA-E-6.1.1 (K3) 应用准则来决定适合自动化的测试

ALTA-E-6.1.2 (K2) 理解从手工测试转换到自动化测试的因素

6.2 识别回归测试中实现自动化需要的步骤

ALTA-E-6.2.1 (K2) 解释实现自动化回归测试需要考虑的因素

6.3 为新功能测试实现自动化需要考虑的因素

ALTA-E-6.3.1 (K2) 解释为新功能测试实现自动化需要考虑的因素

6.4 实现自动化确认测试需要考虑的因素

ALTA-E-6.4.1 (K2) 解释自动化确认测试需要考虑的因素

6.1 自动化的准则

传统上，组织已经开发了手工测试用例。当决定向自动化测试环境转移，需要评估手工测试的当前状态，并决定最有效的方式来将这些测试资产实现自动化。现有的手工测试结构有的适合自动化，有的不适合自动化。在不适合自动化的情况下可能需要重写测试用例。或者，从现有的手工测试用例中提取相关组成部分（例如，输入值、期望值、浏览路径）在自动化测试中重用。一个考虑测试自动化的手工测试策略，允许调整测试结构以迁移到自动化测试。

不是所有的测试用例都能或者都应该被自动化。有时，第一次迭代要手工执行。因此，在从手工测试转移到自动化测试需要考虑两方面的问题：现有的手工测试用例到自动化用例的初始转换，和新增加的手工测试用例随后转换为自动化用例。

并且注意到某些类型的测试只能通过自动化的方式执行（才能有效），比如，可靠性测试、压力测试或者性能测试等。

在测试自动化中，可以不通过用户界面对应用或系统进行测试。在这种情况下，可以通过软件内的接口在集成测试层次进行测试。这些测试用例虽然可以手工执行（例如，通过手工输入命令来触发接口），但这样并不实际。例如，通过自动化手段，可以在一个消息队列系统中插入消息。这样测试可以早些开始（可以在较早期发现缺陷），而此时手工测试还不能进行。

在开始投入自动化测试的人力之前，需要考虑手工测试和自动化测试的实用性和可行性。需要考虑的适用性条件包括但不限于：

- 使用的频率；
- 自动化的复杂度；
- 工具支持的兼容性；
- 测试流程的成熟度；
- 适合自动化的软件产品生命周期阶段；
- 自动化测试环境的持续性；
- 对被测系统（SUT）的控制力。

下面，就每一点进行详细解释。

使用的频率

在考虑是否需要进行自动化时，测试执行频率是一个重要的考虑因素。那些有规律反复执行的测试用例，作为主要或次要发布周期的一部分，由于它们经常被使用，所以是更好的自动化的候选。作为一般性的规则，应用发布周期越多，相应的测试周期也多，自动化测试带来的好处就越大。

只要自动化了功能测试，它们就可以作为回归测试的一部分在后续的发布中使用。对于已经存在的代码基线，在回归测试中使用自动化测试会带来很高的投资回报率（ROI）和风险缓解。

如果一个测试脚本一年运行一次，并且被测系统一年修改一次，为此开发一个自动化测试即不可行也无效率。与其花时间每年调整一次脚本来适应被测试系统的修改，还不如手工进行测试。

自动化的复杂度

如果是测试一个复杂的系统，自动化测试可能会带来巨大的好处，因为手工测试人员可以从执行不断重复的、乏味的、耗时且容易出错的复杂测试步骤中解脱出来。

然而，有些测试脚本很难自动化，或性价比不高。造成这种情况的原因包括：现有的测试自动化解决方案不适合被测系统；为了实现测试自动化，开发大量程序代码并开发 API 以实现自动化需求；在执行测试的过程中，需要处理系统的多样性；与外部系统或专有系统的交互；某些用户体验方面测试；验证自动化脚本所需的时间等。

兼容性和工具支持

用于开发应用程序的开发平台的类型很多。对于测试人员的挑战是需要知道支持特定开发平台有哪些可用的测试工具，并且知道平台能支持到什么程度。组织内可能会使用各种测试工具，包括商业工具，开源工具和自主开发的工具。每个组织对支持测试工具的需求和资源都不同。商业工具的厂商一般提供付费支持，并且当作为市场的领导者时，通常具备包括由专家协助实现测试工具服务的生态系统。开源的工具可能提供的支持例如在线论坛，用户可以通过论坛获得信息、发帖提问。自主开发的工具一般依赖内部现有的员工提供支持。

不能低估工具的兼容性问题。如果不完全了解测试工具和被测系统的兼容性问题就开始测试自动化项目，可能会导致灾难性后果。即使被测系统的大多数测试可以实现自动化，仍然有可能在某些情况下，最关键的测试反而无法实现自动化。

测试过程的成熟度

为了在测试流程中高效的实现自动化，测试过程需要是结构化的、规范的和可重复的。自动化在现有的测试过程中引入了一整套的开发过程，因此要求管理测试代码和相关的组件。

软件产品生命周期阶段是否适合自动化

一个被测系统的产品生命周期可以是几年，甚至是几十年。随着系统的逐步开发，系统的不断变化和扩展，解决缺陷并添加改进内容，以满足最终用户的需求。在系统开发的早期阶段，变更可能太快，无法实现自动化测试解决方案。随着界面布局和控件的优化和增强，在动态变化的环境中创建自动化可能需要连续的返工，而这既不有效也不高效。这类似于试图在行驶中的汽车上更换轮胎；最好等车停下来再换轮胎。对于一个顺序开发环境中的大型系统，当系统稳定并包含一个核心功能时，这是开始实现自动化测试的最佳时机。

