

テスト技術者資格制度
Advanced Level
テスト自動化エンジニア
シラバス 日本語版

Version v2.0.J01

International Software Testing Qualifications Board



著作権について

著作権表示 : © International Software Testing Qualifications Board (以下「ISTQB®」)

ISTQB®は、International Software Testing Qualifications Board の登録商標である。

Copyright © 2024 Test Automation Engineering v2.0 シラバス著者 : Andrew Pollner (議長) 、 Péter Földházi、 Patrick Quilter、 Gergely Ágnesz、 László Szikszai

Copyright © 2016 シラバス著者 : Andrew Pollner (議長) 、 Bryan Bakker、 Armin Born、 Mark Fewster、 Jani Haukinen、 Raluca Popescu、 Ina Schieferdecker

全権利は著者が保有する。著者はここに ISTQB®に著作権を譲渡する。著者（現在の著作権者として）および ISTQB®（将来の著作権者として）は、以下の利用条件に合意した：

本文書からの抜粋は、非商業目的での利用に限り、出典が明記されていれば複写が許可される。認定トレーニングプロバイダーは、著者および ISTQB®がシラバスの出典および著作権者として認められ、当該トレーニングコースの広告において ISTQB®認定メンバーボードから公式認定を受けた後にのみシラバスに言及できることを条件として、本シラバスをトレーニングコースの基礎として使用することができる。

個人または個人のグループは、著者および ISTQB®がシラバスの出典および著作権者として認められることを条件として、本シラバスを記事や書籍の基礎として使用することができる。

本シラバスのその他の利用については、ISTQB®の事前の書面による承諾が必要である。

いかなる ISTQB®認定メンバーボードも、本シラバスを翻訳する際に、翻訳版のシラバスに上記の著作権表示を再掲することを条件として翻訳することができる。

日本語翻訳版の著作権は JSTQB®が有するものである。本書の全部、または一部を無断で複製し利用することは、著作権法の例外を除き、禁じられている。

改訂履歴

ISTQB®

バージョン	日付	備考
2016	2016/10/21	CTAL-TAE 一般公開版リリース
v2.0	2024/05/03	CTAL-TAE v2.0 一般公開版リリース

JSTQB®

バージョン	日付	備考
Version 2016.J01	2020/09/30	Version 2016 の日本語翻訳版
Version 2.0.J01	2026/05/05	v2.0 の日本語翻訳版

目次

著作権について	2
改訂履歴.....	3
目次	4
謝辞	7
0 はじめに	8
0.1 本シラバスの目的	8
0.2 ソフトウェアテストにおけるテスト自動化エンジニア	8
0.3 テスト担当者およびテスト自動化エンジニア向けのキャリアパス	8
0.4 ビジネス成果	9
0.5 試験対象の学習の目的と知識レベル	9
0.6 テスト自動化エンジニア認定試験	10
0.7 認定	10
0.8 標準の取り扱い	11
0.9 最新情報の維持	11
0.10 詳細レベル	11
0.11 本シラバスの構成	11
1 テスト自動化の概要と目標 – 45 分 (K2)	14
1.1 テスト自動化の目的	15
1.1.1 テスト自動化の利点と欠点の説明	15
1.2 ソフトウェア開発ライフサイクル(SDLC)におけるテスト自動化	16
1.2.1 さまざまなソフトウェア開発ライフサイクルモデル (SDLC) におけるテスト自動 化の適用方法の説明	16
1.2.2 テスト対象システム (SUT) に適したテスト自動化ツールの選択	16
2 テスト自動化の準備 – 180 分 (K4)	18
2.1 テスト自動化を可能にするインフラストラクチャ設定の説明	19
2.1.1 テスト自動化の実装を可能にするインフラストラクチャの構成要件の説明	19
2.1.2 異なる環境におけるテスト自動化の活用方法の説明	19

2.2 適切なツールと戦略を選択するための評価プロセス	20
2.2.1 適切なテスト自動化ソリューション（TAS）を決定するためのテスト対象システム（SUT）の分析	20
2.2.2 ツール評価の技術的結果の提示	21
3 テスト自動化アーキテクチャ – 210 分（K3）	23
3.1 テスト自動化で活用されるデザインコンセプト	24
3.1.1 テスト自動化アーキテクチャの主要機能の説明	24
3.1.2 テスト自動化ソリューションの設計方法の説明	25
3.1.3 テスト自動化フレームワークのレイヤー構造化の適用	25
3.1.4 テストケースを自動化するための各種アプローチの適用	26
3.1.5 テスト自動化における設計原則と設計パターンの適用	29
4 テスト自動化の実装 – 150 分（K4）	30
4.1 テスト自動化開発	31
4.1.1 効果的なテスト自動化パイロットおよびデプロイメント活動を支援するガイドラインの適用	31
4.2 テスト自動化開発に関連するリスク	32
4.2.1 テスト自動化のデプロイメントリスクの分析と軽減策の計画	32
4.3 テスト自動化ソリューションの保守性	33
4.3.1 テスト自動化ソリューションの保守性を支援・影響する要因の説明	33
5 テスト自動化の実装とデプロイメント戦略 – 90 分（K3）	35
5.1 CI/CD パイプラインへの統合	36
5.1.1 パイプライン内の各テストレベルでのテスト自動化の適用	36
5.1.2 テストウェアの構成管理の説明	37
5.1.3 API インフラストラクチャに対するテスト自動化の依存関係の説明	38
6 テスト自動化のレポートिंगとメトリクス – 150 分（K4）	39
6.1 テスト自動化データの収集・分析・レポートिंग	40
6.1.1 テスト自動化ソリューション（TAS）およびテスト対象システム（SUT）からのデータ収集方法の適用	40
6.1.2 テスト結果の理解を深めるための、テスト自動化ソリューション（TAS）およびテスト対象システム（SUT）からのデータ分析	42
6.1.3 テスト進捗レポートの作成と公開方法の説明	43

7 テスト自動化ソリューションの検証 – 135 分 (K3)	45
7.1 テスト自動化インフラストラクチャの検証.....	46
7.1.1 テストツールのセットアップを含むテスト自動化環境の検証計画	46
7.1.2 特定の自動テストスクリプトおよびテストスイートの正しい動作の説明	47
7.1.3 テスト自動化が予期せぬ結果を生成する箇所の特定	48
7.1.4 静的解析がテスト自動化コード品質を向上させる方法の説明	48
8 継続的改善 – 210 分 (K4)	49
8.1 テスト自動化の継続的改善の機会	50
8.1.1 データ収集と分析によるテストケース改善の機会の発見.....	50
8.1.2 デプロイされたテスト自動化ソリューションの技術的側面の分析と改善提案の提 示	50
8.1.3 テスト対象システム (SUT) の更新に合わせた自動テストウェアの再構築	53
8.1.4 テスト自動化ツール活用の方会の整理	54
9 参考文献.....	56
付録 A – 学習の目的/知識の認知レベル	60
付録 B – 学習の目的とビジネス成果のトレーサビリティマトリクス	62
付録 C – リリースノート	66
付録 D – ドメイン固有用語.....	67

謝辞

本書は、ISTQB®により 2024 年 5 月 3 日に公開された。

本書は International Software Testing Qualifications Board の Specialist Working Group のテスト自動化タスクフォースによって作成された：Graham Bath（Specialist Working Group 議長）、Andrew Pollner（Specialist Working Group 副議長兼テスト自動化タスクフォース議長）、Péter Földházi、Patrick Quilter、Gergely Ágnesz、László Szikszai。テスト自動化タスクフォースのレビュアー：Armin Beer、Armin Born、Geza Bujdosó、Renzo Cerquozzi、Jan Giesen、Arnika Hryszko、Kari Kakkonen、Gary Mogyorodi、Chris van Bael、Carsten Weise、Marc-Florian Wendland。

テクニカルレビュアー：Gary Mogyorodi

以下の方々が本シラバスのレビュー・コメント・投票に参加した：

Horváth Ágota、Laura Albert、Remigiusz Bednarczyk、Jürgen Beniermann、Armin Born、Alessandro Collino、Nicola De Rosa、Wim Decoutere、Ding Guofu、Istvan Forgacs、Elizabeta Fourneter、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 上級レベル テスト自動化エンジニア（2016 年版）ワーキンググループ：Andrew Pollner（議長）、Bryan Bakker、Armin Born、Mark Fewster、Jani Haukinen、Raluca Popescu、Ina Schieferdecker。

日本語版は、Japan Software Testing Qualifications Board のメンバーおよび以下の日本語翻訳ワーキンググループのメンバーにより作成された。

日本語翻訳ワーキンググループメンバー：

須原 秀敏（ベリサーブ）

江村 禎昭（バルテス・ホールディングス）

伊藤 由貴（MagicPod）

藤原 考功（ベリサーブ）

0 はじめに

0.1 本シラバスの目的

本シラバスは、上級レベル テスト自動化エンジニア (CTAL-TAE) 資格の基礎を定めるものである。ISTQB®は、以下の目的で本シラバスを提供する。

1. メンバーボードに対して、自国語への翻訳および認定トレーニングプロバイダーの認定を行うため。メンバーボードは、自国語のニーズに合わせてシラバスを適合させ、自国の出版物に合わせて参考文献を変更することができる。
2. 認定機関に対して、本シラバスの学習の目的に合わせた自国語の試験問題を作成するため。
3. トレーニングプロバイダーに対して、コースウェアを作成し、適切な教授法を決定するため。
4. 認定候補者に対して、認定試験の準備のため（トレーニングコースの一部として、または独自に）。
5. ソフトウェアおよびシステムエンジニアリングの国際コミュニティに対して、ソフトウェアおよびシステムテストの専門性を向上させ、書籍や記事の基礎とするため。

0.2 ソフトウェアテストにおけるテスト自動化エンジニア

テスト自動化エンジニア資格は、ソフトウェアテストやテスト自動化に関わる幅広い人々を対象としている。例えば、テスト担当者、テストアナリスト、テスト自動化エンジニア (TAE)、テストコンサルタント、テストアーキテクト、テストマネージャー、ソフトウェア開発者などである。また、プロジェクトマネージャー、品質マネージャー、ソフトウェア開発マネージャー、ビジネスアナリスト、IT ディレクター、マネジメントコンサルタントなど、テスト自動化の基本的な理解を必要とする人にも有用である。

本シラバスは、テスト自動化の導入または改善を担うテストエンジニアを主な対象としている。持続可能なテスト自動化ソリューションを実現するための手法と実践を示す。

テスト自動化ソリューションに関する他のガイドラインや参照モデルとしては、選択したソフトウェア開発ライフサイクルやプログラミング技術に対するソフトウェア工学標準、および書式標準がある。本シラバス自体はソフトウェア工学を教えるものではないが、テスト自動化エンジニア (TAE) には、ソフトウェア工学に関するスキル、経験、専門知識が求められる。

さらに、テスト自動化エンジニアは、テスト自動化ソリューションの開発でそれらを適切に活用するために、業界におけるプログラミングおよび文書化の標準とベストプラクティスを理解している必要がある。これらの実践は、テスト自動化ソリューションの保守性、信頼性、およびセキュリティの向上に寄与する。こうした標準は、一般に品質特性に基づいている。

0.3 テスト担当者およびテスト自動化エンジニア向けのキャリアパス

ISTQB®のスキームは、知識の広さと深さの両面から、あらゆるキャリア段階にあるテスト専門家を支援する。ISTQB®テスト自動化エンジニア認定を取得した人は、テスト自動化戦略 (CT-TAS) 資格にも関心を持つことがある。

ISTQB®テスト自動化エンジニア認定を取得した人は、コアの上級レベル資格（テストアナリスト、テクニカルテストアナリスト、テストマネージャー）や、その後のエキスパートレベル（テストマネジメントまたはテストプロセス改善）にも関心を持つことがある。アジャイル環境でのテスト実践スキルを高めたい場合は、アジャイルテクニカルテスト担当者や、スケールされたアジャイルテストリーダーシップ認定の取得も選択肢となる。スペシャリスト分野では、テスト自動化戦略、パフォーマンステスト、セキュリティテスト、AI テスト、モバイルアプリケーションテストなど、特定のテストアプローチや活動に関する領域、あるいは自動車ソフトウェアテストやゲームテストのように特定のドメイン知識を必要とする領域を、より深く学ぶことができる。最新情報については、ISTQB の公式サイト（www.istqb.org）を参照すること。

0.4 ビジネス成果

この節では、テスト自動化エンジニア認定を取得した候補者に期待されるビジネス成果を列挙する。テスト自動化エンジニア認定を取得した候補者は、次のことができる。

TAE-B01	テスト自動化の目的を説明する
TAE-B02	ソフトウェア開発ライフサイクル全体を通じたテスト自動化を理解する
TAE-B03	テスト自動化を可能にするインフラストラクチャの構成を理解する
TAE-B04	適切なツールと戦略を選択するための評価プロセスを理解する
TAE-B05	モジュール化され、スケーラブルなテスト自動化ソリューションを構築するためのデザインコンセプトを理解する
TAE-B06	ソフトウェア開発ライフサイクル内で、パイロットを含むアプローチを選択し、テスト自動化のデプロイメントを計画する
TAE-B07	技術的なニーズを満たす新規または改修されたテスト自動化ソリューションを設計・開発する
TAE-B08	テスト自動化の範囲とアプローチ、およびテストウェアの保守を考慮する
TAE-B09	自動テストが CI/CD パイプライン内でどのように統合されるかを理解する
TAE-B10	ステークホルダーへの情報提供のために、テスト自動化データを収集・分析・レポートする方法を理解する
TAE-B11	テスト自動化インフラストラクチャを検証する
TAE-B12	テスト自動化の継続的改善の機会を定義する

0.5 試験対象の学習の目的と知識レベル

学習の目的はビジネス成果を支援し、テスト自動化エンジニア認定試験の作成に使用される。

一般に、本シラバスのすべての内容は、はじめにと付録を除き、K2、K3、K4 レベルで試験対象となる。つまり、候補者は 8 つの章のいずれかに記載されているキーワードや概念を認識、記憶、または想起することを求められる場合がある。具体的な学習の目的レベルは各章の冒頭に示されており、以下のように分類されている：

- K2：理解
- K3：適用
- K4：分析

学習の目的のさらなる詳細と例は付録 A に記載されている。

章見出しのすぐ下にキーワードとして列挙されているすべての用語は、学習の目的に明示的に記載されていない場合でも、理解しておく必要がある。

0.6 テスト自動化エンジニア認定試験

テスト自動化エンジニア認定試験は本シラバスに基づいている。試験問題への回答には、本シラバスの複数の節に基づく内容が必要になる場合がある。本シラバスのすべての節が試験対象であるが、はじめにと付録は除く。標準や書籍は参考文献として含まれているが、その内容自体は、シラバスで要約されている範囲を超えて試験対象にはならない。

テスト自動化エンジニア認定試験の詳細については、「Exam Structures and Rules v1.1 Compatible with Syllabus Foundation and Advanced Levels and Specialists Modules」を参照すること。

テスト自動化エンジニア試験の受験にあたっては、候補者がソフトウェアテストとテスト自動化に関心を持っていることである。ただし、候補者には以下を強く推奨する：

- ・ ソフトウェアテストエンジニアまたはソフトウェア開発者として 6 か月の経験など、ソフトウェア開発とソフトウェアテストの最低限の基礎知識を持つこと。
- ・ ISTQB 認定メンバーボードのいずれかによって ISTQB 標準に準拠して認定されたコースを受講すること。

