テスト技術者資格制度

Foundation Level シラバス

Version 2018.J03

International Software Testing Qualifications Board

改訂履歴

ISTQB®

バージョン	日付	摘要
ISTQB 2018	2018 年 6 月 4 日	正式公開版
ISTQB 2018	2018 年 4 月 27 日	正式公開バージョン候補
ISTQB 2018	2018 年 2 月 12 日	ベータバージョン候補
ISTQB 2018	2018 年 1 月 19 日	クロスレビュー用内部バージョン 3.0
ISTQB 2018	2018 年 1 月 15 日	コアチーム編集を反映したプレクロスレビュー用内部バージョン 2.9
ISTQB 2018	2017 年 12 月 9 日	アルファレビュー2.5 リリース - 2.0 リリースの技術的編集、新しい内容の追加なし
ISTQB 2018	2017 年 11 月 22 日	アルファレビュー2.0 リリース - テスト技術者資格制度 Foundation Level シラバスメジャーアップデート 2018 ー 詳細については、リリースノートの付録 C を参照
ISTQB 2018	2017 年 6 月 12 日	アルファレビューリリース - 『テスト技術者資格制度 Foundation Level シラバスメジャーアップデート 2018 ー リリースノート』の付録 C を参照
ISTQB 2011	2012 年 4 月 1 日	『テスト技術者資格制度 Foundation Level シラバスメンテナンスリリース - リリースノート』を参照
ISTQB 2010	2010 年 3 月 30 日	『テスト技術者資格制度 Foundation Level シラバスメンテナンスリリース - リリースノート』を参照
ISTQB 2007	2007 年 5 月 1 日	テスト技術者資格制度 Foundation Level シラバスメンテナンスリリース
ISTQB 2005	2005 年 7 月 1 日	テスト技術者資格制度 Foundation Level シラバス
ASQF V2.2	2003 年 7 月	ASQF Syllabus Foundation Level Version 2.2 “ Lehrplan Grundlagen des Software-testens”
ISEB V2.0	1999 年 2 月 25 日	ISEB Software Testing Foundation シラバス V2.0

JSTQB®

Version2018.J03	2019 年 8 月 26 日	誤記の修正
Version2018.J02	2019 年 4 月 5 日	誤記の修正
Version2018.J01	2019 年 3 月 27 日	ISTQB Version2018 の日本語翻訳版
Version2018.J00	2019 年 1 月 10 日	ISTQB Version2018 の日本語翻訳版（公開ベータ版）

Version2011.J02	2012 年 6 月 1 日	用語集（日本語版） Version 2.1.J01 に対応
Version2011.J01	2011 年 3 月 30 日	ISTQB　VER2011 の日本語翻訳版 和訳の表現を全体的に見直し
Version 2007.J01	2009 年 11 月 20 日	用語集に合わせて修正 和訳の表現を全体的に見直し バージョン表記方法の変更
Ver1.1.0	2008 年 7 月 1 日	ISTQB　VER2007 の日本語翻訳版 和訳の表現を全体的に見直し
Ver1.0.1	2006 年 12 月 29 日	用語集に合わせて修正 和訳の表現を全体的に見直し
Ver1.0.0	2006 年 7 月 21 日	用語集に合わせて修正
Ver0.6.3	2006 年 5 月 1 日	JTCB→JSTQB に変更 正式名称を Japan Software Testing Qualifications Board に変更
Ver0.6.2	2006 年 1 月 12 日	4　4.4 の K2 の位置を修正 4　4.4 の最後の行に K3 を追加 4.4 背景の説明の「ブランチ」を削除→デシジョンとブランチが違うように見えるため
Ver0.6.1	2005 年 12 月 17 日	2.1.1 テストレベルの意味について脚注を追加 1.5 レベル 2 以上の独立性があると→独立性が高いと 1.5 独立性のレベルを以下に示す→独立性の度合いを以下に示す
Ver0.6	2005 年 12 月 13 日	ISTQB VER2005 の日本語翻訳版

目　次

謝　辞

このドキュメントは、2018 年 6 月 4 日に開催された ISTQB の総会で正式に発行された。

本ドキュメントは、International Software Testing Qualifications Board の次のチームメンバーにより執筆された。Klaus Olsen (chair), Tauhida Parveen (vice chair), Rex Black (project manager), Debra Friedenberg), Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, Eshraka Zakaria.

本シラバスの作成チームは、技術的編集を行ってくれた Rex Black と Dorothy Graham、レビューチーム、クロスレビューチーム、および提案と意見を提供した各国委員会に感謝をしたい。

次のメンバーが、本シラバスのレビュー、意見表明、および投票に参加した。Tom Adams, Tobias Ahlgren, Xu Aiguo, Chris Van Bael, Katalin Balla, Graham Bath, Gualtiero Bazzana, Arne Becher, Veronica Belcher, Lars Hilmar Bjørstrup, Ralf Bongard, Armin Born, Robert Bornelind, Mette Bruhn-Pedersen, Geza Bujdoso, Earl Burba, Filipe Carlos, Young Jae Choi, Greg Collina, Alessandro Collino, Cui Zhe, Taz Daughtrey, Matthias Daigl, Wim Decoutere, Frans Dijkman, Klaudia Dussa-Zieger, Yonit Elbaz, Ofer Feldman, Mark Fewster, Florian Fieber, David Frei, Debra Friedenberg, Conrad Fujimoto, Pooja Gautam, Thorsten Geiselhart, Chen Geng, Christian Alexander Graf, Dorothy Graham, Michel Grandjean, Richard Green, Attila Gyuri, Jon Hagar, Kobi Halperin, Matthias Hamburg, Zsolt Hargitai, Satoshi Hasegawa, Berit Hatten, Wang Hongwei, Tamás Horváth, Leanne Howard, Chinthaka Indikadahena, J. Jayapradeep, Kari Kakkonen, Gábor Kapros, Beata Karpinska, Karl Kemminger, Kwanho Kim, Seonjoon Kim, Cecilia Kjellman, Johan Klintin, Corne Kruger, Gerard Kruijff, Peter Kunit, Hyeyong Kwon, Bruno Legeard, Thomas Letzkus, Alon Linetzki, Balder Lingegård, Tilo Linz, Hongbiao Liu, Claire Lohr, Ine Lutterman, Marek Majernik, Rik Marselis, Romanos Matthaios, Judy McKay, Fergus McLachlan, Dénes Medzihradszky, Stefan Merkel, Armin Metzger, Don Mills, Gary Mogyorodi, Ninna Morin, Ingvar Nordström, Adam Novak, Avi Ofer, Magnus C Ohlsson, Joel Oliviera, Monika Stocklein Olsen, Kenji Onishi, Francisca Cano Ortiz, Gitte Ottosen, Tuula Pääkkönen, Ana Paiva, Tal Pe'er, Helmut Pichler, Michaël Pilaeten, Horst Pohlmann, Andrew Pollner, Meile Posthuma, Vitalijs Puiso, Salvatore Reale, Stuart Reid, Ralf Reissing, Shark Ren, Miroslav Renda, Randy Rice, Adam Roman, Jan Sabak, Hans Schaefer, Ina Schieferdecker, Franz Schiller, Jianxiong Shen, Klaus Skafte, Mike Smith, Cristina Sobrero, Marco Sogliani, Murian Song, Emilio Soresi, Helder Sousa, Michael Sowers, Michael Stahl, Lucjan Stapp, Li Suyuan, Toby Thompson, Steve Toms, Sagi Traybel, Sabine Uhde, Stephanie Ulrich, Philippos Vakalakis, Erik van Veenendaal, Marianne Vesterdal, Ernst von Düring, Salinda Wickramasinghe, Marie Walsh, Søren Wassard, Hans Weiberg, Paul Weymouth, Hyungjin Yoon, John Young, Surong Yuan, Ester Zabar, and Karolina Zmitrowicz.