随着时间的推移，系统达到其产品生命周期的结束，或者进入退役阶段，或者可能采用新的、更有效的技术重新设计系统。对于一个接近其生命周期结束的系统，不建议使用自动化，因为这样一个短命的计划没有什么价值。然而，对于正在使用不同架构重新设计但保留现有功能的系统，自动化测试环境中定义的数据元素在新旧系统中同样有用。在这种情况下，测试数据是有可能被重用的，并且记录的自动化环境将要与新的架构兼容。

自动化测试环境的持续性

自动化测试环境要具备灵活性和适应性，可以根据被测系统的变化而持续调整。这包含了快速诊断和纠正自动化存在的问题的能力，简便的进行自动化组件维护的能力，灵活添加新功能，新的支持的设施。这些属性是通用测试自动化架构（gTAA）整体设计和实现的主要组成部分。

对被测系统的控制力（前置条件，搭建和稳定性）

测试自动化工程师（TAE）应该能够识别被测系统的控制和可见特性，这些特性将帮助创建有效的自动化测试。否则，测试自动化只能依赖图形界面接口上的交互，结果将导致测试自动化的可维护性降低。请查阅 2.3 为易测试性和自动化进行设计，获得更多信息。

支持 ROI 分析的技术计划

测试自动化可以为团队提供不同程度的好处。然而，实现一个有效的自动化测试解决方案需要大量的人力和费用。在花费大量人力和时间开发自动化测试之前，需要先对目的，潜在的整体收益和实现自动化测试可能的输出进行评估。一旦决定进行自动化，应确定实施这个计划所需的活动，并确定相关费用，以便计算投资回报率。

在向自动化环境转型时，应该做好充分准备，我们需要考虑以下方面：

- 在测试环境中为测试自动化准备测试工具；
- 测试数据和测试用例的正确性；
- 测试自动化工作的范围；
- 培训测试团队进行范式的转变；
- 角色和责任；
- 开发人员和测试自动化工程师的合作；
- 并行工作；
- 测试自动化报告。

在测试环境中为测试自动化准备测试工具

在测试的实验环境中，安装选定的测试工具并确认其功能。这包括下载服务包和更新，选择支持被测系统的合适的安装配置，插件，确保在测试实验环境和自动化开发环境中测试自动化解决方案（TAS）的都能正常工作。

测试数据和测试用例的正确性

必须保证手工测试数据和测试用例的正确性和完整性，以便确保自动化的结果是可预测的。在自动化运行测试时对于输入、导航、同步和验证等都需要详细的数据。

测试自动化的工作范围

为了在自动化方面尽早取得成功，并在可能影响进展的技术问题上获得反馈，以有限的范围开始（测试自动化）将有助于将来的自动化任务。可以针对系统功能的某个能代表整个系统的可操作性的领域完成一个试点项目。试点项目的经验教训将有助于调整未来的时间估算和进度表，并确定需要专门技术资源的领域。试点项目能快速地显示早期自动化的成功，为进一步获得管理支持奠定了基础。

为了有助于实现这个目标，我们应该明智地选择自动化测试用例。我们应该选择测试价值高，实现自动化成本低的用例。回归测试或冒烟测试的测试用例，因为其执行频率高，几乎每天都执行，一旦自动化就可以带来非常大的价值。另一个好的选择是从可靠性测试开始自动化。这些测试通常由需要反复执行的步骤组成，能够发现手工测试难以发现的问题。实现这些可靠性测试需要的工作量很少，但却很快就能显示出其附加值。

这些试点项目使自动化更吸引眼球（节省手工测试工作量，发现严重的问题），为进一步开展自动化铺平了道路（工作和金钱投入）。

此外，应该优先考虑对组织至关重要的测试，因为这些测试最先显示出最大的价值。然而，在这种情况下，作为试点工作的一部分，避免自动化技术上最具挑战性的测试是很重要的。否则，将花费太多精力来开发自动化，而显示的价值却很少。一般性的规则，识别可在大部分应用程序间共享的测试可以为自动化工作保持活力提供动力。

培训测试团队进行范式的转变

测试人员有多种风格：一些领域专家来自最终用户，也有些测试人员是业务分析师，还有些人则具有强大的技术背景，他们能够更好地了解底层的系统架构。为了保证测试的有效性，要组合广泛的人员背景。随着测试团队向自动化转型，角色将变得更加专业化。改变测试团队人员的组成是自动化成功的必要条件，尽早对团队进行有意的变革将有助于减少人员对角色的焦虑，或减少对于被认为是多余的担心。如果处理得当，自动化的转变应该使测试团队的每个人都非常兴奋，并积极准备参与到组织和技术的变革。

角色和责任

测试自动化应该是一个人人都可以参与的活动。但是，这并不等于每个人都有相同的角色。设计、实现和维护自动化测试环境是技术性要求很高的工作，因此应该留给具有较强编程技能和技术背景的个人来做。自动化测试开发出来的测试环境，应该对技术人员和非技术人员都可用。为最大限度地提高自动化测试环境的价值，需要有专门领域知识和测试技能的个人，因为需要开发适当的测试脚本（包括相应的测试数据）。这些脚本可以驱动自动化环境，提供目标测试覆盖。领域专家需要评审报告以确认应用程序功能正确性，而技术专家确保自动化环境正常而高效地运行。这些技术专家也可以是对测试感兴趣的开发人员。软件开发经验对于设计可维护的软件至关重要，在测试自动化中也非常重要。开发人员可以专注于测试自动化框架或测试库。测试用例的实现应该由测试人员负责。