受験要件の注意：ISTQB® Foundation Level の証明書は、ISTQB® テスト自動化エンジニア認定試験を受験する前に取得する必要がある。

0.7 認定

ISTQB® メンバーボードは、本シラバスに従ったコース教材を持つトレーニングプロバイダーを認定することができる。トレーニングプロバイダーは、認定を実施するメンバーボードまたは機関から認定ガイドラインを取得する必要がある。認定コースは本シラバスに適合するものとして認識され、コースの一部として ISTQB® 試験を実施することが許可される。

本シラバスの認定ガイドラインは、Processes Management and Compliance Working Group が公開している一般的な認定ガイドラインに従う。

0.8 標準の取り扱い

テスト自動化エンジニアシラバスには標準（例：IEEE、ISO）が参照されている。これらの参照の目的は、フレームワークを提供すること（品質特性に関する ISO 25010 への参照など）、または読者が希望する場合に追加情報のソースを提供することである。なお、シラバスは標準文書を参照として使用しており、標準文書は試験対象ではない。詳細については第 9 章の参考文献を参照すること。

0.9 最新情報の維持

ソフトウェア業界は急速に変化している。こうした変化に対応し、ステークホルダーが関連する最新情報にアクセスできるようにするため、ISTQB ワーキンググループは ISTQB 公式サイト

(www.istqb.org) にサポート文書や標準の変更情報へのリンクを掲載している。これらの情報は、テスト自動化エンジニアシラバスの試験対象ではない。

0.10 詳細レベル

本シラバスの詳細レベルは、国際的に一貫したコースと試験を可能にすることを目的としている。そのため、本シラバスは以下で構成されている。

- テスト自動化エンジニアスペシャリストの目的を示す一般的な教育目的
- 受講者が想起する必要がある用語のリスト
- 達成すべき認知学習成果を説明する各知識領域の学習の目的
- 受け入れられた文献や標準などのソースへの参照を含む主要概念の説明

シラバスの内容はソフトウェアテストの知識領域全体の説明ではなく、テスト自動化エンジニアのトレーニングコースでカバーされる詳細レベルを反映している。アジャイル手法に従うプロジェクトを含む、すべてのソフトウェアプロジェクトに適用できるテストの概念と技法に焦点を当てている。本シラバスにはアジャイルテストに関連する具体的な学習の目的は含まれていないが、これらの概念がアジャイルプロジェクトや他の種類のプロジェクトにどのように適用されるかについては説明している。

0.11 本シラバスの構成

試験対象となる内容は 8 つの章で構成されている。各章の最上位見出しには所要時間が示されており、章より下位のレベルには時間配分は示されていない。認定トレーニングコースでは、シラバス全体で最低 21 時間の講義時間が必要であり、8 つの章に以下のように配分されている。

- 第 1 章：45 分 — テスト自動化の概要と目標
 - テスト自動化の利点と制約について学ぶ
 - さまざまなソフトウェア開発ライフサイクルモデルにおけるテスト自動化を扱う
 - テスト対象システム（SUT）のアーキテクチャが、テストツールの適合性に与える影響を学ぶ

- 第2章：180分ーテスト自動化の準備
 - テスト対象システム（SUT）のテスト可能性を高めるための設計（可観測性、制御性、明確なアーキテクチャなど）を扱う
 - さまざまな環境でテスト自動化をどのように活用するかを学ぶ
 - 適切なテスト自動化ソリューションを評価するために必要な要因を扱う
 - テスト自動化に関する推奨事項を作成するために必要な技術的観点を学ぶ
- 第3章：210分ーテスト自動化アーキテクチャ
 - テスト自動化アーキテクチャと、テスト自動化ソリューションを構成するコンポーネントを扱う
 - テスト自動化フレームワークにおけるレイヤー構造と、その適用方法を学ぶ
 - テスト自動化ツールを活用するための複数のアプローチを扱う
 - 設計原則と設計パターンをテスト自動化にどう適用するかを学ぶ
- 第4章：150分ーテスト自動化の実装
 - テスト自動化のパイロットプロジェクトを効果的に計画・デプロイする方法を扱う
 - デプロイメントに伴うリスクと、その軽減策について学ぶ
 - テスト自動化コードの保守性を高める要因を扱う
- 第5章：90分ーテスト自動化の実装とデプロイメント戦略
 - CI/CD パイプラインにおける自動テスト実行と、各テストレベルでの適用を学ぶ
 - テスト自動化コンポーネントの構成管理を扱う
 - API や契約テストに関わる依存関係について学ぶ
- 第6章：150分ーテスト自動化のレポートイングとメトリクス
 - SUT とテスト自動化から、分析やレポートイングに必要なデータをどこで収集できるかを学ぶ
 - SUT レポートやテスト自動化データを分析し、故障の原因を明らかにする方法を扱う
 - テストレポートやダッシュボードを用いて、ステークホルダーに情報提供する方法を扱う
- 第7章：135分ーテスト自動化ソリューションの検証
 - テスト自動化のコンポーネントと環境が正しく動作することをどのように検証するかを学ぶ
 - テストスクリプトとテストスイートが正しく実行されていることの確認を扱う
 - どのような場合に根本原因分析を行うべきかを理解する
 - テスト自動化コードの品質を分析する技法を扱う
- 第8章：210分ー継続的改善
 - テストケース改善のために活用できる追加のデータ分析領域を扱う

- テスト自動化ソリューションとその構成要素を改善・更新する方法を学ぶ
- テスト自動化を統合し、合理化する方法の特定について扱う
- テスト支援やセットアップのニーズに対して、ツールがどのように役立つかを学ぶ

1 テスト自動化の概要と目標 — 45 分 (K2)

キーワード

テスト対象システム (SUT) 、テスト自動化、テスト自動化エンジニア (TAE)

第 1 章の学習の目的：

1.1 テスト自動化の目的

TAE-1.1.1 (K2) テスト自動化の利点と欠点を説明する

1.2 ソフトウェア開発ライフサイクルにおけるテスト自動化

TAE-1.2.1 (K2) さまざまなソフトウェア開発ライフサイクルモデル (SDLC) におけるテスト自動化の適用方法を説明する

TAE-1.2.2 (K2) テスト対象システム (SUT) に適したテスト自動化ツールを選択する

1.1 テスト自動化の目的

1.1.1 テスト自動化の利点と欠点の説明

テスト自動化とは、自動テストの実行やテストレポートの作成を含む、次の1つ以上の活動を指す。

- 専用のソフトウェアツールを用いて、テスト実行用のテストスイートを制御・設定する
- テストを自動的に実行する
- 実際の結果を期待結果と比較する

テスト自動化は、テスト対象システム（SUT）と連携しながら重要な機能や能力を提供する。また、さまざまな種類のソフトウェアに適用できる。例えば、ユーザーインターフェース（UI）を持つSUT、UIを持たないSUT、モバイルアプリケーション、ネットワークプロトコル、接続などが含まれる。

テスト自動化には、以下のような多くの利点がある：

- 手動テストと比較して、1回のビルドでより多くのテストを実行できる
- 手動では実行できないテスト（例：リアルタイムレスポンス、リモートテスト、並列テスト）を作成・実行できる
- 手動テストよりも複雑なテストを実施できる
- 手動テストよりも高速に実行できる
- 人的エラーの影響を受けにくい
- テストリソースをより効果的かつ効率的に活用できる
- テスト対象システム（SUT）の品質に関するフィードバックを迅速に提供する
- システムの信頼性（例：可用性、回復性）の向上を助ける
- テストサイクルにわたるテスト実行の一貫性を向上させる

ただし、テスト自動化には以下の潜在的な欠点がある：

- テスト自動化エンジニア（TAE）の採用、新しいハードウェアの購入、トレーニング実施が必要になる場合があるため、プロジェクトに追加コストが発生する
- テスト自動化ソリューションの構築に初期投資が必要である
- テスト自動化ソリューションの開発と保守に時間がかかる
- 成功を確実にするため、テスト自動化の目的を明確に定義する必要がある
- テストの柔軟性が低く、テスト対象システム（SUT）の変更への適応性が低い
- テスト自動化による新たな欠陥を引き起こす可能性がある

テスト自動化には注意しておくべき制限事項がある：

- すべての手動テストが自動化できるわけではない
- 自動テストはプログラムされたことのみを検証する
- テスト自動化は機械が解釈できるテスト結果のみをチェックできる。これは、一部の品質特性が自動化でテストできない可能性があることを意味する

- テスト自動化は自動化されたテストオラクルで検証できるテスト結果のみをチェックできる

1.2 ソフトウェア開発ライフサイクル(SDLC)におけるテスト自動化

1.2.1 さまざまなソフトウェア開発ライフサイクルモデル (SDLC) におけるテスト自動化の適用方法の説明

ウォーターフォール

ウォーターフォールモデルは、線形かつ逐次的な SDLC モデルである。このモデルには明確なフェーズ（要件定義、設計、実装、検証、保守）があり、各フェーズは通常、承認が必要な文書の作成で終了する。テスト自動化の実装は通常、実装フェーズと並行して、またはその後に実施される。ソフトウェアコンポーネントが検証フェーズまでテスト可能にならないため、テストの実行は通常、検証フェーズで行われる。

V 字モデル

V 字モデルは、プロセスが逐次的に実行される SDLC モデルである。プロジェクトが高レベルの要件から低レベルの要件へと定義されるにつれて、それらの要件を妥当性確認するための対応するテストおよび統合活動が定義される。これは CTFL (Certified Tester Foundation Level) 2.2 に記載されているように、従来のテストレベル（コンポーネント、コンポーネント統合、システム、システム統合、受け入れテスト）の起源となっている。各テストレベルにテスト自動化フレームワーク（TAF）を適用することは可能であり、推奨される。

アジャイルソフトウェア開発

アジャイルソフトウェア開発では、テスト自動化を活用できる場面が数多くある。ウォーターフォールモデルや V 字モデルとは異なり、アジャイル開発ではテスト自動化エンジニア（TAE）とビジネス担当者が、ロードマップ、タイムライン、実施予定のテストについて協調して判断できる。また、コードレビュー、ペアプログラミング、頻繁な自動テスト実行などのベストプラクティスが重視される。開発者、テスト担当者、その他のステークホルダーのサイロ化を避けて協働することで、チームは各テストレベルを適切な量と深さの自動化でカバーでき、いわゆる「スプリント内自動化」を実現しやすくなる。詳細は ISTQB CT-TAS (Test Automation Strategy) 3.2 節を参照すること。

1.2.2 テスト対象システム (SUT) に適したテスト自動化ツールの選択

特定のプロジェクトに最適なテストツールを選定するには、まずテスト対象システム（SUT）を分析する必要がある。テスト自動化エンジニア（TAE）は、ツール選定の基準となるプロジェクト要件を明確にする必要がある。

例えば、UI を持つソフトウェアとウェブサービスとでは、必要となるテスト自動化ツールの機能が異なる。そのため、プロジェクトが長期的に何を達成したいのかを理解することが重要である。利用可能なテスト自動化ツールや機能は多岐にわたるが、コストは常に考慮する必要がある。商用の既製ツールを使う場合でも、オープンソース技術に基づくカスタムソリューションを実装する場合でも、導入や運用のプロセスが複雑になる可能性がある。

次に評価すべき観点は、チームの構成とテスト自動化の経験である。テスト担当者のプログラミング経験がほとんどない、またはまったくない場合は、ローコードまたはノーコードのソリューションが有効な選択肢となることがある。

一方、プログラミング知識を持つ技術系テスト担当者の場合、テスト対象システム（SUT）の言語と一致するツールを選択すると有利である。これには、開発者とより効率的に協力してテスト自動化の欠陥をデバッグできることや、チーム間でメンバーの相互育成を行えることなどの利点がある。

2 テスト自動化の準備 — 180 分 (K4)

キーワード

API テスト、GUI テスト、テスト可能性

第 2 章の学習の目的：

2.1 テスト自動化を可能にするインフラストラクチャ構成の理解

TAE-2.1.1 (K2) テスト自動化の実装を可能にするインフラストラクチャの構成要件を説明する

TAE-2.1.2 (K2) 異なる環境におけるテスト自動化の活用方法を説明する

2.2 適切なツールと戦略を選択するための評価プロセス

TAE-2.2.1 (K4) 適切なテスト自動化ソリューション (TAS) を決定するために、テスト対象システム (SUT) を分析する

TAE-2.2.2 (K4) ツール評価の技術的な結果を提示する

2.1 テスト自動化を可能にするインフラストラクチャ設定の説明

2.1.1 テスト自動化の実装を可能にするインフラストラクチャの構成要件の説明

テスト対象システム（SUT）のテスト可能性（**testability**）とは、テストを支援するソフトウェアインターフェースが利用可能であること、例えば SUT の制御性や可観測性（**observability**）を実現する仕組みが備わっていることを指す。こうした仕組みは、SUT の他の機能の設計・実装と並行して設計・実装される必要がある。テスト可能性は通常、システムの非機能要件の一部であるため、主にソフトウェアアーキテクトが対応するが、改善できる領域を特定するためにテスト自動化エンジニア（TAE）が関与することも少なくない。

テスト対象システム（SUT）のテスト可能性（**testability**）を高めるために、構成要件の異なるさまざまなソリューションを活用できる。例えば：

- アクセシビリティ識別子
 - 異なる開発フレームワークでは、これらの識別子を自動生成する場合もあれば、開発者が手動で設定する場合もある
- システム環境変数
 - 管理機能を通じて特定のアプリケーションパラメーターを変更することで、テストをより容易にできる
- デプロイメント変数
 - システム変数と同様であるが、デプロイメント開始前に設定できる

テスト対象システム（SUT）のテスト可能性（**testability**）の設計は以下の側面で構成される：

- 可観測性（**Observability**）：SUT は、その内部状態や振る舞いを把握できるようにするためのインターフェースを提供する必要がある。テストケースは、これらのインターフェースを利用して、実際の結果が期待結果と一致しているかを判断できる。
- 制御性（**Controllability**）：SUT は、SUT に対して操作を実行するためのインターフェースを提供する必要がある。これは、UI 要素、関数呼び出し、通信要素（例：TCP/IP、USB プロトコル）、あるいは環境変数を切り替える物理的または論理的スイッチの電子信号である場合がある。
- アーキテクチャの透明性（**Transparency**）：アーキテクチャのドキュメントは、すべてのテストレベルで可観測性と制御性を提供し、品質を促進する明確で理解可能なコンポーネントとインターフェースを提供する必要がある。

2.1.2 異なる環境におけるテスト自動化の活用方法の説明

異なる種類の自動テストは、異なる環境で実行できる。これらの環境はプロジェクトや開発手法によって異なり、多くのプロジェクトではテストに使用する環境が 1 つ以上存在する。技術的には、これらの環境はコンテナ、仮想化ソフトウェア、その他の手法を用いて構築できる。

考慮すべき環境として、以下が含まれる：

ローカル開発環境

ローカル開発環境は、ソフトウェア開発が最初に行われる場所であり、コンポーネントは自動テストによって機能適合性が検証される。ローカル開発環境では、コンポーネントテスト、GUI テスト、アプリケーションプログラミングインターフェース (API) テストなど、いくつかのテストタイプを実施できる。また、特定のコンピューターの統合開発環境 (IDE) を使用することで、ホワイトボックステストを実行し、できる限り早い段階で不適切なコーディングや品質上の問題を特定することも重要である。