International Software Testing Qualifications Board Foundation Level 作業部会 (Edition 2018) ：Klaus Olsen (chair), Tauhida Parveen (vice chair), Rex Black (project manager), Dani Almog, Debra Friedenberg, Rashed Karim, Johan Klintin, Vipul Kocher, Corne Kruger, Sunny Kwon, Judy McKay, Thomas Müller, Igal Levi, Ebbe Munk, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, Eshraka Zakaria, and Stevan Zivanovic.コアチームは提案に対してレビューチームと各国委員会に感謝をしたい。

International Software Testing Qualifications Board Foundation Level 作業部会(Edition 2011)：Thomas Müller (chair), Debra Friedenberg.コアチームは提案に対してレビューチーム（Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen、Eric Riou du Cosquier Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal)と各国委員会に感謝をしたい。

International Software Testing Qualifications Board Foundation Level 作業部会(Edition 2010)：Thomas Müller (chair), Rahul Verma, Martin Klonk and Armin Beer.コアチームは現在のシラバスへの提案に対し

てレビューチーム(Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veenendaal) と各国委員会に感謝をしたい。

International Software Testing Qualifications Board Foundation Level 作業部会(Edition 2007) ：Thomas Müller (chair) , Dorothy Graham, Debra Friedenberg, Erik van Veenendaal.コアチームは現在のシラバスへの提案に対してレビューチーム(Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, Wonil Kwon) と各国委員会に感謝をしたい。

International Software Testing Qualifications Board Foundation Level 作業部会(Edition 2005) ：Thomas Müller (chair) , Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson, Erik van Veenendaal.コアチームは提案に対してレビューチームと各国委員会に感謝をしたい。

0 イントロダクション

0.1 本シラバスの目的

本シラバスは、国際ソフトウェアテスト資格 Foundation Level のベースとなる。ISTQB は、本シラバスを以下の趣旨で提供する。

1. メンバー委員会に対し、各国語への翻訳および教育機関の認定の目的で提供する。メンバー委員会は、本シラバスを各言語の必要性に合わせて調整し、出版事情に合わせてリファレンスを追加することができる。
2. 認定委員会に対し、本シラバスの学習目的に合わせ、各国語で試験問題を作成する目的で提供する。
3. 教育機関に対し、コースウェアを作成し、適切な教育方法を確定できるようにする目的で提供する。
4. 受験志願者に対し、認定試験準備の目的で提供する（研修コースの一部として、または独立した形態で実施）。
5. 国際的なソフトウェアおよびシステムエンジニアリングのコミュニティに対し、ソフトウェアやシステムをテストする技能の向上を目的とする他、書籍や記事を執筆する際の参考として提供する。

ISTQB では、事前に書面による申請があり ISTQB から承認された場合に限り、第三者がこのシラバスを先に定めた以外の目的での使用を許諾することがある。

0.2 ソフトウェアテスト向けテスト技術者資格制度 Foundation Level

Foundation Level資格は、ソフトウェアテストに関与するあらゆる人々を対象にする。Foundation Level資格の対象者には、テスト担当者、テストアナリスト、テストエンジニア、テストコンサルタント、テストマネージャー、ユーザー受け入れテスト担当者、ソフトウェア開発担当者などが含まれる。このFoundation Level資格は、ソフトウェアテストについて基本的な理解を望む方、例えば、プロダクトオーナー、プロジェクトマネージャー、品質管理マネージャー、ソフトウェア開発マネージャー、ビジネスアナリスト、IT部門長、経営コンサルタントのような人にも適切である。Foundation Level認定資格の保持者はソフトウェアテスト資格での上位レベルに進むことが可能である。

『ISTQB Foundation Level シラバス 日本語版 概要』が別ドキュメントとして用意されており、以下の情報を記載している。

- シラバスのビジネス成果
- 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マトリクス
- 本シラバスの概要

0.3 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、テスト技術者資格制度 Foundation Level の試験問題作成を行うために使用する。

全体を通して、本シラバスのすべての内容は、「イントロダクション」と付録を除いて K1 レベルで試験対象である。つまり、受験志願者は本シラバスの 6 つの章で説明されているキーワードと概念について認識し、記憶し、想起することになる。本シラバスでは各章の先頭で以下の分類にて「学習の目的」を示している。

- K1：記憶
- K2：理解
- K3：適用

学習の目的の詳細と例は付録Bに添付する。

各章の章見出しの下にキーワードとしてリストアップされているすべての用語の定義は、「学習の目的」には明示的に述べられていないとしても「記憶」しておくべき（K1）レベルとなる。

0.4 Foundation Level の認定資格試験

Foundation Level の認定資格試験は本シラバスに基づく。試験問題に対する解答は本シラバスの複数の節を基にしている場合がある。本シラバスのすべての節は、「イントロダクション」と付録を除いて試験対象である。標準、文献、および他の ISTQB シラバスを情報源としているが、それらに関して本シラバス自体の中で要約されている以上の内容は試験対象ではない。

試験の形式は多肢選択式である。問題の数は 40 である。問題の 65%（26 の問題）以上を正解した場合に合格となる。

試験は、認定トレーニングコースの一部として、または（例えば試験センターや公的試験で）独立して実施してもよい。認定トレーニングコースの受講完了は試験のための前提条件ではない。

0.5 認定審査

ISTQBのメンバー委員会にて、教育コースの教材が本シラバスに従っている教育機関を認定する。教育機関はメンバー委員会または認定を行う機関から認定ガイドラインを入手しなければならない。教育コースがシラバスに従っていると認定されると、教育コースの一部としてISTQBの試験を実施することができる。

0.6 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教育と試験を可能にする。このゴールを達成するために本シラバスは以下のようになっている。

- 全般的な教育内容の目的（Foundation Level の意図について説明）
- 用語のリスト（思い出すことができなければならない用語）
- 各知識領域の学習の目的（達成しなければならない知識レベルの学習の成果について説明）
- 教えるべき主要なコンセプトの説明（承認された文献や標準を情報源として含む）

本シラバスの内容はソフトウェアテストの全知識領域の説明ではない。詳細レベルは、Foundation Level のトレーニングコースでカバーされることを示している。本シラバスは、アジャイル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コンセプトとテスト技法に重点を置いて説明している。本シラバスでは、ソフトウェア開発の特定のライフサイクルおよび方法論に関連する個別の学習の目的は取り上げていない。ただし、これらのコンセプトをアジャイルプロジェクト、他の種類のイテレーティブ/インクリメンタルライフサイクル、およびシーケンシャルライフサイクルに適用する方法については説明している。

0.7 本シラバスの構成

6つの章で構成され、すべて試験対象である。各章の一番上の見出しは、章の学習時間を指定している。章より下のレベルでは、時間は指定されていない。認定トレーニングコースでは、本シラバスは16時間45分以上の講義を必要とし、6つの章で以下のように配分する。

- 第 1 章：テストの基礎 - 学習時間 175 分
- 第 2 章：ソフトウェア開発ライフサイクル全体を通してのテスト - 学習時間 100 分
- 第 3 章：静的テスト - 学習時間 135 分
- 第 4 章：テスト技法 - 学習時間 330 分
- 第 5 章：テストマネジメント - 学習時間 225 分
- 第 6 章：テスト支援ツール - 学習時間 40 分

1 テストの基礎 — 175 分

キーワード

カバレッジ、デバッグ、欠陥、エラー、故障、品質、品質保証、根本原因、テスト分析、テストベース、テストケース、テスト完了、テスト条件、テストコントロール、テストデータ、テスト設計、テスト実行、テスト実行スケジュール、テスト実装、テストモニタリング、テスト対象、テスト目的、テストオラクル、テスト計画、テスト手順、テストスイート、テスト、テストウェア、トレーサビリティ、妥当性確認、検証

「テストの基礎」の学習の目的

1.1 テストとは何か？

FL-1.1.1 （K1）テストに共通する目的を識別する。

FL-1.1.2 （K2）デバッグとテストを区別する。

1.2 テストの必要性

FL-1.2.1 （K2）例を挙げてテストの必要性を説明する。

FL-1.2.2 （K2）テストと品質保証の関係を説明し、品質を確保する上でテストがどう貢献するかを、例を挙げて説明する。

FL-1.2.3 （K2）エラー、欠陥、故障を区別する。

FL-1.2.4 （K2）欠陥の根本原因と結果を区別する。

1.3 テストの 7 原則

FL-1.3.1 （K2）テストにおける 7 原則を説明する。

1.4 テストプロセス

FL-1.4.1 （K2）テストを取り巻く状況がテストプロセスに与える影響を説明する。

FL-1.4.2 （K2）テストプロセスにおけるテストの活動と関連するタスクを説明する。

FL-1.4.3 （K2）テストプロセスを支援する多くの作業成果物を区別できる。

FL-1.4.4 （K2）テストベースとテスト作業成果物との間のトレーサビリティを維持することがなぜ必要であるかを説明する。

1.5 テストの心理学

FL-1.5.1 （K1）テストの成否は心理的要素に影響されることを識別する。

FL-1.5.2 （K2）テストの活動とソフトウェア開発の活動で必要となるマインドセットが違うことを説明する。

1.1 テストとは何か？

ソフトウェアシステムは、ビジネス分野（銀行など）から、消費者製品（自動車など）に至るまで、社会を構成する要素として必須となっている。ソフトウェアが期待通りに動かなかった経験は誰もが持っている。ソフトウェアが正しく動作しないと、経済的な損失、時間の浪費、信用の失墜など、さまざまな問題が発生し、時には傷害や死亡事故になることもある。ソフトウェアテストはソフトウェアの品質を評価し、運用環境でソフトウェアの故障が発生するリスクを低減する 1 つの手段である。