开发人员和测试自动化工程师的合作

成功的测试自动化还需要软件开发团队和测试人员的共同参与。开发人员和测试人员需要更紧密地合作以实现测试自动化，这样开发人员能够在他们的开发方法和工具上提供个人的和技术的信息。测试自动化工程师可能会对系统设计和开发人员代码的易测试性产生担忧。特别是当开发人员不遵循标准，使用奇怪的、自己编写的、非常新的库/对象时，情况尤为如此。例如，开发人员可能会选择第三方图形用户界面（GUI）控件，但该控件可能与选定的自动化工具不兼容。总之，一个组织的项目管理团队必须对成功的自动化工作所需的角色和职责有明确的理解。

并行工作

作为转型活动的一部分，很多组织创建了一个并行团队来开始自动化现有的手工测试用例。新的自动化脚本被合并到测试工作中，取代了手动脚本。然而，在这样做之前，通常建议比较和验证自动化脚本执行了与它将要替换的手动脚本相同的测试过程和结果验证。

在许多情况下，需要先对手动脚本进行评估，然后再实现自动化。通过这种评估，确定是否需要重构现有的手工测试脚本结构，以便更高效和更有效的实现自动化。

测试自动化报告

测试自动化解决方案（TAS）可以自动生成各种报告。这些包括单个脚本或脚本中的步骤的成功或失败的状态、总体测试执行统计情况和测试自动化解决方案（TAS）的总体性能。同样重要的是，测试自动化解决方案（TAS）的正确操作需要是可见的，以便报告的任何应用的特定结果都可以被认为是准确和完整的（参见第 7 章：验证测试自动化解决方案（TAS））。

6.2 识别回归测试中实现自动化需要的步骤

回归测试为使用自动化提供了极好的机会。回归测试规模随着今天的功能测试成为明天的回归测试，而日渐增长。总有一天，回归测试的工作量将大于传统的手工测试团队可用的时间和资源。

在准备自动化回归测试的开发步骤时。必须要问自己以下这几个问题：

- 测试将以怎样的频率来执行？
- 回归测试套件中，每个测试的执行时间是多少？
- 测试之间是否有功能重复？
- 测试是否共享数据？
- 测试是否相互依赖？
- 测试执行前需要哪些前置条件？
- 被测系统（SUT）测试覆盖的百分比是多少？
- 当前测试能否无失败地运行？
- 回归测试执行时间过长会发生什么？

下面，针对以上各点进行详细阐述。

测试执行的频率

回归测试中，经常被执行的测试是自动化的最佳选择。这些测试已经开发实现，测试已知的被测系统（SUT）功能，并且能通过自动化极大的减少其执行时间。

测试执行的时间

回归测试中，任何给定的测试或整个测试套件执行所花费的时间是评估自动化测试价值的一个重要参数。一个选项是可以从耗时较多的测试开始实施自动化。这将使每个测试的执行得更快、更有效，同时还增加了自动化回归测试执行的次数。这些是额外的收益，并且可以更频繁地对被测系统（SUT）的质量进行反馈，降低部署风险。

功能的重叠

在对现有的回归测试进行自动化时，最佳实践是识别测试用例之间存在的任何功能重叠，并在可能的情况下减少等价的自动化测试中的重叠。这将进一步提高自动化测试执行时间的效率。随着自动化测试用例变得越来越多，测试的执行效率将变得更加重要。通常，通过自动化开发的测试将采用新的结构，因为它们依赖于可重用组件和共享数据。将现有的手工测试分解成几个较小的自动化测试并不少见。同样，将几个手动测试合并到更大的自动化测试中也可能是更适当的解决方案。手工测试需要先单独评估，以分组进行评估，这样就可以制定有效的转换策略。

数据共享

测试之间经常共享数据。当测试被测系统（SUT）的不同功能时，可能会用到相同的数据。例如，测试用例“A”，它验证了雇员可用的休假时间，而测试用例“B”可能验证了雇员为职业发展而需要学习的课程。每个测试用例使用相同的雇员，但验证的参数不同。在手工测试环境中，通常会在每个手动测试用例中多次复制该雇员数据，而这些测试用例使用该数据验证了雇员的不同信息。然而，在自动化测试实践中，在可能和可行的前提下，尽可能从单个数据源存储和访问共享的数据，以避免重复或引入错误。

测试的相互依赖

在执行复杂的回归测试场景时，一个测试用例可能依赖于一个或多个其他测试用例。这种情况可能很常见，并且经常会发生，例如，某个测试步骤生成一个新的“订单 ID”。随后的测试可能需要验证：a）新的订单在系统中正确显示；B）可以修改订单；或者 C）删除订单成功。在每种情况下，第一次测试中动态生成的“订单 ID”值必须可以被后续的测试捕获并重用。根据测试自动化解决方案（TAS）的设计可以解决这个问题。

测试的前置条件

通常，一个测试用例在不满足前置条件之前是不能成功执行的。这些前置条件可能包括选择正确的数据库或正确的测试数据集，或设置初始值或参数。很多实现测试前置条件所需的步骤，是可以通过自动化完成的。对于这些在执行测试前不能被遗漏的初始化步骤，将初始化步骤自动化给出了更稳定可靠的解决方案。当回归测试被转换为自动化时，这些前置条件应该成为自动化过程的一部分。

被测系统（SUT）的覆盖

每次执行测试时，会运行被测系统（SUT）的部分功能。为了确定整体系统的质量，测试设计要尽可能达到广度和深度的覆盖要求。此外，代码覆盖工具可用于监视自动化测试的执行，以帮助量化测试的有效性。通过自动化回归测试，随着时间的推移，我们可以期望额外的测试提供额外的覆盖。测量这些指标为量化测试本身的价值提供了一种有效的手段。