ビルド環境

その主な目的は、ソフトウェアをビルドし、DevOps エコシステムにおいてビルド結果の正確性を確認するテストを実行することである。この環境はローカル開発環境または継続的インテグレーション/継続的デリバリー (CI/CD) エージェントのいずれかであり、そこでは低レベルのテスト (コンポーネントテスト、コンポーネント統合テスト) と静的解析を、他の環境へ実際にデプロイすることなく実行できる。

統合環境

低レベルテストと静的解析を実施した後、次の段階はシステム統合環境である。ここには、他のシステムと完全に統合され、テスト可能な状態にあるテスト対象システム (SUT) のリリース候補がある。この環境では、UI テストまたは API テストのいずれかによる、完全自動化されたテストスイートを実行できる。この環境ではホワイトボックステストは行わず、ブラックボックステスト (システム統合テストおよび受け入れテスト) のみを実施する。また、この環境は、モニタリングが初めて導入される環境であり、SUT の使用中にバックグラウンドで何が起きているかを確認して、欠陥や故障の効率的な調査を可能にする最初の環境である点も重要である。

プレプロダクション環境

プレプロダクション環境は主に非機能品質特性 (例: 性能効率性) の評価に使用される。非機能テストはあらゆる環境で実行できるが、プレプロダクション環境は本番環境にできるだけ近いため、特に注目される。多くの場合、ビジネスステークホルダーが最終プロダクトを検証するためのユーザー受け入れテストを実施でき、必要に応じて既存の自動テストスイートをここで実行することも可能である。この環境もモニタリングされている。

本番/運用環境

本番環境は、モニタリングと本番環境でのテストを可能にする特定のベストプラクティス (例: カナリアリリース、ブルー/グリーンデプロイメント、A/B テスト) を使用して、ユーザーがデプロイされたシステムを利用しながらリアルタイムで機能品質特性および非機能品質特性を評価するために使用できる。

2.2 適切なツールと戦略を選択するための評価プロセス

2.2.1 適切なテスト自動化ソリューション (TAS) を決定するためのテスト対象システム (SUT) の分析

各テスト対象システム (SUT) は異なるが、成功するテスト自動化ソリューション (TAS) を構築するために分析すべきさまざまな要因や特性がある。SUT を調査する際、テスト自動化エンジニア (TAE) は、そのスコープや特有の機能を踏まえて要件を収集する必要がある。ウェブサービス、モバイル、ウェブアプリケーションなど、アプリケーションの種類によって、必要となるテスト自動化の方式や技術は異なる。調査は、できるだけ多くのリスクとその軽減策を特定し、将来にわたり有効

に機能する TAS を構築するために、手動テスト担当者、ビジネスステークホルダー、ビジネスアナリストなど他の関係者と協力して進めることが推奨される。

テスト自動化のアプローチとアーキテクチャに対する要件では、以下を考慮する必要がある：

- どのテストプロセス活動を自動化すべきか（例：テスト管理、テスト設計、テスト生成、テスト実行）
- どのテストレベルをサポートすべきか
- どのテストタイプをサポートすべきか
- どのテストの役割とスキルセットをサポートすべきか
- どのソフトウェアプロダクト、プロダクトライン、プロダクトファミリーをサポートすべきか（例：実装された TAS のスパンと寿命を定義するため）
- どのような種類の SUT が TAS と互換性を持つ必要があるか
- テストデータの可用性とその品質
- 再現が難しいケースをエミュレートするための方法や手段（例：関係するサードパーティアプリケーション）

2.2.2 ツール評価の技術的結果の提示

テスト対象システム（SUT）が分析され、すべてのステークホルダーから要件が収集された後、これらの要件を満たす可能性のあるテスト自動化ツールが検討される。すべての特定された要件に適合する単一のツールがない場合もあり、ステークホルダーはこの可能性を認識する必要がある。

候補となるツールの評価結果を把握し、さまざまな直接要件・間接要件を比較表に反映させることが有効である。比較表の目的は、特定の要件に照らしてツール間の違いをステークホルダーが確認できるようにすることである。一般に、比較表では列にツール、行に要件を配置し、各セルにその要件に対するツールの特性や優先度に関する情報を記載する。

一般的に、テスト自動化ツールは前の項（2.2.1 項）で特定された要件に適合するかどうかを評価する必要がある。ツールの評価と比較の際に考慮すべき要件には以下が含まれる：

- ツールの言語/技術と IDE ツール
- ツールを構成できること、異なるテスト環境をサポートするかどうか、実行設定の切り替え、動的または静的な設定値を使用するかどうか
- ツール内でテストデータを管理する能力。テストデータ管理は、バージョン管理用の中央リポジトリと統合される場合がある。
- 異なるテストタイプに対して、異なるテスト自動化ツールを選択する必要がある場合がある
- レポート機能を提供する能力。これはプロジェクトのテストレポート要件に合わせる必要がある。
- プロジェクトや組織で使用する他のツール（CI/CD、タスクトラッキング、テスト管理、レポートなど）と統合する能力
- 全体的なテストアーキテクチャを拡張し、ツールのスケーラビリティ、保守性、変更可能性、互換性、信頼性を評価する能力

この比較表は、SUT のテスト自動化に使用する候補ツールまたはツールセットを決定するための有効な情報源となる。

どのツールを使用するか決定方法はプロセスによって異なる場合があるが、提案内容は、適切なステークホルダーに提示して、承認を得る必要がある。

3 テスト自動化アーキテクチャ — 210 分 (K3)

キーワード

振る舞い駆動開発、キャプチャ/プレイバック、データ駆動テスト、汎用テスト自動化アーキテクチャ (gTAA)、キーワード駆動テスト、線形スクリプティング、モデルベースドテスト、構造化スクリプティング、テスト適合レイヤー、テスト自動化フレームワーク、テスト自動化ソリューション、テストハーネス、テストスクリプト、テストウェア、テストステップ、テスト駆動開発

第 3 章の学習の目的：

3.1 テスト自動化で活用されるデザインコンセプト

TAE-3.1.1 (K2) テスト自動化アーキテクチャの主要機能を説明する

TAE-3.1.2 (K2) テスト自動化ソリューションの設計方法を説明する

TAE-3.1.3 (K3) テスト自動化フレームワークのレイヤー構造化を適用する

TAE-3.1.4 (K3) テストケースを自動化するための各種アプローチを適用する

TAE-3.1.5 (K3) テスト自動化における設計原則と設計パターンを適用する

3.1 テスト自動化で活用されるデザインコンセプト

3.1.1 テスト自動化アーキテクチャの主要機能の説明

汎用テスト自動化アーキテクチャ（gTAA）

gTAA は、テスト自動化と、テスト自動化が接続されているシステム（テスト対象システム（SUT）、プロジェクト管理、テスト管理、構成管理）の間の通信の抽象的なビューを提供する高レベルの設計コンセプトである（図 1 参照）。また、テスト自動化アーキテクチャ（TAA）を設計する際にカバーする必要がある機能を示す。

gTAA のインターフェースは以下の通りである：

- SUT インターフェースは、SUT と TAF（テスト自動化フレームワーク）に関しては、3.1.3 項参照）の間の接続を記述する。
- プロジェクト管理インターフェースは、テスト自動化の開発進捗を記述する。
- テスト管理インターフェースは、テストケース定義と自動テストケースのマッピングを記述する。
- 構成管理インターフェースは、CI/CD パイプライン、環境、テストウェアを記述する。

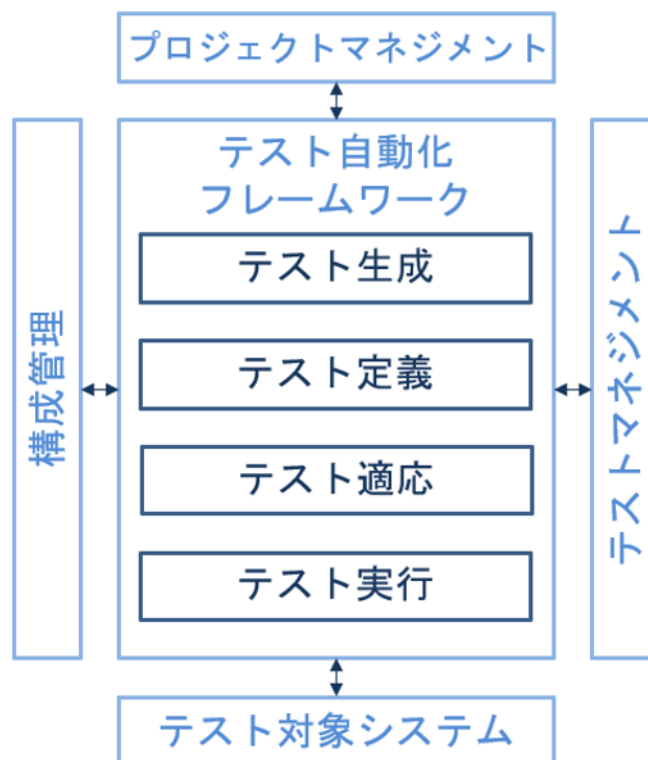


図 1：gTAA（汎用テスト自動化アーキテクチャ）の構成

テスト自動化ツールとライブラリが提供する機能

コアとなるテスト自動化機能は、特定のプロジェクトに必要な機能を満たす利用可能なツールの中から特定し、選定する必要がある。

テスト生成：テストモデルに基づいてテストケースの自動設計をサポートする。モデルベースドテストツールは生成プロセスで活用できる（ISTQB CT-MBT シラバス参照）。テスト生成はオプションの機能である。

テスト定義：テストモデルから任意に派生させることができるテストケースおよび/またはテストスイートの定義と実装をサポートする。テスト定義をテスト対象システム（SUT）および/またはテストツールから分離する。高レベルおよび低レベルのテストを定義する手段が含まれており、テストデータ、テストケース、テストライブラリのコンポーネントまたはそれらの組み合わせで処理される。

テスト実行：テスト実行とテストロギングをサポートする。選択されたテストを自動的に実行するためのテスト実行ツール、およびテストロギングとテストレポートコンポーネントを提供する。

テスト適合：テスト対象システム（SUT）のさまざまなコンポーネントやインターフェースに対して自動テストを適合させるために必要な機能を提供する。API、プロトコル、サービスを通じて SUT に接続するためのさまざまなアダプターを提供する。

3.1.2 テスト自動化ソリューションの設計方法の説明

テスト自動化ソリューション（TAS）は、テスト対象システム（SUT）の機能的、非機能的、技術的要件の理解や、ソリューションを実装するために必要な既存のツールや、新たに必要となるツールによって定義される。TAS は商用またはオープンソースのツールで実装され、SUT 固有のアダプターが必要になる場合がある。

TAA（テスト自動化アーキテクチャ）は、TAS 全体の技術設計を定義する。以下の点を考慮する必要がある：

- テスト自動化ツールとツール固有のライブラリの選定
- プラグインおよび/またはコンポーネントの開発
- 接続性とインターフェース要件の特定（例：ファイアウォール、データベース、URL/接続、モック/スタブ、メッセージキュー、プロトコル）
- テスト管理ツールおよび欠陥管理ツールへの接続
- バージョン管理システムとリポジトリの活用

3.1.3 テスト自動化フレームワークのレイヤー構造化の適用

テスト自動化フレームワーク（TAF）

TAF は TAS の基盤となる要素である。多くの場合、テストハーネス（テストランナーとも呼ばれる）、テストライブラリ、テストスクリプト、およびテストスイートを含む。

TAF のレイヤー

TAF のレイヤーは、テストケース、テストレポーティング、テストロギング、暗号化、テストハーネスなど、同様の目的を持つクラスの境界を明確にする。ただし、目的ごとにレイヤーを導入すると設計は複雑になる。そのため、TAF のレイヤー数は必要最小限に抑えることが推奨される。

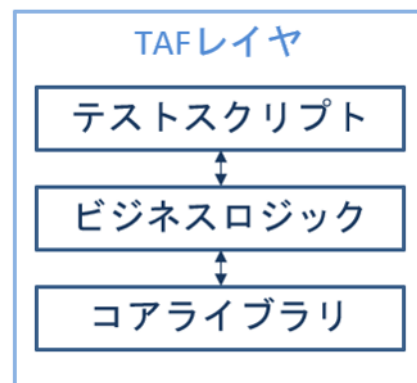


図 2：テスト自動化フレームワークのレイヤー構造

テストスクリプト

テストスクリプトの目的は、テスト対象システム（SUT）のテストケースリポジトリと、テストスイートを定義するための注釈情報を提供することである。テストスクリプトは、テストステップ、ユーザーフロー、または API 呼び出しを含むビジネスロジックレイヤーのサービスを呼び出す。ただし、テストスクリプトからコアライブラリを直接呼び出さないようにする。

ビジネスロジック

テスト対象システム（SUT）に依存するライブラリは、このレイヤーに配置される。これらのライブラリは、コアライブラリのクラスを継承するか、コアライブラリが提供するファサード（外部インターフェース）を利用する（継承とファサードについては 3.1.5 項参照）。ビジネスロジックレイヤーは、SUT や追加設定に対して TAF を実行するためのセットアップにも用いられる。

コアライブラリ

テスト対象システム（SUT）に依存しないすべてのライブラリはこのレイヤーに格納される。コアライブラリは、同じ開発スタックを共有するあらゆる種類のプロジェクトで再利用できる。

テスト自動化のスケールリング

次の例（図 3）は、コアライブラリが複数の TAF に再利用可能な基盤を提供する方法を示している。プロジェクト#1 には、コアライブラリの上に構築された 2 つの TAF があり、別のプロジェクトが既存のコアライブラリを活用してアプリ#3 のテスト用 TAF を構築している。1 人の TAE がプロジェクト#1 の TAF を構築し、2 人目の TAE がプロジェクト#2 の TAF を構築している。

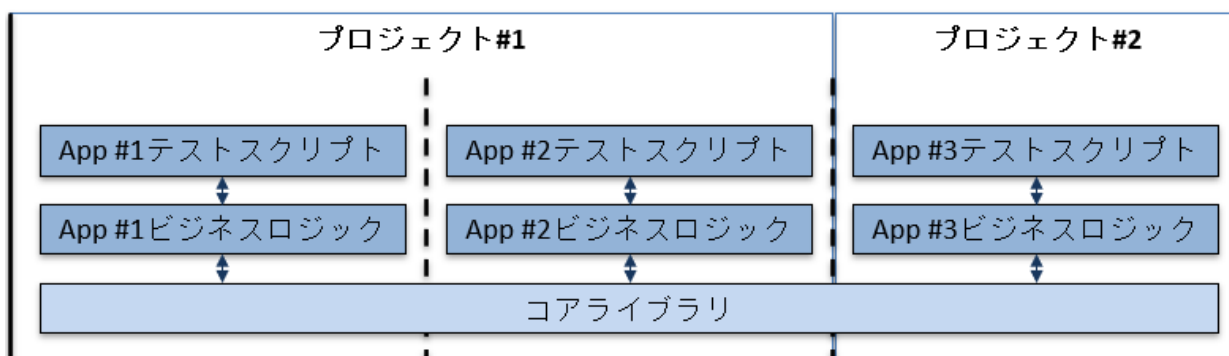


図 3 : 複数の TAF へのコアライブラリの適用

3.1.4 テストケースを自動化するための各種アプローチの適用

チームが選択できる、自動テストケースを作成するための開発アプローチはいくつかある。これらには、対話型スクリプト言語やコンパイル型プログラミング言語が含まれる場合がある。アプローチごとに自動化の利点が異なり、適した状況も異なる。テスト駆動開発（TDD）と振る舞い駆動開発（BDD）は開発手法であるが、適切に実践すれば自動テストケースの開発につながる。