テストに関するよくある誤解の 1 つは、テストはソフトウェアを実行し結果を確認するだけだというものである。1.4 節で説明するように、ソフトウェアテストはさまざまな活動を含むプロセスである。テスト実行（結果の確認を含む）は、それらの活動の 1 つにすぎない。テストプロセスは、テストの計画、分析、設計、実装、テスト進捗と結果の報告、テスト対象の品質評価などの作業を含む。

テスト対象のコンポーネントやシステムを実行することは、動的テストと呼ぶ。テスト対象のコンポーネントやシステムを実行しない場合は、静的テストと呼ぶ。このため、テストは要件、ユーザーストーリー、ソースコードなどの作業成果物をレビューする活動も含む。

テストに関するもう 1 つのよくある誤解は、テストは要件、ユーザーストーリー、またはその他の仕様の検証に重点を置くことがすべてだというものである。テストでは、指定されている要件をシステムが満たすかどうかを確認することに加えて、妥当性確認も行う。妥当性確認では、ユーザーやその他のステークホルダーのニーズを運用環境でシステムが満たしていることを確認する。

テスト活動の編成と実行方法は、ライフサイクルに応じて異なる（2.1 節を参照）。

1.1.1 テストに共通する目的

すべてのプロジェクトで、テストの目的は以下を含む。

- 要件、ユーザーストーリー、設計、およびコードなどの作業成果物を評価する。
- 明確にしたすべての要件を満たしていることを検証する。
- テスト対象が完成し、ユーザーやその他ステークホルダーの期待通りの動作内容であることの妥当性確認をする。
- テスト対象の品質に対する信頼を積み重ねて、所定のレベルにあることを確証する。
- 欠陥の作りこみを防ぐ。
- 故障や欠陥を発見する。
- ステークホルダーが意志決定できる、特にテスト対象の品質レベルについての十分な情報を提供する。
- （以前に検出されなかった故障が運用環境で発生するなどの）不適切なソフトウェア品質のリスクレベルを低減する。
- 契約上、法律上、または規制上の要件や標準を遵守する、そして／またはテスト対象がそのような要件や標準に準拠していることを検証する。

テストの目的は、テスト対象のコンポーネントまたはシステム、テストレベル、ソフトウェア開発ライフサイクルモデルといった要因により異なる。例えば、以下のような相違点がある。

- コンポーネントテストの場合、ソフトウェア中の欠陥を早期に特定して修正するためになるべく多くの故障を見つけることを目的とすることもあれば、コンポーネントテストのコードカバレッジを増やすことを目的とすることもある。

- 受け入れテストの場合、システムが期待通りに動作し、要件を満たすことの確認を目的とすることもあれば、指定期日にシステムをリリースすることのリスクについてステークホルダーに情報を示すことが目的となることもある。

1.1.2 テストとデバッグ

テストとデバッグは異なる。テストを実行することは、ソフトウェアに存在する欠陥に起因する故障を見つけることである。一方、デバッグは、故障の基となる欠陥を見つけて、解析し、取り除く一連の開発の活動である。その後、テスト担当者が確認テストを実施し、修正により欠陥が解決したことを確認する。通常、テスト担当者はテスト対象の初回のテストと最終的な確認テストに責任を持ち、開発担当者はデバッグとコンポーネントテストに責任を持つことになる。ただし、アジャイル開発やライフサイクルによっては、テスト担当者がデバッグやコンポーネントテストに関与することもある。

ISO 標準（ISO/IEC/IEEE 29119-1）には、ソフトウェアテストの概念に関する詳細な情報が記載されている。

1.2 テストの必要性

コンポーネントやシステム、およびドキュメントを厳しくテストすれば、運用環境で問題が発生するリスクを低減できる。システムが稼動する前に欠陥を検出して修正すると、コンポーネントやシステムの品質向上に効果がある。また、契約や法律上の要件、および各業界の標準に合致していることを証明するためにソフトウェアテストが必要になるケースもある。

1.2.1 成功に対するテストの貢献

コンピューターの歴史を通して、運用環境にリリースされたソフトウェアやシステムが欠陥を含んでいるために、故障が起きたりステークホルダーのニーズに合致しなかったりすることは一般的である。ただし、適切なテストレベル、およびソフトウェア開発ライフサイクルの適切な時点で適切なテストの専門知識を適用すれば、適切なテスト技法を使用することで問題を含むリリースの頻度を低減できる。この例には、以下のようなものがある。

- テスト担当者が要件レビューやユーザーストーリーの洗練作業に関与することにより、これらの作業成果物の欠陥を検出できる。要件に対する欠陥の検出と除去を行うことにより、テストができない機能、または正しくない機能が開発されるリスクを低減できる。
- システムの設計時にテスト担当者がシステム設計者と密接に連携して作業することにより、両者が設計とその設計をどうテストするかに対する理解を深めることができる。結果として、基本的な設計の欠陥が混入するリスクを低減でき、テストケースを早い段階で識別できる。
- コード開発時にテスト担当者が開発担当者と密接に連携して作業することにより、両者がコードとそのコードをどうテストするかに対する理解を深めることができる。結果として、コードとテストケースに欠陥が混入するリスクを低減できる。
- テスト担当者はリリース前にソフトウェアを検証および妥当性確認することにより、テストしなければ見逃してしまうかもしれない故障を検出し、故障の原因となる欠陥を除去するプロセス（すなわちデバッグ）を支援できる。結果として、ソフトウェアがステークホルダーのニーズに合致し、要件が満たされる可能性が高まる。

これらの例に加えて、決定したテスト目的（1.1.1 節を参照）を達成することにより、ソフトウェアの開発とメンテナンス全体の成功に貢献できる。

1.2.2 品質保証とテスト

品質保証（または単に *QA*）という用語がテストを意味するために使われることがある。しかし、品質保証とテストは、関連してはいるが同じではない。さらに大きな範囲を表す品質マネジメントという概念が両者を結び付けている。品質マネジメントは、品質に関して組織の方針を示し、組織をコントロールするための活動をすべて含む。品質マネジメントは、品質保証と品質コントロールの両方を含む。品質保証は、一般的に、品質が適切なレベルに確実に到達するよう、適切なプロセスを遵守することに重点を置く。プロセスを適切に遂行すると、これらのプロセスで作成される作業成果物は一般的に高い品質を持つ。これにより、欠陥の混入を防止できる。さらに、根本原因分析を使用して欠陥の原因の検出と除去を行い、プロセスを改善するために振り返りミーティングでの発見事項を適切に利用することが、品質保証の有効性を高めるために重要である。

品質コントロールには、テスト活動を含め、適切なレベルの品質を達成するために役立つさまざまな活動が含まれる。テスト活動は、ソフトウェア開発またはメンテナンスプロセス全体の一部を構成する。品質保証では、テストを含むプロセス全体を適切に実行することが必要となるため、適切なテストを支援する。1.1.1 と 1.2.1 で説明しているように、テストはさまざまな方法で品質の達成に貢献する。

1.2.3 エラー、欠陥、および故障

人間はエラー（誤り）を犯す。そのエラーがソースコードや他の関連する作業成果物の欠陥（フォールトまたはバグ）となる。1 つの作業成果物で欠陥を発生させるエラーは、他の関連する作業成果物でも欠陥を発生させる可能性がある。例えば、要件の導出のエラーは要件欠陥をもたらし、結果的にプログラミングエラーをもたらし、コード内に欠陥をもたらす。

コードの欠陥が実行されると、故障が起きることがあるが、すべての状況で故障が起きるわけではない。例えば、欠陥によっては、故障を起こすためにきわめて特別な入力や事前条件が必要である。そのような故障は発生すること自体が非常に稀であったり、決して発生しなかったりする。

エラーは以下のものを含むさまざまな理由によって発生する。

- 納期のプレッシャー
- 人間の誤りを犯しやすい性質
- プロジェクト参加者の経験不足または技術不足
- プロジェクト参加者間の誤ったコミュニケーション（要件や設計の誤ったコミュニケーションを含む）
- コードの複雑さ、設計の複雑さ、アーキテクチャーの複雑さ、解決すべき根本的な問題の複雑さ、そして／または使用する技術の複雑さ
- システム内またはシステム間のインターフェースに関する誤解（特に、関連するシステムの数が多い場合）
- 新しく不慣れな技術

故障はコード内の欠陥だけでなく、環境条件により起きることもある。例えば、放射能、磁気、電界、汚染がファームウェアの誤動作を起こしたり、ハードウェアの構成変更がソフトウェアの実行に影響を与えたりすることがある。

期待通りではないテスト結果がすべて故障であるとは限らない。テスト実行方法のエラー、テストデータやテスト環境、その他のテストウェアの欠陥、またはその他の理由により、偽陽性（誤検出）をしてしまうことがある。状況によっては、類似のエラーまたは欠陥が偽陰性（検出もれ）となることもある。偽陰性は、検出すべき欠陥を検出しないテストを意味する。偽陽性は、実際には欠陥でないものを欠陥として報告する。