可执行的测试

在将手工回归测试转换为自动化测试之前，务必要验证手工测试的操作是否正确。这将为确保手工测试成功转型为自动化回归测试提供正确的开端。如果因为手动测试编码很差、或使用无效的数据、过时的、与当前被测系统不一致、或因为被测系统（SUT）存在缺陷，而造成手工测试执行不正确。在找到和解决失效的根本原因之前，将这样的手工测试转换为自动化测试将创建出既浪费时间，也没有意义的不能工作的自动化测试。

大型回归测试集

被测系统的回归测试集可能变得非常庞大，以至于测试集在夜间，或周末都不能执行完毕。在这种情况下，如果有多个被测系统（SUT），并发执行测试用例是一个可能的解决方案（对于 PC 应用，这并不是问题；但当被测系统是飞机或太空火箭，情况完全不同）。如果被测系统是很稀有或昂贵的，并发执行是不切实际的选择。在这种情况下，只可能运行部分回归测试。随着时间的推移（几个星期），整个测试套件最终将执行完毕。可以基于风险分析（被测试系统的哪些部分已经发生了改变？），选择执行回归测试套件其中的一部分。

6.3 在新特性测试中实现自动化时要考虑的因素

一般来说，为新功能的测试用例实现自动化更容易，因为新功能的实现仍然未结束（或者更好的情况：尚未启动），测试工程师可以利用自己的知识向开发人员和架构师解释新功能中需要什么，以便能够通过测试自动化解决方案有效且高效地进行测试。
当被测系统增加新特性时，测试人员需要为这些新的特性和相应的需求开发新的测试。测试自动化工程师（TAE）需要征求具有领域知识的测试设计师的反馈，确定当前的测试自动化解决方案是否能够满足新特性的需要。这些分析包括但不限于：现有的方法、第三方开发工具、测试工具等。

对测试自动化解决方案（TAS）的修改，必须针对现有自动化测试件部分进行评估，以便完整记录修改或补充的内容，并且不影响现有测试自动化解决方案（TAS）的行为（或性能）。

如果为实现一项新特性，例如，增加了一个不同的对象类，这可能需要更新或增加测试件部分。此外，必须评估与现有测试工具的兼容性，并在必要时确定可替代解决方案。例如，如果使用关键字驱动的方法，可能需要开发增加的关键字，或修改/扩展现有关键字以覆盖新功能。

支持新环境中的新功能，可能需要评估额外的测试工具。例如，如果现有测试工具只支持 HTML 网页，就可能需要一个新的测试工具。

新的测试需求可能会影响现有的自动化测试和测试件部分。因此，在做任何修改之前，应该在新的或者升级后的被测系统（SUT）上运行现有的自动化测试，验证现有的自动化测试的操作是否适合，并记录变更。这应该包括将相互依赖性映射到其他测试。任何技术的新变化将迫使我们评估当前的测试件部分（包括测试工具、函数库、API 等）以及与当前的测试自动化解决方案（TAS）的兼容性。

当现有需求发生变化时，更新验证这些需求的测试用例的工作应该是项目时间表（工作分解结构）的一部分。需求与测试用例之间的追溯信息将指明需要更新哪些测试用例。这些更新工作应该是整体测试计划的一部分。

最后，需要确定现有的测试自动化解决方案（TAS）是否将继续满足当前被测系统（SUT）的需求。自动化实现技术是否仍然有效？是否需要一个新结构？是否可以通过扩展当前能力做到这一点？

刚引入新功能的时候，是测试工程师确保新功能可测性的一个好机会。在设计阶段，应该考虑通过提供测试接口来实现可测性，可以通过脚本语言或测试自动化工具来利用这些接口验证新功能。详见第 2.3 节的易测试性和自动化设计中获取更多信息。

6.4 实现自动化确认测试需要考虑的因素

根据报告的缺陷进行代码修复之后，需要执行确认测试。测试人员通常执行重现缺陷的必要步骤来验证缺陷不再存在。

缺陷有时会在后面的版本中再次出现（这可能预示着在配置管理方面有问题），因此确认测试是自动化的首要选择。使用自动化将有助于减少确认测试的执行时间。可以在现有的自动回归测试集中，增加或补充确认测试用例。

自动化的确认测试通常限定很小的功能范围。一旦报告发现缺陷，重现缺陷的步骤也明确，就可以在这样的时刻实现确认测试自动化。自动的确认测试可以合并进一个标准的自动化回归测试套件中。或者，在可行的情况下，在现有的自动化测试中加入确认测试的步骤。使用任何一种方法，都可以体现自动缺陷确认测试的价值。

跟踪自动确认测试可以获得修复缺陷所花费的时间和周期数的额外报告。

除了确认测试外，还需要进行回归测试，以确保修复缺陷没有引入新的缺陷。此时需要通过变更影响分析来确定回归测试的适当范围。

7. 验证测试自动化解决方案（TAS） -120 分钟

关键字

验证（verification）

验证测试自动化解决方案（TAS）的学习目标

7.1 验证自动化测试环境组件

ALTA-E-7.1.1 (K3) 验证包括测试工具设置的自动化测试环境的正确性

7.2 验证自动化测试套

ALTA-E-7.2.1 (K3) 验证给定的自动化测试脚本和/或测试套件的行为正确性

7.1 验证自动化测试环境组件

测试自动化团队需要验证自动化测试环境按预期工作。这些检查一般在自动化测试启动前完成。

可以采取若干步骤来完成自动化测试环境组件的验证工作。这些步骤的具体描述如下：

测试工具安装、设置、配置和定制

测试自动化解决方案（TAS）由许多组件构成。这些组件都必须确保可靠和可重复执行。测试自动化解决方案（TAS）的核心包括可执行的组件、对应的功能库以及支撑数据和配置文件。配置测试自动化解决方案（TAS）的过程可能包括从使用自动化的安装脚本到手动将文件放置到对应的目录。测试工具，如同操作系统和其他应用软件一样，经常会有补丁包或者可选/必选的插件来确保对于给定被测系统（SUT）环境的兼容性。