キャプチャ/プレイバック（Capture/Playback）

キャプチャ/プレイバックは、一連のアクションが手動で実行される間にテスト対象システム（SUT）とのやり取りをキャプチャするアプローチである。これらのツールはキャプチャ中にテストスクリプトを生成し、使用するツールによっては、テスト自動化コードを変更できる場合がある。コードを公開しないツールはノーコードテスト自動化と呼ばれることがあり、コードを公開するツールはローコードテスト自動化と呼ばれる。

利点

- 初期設定と使用が容易である

欠点

- 保守、拡張、変化への対応が難しい
- テストケースのキャプチャ中にテスト対象システム（SUT）が利用可能である必要がある
- スcopeが小さく、変更頻度の低い SUT にのみ実用的である
- キャプチャされた SUT 実行は、キャプチャが取得されたときの SUT バージョンに大きく依存する

- 既存の構成要素を再利用する代わりに各テストケースを録画することは時間がかかる

線形スクリプティング (Linear Scripting)

線形スクリプティングは、テスト自動化エンジニア (TAE) が作成したカスタムテストライブラリを必要とせず、テストスクリプトの作成と実行に使用されるプログラミング活動である。TAE は、キャプチャ/プレイバックツールで記録されたテストスクリプトを活用し、それを変更することができる。

利点

- テストスクリプトの設定と作成が容易である
- キャプチャ/プレイバックと比較して、テストスクリプトをより容易に変更できる

欠点

- 保守、拡張、変化への対応が難しい
- テストケースのキャプチャ中にテスト対象システム (SUT) が利用可能である必要がある
- スcopeが小さく、変更頻度の低い SUT にのみ実用的である
- キャプチャ/プレイバックと比較して、ある程度のプログラミング知識が必要である

構造化スクリプティング (Structured Scripting)

再利用可能な要素、テストステップ、およびユーザージャーニーを含むテストライブラリが導入される。このアプローチではテストスクリプトの作成と保守にプログラミング知識が必要である。

利点

- 保守、拡張、移植、適応、変更への追従が容易である
- ビジネスロジックをテストスクリプトから分離できる

欠点

- プログラミング知識が必要である
- TAF 開発とテストウェアの定義への初期投資は時間がかかる

テスト駆動開発 (Test-Driven Development)

テスト対象システム (SUT) の新機能が実装される前に、テストケースは開発プロセスの一部として定義される。TDD アプローチは、テスト、コーディング、リファクタリングであり、レッド、グリーン、リファクタリングとして知られている。開発者は失敗する (レッド) テストケースを 1 つ特定して作成する。次にテストケースを満たす機能を開発する (グリーン)。その後、コードを最適化し、クリーンコード原則に準拠させるようにリファクタリングする。このプロセスは、次のテスト、次の機能追加へと繰り返される。

利点

- コンポーネントレベルのテストケース開発を簡素化する
- コード品質とコードの構造を向上させる
- テスト可能性 (testability) を向上させる
- 望ましいコードカバレッジを達成しやすくする

- 上位テストレベルへの欠陥の伝播を減らす
- 開発者、ビジネス担当者、テスト担当者間のコミュニケーションを向上させる
- GUI テストや API テストでまだ検証されていないユーザーストーリーについても、TDD に従うことで、終了基準をより早く満たしやすくなる

欠点

- 最初は TDD に慣れるのに時間がかかる
- TDD を適切に実行しないと、コード品質への誤った確信につながる可能性がある

データ駆動テスト (Data-Driven Testing)

データ駆動テスト (DDT) は構造化スクリプティングアプローチを基盤としている。テストスクリプトにはテストデータ (例: .csv ファイル、.xlsx ファイル、データベースダンプ) が提供される。これにより、異なるテストデータで同じテストスクリプトを複数回実行できる。

利点

- データ供給源を通じてテストケースを素早く簡単に拡張できる
- 新しい自動テストの追加コストを大幅に削減できる
- テストアナリストは、テストを説明する 1 つ以上のテストデータファイルを入力することで、自動テストを指定できる。これにより、テストアナリストはテクニカルテストアナリストへの依存を減らして自動テストを指定する自由度が増す。

欠点

- 適切なテストデータ管理が必要になる場合がある

キーワード駆動テスト (Keyword-Driven Testing)

キーワード駆動テスト (KDT) のテストケースは、キーワードとキーワードが操作するテストデータから導き出されたテストステップのリストまたはテーブルで構成される。キーワードはユーザーの観点から定義される。この手法は多くの場合、データ駆動テスト (DDT) を基盤として構築される。

利点

- KDT アプローチに従うことで、テストアナリストとビジネスアナリストが自動テストケースの作成に参加できる
- KDT はテスト自動化とは独立して手動テストにも使用できる (キーワード駆動テストに関する ISO/IEC/IEEE 29119-5 規格を参照)

欠点

- キーワードの実装と保守は、スコープが拡大するにつれて困難になる可能性がある複雑なタスクであり、テスト自動化エンジニア (TAE) がカバーする必要がある
- 小規模なシステムには多大な労力がかかる

振る舞い駆動開発 (Behavior-Driven Development)

振る舞い駆動開発 (BDD) は、自動テストケースとして使用でき、フィーチャーファイルに保存できる受け入れ基準を策定するために、自然言語フォーマット (given、when、then) を活用する。BDD ツールはその言語を理解してテストケースを実行できる。

利点

- 開発者、ビジネス担当者、テスト担当者間のコミュニケーションを向上させる
- 自動化された BDD シナリオはテストケースとして機能し、仕様のカバレッジを確保する
- BDD はテストピラミッドの異なるレベルで複数のテストタイプを生成するのに活用できる

欠点

- 追加のテストケース（通常は否定のテスト条件やエッジケース）は、依然としてチーム（通常はテストアナリストまたは TAE）が定義する必要がある
- 多くのチームが BDD を自然言語でテストケースを書く方法のみだと混同し、アプローチ全体にビジネス担当者や開発者を巻き込まない
- 自然言語のテストステップの実装と保守は TAE にとって複雑な作業である
- 過度に複雑なテストステップはデバッグを困難でコストのかかる活動に変えてしまう

3.1.5 テスト自動化における設計原則と設計パターンの適用

テスト自動化はソフトウェア開発活動である。したがって、設計原則と設計パターンはソフトウェア開発者と同様に TAE にとっても重要である。

オブジェクト指向プログラミング原則

オブジェクト指向プログラミングには、カプセル化、抽象化、継承、ポリモーフィズムという 4 つの主要原則がある。

SOLID 原則

これは単一責任（Single responsibility）、開放/閉鎖（Open-closed）、リスコフの置換（Liskov substitution）、インターフェース分離（Interface segregation）、依存性逆転（Dependency inversion）の頭字語である。これらの原則はコードの読みやすさ、保守性、スケーラビリティを向上させる。

デザインパターン

多くのデザインパターンの中で、TAE に最も重要な 3 つがある。

ファサードパターンは実装の詳細を隠し、テスト担当者がテストケースに必要なものだけを公開する。シングルトンパターンは、テスト対象システム（SUT）と通信するドライバーを 1 つに限定するためによく使用される。

ページオブジェクトモデルでは、作成されたクラスファイルをページモデルと呼ぶ。SUT の構造が変更されるたびに、TAE は各テストケースのロケーターを更新するのではなく、ページモデル内の 1 か所のみを更新するだけで済む。

フローモデルパターンはページオブジェクトモデルの拡張である。ページオブジェクトモデルの上に追加のファサードを導入し、ページオブジェクトと連携するすべてのユーザーアクションを格納する。ダブルファサード設計を導入することにより、フローモデルパターンはテストステップを複数のテストスクリプトで再利用できるため、より高い抽象化と保守性を実現する。

4 テスト自動化の実装 — 150 分 (K4)

キーワード

リスク、テストフィクスチャ

第 4 章の学習の目的：

4.1 テスト自動化開発

TAE-4.1.1 (K3) 効果的なテスト自動化パイロットおよびデプロイメント活動を支援するガイドラインを適用する

4.2 テスト自動化開発に関連するリスク

TAE-4.2.1 (K4) テスト自動化のデプロイメントリスクを分析し、軽減策を計画する

4.3 テスト自動化ソリューションの保守性

TAE-4.3.1 (K2) テスト自動化ソリューションの保守性を支援・影響する要因を説明する

4.1 テスト自動化開発

4.1.1 効果的なテスト自動化パイロットおよびデプロイメント活動を支援するガイドラインの適用

テスト自動化のパイロットでは、妥当性確認の範囲を定義することが重要である。パイロットプロジェクトの実施にはそれほど長い時間はかからない場合があるが、その結果はプロジェクトの方向性に大きな影響を与える可能性がある。

テスト対象システム（SUT）に関して収集された情報とプロジェクトの要件に基づいて、テスト自動化を最適化するためのガイドラインを設定するには、以下を評価する必要がある：

- 採用するプログラミング言語
- 適切な商用既製ツール/オープンソースツール
- カバーするテストレベル
- 選択されたテストケース
- テストケース開発アプローチ

上記の項目に基づいて、テスト自動化エンジニア（TAE）は、採用すべき初期アプローチを定義できる。要件に応じて、異なるアプローチの利点と欠点を示す複数の初期プロトタイプを作成することもできる。その結果に基づき、TAE は進むべき方向を判断できる。

タイムラインの定義は、スケジュールを守り、パイロットの成功を確保するうえで重要である。一般的には、パイロットプロジェクトの進捗を定期的に確認し、リスクを特定して軽減することが推奨される。

パイロット中は、ソリューションと実装済みコードを CI/CD に統合することも推奨される。これにより、テスト対象システム（SUT）、テスト自動化ソリューション（TAS）、または組織内の異なるツールの全体的な統合における問題が早期に明らかになる可能性がある。

テストケースの数が増えるにつれて、TAE は、初期の CI/CD セットアップを見直し、異なる方法・異なるタイミングでテストを実行できるように変更することを検討できる。

また、パイロットプロジェクト中には、以下のような技術的でない側面を評価する必要がある：

- チームメンバーの知識と経験
- チーム構成
- ライセンスおよび組織のルール
- テストケース自動化の対象として計画しているテストの種類とテストレベル

パイロットプロジェクトが完了したら、TAE とテストマネージャーは、成功または失敗を評価し、適切な判断を下すために、その取り組み全体を評価する必要がある。

4.2 テスト自動化開発に関連するリスク

4.2.1 テスト自動化のデプロイメントリスクの分析と軽減策の計画

テスト自動化フレームワーク（TAF）とテスト対象システム（SUT）のインターフェースは、アーキテクチャ設計の一部として考慮される必要がある。その後、パッケージング、テストロギング、テストハーネスツールを選択できる。

パイロットの実装中に、テスト自動化のコードの拡張性と保守性を考慮する必要がある。これらはパイロット評価フェーズの重要な要素であり、最終決定に重大な影響を与える可能性がある。

パイロットを通じて、さまざまなデプロイメントリスクを特定できる：

- ファイアウォールの開放
- リソース使用率（例：CPU、RAM）

ファイアウォールの問題、リソース使用率、ネットワーク接続および信頼性などのデプロイメントリスクに備える必要がある。これらはテスト自動化に厳密には関連していないが、テスト自動化エンジニア（TAE）は開発プロセスで信頼性が高く有益な品質ゲートを提供するためにすべての条件が満たされていることを確認する必要がある。

モバイルテスト自動化のために実際のデバイスを使用するケースは、その一例である。モバイルデバイスは電源が入っていて、テスト中に動作するのに十分なバッテリーがあり、ネットワークに接続されており、SUT にアクセスできる必要がある。

技術的なデプロイメントリスクには以下が含まれる：

- パッケージング
- ロギング
- テスト構造化
- 更新

パッケージング

テスト自動化のバージョン管理が SUT にとって重要であるように、パッケージングについても考慮する必要がある。テストウェアは組織全体で共有するために、オンプレミスまたはクラウドのリポジトリにアップロードする必要がある場合がある。

ロギング

テストロギングはテスト結果に関する情報の大部分を提供する。テスト自動化にはいくつかのテストロギングレベルがあり、それぞれさまざまな理由で有用である：

- 致命的（Fatal）：このレベルはテスト実行を中断させる可能性のあるエラーイベントをログに記録するために使用される
- エラー（Error）：このレベルは条件または操作が失敗し、テストケースも失敗する場合に使用される
- 警告（Warn）：このレベルは予期しない条件/アクションが発生したがテストケースのフローを壊さない場合に使用される

- 情報 (Info) : このレベルはテストケースの基本情報とテスト実行中に何が起こるかを表示するために使用される
- デバッグ (Debug) : このレベルは基本ログには通常必要ないが、テスト失敗の調査中に役立つ実行固有の詳細を格納するために使用される
- トレース (Trace) : このレベルは Debug に似ているが、さらに詳細な情報を含む

テスト構造化

TAS の最も重要な部分はテストハーネスとそれに含まれるテストフィクスチャ (テストを実行するために利用可能でなければならない項目) である。テストフィクスチャはテスト環境とテストデータの制御に自由を与える。事前条件と事後条件はテスト実行に対して定義でき、テストケースはいくつかの方法でテストスイートにグループ化できる。これらの側面はパイロットプロジェクト中の評価においても重要である。さらに、テストフィクスチャにより、反復可能で自己完結した自動テストの作成が可能になる。

更新

最も一般的な技術的リスクの 1 つは、テストハーネス (例: エージェント) の自動更新とデバイスのバージョン変更である。これらのリスクは、適切な電源供給、適切なネットワーク接続、適切なデバイス構成計画を持つことで軽減できる。

4.3 テスト自動化ソリューションの保守性

4.3.1 テスト自動化ソリューションの保守性を支援・影響する要因の説明

保守性は、プログラミング標準や TAE 同士の共通認識に大きく影響される。基本原則としては、Robert C. Martin によるクリーンコード原則に従うことが挙げられる。(Robert C Martin, "Clean Code: A Handbook of Agile Software Craftsmanship", 2008)

簡潔に言えば、クリーンコード原則の要点は以下の点を強調している:

- クラス、メソッド、変数に意味のある名前を付けるための共通の命名規則を使用する
- 論理的で共通のプロジェクト構造を使用する
- ハードコーディングを避ける
- メソッドの入力パラメーターが多くなりすぎないようにする
- 長くて複雑なメソッドを避ける
- ロギングを使用する
- 有益で必要な場合にデザインパターンを使用する
- テスト可能性 (testability) に焦点を当てる

命名規則は、変数の意味を識別するのに非常に役立つ。例えば、「loginButton」、「resetPasswordButton」などの理解可能な変数名を持つことで、TAE はどのコンポーネントを使用すべきかを理解しやすくなる。

ハードコーディングとは、値をソフトウェア内に直接埋め込み、後で容易に変更できないようにすることである。データ駆動テストを使用することで回避できる。テストデータはより簡単に保守できる