1.2.4 欠陥、根本原因、および影響

欠陥の根本原因とは、欠陥を埋め込んだ最初の行為または条件のことである。欠陥を分析して根本原因を識別し、類似の欠陥が今後発生しないようにしなければならない。最も重大な根本原因に焦点をあてることはプロセス改善につながり、今後に大量の欠陥が作りこまれるのを防止できる。

例えば、1 行のコードの間違いが間違った利子の支払いを引き起こし、顧客の不満を招いたとする。欠陥のあるコードは、プロダクトオーナーが利子の計算方法を誤解しており、ユーザーストーリーが曖昧となったために埋め込まれた。多くの割合の欠陥が利子の計算に存在し、これらの欠陥の根本原因が類似の誤解により発生していた場合、プロダクトオーナーが利子計算に関する知識を習得することで、そのような欠陥を埋め込まないようにできる。

この例では、「顧客の不満」が影響で、「間違った利子の支払い」が故障である。「コードでの不適切な計算」は欠陥であり、おおもとの欠陥であるユーザーストーリーの曖昧さが欠陥をもたらした。おおもとの欠陥の根本原因はプロダクトオーナーの知識不足であり、ユーザーストーリーの作成時に誤りを犯した。根本原因分析のプロセスは、『ISTQB-ETM Expert Level Test Management Syllabus』と『ISTQB-EITP Expert Level Improving the Test Process Syllabus』で説明する。

1.3 テストの 7 原則

これまで 50 年以上にわたり、さまざまなテストの原則が提唱され、あらゆるテストで共通に使える一般的なガイドラインとなってきた。

1.テストは欠陥があることは示せるが、欠陥がないことは示せない

テストにより、欠陥があることは示せるが、欠陥がないことは証明できない。テストにより、ソフトウェアに残る未検出欠陥の数を減らせるが、欠陥が見つからないとしても、正しさの証明とはならない。

2.全数テストは不可能

すべてをテストすること（入力と事前条件の全組み合わせ）は、ごく単純なソフトウェア以外では非現実的である。全数テストの代わりに、リスク分析、テスト技法、および優先度によりテストにかける労力を集中すべきである。

3.早期テストで時間とコストを節約

早い段階で欠陥を見つけるために、静的テスト活動と動的テスト活動の両方をソフトウェア開発ライフサイクルのなるべく早い時期に開始すべきである。早期テストは、*シフトレフト*とも呼ばれる。ソフトウェア開発ライフサイクルの早い時期にテストを行うことにより、コストを低減または削減できる（3.1 節を参照）。

4.欠陥の偏在

リリース前のテストで見つかる欠陥や運用時の故障の大部分は、特定の少数モジュールに集中する。

テストの労力を集中させるために欠陥の偏在を予測し、テストや運用での実際の観察結果に基づいてリスク分析を行う。（原則 2 で触れたことと同様）。

5.殺虫剤のパラドックスにご用心

同じテストを何度も繰り返すと、最終的にはそのテストでは新しい欠陥を見つけられなくなる。この「殺虫剤のパラドックス」を回避するため、テストとテストデータを定期的に見直して、改定したり新規にテストを作成したりする必要がある（殺虫剤を繰り返し使用すると効果が低減するのと同様に、テストにおいても欠陥を見つける能力は低減する）。ただし、自動化されたリグレッションテストの場合は、同じテストを繰り返すことでリグレッションが低減しているという有益な結果を示すことができる。

6.テストは状況次第

状況が異なれば、テストの方法も変わる。例えば、安全性が重要な産業用制御ソフトウェアのテストは、e コマースモバイルアプリケーションのテストとは異なる。また、アジャイルプロジェクトとシーケンシャルライフサイクルプロジェクトでは、テストの実行方法が異なる（2.1 節を参照）。

7.「バグゼロ」の落とし穴

テスト担当者は可能なテストすべてを実行でき、可能性のある欠陥すべてを検出できると期待する組織があるが、原則 2 と原則 1 により、これは不可能である。また、大量の欠陥を検出して修正するだけでシステムを正しく構築できると期待することも誤った思い込みである。例えば、指定された要件すべてを徹底的にテストし、検出した欠陥すべてを修正しても、使いにくいシステム、ユーザーのニーズや期待を満たさないシステム、またはその他の競合システムに比べて劣るシステムが構築されることがある。

これらのテスト原則およびその他のテスト原則の例については、『Myers 2011』、『Kaner 2002』、および『Weinberg 2008』を参照されたい。

1.4 テストプロセス

万能なソフトウェアテストプロセスというものはない。しかし、テストの目的を達成する確率を高くする、汎用的なテスト活動のセットというものが複数ある。これらテスト活動のセットがテストプロセスである。特定の状況に対する適切なソフトウェアテストプロセスは、多くの要因を考慮して決定する。テストプロセスの中で、どのテストの活動を、どのように、いつ実施するかは、組織のテスト戦略に含まれている場合がある。

1.4.1 状況に応じたテストプロセス

組織のテストプロセスに影響する状況のいくつかを次に示す（すべてが示されているわけではない）。

- 使用するソフトウェア開発ライフサイクルモデルとプロジェクト方法論
- 考慮対象のテストレベルとテストタイプ
- プロダクトとプロジェクトのリスク
- ビジネスドメイン
- 次を含む運用上の制約：
  - 予算とリソース
  - 期間
  - 複雑さ
  - 契約上および規制上の要件
- 組織のポリシーと実践例

- 要求される組織内の標準と組織外の標準

以降の節では、組織のテストプロセスの一般的な側面を、以下に列挙した項目ごとに説明する。

- テストの活動とタスク
- テスト作業成果物
- テストベースとテスト作業成果物との間のトレーサビリティ

（考慮対象のテストレベルまたはテストタイプの）テストベースに、計測可能なカバレッジ基準を定義しておくと、非常に役立つ。カバレッジ基準は重要業績評価指標（KPI）として効果的に機能して活動を推進し、ソフトウェアテストの目的の達成度を示す（1.1.1 節を参照）。

例えば、モバイルアプリケーションのテストベースは、要件のリストとサポート対象のモバイルデバイスのリストを含むとする。その場合、各要件はテストベースの要素であり、サポート対象の各デバイスもテストベースの要素である。カバレッジ基準は、テストベースの各要素に対して 1 つ以上のテストケースが必要だとする。これらのテストの実行結果により、ステークホルダーは、指定されている要件が満たされているかどうかと、サポート対象デバイスで故障が検出されたかどうかを知ることができる。

ISO 標準（ISO/IEC/IEEE 29119-2）には、テストプロセスの詳細情報が記載されている。

1.4.2 テストの活動とタスク

テストプロセスを構成する主な活動のグループは以下の通りである。

- テスト計画
- テストのモニタリングとコントロール
- テスト分析
- テスト設計
- テスト実装
- テスト実行
- テスト完了

各グループは、以降の節で説明する活動で構成する。各グループ内の各活動は、さらに複数の個別のタスクで構成する。これらのタスクはプロジェクトまたはリリースごとに異なる。

さらに、これらのグループの多くは論理的には順次行われるが、繰り返されることが多い。例えば、アジャイル開発では、計画作業を継続的に行いながら、ソフトウェアの設計、ビルド、テストが連続して行われる活動を、小さな単位で繰り返し行う。このため、この開発アプローチでは、テストの活動も反復的に継続して行う。シーケンシャル開発でも、活動を論理的な順序で段階的に行う場合、重なりあって実行したり、組み合わせて実行したり、同時に実行したり、実行を省略したりするため、システムやプロジェクトの状況に合わせてこれらの主要な活動をテーラリングする必要がある。

テスト計画

テスト計画では、テストの目的と、状況により課せられる制約内でテストの目的を達成するためのアプローチを定義する（例えば、適切なテスト技法とタスクを指定し、納期に間に合うようにテストスケジュールを作成する）。テスト計画書は、モニタリングとコントロールの活動からのフィードバックに応じて更新をする。テスト計画については、5.2 節でさらに説明する。

テストのモニタリングとコントロール

テストモニタリングは、テスト計画書で定義したテストモニタリングのメトリクスを使用して、テスト計画書の内容と実際の進捗を継続的に比較する活動である。テストコントロールは、テスト計画書の目的に合致させるために対策を講じる活動で、テスト計画書の継続的な更新活動も含む。テストのモニタリングとコントロールは、終了基準の評価により支援される。終了基準は、ライフサイクルによっては「完了（done）の定義」とも呼ばれる（『ISTQB テスト技術者資格制度 Foundation Level Extension シラバス アジャイルテスト担当者 日本語版 Version 2014.J01』を参照）。例えば、テストレベルによっては、テスト実行の終了基準を評価する際に、以下の活動を行う。