如果从中心存储库中自动化安装或者复制测试工具，则有明显的优势，这种方式可以确保不同被测系统（SUT）上的测试都能正确地使用合适和相同版本的测试自动化解决方案（TAS）以及相同的测试自动化解决方案（TAS）配置。测试自动化解决方案（TAS）的升级也可以通过该中心存储库进行。中心存储库的使用以及测试自动化解决方案（TAS）的版本升级流程应该与标准开发工具相同。

已知通过或者失败的测试脚本

如果应该通过的测试用例执行失败，预示着有根本性的错误发生并且需要立即纠正。反之，如果应该失败的测试用例通过了，我们需要识别未正常工作的组件。验证生成日志文件的正确性、性能度量的正确性、测试用例/脚本的自动设置和清除等，这些工作都是非常重要的。执行若干个不同类型和级别的测试用例也是有帮助的，如功能测试、性能测试和组件测试等。这项活动应该在框架层面执行。

测试环境设置 / 清理的可重复性

测试自动化解决方案（TAS）可能会被部署在不同的系统和服务器上。为了确保测试自动化解决方案（TAS）在每种环境下能正确地工作，需要一种系统性的在给定环境中装载和卸载测试自动化解决方案（TAS）。当测试自动化解决方案（TAS）的构建和重复构建在相同或多个环境中无明显差异的运作时，则可以认为测试自动化解决方案（TAS）具备了不同环境下的可重复性。测试自动化解决方案（TAS）组件的配置管理确保了给定的配置能够被可靠的创建。

测试环境和组件的配置

理解并文档化组成测试自动化解决方案（TAS）的各个组件将提供必要的知识。当在被测系统（SUT）的环境改变时，这些必要的知识能帮助分析测试自动化解决方案（TAS）的哪些方面会受到影响或需要变更。

对内部和外部系统/接口的连接

一旦在给定被测系统（SUT）环境下安装了测试自动化解决方案（TAS），在实际使用被测系统（SUT）之前应该执行一系列检查或者前置条件来确认内部和外部系统/接口的连通。自动化建立前置条件是确保测试自动化解决方案（TAS）的正确安装和配置的基础。

自动化测试工具的侵入

测试自动化解决方案（TAS）通常是与被测系统（SUT）紧密耦合的。这种设计是有意为之，以确保两者之间的高度兼容性，尤其是有关图形用户界面（GUI）级别的交互时。然而，这种紧密集成也可能存在负面作用，包括：当测试自动化解决方案（TAS）驻留在被测系统（SUT）环境中时引起被测系统（SUT）的行为不同；测试自动化解决方案（TAS）操作时，被测系统（SUT）会有与手工操作时不同的行为；当测试自动化解决方案（TAS）在被测系统（SUT）环境中，或者通过测试自动化解决方案（TAS）驱动被测系统（SUT）时，被测系统（SUT）的性能受到影响。

入侵的程度随着选择的自动化测试方式而不同。举例来说，

- 通过外部接口与被测系统（SUT）进行交互时，入侵程度会非常低。外部接口可以是电子信号（如物理交换机），USB 设备的 USB 信号（如键盘），通过这种方式可以很好地模拟最终用户，这种方式下的被测系统（SUT）软件不需要为测试目的而进行修改。被测系统（SUT）的行为和时间特性没有因为测试方法受到影响。但是通过这种方式来和被测系统（SUT）进行交互会非常复杂。可能需要专用硬件以及硬件描述语言等来与被测系统（SUT）交互。对于纯软件系统来说这不是一种典型的做法，但对于带有嵌入式软件的产品来说这种做法很常见。
- 当在图形用户界面（GUI）层与被测系统（SUT）进行交互时，被测系统（SUT）环境要适宜于注入用户界面（UI）命令和提取测试用例所需的信息。被测系统（SUT）的行为不是直接改变的，，而是受时间影响，这会对被测系统（SUT）行为产生影响。相比于上文中的方式，这种方式的入侵程度要高一些，但是与被测系统（SUT）交互相对不那么复杂。通常商业化的现成工具可以用于这类的自动化。
- 与被测系统（SUT）的交互也可以通过软件中的测试接口或者通过软件提供的现有接口来完成。如何确保这些接口（APIs）的可用性是易测试性设计的重要部分。这种方式的入侵程度会非常高。自动化测试可能使用了最终用户根本不会使用的接口（测试接口）或者使用与实际环境完全不同的接口。另一方面，通过接口（API）来实施自动化测试是非常简便和低成本的。只要理解了这种方式的潜在风险，通过测试接口来进行测试不失为一种行之有效的方式。

高入侵程度时，如果测试显示失败，实际使用情况下却未必如此。这种情况的发生，对测试自动化方案的信心就会急剧下降。开发人员可能要求通过自动化测试发现的失败应尽可能再通过手工测试进行复现，以协助分析。

框架组件测试

与软件开发项目一样，自动化框架组件也需要进行单独的测试和验证。这项工作包括了功能和非功能测试（性能、资源利用率和易用性等）。

例如，图形用户界面（GUI）系统中提供对象验证的组件，就需要对广泛的对象类进行测试，以确保对象验证功能正确执行。同样的，错误日志和报告应该能够产生有关自动化执行状态和被测系统（SUT）行为的准确信息。