共通ソースから取得される。ハードコーディングは開発時間を短縮するが、データが頻繁に変更されると保守に時間がかかる可能性があるため、使用は推奨されない。頻繁には変更されない変数には定数を使用することも推奨される。そうすることで、保守が必要なソースと場所を減らすことができる。デザインパターンの活用も強く推奨される。デザインパターン（3.1.5 項に記載）は、適切に使用される限り、構造化され適切に保守可能なテスト自動化コードの実装を可能にする。

テスト自動化コードの高品質を確保するために、静的解析ツールを活用することが推奨される。IDE で一般的に使用されるコードフォーマッターはテスト自動化コードの読みやすさを向上させる。

クリーンコード原則以外にも、バージョン管理で合意されたブランチ構造と戦略を使用することが推奨される。機能、リリース、欠陥修正に異なるブランチを使用することは、ブランチの内容を理解するのに役立つ。

5 テスト自動化の実装とデプロイメント戦略 — 90 分 (K3)

キーワード

契約テスト

第 5 章の学習の目的：

5.1 CI/CD パイプラインへの統合

TAE-5.1.1 (K3) パイプライン内の各テストレベルでテスト自動化を適用する

TAE-5.1.2 (K2) テストウェアの構成管理を説明する

TAE-5.1.3 (K2) API インフラストラクチャに対するテスト自動化の依存関係を説明する

5.1 CI/CD パイプラインへの統合

5.1.1 パイプライン内の各テストレベルでのテスト自動化の適用

テスト自動化の主な利点の 1 つは、実装されたテストを無人で実行できることであり、パイプライン内での実行に適している。これは **CI/CD** パイプライン、または定期的にテストを実行するために使用されるパイプラインを通じて実現できる。

テストレベルは通常以下のように統合される：

- **TAF/TAS** の構成テストは、ビルド中に実行される一種のコンポーネントテストと見なすことができる。これらのテストは、テスト自動化プロジェクト (**TAF/TAS**) のビルド時に実行され、テストスクリプトで使用されるすべてのファイルへのパスが正しく、ファイルが実際に存在し、指定された場所にあることを確認する。
- コンポーネントテストは、個々のコンポーネント（例：ライブラリクラス、ウェブコンポーネント）で実行されるため、パイプラインのビルドステップの一部になる。パイプラインの品質ゲートとして機能するため、継続的インテグレーションパイプラインの重要な部分である。
- コンポーネント統合テストは、低レベルコンポーネントまたはテスト対象システム (**SUT**) のテストである場合、継続的インテグレーションパイプラインの一部として組み込むことができる。そのような場合、これらのテストとコンポーネントテストは一緒に実行される。
- システムテストは多くの場合、継続的デプロイメントパイプラインに統合でき、デリバリーされる **SUT** の最終的な品質ゲートとして機能する。
- 異なるシステムコンポーネント間のシステム統合テストは、多くの場合、品質ゲートとして継続的デリバリーパイプラインの一部になる。これらのシステム統合テストにより、別々に開発されたシステムコンポーネントが連携して動作することを確認できる。

多くの最新の継続的インテグレーションシステムは、継続的デリバリーパイプラインのビルドとデプロイメントのフェーズを区別している。そのような場合、コンポーネントテストとコンポーネント統合テストは最初のビルドフェーズの一部である。この最初のフェーズが成功した場合（つまり、ビルドとテストと一緒に完了した場合）、コンポーネントまたは **SUT** がデプロイされる。

システム統合テスト、システムテスト、受け入れテストの場合、それらをパイプラインに統合する主な 2 つのアプローチがある：

1. テストケースはコンポーネントデプロイメント後のデプロイメントフェーズの一部として実行される。これは、テスト結果に基づいてデプロイメントが失敗し、ロールバックされる可能性があるため有効である。ただし、この場合、テストを再実行する必要がある場合は再デプロイメントが必要である。
2. テストケースは、成功したデプロイメントによって起動される別のパイプラインとして実行される。これは、デプロイメントごとに異なるテストスイートやさまざまなテスト自動化コードを実行することが想定される場合に有効である。この場合、テストは品質ゲートとして機能しない。そのため、失敗したデプロイメントをロールバックするには、通常、別の手動対応が必要になる。
この場合、デプロイメント確認用として少数の単純な自動テストスクリプトを使用し、**SUT** が正しくデプロイされたことを確認できる。ただし、これらの自動テストスクリプトは広範な機能適合性を検証するものではない。

パイプラインは、他のテスト自動化の目的にも使用できる。例えば：

- 定期的に異なるテストスイートを実行する：リグレッションテストスイートは毎晩実行できる（つまり、ナイトリーリグレッション）。特に長時間実行されるテストスイートの場合、チームは翌朝、**SUT** の品質状態を把握できる。
- 非機能テストの実行：継続的デプロイメントパイプラインの一部として、または別途、性能効率性などシステムの特定の非機能品質特性を定期的に監視するために実行される。

5.1.2 テストウェアの構成管理の説明

構成管理はテスト自動化の不可欠な要素である。自動化は、多くの場合、複数のテスト環境やテスト対象システム（**SUT**）の異なるバージョンで実行されるためである。

テスト自動化における構成管理には以下が含まれる：

- テスト環境設定
- テストデータ
- テストスイート/テストケース

テスト環境設定

開発パイプラインで使用される各テスト環境は、さまざまな **URL** や認証情報など、異なる設定を持つ場合がある。テスト環境設定は通常テストウェアとともに格納される。ただし、テスト自動化が複数のプロジェクトまたは同一プロジェクトの複数のテスト自動化フレームワーク（**TAF**）で使用される場合、テスト環境設定は共通のコアライブラリまたは共有リポジトリの一部として格納される。

テストデータ

テストデータもテスト環境または **SUT** のリリースと機能セットに固有のものになる場合がある。テスト環境設定と同様に、テストデータは通常、小規模な **TAF** とともに格納されるが、テストデータ管理システムも使用できる。

テストスイート/テストケース

一般的には、スモークテストやリグレッションテストなど、目的に基づいて複数のテストスイートを構成する。これらのテストスイートは、多くの場合、異なるパイプラインやテスト環境を利用して、別々のテストレベルで実行される。

SUT の各リリースは、特定のリリースの品質を評価するテストケースとテストスイートを含む機能セットを決定する。テストウェアにはこれを処理するための異なる選択肢がある：

- 機能トグル設定（実行時に機能の有効／無効を切り替える仕組み）は、各リリースまたはテスト環境ごとに定義できる。各機能をテストするためのテストケースとテストスイートが用意される。機能トグルはテストウェアでを使用して、特定のリリース/テスト環境でどのテストスイートを実行するかを特定できる。
- テストウェアは、同じリリースバージョンの **SUT** とともにリリースすることもできる。このようにして、**SUT** バージョンと、それをテストするテストウェアとを完全に一致させることができる。このようなリリースは通常、タグまたはブランチを使用した構成管理システムを使用して実装される。

5.1.3 API インフラストラクチャに対するテスト自動化の依存関係の説明

API テスト自動化を実施する際、適切な戦略を構築するために依存関係に関する以下の情報を把握しておくことが重要である：

- **API 接続**：自動テスト可能なビジネスロジックと API 間の関係を理解する
- **API ドキュメント**：すべての関連情報（例：パラメーター、ヘッダー、要求-応答オブジェクトの異なる種類）を備えた、テスト自動化のベースラインとして機能する

統合された自動 API テストは、開発者またはテスト自動化エンジニア（TAE）のいずれかが実施できる。ただし、シフトレフトの考え方にに基づき、異なるテストレベルで役割を分担してテストを実施することが推奨される。ISTQB CTFL シラバスでは、コンポーネント統合テストとシステム統合テストが言及されており、契約テストと呼ばれるベストプラクティスによって拡張できる。

契約テスト

契約テストは、サービス同士が通信できること、およびサービス間で共有されるデータが、定義された一連のルールに従っていることを検証する統合テストの一種である。契約テストを使用することで、2つの独立したシステム（例：2つのマイクロサービス）が相互に通信可能であることを保証できる。スキーマ検証に加え、両者が将来変更されても許容される相互作用について合意しておくことが求められる。各サービス間でやり取りされる相互作用を記録して契約として保存し、双方がそれに準拠しているかを検証するために利用される。このテストタイプの主な利点の1つは、基盤となるサービスに起因する欠陥を SDLC の早い段階で発見でき、その原因を特定しやすくなることである。

契約テストのコンシューマー駆動型アプローチでは、コンシューマーが、プロバイダーがそのコンシューマーからのリクエストにどのように応答すべきか、という期待を定義する。契約テストのプロバイダー駆動型アプローチでは、プロバイダーはそのサービスがどのように動作するかを示す契約を作成する。

6 テスト自動化のレポートイングとメトリクス — 150 分 (K4)

キーワード

測定、メトリクス、テストログ、テスト進捗レポート、テストステップ

第 6 章の学習の目的：

6.1 テスト自動化データの収集・分析・レポートイング

TAE-6.1.1 (K3) テスト自動化ソリューション (TAS) およびテスト対象システム (SUT) からのデータ収集方法を適用する

TAE-6.1.2 (K4) テスト結果の理解を深めるために、テスト自動化ソリューション (TAS) およびテスト対象システム (SUT) からのデータを分析する

TAE-6.1.3 (K2) テスト進捗レポートの作成と公開方法を説明する

6.1 テスト自動化データの収集・分析・レポーティング

6.1.1 テスト自動化ソリューション (TAS) およびテスト対象システム (SUT) からのデータ収集方法の適用

データは以下のソースから収集できる：

- SUT ログ
 - ウェブ/モバイル UI
 - API
 - アプリケーション
 - ウェブサーバー
 - データベースサーバー
- TAF ログ (監査証拠の提供のため)
- ビルドログ
- デプロイメントログ
- 本番環境のデータモニタリングするための本番ログ (ISTQB CT-PT シラバス、2.3 項参照)
 - トレンド分析を実施するための本番環境でのパフォーマンス監視
 - 性能テスト環境での性能効率性テストログ (例：負荷テスト、ストレステスト、スパイクテスト)
- スクリーンショットと画面録画 (自動化ツールの標準またはサードパーティ製)

TAS は自動テストウェアを中核としているため、その利用状況に関する情報を記録できるように拡張できる。基盤となるテストウェアにこのような機能を追加すれば、すべての高レベル自動テストスクリプトで活用できる。例えば、テスト実行の開始時刻と終了時刻を記録するように基盤となるテストウェアを拡張すれば、すべてのテストに適用できる。

測定とテストレポート生成を支援するテスト自動化の機能

多くのテストツールのスクリプト言語には、測定や、個々のテストおよびテストスイート全体について、実行前・実行中・実行後に情報を記録・ログ出力する機能があり、レポーティングを支援する。

一連のテスト実行それぞれのテストレポーティングには、以前のテスト実行のテスト結果を考慮して、テスト成功率の変化などのトレンドを強調できる分析機能が必要である。

テスト自動化は通常、テスト実行とテスト検証の両方を自動化する必要がある。テスト検証は実際の結果の特定の要素を期待される結果と比較することで達成される。この比較はアサーションを使用するテストツールによって行われるのが最善である。この比較結果として報告される情報のレベルを考慮する必要がある。テストステータスが正しく決定されること (つまり、合格または不合格) が重要である。不合格ステータスの場合、故障の原因に関する詳細情報が必要になる (例：スクリーンショット)。

テストの実際の結果と期待結果との差異は、常に明確とは限らない。ツールによっては、日時など想定内の差異を除外し、予期しない差異に絞って比較結果を強調表示できるため、分析に役立つ。

テストロギング

テストログは、TAS と SUT 内の潜在的な欠陥を分析するためによく使用される情報源である。以下の節では、TAS と SUT で分類されたテストロギングの例を示す。

TAS のロギング

コンテキストにより、TAF またはテスト実行が以下を含む情報のロギングを担当するかどうかが決まる：

- 開始時刻と終了時刻を含む、現在テスト実行中のテストケース
- テスト実行のステータス。故障はテストログで簡単に識別できるが、TAF もこの情報を持ち、ダッシュボードを通じて報告する必要がある。テスト実行のステータスは、合格、不合格、または TAS 故障のいずれかである。ステータスが不確定な場合もあり、組織はそれらに対して明確で一貫した定義を設けることが重要である。TAS 故障という分類は、欠陥が SUT 側にならない状況に適用される。
- タイミング情報を含む、テストログの低レベルの詳細（例：重要なテストステップのロギング）
- テストケースにおいて、サードパーティツールを用いて特定された SUT に関する動的情報（例：メモリリーク）。故障が検出されたときに実行されたテストスイートとともに、実際の結果と故障をログに記録する必要がある。
- 信頼性テストやストレステストのように多数のテストサイクルが実行される場合、テストケースが何回実行されたかを簡単に判断するためにカウンターをログに記録する必要がある。
- テストケースにランダムな要素がある場合（例：ランダムなパラメーターや状態遷移テストのランダムなテストステップ）、ランダムな数値/選択をログに記録する必要がある。
- 同じテストステップと同じタイミングでテストを再実行できるように、テストケースが実行するすべてのアクションをログに記録し、テストログまたはその一部を再現可能な形式で保持する必要がある。これは、特定の故障を再現し、追加情報をキャプチャするのに役立つ。テストケースのアクション情報は SUT によってもログに記録され、顧客が特定した故障を再現する際に使用できる。顧客がテストスイートを実行すると、テストログ情報がキャプチャされ、欠陥のトラブルシューティング時に開発チームが再生できる。
- テスト実行中に保存されたスクリーンショットは、根本原因分析に活用できる。
- テストスイートが故障を引き起こすたびに、TAS は欠陥の分析に必要なすべての情報が利用可能な状態で保存されていること、また可能であればテストの継続に関する情報も記録する必要がある。TAS は関連するクラッシュダンプとスタックトレースを保存する必要がある。また、上書きされる可能性のあるテストログ（例：SUT 上のテストログによく使用されるサイクリックバッファ）は、後の分析に使用できる場所に格納する必要がある。
- カラーを使用することで、異なる種類のテストログ情報を区別するのに役立つ（例：欠陥は赤、進捗情報は緑）。

SUT のロギング

SUT ログとテスト自動化の結果を照合することで、SUT および TAS 内の欠陥の根本原因の特定を支援する。

- SUT で欠陥が特定された場合、日時スタンプ、欠陥のソースロケーション、エラーメッセージなど、欠陥の分析に必要なすべての情報をログに記録する必要がある。
- システムの起動時に、異なるソフトウェア/ファームウェアバージョン、SUT の設定、オペレーティングシステムの構成などからなる構成情報をファイルにログに記録する必要がある。
- テスト自動化を使用することで、SUT ログは簡単に検索可能になる。TAS のテストログで特定された故障は、SUT のテストログでも容易に特定できる必要があり、その逆も同様である。追加のツールの有無に関わらず、さまざまなテストログをタイムスタンプで同期することで、故障が報告されたときに何が起こったかの相関分析が容易になる。

他のサードパーティツールとの統合（例：スプレッドシート、XML、ドキュメント、データベース、レポートツール）

自動テストケースの実行結果を、他のツールで追跡とレポーティングに活用する場合（例：トレーサビリティ情報の更新）、サードパーティツールに適した形式で情報を提供することが可能である。これは多くの場合、既存のテストツール機能（例：テストレポーティングのエクスポート形式）、または他のソフトウェアと一致する形式で出力するカスタマイズされたレポーティングを作成することで達成される。