- 特定のカバレッジ基準に対してテスト結果とログをチェックする。
- テスト結果とログに基づいて、コンポーネントまたはシステムの品質のレベルを評価する。
- さらなるテストが必要かどうかを判断する（例えば、プロダクトリスクカバレッジで目標とした特定のレベルを当初予定したテストで達成できない場合、さらなるテストの作成および実行が必要になる）。

計画に対するテスト進捗は、テスト進捗レポートを使用してステークホルダーへ伝える。このレポートで伝える内容には、計画からの逸脱や、テストの中止を決定するために必要な情報を含む。

テストのモニタリングとコントロールについては、5.3 節でさらに説明する。

テスト分析

テスト分析では、テスト可能なフィーチャーを識別し、テスト条件を決めるためにテストベースを分析する。言い換えると、テスト分析では計測可能なカバレッジ基準から見た「何をテストするか」を決定する。

テスト分析の主な活動は以下の通りである。

- テストレベルごとに適切なテストベースを分析する。
    - 要件仕様。ビジネス要件、機能要件、システム要件、ユーザーストーリー、エピック、ユースケースの他、コンポーネントやシステムに期待される機能および非機能の動作を指定する類似の作業成果物などがある。
    - 設計および実装情報。システムやソフトウェアに関するアーキテクチャー図もしくはドキュメント、設計仕様、コールフロー、モデル図（UML や ER 図など）、インターフェース仕様、コンポーネントもしくはシステムの構造を指定する類似の作業成果物などがある。
    - コンポーネントまたはシステム実装そのもの。コード、データベースのメタデータやクエリー、インターフェースなどがある。
    - リスク分析レポート。コンポーネントやシステムの機能、非機能、構造の各側面を考慮する。
- テストベースとテストアイテムを評価して、以下のようなさまざまな種類の欠陥を識別する。
    - 曖昧
    - 欠落
    - 不整合
    - 不正確
    - 矛盾
    - 冗長なステートメント
- テストすべきフィーチャーとフィーチャーのセットを識別する。
- テストベースの分析に基づいて、各フィーチャーのテスト条件を決めて優先度を割り当てる。この際には、機能/非機能/構造の特性、他のビジネス/技術的要因、リスクのレベルを考慮する。

- テストベースの各要素と関連するテスト条件の間に双方向のトレーサビリティを確立する（1.4.3 節と 1.4.4 節を参照）。

テスト分析プロセスにてブラックボックス、ホワイトボックス、および経験ベースのテスト技法を適用することは（第 4 章を参照）、重要なテスト条件の欠落を防止し、精度と正確性が高いテスト条件の特定に役立つ。

テスト分析では、テストチャーターのテスト目的として使用するテスト条件を作り出す場合がある。経験ベースのテストの種類によっては、テストチャーターが典型的な作業成果物となる（4.4.2 節を参照）。これらのテスト目的がテストベースに対して追跡可能な場合、経験ベースのテストにおけるカバレッジの達成度を計測できる。

テスト分析で欠陥を検出できることは大きな利点だといえる。特に他のレビュープロセスが使用されていない場合、および／またはテストプロセスがレビュープロセスと密接に関連付けられている場合に利点となる。この場合、テスト分析では、要件に一貫性があり、適切に表現され、完全であることを検証できるだけでなく、顧客、ユーザー、およびその他のステークホルダーのニーズが適切に要件に反映されていることの妥当性確認ができる。例えば、コーディングの前にユーザーストーリーや受け入れ基準からテスト条件やテストケースを作成する振る舞い駆動開発（BDD）や受け入れテスト駆動開発（ATDD）などの技法は、ユーザーストーリーや受け入れ基準を検証および妥当性確認をして、欠陥を検出する（『ISTQB テスト技術者資格制度 Foundation Level Extension シラバス アジャイルテスト担当者 日本語版』を参照）。

テスト設計

テスト設計では、テスト条件を高位レベルテストケース、高位レベルテストケースのセット、およびその他のテストウェアへ落とし込む。つまり、テスト分析は「何をテストするか」を決定し、テスト設計は「それをどうテストするか」を決定する。

テスト設計の主な活動は以下の通りである。

- テストケースおよびテストケースのセットを設計し、優先度を割り当てる。
- テスト条件とテストケースを支援するために必要なテストデータを識別する。
- テスト環境を設計し、必要なインフラストラクチャーやツールを識別する。
- テストベース、テスト条件、テストケース、テスト手順の間で双方向のトレーサビリティを確立する（1.4.4 節を参照）。

テスト設計の間にテスト条件をテストケースおよびテストケースのセットへ落とし込む際には、さまざまなテスト技法を使用する（第 4 章を参照）。

テスト分析と同じように、テスト設計はテストベースに含まれる類似の欠陥を識別することもある。また、テスト設計時に欠陥を識別できることは、テスト分析時と同じように大きな利点だといえる。

テスト実装

テスト実装では、テスト実行に必要なテストウェアを作成、および／または完成させる。そこにはテストケースの順序を決定してテスト手順を作成することが含まれる。つまり、テスト設計は「それをどうテストするか」の答えであり、テスト実装は「テストの実行に必要なものすべてを準備したか」の答えである。

テスト実装の主な活動は以下の通りである。

- テスト手順を開発して優先度を割り当てる。場合によっては、自動化のテストスクリプトを作成する。

- テスト手順や（存在する場合）テストスクリプトからテストスイートを作成する。
- 効率的にテスト実行ができるように、テスト実行スケジュール内でテストスイートを調整する（5.2.4 節を参照）。
- テスト環境（これには、テストハーネス、サービスの仮想化、シミュレーター、およびその他のインフラストラクチャーアイテムが含まれる）を構築する。また、必要なものすべてが正しくセットアップされていることを確認する。
- テストデータを準備し、テスト環境に適切に読み込ませてあることを確認する。
- テストベース、テスト条件、テストケース、テスト手順、テストスイートの間での双方向トレーサビリティを検証し更新する（1.4.4 節を参照）。

テスト設計とテスト実装は一緒に行うことがある。

探索的テストやその他の種類の経験ベースのテストの場合、テスト設計とテスト実装は、テスト実行の一部となり、ドキュメントもまとめて作る場合がある。探索的テストでは（テスト分析の一環として作り出した）テストチャーターに基づいて、テスト設計とテスト実装を行いながらテスト実行をする（4.4 節を参照）。

テスト実行

テスト実行では、テスト実行スケジュールに従ってテストスイートを実行する。

テスト実行の主な活動は以下の通りである。

- テストアイテムまたはテスト対象、テストツール、テストウェアの ID とバージョンを記録する。
- 手動で、またはテスト実行ツールを使用してテストを実行する。
- 実行結果と期待結果を比較する。
- 不正を分析して、可能性のある原因を特定する（故障はコード内の欠陥によって発生する可能性があるが、偽陽性が発生することもある（1.2.3 節を参照））。
- 故障を観察し、観察に基づいて欠陥を報告する（5.6 節を参照）。
- テスト実行の結果（合格、不合格、ブロックなど）を記録する。
- 不正への対応の結果、または計画したテストの一環として、テスト活動を繰り返す（修正したテストケースによるテスト、確認テスト、および／またはリグレッションテストの実行）。
- テストベース、テスト条件、テストケース、テスト手順、テスト結果の間で双方向のトレーサビリティを検証し更新する。

テスト完了

テスト完了の活動では、完了したテストの全活動のデータを集め、プロジェクトから得たこと、テストウェア、およびその他の関連する情報すべてをまとめる。テスト完了活動は、ソフトウェアシステムがリリースされたとき、テストプロジェクトが完了（または打ち切り）したとき、アジャイルプロジェクトのイテレーションが（例えば、振り返りミーティングの一環として）完了したとき、テストレベルが完了したとき、メンテナンス版のリリースが完了したときなどに実施する。

テスト完了の主な活動は以下の通りである。

- すべての欠陥レポートがクローズしていることを確認する。または、テスト実行の終了時に未解決として残されている欠陥について変更要求またはプロダクトバックログアイテムを作成する。
- テストサマリーレポートを作成して、ステークホルダーに提出する。
- テスト環境、テストデータ、テストインフラストラクチャー、その他のテストウェアを次回も使えるように整理し保管する。

- テストウェアをメンテナンスチーム、他のプロジェクトチーム、および／またはその使用により利益を得る可能性のある他のステークホルダーに引き継ぐ。
- 完了したテスト活動から得られた教訓を分析し、次回のイテレーションやリリース、プロジェクトのために必要な変更を決定する。
- 収集した情報をテストプロセスの成熟度を改善するために利用する。

1.4.3 テスト作業成果物

テスト作業成果物は、テストプロセスの一環として作成する。組織ごとにテストプロセスの現場での適用のされ方は大きく異なるため、テストプロセスで作成する作業成果物の種類、それらの体系や使われ方、成果物名も大きく異なる。本シラバスでは、前述したテストプロセス、本シラバスまたは ISTQB 標準用語集で説明する作業成果物に従う。テスト作業成果物のガイドラインとして、ISO 標準（ISO/IEC/IEEE 29119-3）を使用することもできる。