非功能测试的例子有：了解框架性能的下降和系统资源使用率等，这些信息可以协助发现诸如内存泄漏等问题。还有框架内和/或框架外的各组件的互操作性。

7.2 验证自动化测试套件

自动化测试套件应该针对完整性、一致性和行为正确性进行测试。通过实施各种类型的验证检查，可以确保自动化测试套件在任意给定时刻能正常运行或者适合使用。

可以采取以下一系列步骤来验证自动化测试套件，包括:

- 执行已知通过/失败的测试脚本；
- 检查测试套件；
- 验证针对框架新特性的新测试用例；
- 考虑测试的可重复性；
- 检查自动化测试套件中是否有足够的验证点。

上述步骤详细介绍如下：

执行已知通过或者失败的测试脚本

当应该通过的测试用例失败时，预示着发生了根本性的错误并且需要尽快纠正。相反地，如果应该失败的用例套件却通过了，必须标识这样的测试用例为不能正常工作。验证日志文件生成的正确性、性能数据以及自动化测试用例/脚本的设置和清理是非常重要的。执行若干个不同类型和级别的测试用例也是有帮助的（例如，功能测试、性能测试和组件测试等）。

检查测试套件

检查测试套件的完整性（所有的测试用例都包含了预期结果、具备测试数据）以及其版本对应框架和被测系统（SUT）是正确的。

验证针对框架新特性的新测试用例

当测试自动化解决方案（TAS）的某个新特性在测试用例中第一次实际使用时，必须密切关注并验证该特性可以正常工作。

考虑测试的可重复性

当重复执行测试时，测试结果和判断应该始终是相同的。如果在测试集合中，有些测试用例运行结果不可靠，例如存在竞争条件，那么这些测试用例需要从现役的自动化测试套件中移出，单独分析以发现根本原因。否则的话，重复去执行这些测试并分析问题会浪费时间。

间歇性的失效应被分析。问题可能来自测试用例自身或者来自框架（甚至有可能来自被测系统（SUT）自身）。可以通过分析测试用例、框架和被测系统（SUT）的日志文件来确定问题产生的根本原因，可能还需要进行调试。测试分析师，软件开发人员和领域专家的支持也可以帮助定位根本原因。

检查自动化测试套件和/或测试用例中有足够的验证点

应该能够验证自动化测试套件已经执行并且取得了预期结果，也必须有证据来证明测试套件和/或测试用例按照预期执行。这样的证据可以包括：每个测试用例开始和结束时的日志、每个已完成测试用例的测试执行状态的记录、或者是验证后置条件已被满足等。

8. 持续改进-150 分钟

关键字

维护（maintenance）

持续改进的学习目标

8.1 改进测试自动化的选项

ALTA-E-8.1.1 (K4) 分析已经部署的测试自动化解决方案的技术方面的问题，并提供推荐的改进内容。

8.2 使自动化测试适应环境和被测系统（SUT）的变化

ALTA-E-8.2.1 (K4) 分析自动化测试件，包括测试环境组件、工具和支持函数库，以便为了适应特定测试环境或被测系统（SUT）的变化，梳理清楚应该在哪些方面进行强化和提升。

8.1 改进测试自动化的选项

除了保证测试自动化解决方案（TAS）和被测系统（SUT）同步所做的日常维护工作之外，还有很多典型的改进测试自动化解决方案（TAS）的机会。改进测试自动化解决方案（TAS）可以获得一系列的好处，包括更高的效率（进一步减少手动干预）、更好的易用性、增强的功能，以及提高对测试活动的支持。如何改进测试自动化解决方案（TAS）要看这些改进措施能够给项目带来多大的收益。

可以考虑对测试自动化解决方案（TAS）进行改进的领域包括：脚本、验证、架构、预处理和后期处理、文档和工具支持。下面将更详细地介绍这些内容。

脚本

如 3.2.2 小节所述，脚本编写方法多样，从简单的结构化方法、数据驱动方法，直到更复杂的关键字驱动。对于新的自动化测试，升级现有测试自动化解决方案（TAS）脚本编写方法可能是适合的。新的脚本编写方法可以应用到现有自动化测试中，或至少可以应用到需要最大维护工作量的测试中。

除了彻底的改变脚本编写方法之外，测试自动化解决方案（TAS）改进还可以聚焦在脚本的实现上。例如：

- 对测试用例/步骤/过程的重复部分进行评估，以巩固自动化测试。
  包含相似操作步骤的测试用例，不应该多次反复实现这些步骤。这些步骤应该定义成一个函数并添加到库中，以便复用。这些库函数还可以被不同的测试用例所使用。这样提高了测试件的可维护性。如果测试步骤不完全相同但很相似，可以使用参数化的技术。
  注：这是关键字驱动测试中的一种典型做法。
- 建立测试自动化解决方案（TAS）和被测系统（SUT）的错误恢复过程。
  如果在执行测试用例的过程中发生错误，测试自动化解决方案（TAS）应当能够从错误状态中恢复，继续执行下一个测试用例。如果是被测系统（SUT）发生错误，测试自动化解决方案（TAS）需要能对被测系统（SUT）实行必要的恢复操作（例如，重新启动整个被测系统（SUT））。
- 评估等待机制以确保使用最佳类型。
  有三种的常见的等待机制：
  1. 硬编码等待（等待一定的毫秒数）：这可能是导致许多测试自动化出现问题的根本原因。
  2. 动态轮询等待：这个方法更加灵活、有效。例如：检查某一状态是否变化或某操作是否已经完成。
     - 只等待所需时间，没有浪费测试时间；
     - 如果出于某种原因，这个过程需要更长的等待时间，轮询将一直等到条件成立为止。记住要包含超时机制，否则当出现问题时，测试用例可能会永远等待。
  3. 一个更好的方法是订阅被测系统（SUT）事件机制。这比其他两个方式都要更加可靠。但是测试脚本语言需要支持事件订阅，并且被测系统（SUT）需要将这些事件提供给测试应用程序。记住要包含超时机制，否则出现问题时，测试用例可能会永远等待。
- 将测试件视为软件。