テスト結果の可視化

テスト結果はチャートを用いて可視化できる。テスト実行/テスト自動化の全体的なステータスを示すために信号機などの色付きアイコンを使用することが推奨される。報告された情報に基づいて意思決定ができるようになる。経営層は特にテスト結果を確認し意思決定に役立てるために視覚的なサマリーに関心がある。さらに詳しい情報が必要な場合は、ドリルダウンすることで確認できる。

6.1.2 テスト結果の理解を深めるための、テスト自動化ソリューション（TAS）およびテスト対象システム（SUT）からのデータ分析

テスト実行後は、**SUT** と **TAS** の両方における潜在的な故障を特定するため、テスト結果を分析することが重要である。そのような分析では、**TAS** から収集されたデータが主要であり、**SUT** から収集されたデータは補助的な情報源として使われる。

- 適切なテスト自動化のサイジングをサポートするためのテスト環境データの分析（例：クラウド内）
 - クラスターとリソース（例：CPU、RAM）
 - シングルブラウザー対マルチブラウザー（クロスブラウザー）テスト実行
- 以前のテスト実行のテスト結果を比較する
- ソフトウェアの使用状況を監視するために、ウェブログをどのように活用するかを決定する

テスト実行の故障を分析する必要がある。潜在的な問題が存在するためである：

1. 同じ故障が以前のテスト実行で発生したかどうかを確認する。これは **SUT** または **TAS** の既知の欠陥である可能性がある。**TAS** はテストケースのテスト結果の履歴をログに記録するように構築でき、分析を支援する。
2. その欠陥が既知でない場合は、テストケースと、そのテスト対象箇所を特定する。テストケース自体から内容が読み取れる場合もあり、そうでなければテストケースはその ID に基づいてテスト管理システムで特定できる。その ID はテスト実行とともにログに記録される。
3. テストケースのどのテストステップで故障が発生したかを特定する。**TAS** はこれをログに記録する。
4. **SUT** の状態に関するテストログを分析して、期待される結果と一致するかどうかを確認する。スクリーンショット、API およびネットワークログ、または **SUT** の状態を示すログを活用する。
5. **SUT** の状態が期待されたものと異なる場合、欠陥管理システムに欠陥を記録する。必要なすべての欠陥情報と、欠陥であることを正当化するログを含めることを確認する。

故障が発生した場合でも、**SUT** の実際の結果と期待結果が一致していることがある。この場合、**TAS** におそらく欠陥があり、修正が必要であるか、見えない不一致が存在する。

もう 1 つ起こり得る状況として、テスト実行中にテスト環境が利用できない場合、または部分的にしか利用できない場合に発生する可能性がある。この場合、すべてのテストケースが同じ欠陥で失敗するか、システムの一部がダウンしている場合は一見本当の故障で失敗する可能性がある。このような

欠陥の根本原因を特定するために、SUT ログを分析することが有効である。テスト実行時にテスト環境の停止があったかどうかを示される。

SUT がユーザー操作 (UI セッションまたは API 呼び出し) の監査ログを実装している場合、テスト結果の分析に役立つ。通常、ユニークな ID が各リクエストに付与され、システム内の後続の呼び出しと統合に同じ ID が使用される。このようにして、リクエストや関連処理に付与されたユニークな ID を知ることで、システムの動作を観察・追跡できる。

このユニークな ID は通常、相関 ID またはトレース ID と呼ばれる。これはテスト結果の分析を助けるために TAS でログに記録できる。

6.1.3 テスト進捗レポートの作成と公開方法の説明

テストログは、テストケースおよび/またはテストスイートのテストステップ、実行されたアクション、期待される応答に関する詳細情報を提供する。ただし、テストログだけでは全体的なテスト結果を把握することは困難である。そのために、テストレポート生成機能が必要である。テストスイートの実行後、簡潔なテスト進捗レポートを作成して公開する必要がある。レポート生成ツールを使用できる。

テスト進捗レポートの内容

テスト進捗レポートには、各ステークホルダーに適した形式で、テスト結果、SUT 情報、テストが実行されたテスト環境のドキュメントが含まれている必要がある。

どのテストが失敗したか、その失敗の理由を示すことが必要である。トラブルシューティングを容易にするために、テスト実行の履歴と、誰が報告したか (一般的には作成者または最終更新者) を把握しておくことが重要である。責任者は故障の原因を調査し、それに関連する欠陥を報告し、欠陥の修正をフォローアップし、修正が正しく実装されたことをテストする必要がある。

テストレポート生成は TAF コンポーネントの故障を診断するためにも使用される。

テストレポートの公開

テストレポートは、関連するすべてのステークホルダーに公開する必要がある。ウェブサイト、クラウド、またはオンプレミス環境にアップロードしたり、メーリングリストに送信したり、テスト管理ツールなど他のツールに登録したりできる。これにより、各人がメールやチャットボットの投稿メッセージで通知を受け取るように設定していれば、レポートがレビュー・分析されやすくなる。

オプションとして、SUT の問題のある部分を特定し、テストレポートの履歴を保持することで、頻繁なリグレーションが発生するテストケースやテストスイートに関する統計を収集し、トレンド分析に活用することができる。

レポートするステークホルダーには以下が含まれる：

- 経営層ステークホルダー
 - 典型的な役割：ソリューションまたはエンタープライズアーキテクト、プロジェクト/デリバリーマネージャー、プログラママネージャー、テストマネージャー、またはテストディレクター
- 運用ステークホルダー
 - 典型的な役割：プロダクトオーナー/マネージャー、ビジネス担当者、またはビジネスアナリスト

- 技術系ステークホルダー

- 典型的な役割：チームリーダー、スクラムマスター、ウェブ管理者、データベース開発者/管理者、テストリーダー、TAE、テスト担当者、または開発者

テストレポートは受信者によって内容や詳細が異なる場合がある。技術系ステークホルダーは低レベルの詳細に関心がある。経営層は前回のテスト実行以降に追加されたテストケースの数、合格/不合格比率の変化、TAS と SUT の信頼性などのトレンドに焦点を当てる。運用ステークホルダーは通常、プロダクト使用に関連するメトリクスをより重視する。

ダッシュボードの作成

最新のレポートングツールは、ダッシュボード、カラフルなチャート、詳細なログコレクション、自動化されたテストログ分析を通じていくつかのレポートングオプションを提供する。市場には多くの利用可能なツールがある。

これらのツールは、パイプライン実行テストログ、プロジェクト管理ツール、コードリポジトリなどの情報源からのデータ集約をサポートする。これらのツールが提供するデータの可視化は、ステークホルダーがトレンドを確認し、それに応じて意思決定するのに役立つ。これらのトレンドには、欠陥の集中傾向、特定のテスト環境への欠陥の広がり増加/減少、SUT のパフォーマンス低下、ビルドの信頼性などが含まれる。

テストログの人工知能/機械学習分析

近年、一部のテスト自動化ツールは機械学習（ML）アルゴリズムを組み込んでいる。また、それに基づいて構築されているものもある。テストログ内の大量のデータを自動分析することで、TAE が壊れたロケータを見つけ、テスト失敗の理由を特定し（SUT の欠陥か TAS の欠陥かを見極め）、テストレポートングのために共通の欠陥をグループ化するのに費やす時間を削減するのに役立つ（ISTQB CT-AI シラバス参照）。

7 テスト自動化ソリューションの検証 — 135 分 (K3)

キーワード

静的解析

第 7 章の学習の目的：

7.1 テスト自動化インフラストラクチャの検証

TAE-7.1.1 (K3) テストツールのセットアップを含むテスト自動化環境の検証を計画する

TAE-7.1.2 (K2) 特定の自動テストスクリプトおよびテストスイートの正しい動作を説明する

TAE-7.1.3 (K2) テスト自動化が予期せぬ結果を生成する箇所を特定する

TAE-7.1.4 (K2) 静的解析がテスト自動化コード品質を向上させる方法を説明する

7.1 テスト自動化インフラストラクチャの検証

7.1.1 テストツールのセットアップを含むテスト自動化環境の検証計画

テスト自動化環境と、テスト自動化ソリューション (TAS) を構成するその他すべてのコンポーネントが期待どおりに機能するかどうかは、あらかじめ検証しておく必要がある。こうした確認は、例えばテスト自動化を開始する前に実施する。テスト自動化環境のコンポーネントを検証するために、いくつかのステップを踏むことができる。それぞれについて以下で詳しく説明する。

テストツールのインストール、セットアップ、設定、カスタマイズ

TAS は多くのコンポーネントで構成されている。信頼性が高く、繰り返し可能な動作を保証するために、それらが確実に機能する必要がある。TAS の中核には、実行可能コンポーネント、対応する関数ライブラリ、サポート用データ、設定ファイルがある。TAS を設定する方法には、自動インストールスクリプトの利用から、対応するフォルダーへのファイルの手動配置まで、さまざまなものがある。テストツールは、オペレーティングシステムや他のソフトウェアと同様に、定期的にサービスパックが提供されたり、特定のテスト対象システム (SUT) 環境との互換性を確保するためのオプションまたは必須のアドオンが必要になったりする場合がある。

自動インストールまたはリポジトリからのコピーには利点がある。これにより、それが適切な場合には、異なる SUT に対するテストが同じバージョン・同じ設定の TAS で実行されたことを保証できる。TAS のアップグレードはリポジトリを通じて行うことができる。リポジトリの使用と TAS の新しいバージョンへのアップグレードプロセスは、標準的な開発ツールと同じである必要がある。

テスト環境のセットアップ/ティアダウンの再現性

TAS は、さまざまなシステムやサーバー上に実装され、CI/CD パイプラインを支援する。TAS が各テスト環境で正しく機能することを確保するには、特定のテスト環境に対して TAS をロード・アンロードするための体系的なアプローチが必要である。TAS の構築と再構築が、複数のテスト環境の内外で同じように動作することを確認できれば、再現性が確保されたことになる。TAS のコンポーネントを構成管理下に置くことで、特定の設定を確実に再現できるようになる。これが達成されると、TAS を構成するさまざまなコンポーネントを文書化することで、SUT 環境が変更された場合に、TAS のどの側面が影響を受けるか、または変更が必要かについての知識を得ることができる。

内部および外部システム/インターフェースとの接続性

特定の SUT 環境に TAS をインストールした後は、SUT を使用する前に、内部システム、外部システム、および各種インターフェースに接続可能であることを確認する一連のチェックや事前条件を実施する必要がある。例えば、サーバーにログインし、テスト自動化ツールを起動し、そのツールが SUT にアクセスできることを確認し、設定内容を手動で確認し、システム間のテストロギングやテストレポート生成に必要な権限が適切に設定されていることを確認することは、有効なプラクティスである。テスト自動化の事前条件を確立することは、TAS が正しくインストールされ、設定されていることを確認するうえで不可欠である。

TAF コンポーネントテスト

あらゆるソフトウェア開発プロジェクトと同様に、TAF のコンポーネントは個別にテストおよび検証される必要がある。これには、機能テストおよび非機能テスト（例：性能効率性、リソース使用率）が含まれる場合がある。例えば、GUI システム上でオブジェクト検証を提供するコンポーネントは、オブジェクト検証機能が正しく動作することを確認するために、幅広いオブジェクトクラスに対してテストを行う必要がある。同様に、テストログおよびテストレポートは、テスト自動化の状態および SUT の動作に関する正確な情報を出力しなければならない。非機能テストの例としては、TAF の性能劣化の把握、メモリリークなどの欠陥を示す可能性があるシステムリソースの使用率の監視、TAF 内外のコンポーネント間における相互運用性の欠如の確認などが挙げられる。

7.1.2 特定の自動テストスクリプトおよびテストスイートの正しい動作の説明

自動テストスイートは、完全性、一貫性、正しい動作について検証される必要がある。自動テストスイートが常に利用可能であり、使用に適していることを確認するために、さまざまな種類の検証チェックを適用することができる。

自動テストスイートを検証するためのステップとして以下が含まれる：

- テストスイートの構成を確認する
- テスト自動化フレームワーク（TAF）の新機能に焦点を当てた新しいテストを検証する
- テストの再現性を考慮する
- 自動テストツールの干渉を考慮する

それぞれについて以下で詳しく説明する。

テストスイートの構成を確認する

完全性（例：テストケースすべてに期待される結果があり、テストデータが存在する）と、TAF とテスト対象システム（SUT）のバージョンが正しいかを確認する。

TAF の新機能に焦点を当てた新しいテストの検証

TAF の新機能がテストケースで初めて使用される場合、機能が正しく動作していることを確認するために、それを検証し、慎重に監視する必要がある。

テストの再現性の考慮

テストを繰り返す場合、テスト結果は常に同じである必要がある。信頼性の高いテスト結果を与えないテストケース（例：競合状態のため）はアクティブな自動テストスイートから外して、根本原因を特定するために別途分析する必要がある。さもないと、これらのテスト実行の同じ失敗を分析するために繰り返し時間が費やされる。

自動テストツールの干渉性の考慮

TAS は多くの場合、SUT と密接に結合される。これは、相互作用のレベルに関する互換性を高めるための意図的な設計である。しかし、この密接な統合は好ましくない結果を招くこともある。例えば、TAS が SUT 環境内に配置されている場合、SUT の機能が手動でテストする場合とは異なる可能性があり、パフォーマンスにも影響を与えることがある。

干渉のレベルが高い場合、本番環境では明らかでない故障がテスト中に現れる可能性がある。これが自動テストで故障を引き起こす場合、TAS への信頼は大幅に低下する可能性がある。開発者は、テ

ト自動化によって特定された故障を分析のために、可能であれば手動で再現することを要求する場合がある。

7.1.3 テスト自動化が予期せぬ結果を生成する箇所の特定

テストスクリプトが予期せず失敗または合格する場合、根本原因分析を行う必要がある。これには、テストスクリプトのテストログ、パフォーマンスデータ、セットアップ、ティアダウンの検査が含まれる。

また、個別のテストを切り分けて実行することも有効である。断続的な故障は分析が困難である。欠陥はテストケース、テスト対象システム (SUT)、TAF、ハードウェア、またはネットワークにある可能性がある。システムリソースの監視により、根本原因の手がかりが得られる場合がある。テストケース、SUT、TAF のテストログファイル分析が欠陥の根本原因の特定に役立つ。デバッグも必要になる場合がある。根本原因の特定を支援するには、テストアナリスト、ビジネスアナリスト、開発者、またはシステムエンジニアからのサポートが必要な場合がある。

すべてのアサーションが適切に配置されているか確認することも重要である。欠落しているアサーションは、不確定なテスト結果につながる可能性がある。

7.1.4 静的解析がテスト自動化コード品質を向上させる方法の説明

静的コード解析はプログラムコードの脆弱性と欠陥を見つけるのに役立つ。これにはテスト対象システム (SUT) またはテスト自動化フレームワーク (TAF) が含まれる場合がある。

自動スキャンはコードを検査してリスクを軽減できる。これにより、SUT の欠陥を見直すことができ、コード品質標準が満たされ、適切に適用されていることを確認できる。これはまた、プロアクティブな欠陥検出技法とも見なされ、DevSecOps 実装（つまり、セキュリティに重点を置いた DevOps）で重要な役割を果たす。これらのスキャンはパイプラインを通じて SDLC の早い段階で発生し、開発チームに即座のフィードバックを提供する。