本節で説明するテスト作業成果物の多くは、テストマネジメントツールや欠陥マネジメントツールを使用して記録およびマネジメントができる（第 6 章を参照）。

テスト計画の作業成果物

テスト計画の作業成果物には、通常、1 つ以上のテスト計画書が含まれる。テスト計画書にはテストベースに関する情報が含まれる。他のテスト作業成果物としてはトレーサビリティに関する情報（以降の説明と 1.4.4 節を参照）や、テストのモニタリングとコントロールで使用する終了基準（または完了（done）の定義）がある。テスト計画書は 5.2 節で説明する。

テストのモニタリングとコントロールの作業成果物

テストのモニタリングとコントロールの作業成果物には、一般的にさまざまな種類のテストレポートが含まれる。これらのテストレポートには、（随時、および／または定期的に作成する）テスト進捗レポートと（さまざまな完了マイルストーンで作成する）テストサマリーレポートの両方が含まれる。すべてのテストレポートは、レポート作成時点で提供可能になったテスト実行結果の要約を含む、読み手に合わせたテスト進捗についての詳細情報を提供すべきである。

テストのモニタリングとコントロールの作業成果物は、タスクの完了、リソースの割り当てや使用状況、工数などプロジェクトマネジメントにて取り扱う事項にも言及する。

テストのモニタリングとコントロール、およびこれらの活動で作成する作業成果物については、本シラバスの 5.3 節でさらに説明する。

テスト分析の作業成果物

テスト分析の作業成果物には、決定して優先順位を付けたテスト条件が含まれる。これらのテスト条件は、カバーするテストベースの具体的な要素との間で双方向のトレーサビリティを確立することが理想である。探索的テストのテスト分析では、テストチャーターを作成する場合がある。テスト分析では、テストベースに含まれる欠陥を検出した結果をレポートする場合もある。

テスト設計の作業成果物

テスト設計の結果は、テスト分析で決定したテスト条件を実行するためのテストケースとテストケースのセットになる。具体的な入力データと期待結果の値を記載しない高位レベルテストケースを設計することがよい実践例だといえる。高位レベルテストケースは、テストサイクルに応じた具体的な値を指定して複数のテストサイクルで再利用できる上に、テストケースの範囲を適切に文書化できる。各テストケースは、カバーするテスト条件との間で双方向のトレーサビリティを確立することが理想である。

テスト設計では、必要なテストデータの設計、および／または識別も行う。また、テスト環境の設計およびインフラストラクチャーとツールの識別も行う。ただし、これらを文書化する度合いは、状況によって大幅に異なる。

テスト設計では、テスト分析で決定したテスト条件をさらに洗練する場合もある。

テスト実装の作業成果物

テスト実装の作業成果物は、以下の通りである。

- テスト手順とそれらの順序付け
- テストスイート
- テスト実行スケジュール

テスト実装が完了した際には、テストケースとテスト条件の全体について、テスト手順とテストベースの特定の要素との間の双方向のトレーサビリティを介して、テスト計画書で定めたカバレッジ基準の達成度が把握できるようになることが理想である。

テスト実装では、ツールを使用して作成する作業成果物、例えばサービス仮想化などを作成することもあれば、ツールで使用される作業成果物、例えば自動化されたスクリプトなどを作成することもある。

テスト実装では、テストデータやテスト環境の作成や検証を行うこともある。データおよび／または環境の検証結果に関するドキュメントの完成度は、状況によって大幅に異なる。

テストデータは、テストケースの具体的な入力値や期待結果を割り当てる役割となる。具体値と値に関する明示的な指示で、高位レベルテストケースを実行可能な低位レベルテストケースにできる。同じ高位レベルテストケースであっても、異なるリリースのテスト対象で実行する場合、異なるテストデータを使用する場合がある。具体的なテストデータに関連する具体的な期待結果は、テストオラクルを使用して識別する。

探索的テストでは、テストケースとテストベースが持つ特定の要素に対するトレーサビリティが文書化される度合いは、状況によって大幅に異なり、テスト設計とテスト実装の作業成果物をテスト実行時に作成することもある。

テスト実装では、テスト分析で決定したテスト条件をさらに洗練する場合もある。

テスト実行の作業成果物

テスト実行の作業成果物は、以下の通りである。

- テストケースまたはテスト手順のステータスに関するドキュメント（例えば、テスト実行可能、合格、不合格、ブロック、意図的にスキップなど）
- 欠陥レポート（5.6 節を参照）
- テストアイテム、テスト対象、テストツール、テストウェアに関するドキュメント

テスト実行の活動が完了した際には、テスト手順との間の双方向のトレーサビリティを介して、テストベースの各要素のステータスが判定および報告できることが理想である。例えば、要件ごとに、計画したすべてのテストが合格か不合格か、そして／または関連付けられた欠陥があるか、そして計画したテストをすべて完了したかを報告できる。トレーサビリティにより、テストカバレッジ基準が満たされたことを検証でき、テストの結果をステークホルダーが理解できる言葉で報告できる。

テスト完了の作業成果物

テスト完了の作業成果物には、テストサマリーレポート、後続するプロジェクトまたはイテレーションを改善するためのアクションアイテム（アジャイルプロジェクトの振り返りから得られた教訓に従うことなど）、変更要求またはプロダクトバックログアイテム、変更に対応して最終的に完成したテストウェアなどがある。

1.4.4 テストベースとテスト作業成果物との間のトレーサビリティ

1.4.3 節で説明しているように、テストの作業成果物とそれらの名称は、状況によって大きく異なる。効果的なテストのモニタリングとコントロールを実装するためには、名称の違いがあるにしろ、前述した通り、テストベースの各要素、および関連するテスト作業成果物との間のトレーサビリティを、テストプロセス全体を通して確立、および維持することが重要である。テストカバレッジの評価に加えて、優れたトレーサビリティが役に立つことは以下の通りである。

- 変更の影響度を分析する。
- テストを監査可能にする。
- IT ガバナンス基準を満たす。
- テストベースの要素のステータス（例えば、テストが合格となった要件、テストが不合格となった要件、テストを保留している要件）を含めることで、テスト進捗レポートとテストサマリーレポートの理解しやすさを向上する。
- テストの技術的な側面をステークホルダーにとってなじみのある言葉で説明する。
- ビジネスゴールに対するプロダクトの品質、プロセス能力およびプロジェクト進捗の評価に関する情報を提供する。

本節で概説しているテスト作業成果物の一部、またはすべてに合致するテスト作業成果物モデルを提供するテストマネジメントツールもある。また、独自のマネジメントシステムを構築し、作業成果物を体系化して、必要な情報のトレーサビリティを確保している組織もある。

1.5 テストの心理学

ソフトウェア開発は、ソフトウェアテストを含めて人が行うため、人の心理がソフトウェアテストに重要な影響を及ぼす。

1.5.1 人の心理とテスト

要件のレビュー、およびユーザーストーリーの洗練をするセッションなど、静的テスト中に欠陥を見つけることや、動的テスト実行中に故障を見つけることは、プロダクトおよびプロダクトの開発担当者に対する非難と解釈されることがある。確証バイアスと呼ばれる人の心理の要素は、持っている信念に合わない情報を受け入れがたくする。例えば、開発担当者は、コードは正しいという思い込みがあるので、確証バイアスにより、コードが正しくないということを受け入れがたい。確証バイアスに加えて、他の認知バイアスがテストからの情報の理解または受け入れを困難にする場合がある。さらに、テストからの情報は悪いニュースを含んでいることが多く、悪いニュースをもたらす人は非難される傾向がある。

テストはプロジェクトの進捗とプロダクトの品質に大きく貢献するが、これらの心理的要因により、破壊的な活動と見られる場合がある（1.1 節と 1.2 節を参照）。これらの心理的傾向を軽減するためには、欠陥や故障に関する情報を建設的な方法で伝えるべきである。こうすることで、テスト担当者とアナリスト、プロダクトオーナー、設計者、開発担当者との間の緊張を緩和できる。このことは、静的テストと動的テストの両方に当てはまる。

テスト担当者およびテストマネージャーは、欠陥、故障、テスト結果、テストの進捗、リスクの情報を効果的に伝えることができ、同僚と建設的な関係を構築するための優れた対人スキルが必要となる。コミュニケーションを適切に行う方法には以下のものがある。

- 対決ではなく、協調姿勢で開始する。全員のゴールは、高品質のシステムであることを再認識するとよい。
- テストの利点を強調する。例えば、開発担当者に対しては、欠陥情報は、作業成果物の品質向上と開発担当者のスキル向上に役立つ。組織に対しては、欠陥をテストで検出して修正すると、時間と経費の節約になり、プロダクト品質の全体的なリスクも減る。
- テスト結果や他の発見事項は、中立な立場で、事実に焦点をあてて伝え、欠陥を作りこんだ担当者を非難しないようにする。客観的かつ事実に基づいた欠陥レポートを書いたり、発見事項をレビューしたりする。
- 他人の気持ちや、他人が情報に対して否定的に反応した理由を理解するように努力する。
- 自分の言ったことを他人が理解し、他人の言ったことを自分が理解していることを確認する。