测试件的开发和维护是软件开发的形式之一。因此，良好的编码实践（例如，使用编码规范、静态分析、代码评审）应该应用于测试件的开发和维护中。甚至让软件开发人员（而不是测试工程师）开发测试件的某些部分（例如，库）可能是一个好主意；

- 评估现有的脚本并做修订或剔除。
  有些脚本会带来麻烦（例如，偶发失败，或者维护成本大），明智的做法是重新设计这些脚本，其他不再产生价值的脚本可以从测试件中删除。

测试执行

如果一个自动化回归测试套件花费整晚却还没有跑完，也不应感到意外。当测试执行耗时过长，意味着可能需要同时在多个系统上并发执行测试，但这并不总是可行。如果被测系统很昂贵，就会有限制条件：所有的测试都必须在单个系统上执行。这时需要将回归测试套件分割成多个部分，每个部分在一个限定的时间段内执行（例如，一个晚上）。后续分析自动化测试覆盖率的时候，可能会发现一些重复内容。删除重复部分可以减少执行时间并提升效率。

验证

在创建新的验证功能之前，应采用一套标准验证方法，以供所有自动化测试使用。这将避免在多个测试中反复重新实现验证操作。当验证方法不完全相同但相似时，使用参数化的方法将有助于功能被多种类型的对象使用。

架构

可能需要对架构进行变更来支持被测系统（SUT）的易测试性上的改进，可能需要同时改变被测系统（SUT）架构和自动化架构，也可能只改变其中一个。这样的改进能为测试自动化提供主要的改善，但可能需要重大的变更和对被测系统（SUT）/测试自动化解决方案（TAS）进行大量的改造和投入。例如，如果需要改变被测系统（SUT）以提供专供测试的 API，那么测试自动化解决方案（TAS）也应该相应地进行重构。在后期添加这些特性可能相当昂贵，所以最好是在自动化早期阶段进行规划（并且是被测系统（SUT）的开发早期阶段–参见 2.3 章节为易测试性与自动化进行设计）。

预处理和后期处理

提供标准的设置和清理任务，这些也称为预处理（设置）和后期处理（清理）。这就为每个自动化测试节省了不少重复的工作量，不仅降低了维护成本，而且减少了实现新的自动测试所需的工作量。

文档

这包括所有形式的文档：脚本文档（脚本的用途、使用方式等）、测试自动化解决方案（TAS）的用户文档以及测试自动化解决方案（TAS）生成的报告和日志。

测试自动化解决方案（TAS）特性

添加额外的测试自动化解决方案（TAS）特性和功能，如详细报告、日志、与其它系统集成等。切记只增加确有必要的新特性。添加没必要的特性只会增加系统的复杂性并降低其可靠性和可维护性。

测试自动化解决方案（TAS）的更新和升级

通过更新或升级到测试自动化解决方案（TAS）新版本，测试用例可以使用新的功能（或故障可能被修正）。更新框架（通过升级现有测试工具或引入新工具）的风险在于，可能会对现有的测试用例产生负

面影响。在推出新版本之前，先运行试点测试用例来测试新版本的测试工具。试点测试用例应涵盖不同环境下的不同应用程序、不同测试类型的自动化测试。

8.2 实现测试自动化改进措施的计划

更改现有的测试自动化解决方案（TAS）需要进行仔细的规划和调研。创建由测试自动化框架（TAF）和组件库构成健壮的测试自动化解决方案（TAS），已经耗费了大量精力。任何变更，无论它有多么微不足道，都可能对测试自动化解决方案（TAS）的可靠性和性能产生极大的影响。

识别测试环境组件的更改

评估需要做出哪些改变和改进。这些变更是否需要更改测试软件、定制的功能库以及操作系统？每一项变化都会影响测试自动化解决方案（TAS）的性能。总体目标是确保自动化测试的有效持续运行。因此应逐步实施变更，以便通过运行适量的测试脚本就可以衡量变更对测试自动化解决方案（TAS）的影响。一旦发现不存在负面影响，该变更就可以全面实施。最后一步，是运行全回归测试，以验证变更不会对自动化脚本产生不利影响。在执行这些回归脚本时，可能会发现错误。确定这些错误的根本原因（通过报告、日志、数据分析等）将提供一种手段来确保它们不是由自动化改进活动产生的。

提高测试自动化解决方案（TAS）核心函数库的效率和效能

随着测试自动化解决方案（TAS）的成熟，会有新的方法来更有效地执行任务。这些新技术（包括优化函数代码、使用较新的操作系统库等）需要与当前项目和所有项目使用的核心函数库有机地结合。

对作用于相同控件类型的多个函数可进行合并

在自动化测试运行期间发生的很大一部分是图形用户界面（GUI）中控件的交互查询。该交互查询用于提供控件的信息（例如，可见/不可见，启用/未启用，大小和维度，数据等）。通过这些信息，自动化测试可以从下拉列表中选择一个项目，或者将数据输入到一个字段中，又或者是从一个字段中读取一个值。有很多函数可以作用于控件从而获取这些信息。其中一些函数是非常定制化的，而其他函数更具通用性。例如，某个函数只能操作下拉菜单。又或者，某个函数（在测试自动化解决方案（TAS）中创建和使用的函数）可以操作多个功能，只是将某个功能作为其参数。因此，测试自动化工程师（TAE）需要将一些函数合并成为较少的函数，这样就可以在实现相同结果的同时将维护需求最小化。