欠陥に関する出力は通常、クリティカル、高、中、低の重要度で分類されるため、開発チームはどの欠陥を修正するかを優先順位付けできる。特定の静的解析ツールには、見つかった欠陥に対処するためのコード修正候補を提供する機能もある。問題のあるコード行を開発チームに提示し、開発者が実装するための可能な解決策を提供する。さらに、これらのツールは TAE を支援する。品質を測定し、コードにコメントを付けるべき箇所を提案し、最適化されたリソース処理のためのコード設計を改善し（例：try/catch ブロックの利用、より良いループ構造）、不良なライブラリ呼び出しを削除する。

テスト自動化ツールではプログラミング言語を使用するため、不十分なテスト自動化コードが SDLC に導入されるリスクがある。簡単な例として、テストを自動化する際にはユーザー名とパスワードを扱うのが一般的である。TAE が、1 つまたは複数のテストスクリプト内に平文のパスワードを誤って含めてしまう可能性もある。静的解析ツールはテスト自動化コードにも有益である。例えば、コード内の平文パスワードなどのセキュリティ違反を分析するために使用できる。静的解析ツールは、テスト自動化ソフトウェアで使用されるものを含む多くのプログラミング言語をサポートする。したがって、TAE がコードスキャンのベストプラクティスをテスト自動化コードにも適用することが不可欠である。テスト自動化コードは必ずしも全体のソフトウェアと一緒にデプロイされるわけではないが、パスワードを含む自動テストスクリプトが発見された場合には明確な潜在的脆弱性となる（詳しくは ISTQB CT-SEC シラバスを参照）。

8 継続的改善 — 210 分 (K4)

キーワード

スキーマ検証、テストヒストグラム

第 8 章の学習の目的：

8.1 テスト自動化の継続的改善の機会

TAE-8.1.1 (K3) データ収集と分析によるテストケース改善の機会を発見する

TAE-8.1.2 (K4) デプロイされたテスト自動化ソリューションの技術的側面を分析し、改善提案を提示する

TAE-8.1.3 (K3) テスト対象システム (SUT) の更新に合わせて自動テストウェアを再構築する

TAE-8.1.4 (K2) テスト自動化ツール活用を整理する

8.1 テスト自動化の継続的改善の機会

8.1.1 データ収集と分析によるテストケース改善の機会の発見

以下に説明するアプローチでさまざまな種類のデータを考慮することで、データ収集と分析を改善できる。

テストヒストグラム

テストヒストグラムとして表現された視覚的なテストデータレポートは、テストケースのデータ傾向に関する改善の手がかりを与える。多くの CI/CD ツールやテストレポートツールには、異なるテスト結果と、それに対応するテストデータ（例：例外ログ、エラーメッセージ、スクリーンショット）を表示する機能がある。これにより、テスト自動化エンジニア（TAE）は改善可能な領域を判断できる。テストヒストグラムを利用することで、TAE は壊れやすいテストケースを特定し、追加改善のためにリファクタリングするか、実装そのものを見直すかを判断できる。

人工知能

テストとテスト自動化をサポートするために人工知能（AI）を活用することも、1 つの改善機会となる。例えば、UI テストケースでは、データには入力として処理できる UI ロケータ値も含まれる。最近の最先端ツールは、使用されている特定のロケータが変更されたかどうかを検出する機能を持っている。機械学習（ML）と画像認識に基づいて、新しいセクターを特定し、自己修復アルゴリズムを使用してテストケースを修正し、変更されたロケータをテストレポートに含めることができる。これにより、バージョン管理の変更とコード保守などのフォローアップ手順を加速できる。

スキーマ検証

スキーマ検証は API データ分析（例：ターゲットエンドポイントから取得したプロパティ）およびデータベース分析（例：許容するデータ型や値の範囲などのソフトウェアフィールドの検証ルール）に適用できる。

スキーマ検証を使用することで、TAS はレスポンスが実際のビジネス仕様と一致するかどうかをチェックできる。この種のチェックは、必須のレスポンス要素がサービスレスポンスに存在するかどうか、およびそのオブジェクト型がスキーマに定義されているものと一致するかどうかを判断するために使用できる。スキーマが壊れている場合、ソリューションは実際の検証結果を返すため、TAE が問題の根本原因を特定するのに役立つ。

例：API にはレスポンスに 6 つの必須要素があり、すべて文字列でなければならない。スキーマ検証ツールを使用すると、これらの型が文字列であるかどうか、またその値が null でないかどうかを個別にアサーションで記述する必要がない。スキーマ検証はこれらのチェックを処理し、実装されたテスト自動化コードをより短くし、バックエンドサービスの欠陥検出の効率を高める。

8.1.2 デプロイされたテスト自動化ソリューションの技術的側面の分析と改善提案の提示

テスト自動化ソリューション（TAS）をテスト対象システム（SUT）と同期させるために必要な継続的な保守タスクに加えて、TAS を改善する多くの機会がある。これらの改善は、より高い効率性（例：手動介入のさらなる削減）、使いやすさの向上、機能の拡充、テストへのサポート改善など、さまざまな利点を達成するために行われる。TAS の改善方法の決定は、どの機能がプロジェクトに最も価値を追加するかによって影響される。

改善を検討できる TAS の特定の領域には、スクリプティング、テスト実行、検証、TAA、TAF、セットアップとティアダウン、ドキュメント、TAS 機能、TAS の更新とアップグレードが含まれる。これらについて以下で詳しく説明する。

スクリプティング

スクリプティング技法は線形スクリプティングからデータ駆動テストアプローチ、そしてより洗練されたキーワード駆動テストアプローチまでさまざま、3.1.4 項に記載されている。すべての新しい自動テストに対して現在の TAS スクリプティング技法をアップグレードすることが適切な場合がある。この技法は既存のすべての自動テスト、または少なくとも最も多くの保守作業を含むものにも後付けで適用することができる。

テストスクリプトの実装に焦点を当てた、別の TAS 改善領域がある。例えば：

- テストスクリプト/テストケース/テストステップの重複部分を評価し、自動テストを統合する。同様のアクションシーケンスを含むテストケースは、これらのテストステップを複数回実装すべきではない。これらのテストステップは関数にして、再利用できるようにライブラリに追加する必要がある。これらのライブラリ関数は、異なるテストケースで使用できる。これにより、テストウェアの保守性が向上する。テストステップが同一ではないが類似している場合、パラメーター化が必要になる場合がある。補足：これはキーワード駆動テスト (keyword-driven testing) での典型的なアプローチである。
- TAS と SUT の故障回復プロセスを確立する。テストスイートの実行中に故障が発生した場合、TAS は次の可能なテストに進むためにこの状態から回復できる必要がある。SUT で故障が発生した場合、TAS は実行可能で実用的な場合に SUT 上で必要な回復アクション（例：SUT の再起動）を実行する必要がある。
- 待機メカニズムを評価し、最適なタイプが使用されていることを確認する。一般的な待機メカニズムが 3 種類ある：
 - ハードコードされた待機（つまり、特定のミリ秒数待機する）は、ソフトウェアの応答時間の予測が不可能であるため、多くのテスト自動化欠陥の根本原因となる可能性がある。
 - ポーリングによる動的待機（例：特定の状態変化またはアクションが行われたことを確認する）は、より柔軟であり効率的である：
 - ・ TAS は必要な時間だけ待機し、テスト時間が無駄にならない
 - ・ プロセスが予想より長くかかる場合、ポーリングは条件が真になるまで待機する。欠陥がある場合にテストが永遠に待機しないように、タイムアウトメカニズムを含めることを忘れないこと。
 - さらに良い方法は、SUT のイベント機構を登録することである。これは他の 2 つのオプションよりもはるかに信頼性が高いが、テストスクリプティング言語がイベントの登録をサポートする必要がある、SUT はこれらのイベントを TAS に提供する必要がある。欠陥がある場合にテストが永遠に待機しないように、タイムアウトメカニズムも必要である。

テスト実行

自動化されたリグレーションテストスイートが、テスト実行に時間がかかりすぎるために完了しない場合、可能であれば異なるテスト環境で並行してテストすることが必要な場合がある。テストに高価

なシステムが使用されている場合、すべてのテストを単一のターゲットシステムで実施する必要があるという制約がある可能性がある。リグレッションテストスイートを複数の部分に分割し、それぞれが定義された期間内（例：1 晩）に実行されるようにする必要がある場合がある。テスト自動化のカバレッジを分析すると、重複しているテストが発見される場合がある。この重複を削除することでテスト実行時間を削減し、さらなる効率を生み出すことができる。CI/CD の場合、テスト実行時間を最適化するためにバッチジョブを並行して実行することが良いプラクティスである。また、手動介入を減らし、開発プロセスを加速するために、例えば毎朝など特定の時間に異なるパイプラインを実行するように自動化されたバッチジョブをスケジュールすることも良いプラクティスである。

検証

新しい検証関数を作成する前に、すべての自動テストで使用するための標準的な検証方法のセットを採用することが推奨される。これにより、複数のテストにわたる検証アクションの再実装を避けることができる。検証方法が同一ではなく類似している場合、パラメーター化を使用することで、関数を複数の種類のオブジェクトで使えるようになる。

TAA

SUT のテスト可能性の改善をサポートするために TAA を変更することが必要な場合がある。これらの変更は SUT のアーキテクチャおよび/または TAS の TAA で行われる場合がある。これはテスト自動化の大幅な改善をもたらすことができるが、SUT または TAS への大きな変更と投資が必要になる場合がある。例えば、SUT がテスト用の API を提供するように変更される場合、TAS もそれに応じてリファクタリングされる必要がある。このような機能を SDLC の後期に追加することはかなり費用がかかる可能性がある。テスト自動化の開始時と SUT の SDLC の初期フェーズでこれを検討することが非常に望ましい。

TAF

TAF で使用されるコアライブラリの新しいバージョンがしばしば出てくる。これらが大きな更新である場合があり、最新バージョンは TAF の依存関係リストにすぐには参照できない場合がある。これは、これらのライブラリを使用している多くのチームのテストを壊してしまう可能性があるためである。したがって、最初にパイロットと影響分析を実施する方が望ましい。その後、採用計画を策定する。すべてのチームがコアライブラリ層のビルドファイルで依存関係を更新することで同時に新しいバージョンを採用するか、各チームがビジネスロジック層でいつ更新するかを個別に決定するかのどちらかの採用方法がある。最終的に、すべてのチームが新しいバージョンのコアライブラリを受け入れる準備ができたなら、コアライブラリ層で依存関係を更新できる（詳細は 3.1.3 項参照）。

セットアップとティアダウン

各テストスクリプトまたはテストスイートの前後に繰り返されるアクションと設定は、セットアップまたはティアダウンメソッドに移動する必要がある。このようにして、コードに影響する変更はすべて 1 か所で更新でき、保守作業が削減される。例えば、ウェブサービス呼び出しは UI テストの事前条件や事後条件（例：ユーザー登録、ユーザークリーンアップ、プロフィールセットアップ）を満たすために使用される。

ドキュメント

ここでいうドキュメントには、テスト自動化コードが何を行うか、どのように使用すべきかを示す説明資料、TAS のユーザードキュメント、TAS が生成するテストレポートやテストログなど、テスト自動化に関するあらゆる文書が含まれる。

TAS 機能

TAS 機能と関数を追加する。例えば、詳細なテストレポート、テストログ、他のシステムへの統合などである。なお、使用される新機能のみを追加するべきである。未使用の機能を追加すると複雑さが増し、信頼性と保守性が低下する。

TAF の更新とアップグレード

TAF を新しいバージョンに更新またはアップグレードすることで、テストケースで使用できる新しい関数が利用可能になったり、故障が修正されたりする場合がある。ただしリスクもあり、既存のテストツールをアップグレードするか新しいものを導入することで TAF を更新することが、既存のテストケースに悪影響を与える可能性がある。新しいバージョンのテストツールをロールアウトする前に、サンプルテストを実行してテストツールの最新バージョンをテストすることが推奨される。サンプルテストは、異なる SUT、異なるテストタイプ、適切で異なるテスト環境にて実行される代表的な自動テストである必要がある。

8.1.3 テスト対象システム (SUT) の更新に合わせた自動テストウェアの再構築

既存のテスト対象システム (SUT) に一連の変更が加えられた後には、TAF とコンポーネントライブラリを含む TAS への更新が必要になる。どんなに小さな変更であっても、TAS の信頼性とパフォーマンスに広範な悪影響を与える可能性がある。

テスト環境コンポーネントの変更の特定

必要な変更や改善内容を評価する。これらはテストウェア、カスタマイズされた関数ライブラリ、またはオペレーティングシステムの変更を伴うのか。いずれも TAS のパフォーマンスに影響する。全体的な目標は、自動テストが効率的に実行され続けることを保証することである。変更は、最小実行可能プロダクトの考え方に基いて段階的に進める必要がある。これにより、限られたテストスクリプトの実行を通じて TAS への影響を測定できる。副作用がないことが確認できたら、本格的に変更を実装する。完全なリグレーション実行は、その変更が自動テストスクリプトに悪影響を与えていないことを検証するための最後のステップである。これらのリグレーションテスト実行中に故障が見つかる場合がある。こうした故障の根本原因を特定すること（例：テストレポート、テストログ、テストデータ分析を通じて）は、改善活動によって新たな問題が発生していないことを確認する手段となる。

コア TAS 関数ライブラリの効率と効果の向上

TAS が成熟するにつれて、タスクをより効率的に実行する新しい方法が発見される。これらの新しい技法（例：関数のコードの最適化、新しいオペレーティングシステムライブラリの使用）は、現在のプロジェクトと将来のプロジェクトで使用されるコア関数ライブラリに組み込む必要がある。

同じコントロールタイプに作用する複数の関数を統合対象として特定する

自動テスト実行中に起こることの大部分は、GUI のコントロールの確認作業である。この確認作業は、コントロールに関する情報（例：表示/非表示、有効/無効、サイズと寸法、データ）を提供する。この情報を使用して、自動テストはドロップダウンリストからアイテムを選択し、フィールドにデータを入力し、フィールドから値を読み取ることができる。コントロールに作用してこの情報を引き出す

複数の関数がある。一部の関数には非常に特殊化されたものがあれば、汎用化されたものもある。例えば、ドロップダウンリストにのみ機能する特定の関数がある場合がある。または、関数にパラメータを指定することで複数の機能をする関数が存在することもある。したがって、TAE は複数の関数を統合し、少ない関数を使用して同じ結果を達成させ、保守を最小化することができる。

SUT の変更を収容するための TAA のリファクタリング

TAS のライフサイクルを通じて、SUT の変更に対応するために TAA の変更が必要になる。SUT が発展・成熟するにつれて、それを支える基盤である TAA も、必要な機能を継続的に提供できるように見直していく必要がある。機能を拡張する際には、後付けで実装するのではなく、TAA の観点から分析し、変更するよう注意を払う必要がある。これにより、新しい SUT 機能が追加され、それに伴って追加のテストスクリプトが必要になった場合でも、それらを支える互換性のあるコンポーネントが整備されることを確実にできる。