典型的なテスト目的については 1.1 節で前述しているが、テスト目的を適切かつ明確に定義することが心理的に大きな影響を持つ。人は、チーム、マネジメント、他のステークホルダーが設定した目的に自身の計画と振る舞いを合わせる傾向がある。テスト担当者も個人的偏見を最小限にして、これらの目的に忠実であることが重要である。

1.5.2 テスト担当者と開発担当者のマインドセット

開発担当者とテスト担当者は、異なった考え方をすることが多い。開発担当者の主目的は、プロダクトを設計して構築することである。テストの目的は、これまでに説明しているように、プロダクトの検証と妥当性確認、リリース前の欠陥の検出などである。このように目的が異なっており、異なるマインドセットを必要とする。両者のマインドセットを組み合わせることで、より高いレベルのプロダクト品質を達成できる。

マインドセットによって、意志決定と問題解決において前提とするものと優先する方法が異なる。テスト担当者のマインドセットには、好奇心、プロとしての悲観的な考え、批判的な視点、細部まで見逃さない注意力、良好で建設的なコミュニケーションと関係を保つモチベーションが必要となる。テスト担当者のマインドセットは経験を積むに従って成長し成熟する傾向がある。

開発担当者のマインドセットはテスト担当者と同じ要素をいくつか含むこともあるが、開発担当者は、解決策の設計と構築により大きな関心があり、解決策に誤りがあるかには関心があまりない。また、確証バイアスが、自分自身の作業中に犯した誤りを見つけることを困難にする。

開発担当者は、正しいマインドセットを持って自身のコードをテストする必要がある。さまざまなソフトウェア開発ライフサイクルモデルに応じて、テスト担当者とテスト活動を編成する方法は異なることが多い。特に大規模、複雑、またはセーフティクリティカルなシステムにおいては、独立したテスト担当者がテスト活動のいくつかを担うと効果的な欠陥検出ができる。独立したテスト担当者は、作業成果物の作成者（ビジネスアナリスト、プロダクトオーナー、設計者、プログラマー）とは異なる認知バイアスと観点を持つためである。

2 ソフトウェア開発ライフサイクル全体を通してのテスト | 100 分

キーワード

受け入れテスト、アルファテスト、ベータテスト、市販ソフトウェア（COTS）、コンポーネント統合テスト、コンポーネントテスト、確認テスト、契約による受け入れテスト、機能テスト、影響度分析、統合テスト、メンテナンステスト、非機能テスト、運用受け入れテスト、リグレッションテスト、規制による受け入れテスト、シーケンシャル開発モデル、システム統合テスト、システムテスト、テストベース、テストケース、テスト環境、テストレベル、テスト対象、テスト目的、テストタイプ、ユーザー受け入れテスト、ホワイトボックステスト

本章の学習の目的

2.1 ソフトウェア開発ライフサイクルモデル

FL-2.1.1 （K2）ソフトウェア開発ライフサイクルにおけるソフトウェア開発活動とテスト活動の関係を説明する。

FL-2.1.2 （K1）ソフトウェア開発ライフサイクルモデルを、プロジェクトやプロダクトの特性に合わせて調整する必要がある理由を識別する。

2.2 テストレベル

FL-2.2.1 （K2）目的、テストベース、テスト対象、典型的な欠陥と故障、アプローチや責務の観点から、さまざまなテストレベルを比較する。

2.3 テストタイプ

FL-2.3.1 （K2）機能テスト、非機能テスト、ホワイトボックステストを比較する。

FL-2.3.2 （K1）あらゆるテストレベルで機能テスト、非機能テスト、ホワイトボックステストが行われることを認識する。

FL-2.3.3 （K2）確認テストとリグレッションテストの目的を比較する。

2.4 メンテナンステスト

FL-2.4.1 （K2）メンテナンステストを引き起こすものを要約する。

FL-2.4.2 （K2）メンテナンステストにおける影響度分析の役割を説明する。

2.1 ソフトウェア開発ライフサイクルモデル

ソフトウェア開発ライフサイクルモデルは、ソフトウェア開発プロジェクトの各段階で実施する活動の種類と、各活動間の論理的および時系列的な関連を表している。ソフトウェア開発ライフサイクルモデルには多くの種類があり、モデルが異なれば、テストに対するアプローチも異なる。

2.1.1 ソフトウェア開発とソフトウェアテスト

ソフトウェア開発ライフサイクルモデルを正しく理解し、適切にテスト活動を行うことができるようにすることは、テスト担当者の役割全体の中でも重要な部分となる。

どのようなソフトウェア開発ライフサイクルモデルにも、よいテストを行うときの特徴がいくつかある。

- 各開発活動に対応してテスト活動がある。
- 各テストレベルには、そのレベル特有の目的がある。
- 各テストレベルのテスト分析や設計は、対応する開発活動の間に開始する。
- テスト担当者は要件と設計を定義および洗練する討議に参加し、作業成果物（要件、設計、ユーザーストーリーなど）のドラフトができたらすぐにそのレビューに関与する。

早期テストのテスト原則に従い、どのようなソフトウェア開発ライフサイクルモデルであるかに関係なく、テスト活動はライフサイクルの早い段階に開始すべきである。

本シラバスでは、一般的なソフトウェア開発ライフサイクルモデルを以下のモデルに分類する。

- シーケンシャル開発モデル
- イテレーティブ開発モデルとインクリメンタル開発モデル

シーケンシャル開発モデルは、ソフトウェア開発プロセスを順次実行する活動として表す。これは、開発プロセスのあらゆるフェーズが直前のフェーズの完了とともに始まることを意味する。理論的にはフェーズが重なることはないが、実際には、後続のフェーズからフィードバックを早期に行うのが効果的である。

ウォーターフォールモデルでは、開発活動（例えば、要件分析、設計、コーディング、テストなど）は、逐次完了する。このモデルでは、テスト活動はその他の開発活動がすべて完了した後に実行する。

V 字モデルは、ウォーターフォールモデルと異なり、開発プロセス全体にテストプロセスを統合しており、早期テストの原則を実装している。また、V 字モデルでは、早期テストの裏付けとして、各開発フェーズにテストレベルが対応している（テストレベルについては、2.2 節を参照）。テスト実行は、関連付けられているテストレベルごとに順次進めていくが、重複することもある。

シーケンシャル開発モデルでは、フィーチャーが完全に揃ったソフトウェアを提供するが、ステークホルダーやユーザーが利用できるまでには、典型的に数か月から数年を要する。

インクリメンタル開発は、システムを分割し、分割単位ごとに要件の確定、設計、構築、テストを行う。これは、ソフトウェアのフィーチャーが徐々に増加していくことを意味する。増加させていくフィーチャーの分割単位の大きさは手法によってさまざまであり、ユーザーインターフェース画面の 1 つの変更や新しい検索条件の追加のように小さい場合もある。

イテレーティブ開発モデルでは、グループにしたフィーチャー群を、一連のサイクル（多くの場合、固定された期間）の中で一緒に仕様化、設計、構築、テストする。イテレーション（反復）では、プロジェクトで決めた変更範囲に加えて、以前のイテレーションで開発したフィーチャーの変更を含める場合

がある。各イテレーションでは、動作するソフトウェアが提供される。提供されるソフトウェアは、段階的にフィーチャーを追加して成長させていく。完成したソフトウェアが提供されるか、開発が中止するまでイテレーションを継続する。

この例には、以下のようなものがある。

- ラショナル統一プロセス（RUP）：各イテレーションは他と比べて長期（例えば 2～3 か月）になる傾向にあり、フィーチャー増分は、関連するフィーチャーで構成するいくつかのグループなど、期間に対応して大きくなる傾向にある。
- スクラム：各イテレーションは他と比べて短期（例えば、数時間、数日、数週間）になる傾向にあり、フィーチャーの増分は、わずかな機能強化、および／またはいくつかの新しいフィーチャーなど、小さくなる傾向にある。
- カンバン：イテレーションの期間の長さが固定されているかどうかに関係なく実装できるので、完了時に単一の機能強化やフィーチャーをリリースすることも、複数のフィーチャーをまとめて一括してリリースすることもできる。
- スパイラル（プロトタイピング）：増分を試行しながら追加する。増分は、後続の開発作業で大幅に作り直すことも破棄することもある。