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，需要作出适当改变以适应被测系统（SUT）的变化。随着被测系统（SUT）的发展和成熟，底层的测试自动化架构（TAA）也将不断发展，以确保有支持被测系统（SUT）的能力。在扩展功能时，必须注意不要以补丁方式实现，而是在自动化解决方案的体系结构层面进行分析和更改。这将确保当新的被测系统（SUT）功能需要额外脚本时，已经有兼容的组件来支持这些新自动化的测试用例。

命名规则和标准化

一旦发生变更，新的自动化代码和函数库的命名规则必须与预先定义的标准相一致（参见第 4.3.2 节范围和方法）。

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脚本的评估，包括脚本的使用情况及其持续使用的价值。例如，如果某些测试是复杂且耗时的，将其分解成几个较小的测试可能更加可行且有效率。删除不经常执行或根本不执行的脚本，将降低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明确。

9. 参考文献

9.1 标准

测试自动化标准包括但不限于：

- TTCN-3（The Testing and Test Control Notation）是由 ETSI（欧洲电信标准协会）和 ITU（国际电信联盟）维护的全球适用的标准测试语言,他的组成如下：
  - ES 201 873-1: TTCN-3 核心语言
  - ES 201 873-2: TTCN-3 表格显示格式 (TFT)
  - ES 201 873-3: TTCN-3 图形显示格式 (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
- IEEE（电气与电子工程师学会）的自动测试标记语言（ATML）由以下组成：
  - IEEE Std 1671.1: 测试描述
  - IEEE Std 1671.2: 仪器描述
  - IEEE Std 1671.3: UUT 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: 信号和测试的定义
  - IEEE Std 1636.1: 测试结果
- ISO/IEC/IEEE 29119-3
- OMG（对象管理组）的 UML 测试配置文件（UTP）为以下各项指定测试规范概念：
  - 测试架构
  - 测试数据
  - 测试行为
  - 测试日志
  - 测试管理

9.2 ISTQB 文档

标识	参考文档
ISTQB-AL-TM	ISTQB 测试工程师认证，高级大纲，测试经理，2012 版，可以从 [ISTQB-Web]查阅
ISTQB-AL-TTA	ISTQB 测试工程师认证，高级大纲，技术测试分析师，2012 版，可以从 [ISTQB-Web]查阅
ISTQB-EL-CEP	ISTQB 专家级认证扩展，可以从 [ISTQB-Web]查阅
ISTQB-EL-Modules	ISTQB 专家级模块概述，1.2 版，2013 年 8 月 23 日，可以从 [ISTQB-Web]查阅
ISTQB-EL-TM	ISTQB 专家级–测试管理大纲，2011 版，可以从 [ISTQB-Web]查阅
ISTQB-FL	ISTQB 基础级大纲，2011 版，可以从 [ISTQB-Web]查阅
ISTQB-Glossary	ISTQB 专业术语，2.4 版，2014 年 7 月 4 日，可以从 [ISTQB-Web]查阅

9.3 商标

本文档使用以下注册商标和服务标识：

ISTQB®是国际软件测试认证委员会的注册商标。

9.4 书籍

标识	参考书籍
[Baker08]	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
[Dustin09]	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
[Dustin99]	Efriede Dustin, Jeff Rashka, John Paul, “Automated Software Testing: introduction, management, and performance”, Addison-Wesley, 1999, ISBN-10: 0201432870, ISBN-13: 9780201432879
[Fewster&Graham12]	Mark Fewster, Dorothy Graham, “Experiences of Test Automation: Case Studies of Software Test Automation”, Addison-Wesley, 2012
[Fewster&Graham99]	Mark Fewster, Dorothy Graham, “Software Test Automation: Effective use of test execution tools”, ACM Press Books, 1999, ISBN-10: 0201331403, ISBN-13: 9780201331400
[McCaffrey06]	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

[Mosley02] Daniel J. Mosley, Bruce A. Posey, “Just Enough Software Test Automation”, Prentice Hall, 2002, ISBN-10: 0130084689, ISBN-13: 9780130084682

[Willcock11] 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

9.5 参考网站

标识	参考
ISTQB-Web	国际软件测试认证委员会的官网。可以参考本网站最新的 ISTQB 术语和大纲：www.istqb.org

10. 培训机构的注意事项

10.1 培训时间

本大纲的各章按分钟分配时间。目的是指导认证课程各个部分的相对时间比例，同时对讲授各个部分给出了最少时间。

培训机构可以用更多的时间进行授课，考生也可以花费更多的时间阅读和研究。课程表可以与大纲的章节顺序不同。并不需要在连续的时间段讲授此课程。

下表给出了各章讲授和练习的时间指南（全部时间以分钟为单位）。

章节	时间（分钟）
0. 简介	0
1. 测试自动化的介绍和目标	30
2. 测试自动化准备	165
3. 通用测试自动化结构	270
4. 部署风险和应急处理	150
5. 测试自动化报告与度量	165
6. 从手工测试向自动化测试环境转型	120
7. 验证测试自动化解决方案（TAS）	120
8. 持续改进	150
总计	1170

课程总计天数为：2 天 5 小时 30 分钟（按照平均 7 小时/工作日计算）。

10.2 在培训场所的实践练习

无需在培训场所实施的练习。

10.3 电子学习规范

本大纲的所有部分都适合采用电子学习的方式来实施。