命名規則と標準化

変更が導入されるにつれて、新しいテスト自動化コードと関数ライブラリの命名規則は、既に定義された標準と一致している必要がある（詳細は 4.3.1 項参照）。

SUT 改訂/廃止のための既存テストスクリプトの評価

変更と改善のプロセスには、既存のテストスクリプトの使用状況と継続的な価値の評価が含まれる。例えば、特定のテストが複雑で実行に時間がかかる場合、それらをより小さなテストに分解することがより実用的で効率的になる可能性がある。実行頻度が低いまたはまったく実行されないテストを廃止対象にすることで、TAS の複雑さを軽減し、保守が必要なものの明確さを向上させる。

8.1.4 テスト自動化ツール活用のお機会の整理

実際のテスト以外にも、テスト自動化は以下のような特定のテストに限定されない活動を支援できる：

環境のセットアップと制御

特定のテストスクリプト（例：テストデータ作成）をセットアップメソッドで活用して、新しいテスト環境に異なるテストデータを作成できる。異なるデータ入力に基づいて複数のプロファイルを持つユーザーを作成する必要がある状況では、チームはウェブサービスエンドポイントを呼び出してこれらのユーザーを登録する自動テストスクリプトを活用できる。テストスクリプトはテストインフラストラクチャのセットアップを制御でき、処理後のクリーンアップにも活用できる。これにより時間が節約され、各新しいテスト環境に適切なユーザーが存在することが確保される。例として、異なるテストログや他のテストウェアをテスト環境から自動的に削除し、ハウスキーピング（テストデータ・環境のクリーンアップ）とテスト環境の利用をより効率的にすることができる。

データエイジング

テスト自動化はテスト環境のテストデータを操作するために使用できる。例えば、データベースでは、日付フィールドを確認・制御して、年の観点から最新の状態を保つことができる。

スクリーンショットと動画生成

最新の UI テスト自動化ツールには、特定の条件でスクリーンショットや動画を作成してそれらを保存する組み込み機能がある。これらのテストツールにより、チームはビジネスを支援して、ソフトウ

エアリリースのドキュメントやマーケティング目的のための実際の使用状況のスクリーンショットや動画を作成することができる。

9 参考文献

標準

テスト自動化の標準には以下が含まれるが、これに限らない：

IEEE（電気電子学会）による自動テストマークアップ言語（ATML）で構成される

- IEEE Std 1671.1 : テスト記述
- IEEE Std 1671.2 : 機器記述
- IEEE Std 1671.3 : UUT 記述
- IEEE Std 1671.4 : テスト設定記述
- IEEE Std 1671.5 : テストアダプター記述
- IEEE Std 1671.6 : テストステーション記述
- IEEE Std 1641 : 信号とテストの定義
- IEEE Std 1636.1 : テスト結果

ISO/IEC/IEEE 29119-5 (2016) ソフトウェアおよびシステムエンジニアリング — ソフトウェアテスト — 第 5 部：キーワード駆動テスト

ISO/IEC 30130:2016 (E) ソフトウェアエンジニアリング — ソフトウェアテストツールの能力

ETSI（欧州電気通信標準化機構）および ITU（国際電気通信連合）によるテスト記述・テスト制御記法（TTCN-3）で構成される

- ES 201 873-1: TTCN-3 コア言語
- ES 201 873-2: TTCN-3 表形式表現フォーマット（TFT）
- ES 201 873-3: TTCN-3 グラフィカル表現フォーマット（GFT）
- ES 201 873-4: TTCN-3 動作セマンティクス
- ES 201 873-5: TTCN-3 ランタイムインターフェース（TRI）
- ES 201 873-6: TTCN-3 制御インターフェース（TCI）
- ES 201 873-7: TTCN-3 での ASN.1 の使用
- ES 201 873-8: TTCN-3 での IDL の使用
- ES 201 873-9: TTCN-3 での XML の使用
- ES 201 873-10: TTCN-3 ドキュメント
- ES 202 781: 拡張機能：設定とデプロイメントサポート
- ES 202 782: 拡張機能：TTCN-3 パフォーマンスとリアルタイムテスト
- ES 202 784: 拡張機能：高度なパラメーター化
- ES 202 785: 拡張機能：振る舞い型
- ES 202 786: 拡張機能：連続信号を持つインターフェースのサポート

- ES 202 789:拡張機能：拡張 TRI

OMG（オブジェクト管理グループ）による UML テストプロファイル（UTP）は、以下のテスト仕様コンセプトを規定する

- テストアーキテクチャ
- テストデータ
- テスト動作
- テストロギング
- テスト管理

ISTQB®ドキュメント

識別子	参照
ISTQB-AL-TTA	ISTQB Certified Tester, Advanced Level Syllabus, Technical Test Analyst, Version 4.0, June 2021, available from [ISTQB-Web] 日本では「テスト技術者資格制度 Advanced Level シラバス 日本語版概要 Version 2012」として発行されている。
ISTQB-FL	ISTQB Certified Tester, Foundation Level Syllabus, Version 4.0, April 2023, available from [ISTQB-Web] 日本では「FL シラバス（CTFL）」として発行されている。
ISTQB-MBT	ISTQB Certified Tester, Model-Based Testing Syllabus, Version 1.0, October 2015, available from [ISTQB-Web] 日本では「FL シラバス モデルベースドテスト担当者（CT-MBT）」として発行されている。
ISTQB-PT	ISTQB Certified Tester, Performance Testing Syllabus, December 2018, available from [ISTQB-Web] 日本では「FL シラバス 性能テスト担当者（CT-PT）」として発行されている。
ISTQB-SEC	ISTQB Certified Tester, Security Tester Syllabus, Version 1.0, March 2016, available from [ISTQB-Web]
ISTQB-TAS	ISTQB Certified Tester, Test Automation Strategy Syllabus, February 2024, available from [ISTQB-Web]
ISTQB-Glossary	ISTQB Glossary of Terms, available online from [ISTQB-Web]

書籍

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
Efride 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
Efride 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 日本では「システムテスト自動化 標準ガイド」（翔泳社, 2014 年）として発行されている。
Robert C Martin, "Clean Code: A Handbook of Agile Software Craftsmanship", 2008, ISBN-10: 9780132350884 日本では「Clean Code アジャイルソフトウェア達人の技」（アスキー・メディアワークス, 2009 年）として発行されている。
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

記事

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 "DevSecOpsFundamentals 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.pnsgc.org/docs/PROP53522057-FoldhaziDraftFinal.pdf
Thomas Pestak, William Rowell, PhD, "Automated Software Testing Practices and Pitfalls", September 2018, https://www.ait.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.ait.edu/stat/statcoe_files/0214simp%20%20AST%20IG%20for%20Managers%20and%20Practitioners.pdf
The Selenium Project, "Page object models", https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/

付録 A — 学習の目的/知識の認知レベル

以下の学習の目的は本シラバスに適用されるものとして定義されている。シラバスの各トピックは、そのための学習の目的に従って試験される。

学習の目的は、以下に列举された知識の認知レベルに対応するアクション動詞で始まる。

レベル 2 : 理解 (K2)

候補者はトピックに関連するステートメントの理由や説明を選択でき、テストの概念について要約、比較、分類、例を示すことができる。

アクション動詞 : 分類する、比較する、区別する、識別する、説明する、例を示す、解釈する、要約する

例	注記
テストツールをその目的とサポートするテスト活動に従って分類する。	
異なるテストレベルを比較する。	類似点、相違点、またはその両方を探すために使用できる。
テストとデバッグの違いを区別する。	概念間の相違点を探す。
プロジェクトリスクとプロダクトリスクを識別する。	2つ（またはそれ以上）の概念を別々に分類できるようにする。
テストプロセスへのコンテキストの影響を説明する。	
テストが必要な理由の例を示す。	
特定の故障のプロファイルから欠陥の根本原因を推測する。	
作業成果物レビュープロセスの活動を要約する。	

レベル 3 : 適用 (K3)

候補者は、慣れ親しんだタスクに直面したときに手順を実行するか、正しい手順を選択して特定のコンテキストに適用することができる。

アクション動詞 : 適用する、実装する、準備する、使用する

例	注記

境界値分析を適用して、特定の要件からテストケースを導出する。	手順/技法/プロセスなどを参照する必要がある。
技術的および管理的要件をサポートするためのメトリクス収集方法を実装する。	
モバイルアプリのインストーラビリティテストを準備する。	
トレーサビリティを使用して、テスト目的、テスト戦略、テスト計画との完全性と一貫性についてテスト進捗を監視する。	技法または手順を使用できることを候補者に求める LO で使用できる。「適用する」と同様。

レベル 4 : 分析 (K4)

候補者は、より深い理解のために手順や技法に関連する情報をその構成要素に分解でき、事実と推論を区別できる。典型的な適用は、文書、ソフトウェア、またはプロジェクト状況を分析し、問題またはタスクを解決するための適切なアクションを提案することである。

アクション動詞：分析する、解体する、概要を述べる、優先順位を付ける、選択する。

例	注記
特定の目標を達成するためにどのブラックボックスまたは経験ベースのテスト技法を適用すべきかを決定するために、特定のプロジェクト状況を分析する。	分析の測定可能な目標と組み合わせた場合にのみ試験対象となる。「xx を xx するために分析する」（または類似の）形式である必要がある。
関連するプロダクトリスクに基づいて実行のために特定のテストスイートのテストケースに優先順位を付ける。	
特定の要件セットを検証するための適切なテストレベルとテストタイプを選択する。	分析を必要とする選択が必要な場合に使用される。

参照

(学習の目的の認知レベルについて)

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

付録 B —学習の目的とビジネス成果のトレーサビリティマトリクス

この節では、テスト自動化エンジニアスペシャリストのビジネス成果と学習の目的の間のトレーサビリティを示す。

テスト自動化エンジニアスペシャリストのビジネス成果			B 0 1	B 0 2	B 0 3	B 0 4	B 0 5	B 0 6	B 0 7	B 0 8	B 0 9	B 1 0	B 1 1	B 1 2
第 1 章	テスト自動化の概要と目標													
1.1	テスト自動化の目的													
1.1.1	テスト自動化の利点と欠点を説明する	2	X											
1.2	ソフトウェア開発ライフサイクルにおけるテスト自動化													
1.2.1	さまざまなソフトウェア開発ライフサイクルモデル (SDLC) におけるテスト自動化の適用方法を説明する	2		X										
1.2.2	テスト対象システム (SUT) に適したテスト自動化ツールを選択する	2		X										
第 2 章	テスト自動化の準備													
2.1	テスト自動化を可能にするインフラストラクチャ構成の理解													
2.1.1	テスト自動化の実装を可能にするインフラストラクチャの構成要件を説明する	2			X									
2.1.2	異なる環境におけるテスト自動化の活用方法を説明する	2			X									
2.2	適切なツールと戦略を選択するための評価プロセス													
2.2.1	適切なテスト自動化ソリューション (TAS) を決定するために、テスト対象システム (SUT) を分析する	4				X								
2.2.2	ツール評価の技術的な結果を提示する	4				X								

テスト自動化エンジニアスペシャリストのビジネス成果			B 0 1	B 0 2	B 0 3	B 0 4	B 0 5	B 0 6	B 0 7	B 0 8	B 0 9	B 1 0	B 1 1	B 1 2
第3章	テスト自動化アーキテクチャ													
3.1	テスト自動化で活用されるデザインコンセプト													
3.1.1	テスト自動化アーキテクチャの主要機能を説明する	2					X							
3.1.2	テスト自動化ソリューションの設計方法を説明する	2					X							
3.1.3	テスト自動化フレームワークのレイヤー構造化を適用する	3					X							
3.1.4	テストケースを自動化するための各種アプローチを適用する	3					X							
3.1.5	テスト自動化における設計原則と設計パターンを適用する	3					X							
第4章	テスト自動化の実装													
4.1	テスト自動化開発													
4.1.1	効果的なテスト自動化パイロットおよびデプロイメント活動を支援するガイドラインを適用する	3						X						
4.2	テスト自動化開発に関連するリスク													
4.2.1	テスト自動化のデプロイメントリスクを分析し、軽減策を計画する	4							X					
4.3	テスト自動化ソリューションの保守性													
4.3.1	テスト自動化ソリューションの保守性を支援・影響する要因を説明する	2								X				
第5章	テスト自動化の実装とデプロイメント戦略													
5.1	CI/CD パイプラインへの統合													
5.1.1	パイプライン内の各テストレベルでテスト自動化を適用する	3									X			
5.1.2	テストウェアの構成管理を説明する	2									X			

テスト自動化エンジニアスペシャリストのビジネス成果			B 0 1	B 0 2	B 0 3	B 0 4	B 0 5	B 0 6	B 0 7	B 0 8	B 0 9	B 1 0	B 1 1	B 1 2
5.1.3	API インフラストラクチャに対するテスト自動化の依存関係を説明する	2									X			
第 6 章	テスト自動化のレポートニングとメトリクス													
6.1	テスト自動化データの収集・分析・レポートニング													
6.1.1	テスト自動化ソリューション（TAS）およびテスト対象システム（SUT）からのデータ収集方法を適用する	3										X		
6.1.2	テスト結果の理解を深めるために、テスト自動化ソリューション（TAS）およびテスト対象システム（SUT）からのデータを分析する	4										X		
6.1.3	テスト進捗レポートの作成と公開方法を説明する	2										X		
第 7 章	テスト自動化ソリューションの検証													
7.1	テスト自動化インフラストラクチャの検証													
7.1.1	テストツールのセットアップを含むテスト自動化環境の検証を計画する	3											X	
7.1.2	特定の自動テストスクリプトおよびテストスイートの正しい動作を説明する	2											X	
7.1.3	テスト自動化が予期せぬ結果を生成する箇所を特定する	2											X	
7.1.4	静的解析がテスト自動化コード品質を向上させる方法を説明する	2											X	
第 8 章	継続的改善													
8.1	テスト自動化の継続的改善の機会													
8.1.1	データ収集と分析によるテストケース改善の機会を発見する	3												X

テスト自動化エンジニアスペシャリストのビジネス成果			B 0 1	B 0 2	B 0 3	B 0 4	B 0 5	B 0 6	B 0 7	B 0 8	B 0 9	B 1 0	B 1 1	B 1 2
8.1.2	デプロイされたテスト自動化ソリューションの技術的側面を分析し、改善提案を提示する	4												X
8.1.3	テスト対象システム（SUT）の更新に合わせて自動テストウェアを再構築する	3												X
8.1.4	テスト自動化ツール活用を機会を整理する	2												X

付録 C — リリースノート

ISTQB® テスト自動化エンジニア シラバス 2024 は、2016 年版の大幅な改訂・全面改定版である。このため、章や節ごとの詳細なリリースノートは設けていない。テスト自動化エンジニアの役割に関するコンテンツが大幅に拡充される一方、戦略的なコンテンツは新たに策定された ISTQB® テスト自動化戦略シラバス 2024 に移管された。

付録 D — ドメイン固有用語

用語名	定義
DevSecOps	開発、セキュリティ、運用を組み合わせ、高度な自動化を使用する手法。
フローモデルパターン (flow model pattern)	作業ドメイン、そのコンポーネント、およびそれらの間の相互接続の高レベルビュー。
ユーザージャーニー (user journey)	ユーザーがテスト対象システムとどのように連携するかを、ユーザー体験の観点から示す一連のステップ。
ページオブジェクトパターン (page object pattern)	テスト保守の向上とコードの重複削減のための、テスト自動化におけるデザインパターン。
バージョン管理 (version control)	ソースコードの特定のバージョンをチェックインして格納するプロセス。