これらの手法を使用してコンポーネントやシステムを開発する場合、開発全体を通してテストレベルの重複や繰り返しが発生することが多い。各フィーチャーがイテレーションを通して提供されるまでに複数のテストレベルでテストされていることが望ましい。チームは、複数のテストレベルを大幅に自動化するために、継続的デリバリーまたは継続的デプロイメントのデリバリーパイプラインを使用する場合がある。これらの手法を導入するチームには、これまでのテストにおける仕事内容の変更、およびテスト担当者と開発担当者の関係を変更できる自己組織化されたチームの概念が必要となる。

これらの手法は、システムの段階的な成長を可能にする。段階的に成長するシステムは、フィーチャーごと、イテレーションごと、または従来のようにメジャーリリースの形態でエンドユーザーに提供される場合がある。ソフトウェアを分割単位でエンドユーザーにリリースするかは別として、システムが成長するにつれてリグレッションテストの重要性は増大する。

シーケンシャルモデルとは対照的に、イテレーティブモデルおよびインクリメンタルモデルでは、使用可能なソフトウェアを数週間、極端な場合には数日で提供できるが、要件が完全に満たされた製品の提供には数か月から極端な場合には数年を要する。

アジャイル開発におけるソフトウェアテストの詳細については、『ISTQB テスト技術者資格制度 Foundation Level Extension シラバス アジャイルテスト担当者 日本語版』、『Black 2017』、『Crispin 2008』、『Gregory 2015』を参照されたい。

2.1.2 状況に応じたソフトウェア開発ライフサイクルモデル

ソフトウェア開発ライフサイクルモデルは、プロジェクトやプロダクトの特性に応じて選択および調整する必要がある。ソフトウェア開発ライフサイクルモデルの選択と調整は、プロジェクトのゴール、開発するプロダクトの種類、ビジネス上の優先度（市場に参入するまでの時間など）、識別したプロダクトリスクおよびプロジェクトリスクに基づいて適切に行わなければならない。例えば、小規模な内部向けの管理システムの開発とテストは、自動車のブレーキ制御システムなどのセーフティクリティカルシステムの開発とテストとは異なる。別の例としては、組織上および文化的な問題によりチームメンバー間のコミュニケーションが阻害され、イテレーティブ開発が妨げられる場合がある。

プロジェクトの状況によっては、テストレベルおよび／またはテスト活動を組み合わせたり再編成したりすることが必要である。例えば、市販ソフトウェア（COTS）製品を大規模なシステムに組み込む場合、システム統合テストレベル（インフラストラクチャーやその他のシステムとの統合）と受け入れテストレベル（ユーザー受け入れテストや運用受け入れテストでの、機能テストや非機能テストと併せて）で

相互運用性テストを購入者が実施する必要がある。テストレベルの詳細については 2.2 節を、テストタイプの詳細については 2.3 節を参照されたい。

さらに、ソフトウェア開発ライフサイクルモデルを組み合わせる場合がある。例えば、バックエンドシステムの統合のための開発とテストに V 字モデルを使用し、フロントエンドのユーザーインターフェース（UI）と機能の開発とテストにアジャイル開発モデルを使用する場合がある。プロジェクトの初期ではプロトタイピングを使用し、試行フェーズが完了した後ではインクリメンタル開発モデルを採用する場合がある。

IoT（モノのインターネット）システムは、デバイス、プロダクト、サービスなどの多くのさまざまな開発対象で構成されるが、開発対象ごとに個別のソフトウェア開発ライフサイクルモデルを適用するのが一般的である。このことは、IoT システムを複数バージョン開発するときに特化した課題があることを示している。さらに、各開発対象のソフトウェア開発ライフサイクルでは、後期となる本番運用開始後のフェーズ（運用フェーズ、更新フェーズ、廃止フェーズ）がきわめて重要になる。

2.2 テストレベル

テストレベルは、系統的にまとめ、マネジメントしていくテストの活動のグループである。各テストレベルは、1.4 節で説明した活動で構成されるテストプロセスのインスタンスであり、個々のユニットまたはコンポーネントから完成したシステムまでの特定の開発レベル、または必要に応じてシステムオブシステムズまでの開発レベルのソフトウェアに関連している。テストレベルはソフトウェア開発ライフサイクル内の他の活動と関連付けられる。本シラバスでは、以下のテストレベルを使用する。

- コンポーネントテスト
- 統合テスト
- システムテスト
- 受け入れテスト

テストレベルは、以下の属性で特徴付けられる。

- 特定の目的
- テストベース（テストケースの導出時に参照）
- テスト対象（すなわち、テストするものは何か）
- 典型的な欠陥と故障
- （そのテストレベルでの）アプローチと責務

テストレベルそれぞれに適切なテスト環境が必要である。例えば、受け入れテストでは本番環境に類似したテスト環境が理想であり、コンポーネントテストでは、開発担当者は彼ら所有の開発環境を使用するのが一般的である。

2.2.1 コンポーネントテスト

コンポーネントテストの目的

コンポーネントテスト（ユニットテストまたはモジュールテストとも呼ばれる）は、個別にテスト可能なコンポーネントに焦点をあてる。コンポーネントテストの目的は以下の通りである。

- リスクの軽減
- コンポーネントの機能的/非機能的振る舞いが設計および仕様通りであることの検証
- コンポーネント品質に対する信頼の積み上げ
- コンポーネントに存在する欠陥の検出
- 欠陥がより高いテストレベルまで見逃されることの防止

特にコード変更が継続して行われるインクリメンタル開発モデルやイテレーティブ開発モデル（アジャイルなど）では、コンポーネントテストのリグレッションテストを自動化して、変更が既存のコンポーネントを破壊していないという信頼を積み重ねていくことが重要である。

多くの場合、コンポーネントテストは、ソフトウェア開発ライフサイクルモデルやシステムの特性により、システムの他の部分と切り離したテストが可能である。この場合、モックオブジェクト、サービス仮想化、ハーネス、スタブ、ドライバなどを使う。コンポーネントテストは、機能性（例えば計算の正しさ）、非機能特性（例えばメモリリークの検出）、構造の属性（例えばデシジョンテスト）も含む。

テストベース

コンポーネントテストでテストベースとして使用できる作業成果物の例：

- 詳細設計
- コード
- データモデル
- コンポーネント仕様

テスト対象

コンポーネントテストの典型的なテスト対象の例：

- コンポーネント、ユニット、またはモジュール
- コードとデータ構造
- クラス
- データベースモジュール

典型的な欠陥と故障

コンポーネントテストの典型的な欠陥と故障の例：

- 正しくない機能（例えば、設計仕様の記述とは異なる）
- データフローの問題
- 正しくないコードとロジック

検出した欠陥は、形式に沿った欠陥マネジメントを行わず、見つけたらすぐに修正するのが一般的である。ただし、開発担当者が欠陥を報告すると、根本原因分析やプロセス改善にとって重要な情報が得られることになる。

アプローチと責務

コンポーネントテストはコードを開発した開発担当者が行うことが一般的であるが、そうでない場合、少なくともテスト対象のコードにアクセスできる必要がある。開発担当者は欠陥の検出と修正を行って、開発したコンポーネントを差し替えることができる。また、コンポーネント用のコードを記述した後で、テストを記述し実行することが多い。ただし、特にアジャイル開発では、アプリケーションコードを記述する前に、自動化されたコンポーネントのテストケースを記述する場合がある。

例えば、高度にイテレーティブであるテスト駆動開発（TDD）のサイクルは、自動化されたテストケースの開発、小規模なコードの構築と統合、コンポーネントテストの実行、問題の修正、コードのリファクタリングで構成される。このプロセスは、コンポーネントを完全に構築し、すべてのコンポーネントテストが合格となるまで継続する。テスト駆動開発は、テストファーストアプローチの一例である。テスト駆動開発はエクストリームプログラミング（XP）がオリジナルであるが、アジャイル開発の他のアプローチおよびシーケンシャルライフサイクルにも普及している（『ISTQB テスト技術者資格制度 Foundation Level Extension シラバス アジャイルテスト担当者 日本語版』を参照）。

2.2.2 統合テスト

統合テストの目的

統合テストは、コンポーネントまたはシステム間の相互処理に焦点をあてる。統合テストの目的は以下の通りである。

- リスクの軽減
- インターフェースの機能的/非機能的振る舞いが設計および仕様通りであることの検証
- インターフェース品質に対する信頼の積み上げ
- 欠陥の検出（インターフェース自体、コンポーネントに内在、またはシステムに内在）
- 欠陥がより高いテストレベルまで見逃されることの防止

コンポーネントテストと同様に、統合テストのリグレッションテストを自動化する場合があり、自動化したリグレッションテストにより、変更が既存のインターフェース、コンポーネント、システムを破壊しないという信頼を提供できる。

本シラバスで説明しているように、統合テストには 2 つの異なるレベルがあり、以下のようにさまざまな大きさのテスト対象物に対して実施する。

- コンポーネント統合テストは、統合するコンポーネント間の相互処理とインターフェースに焦点をあててテストする。コンポーネントテストの後に実施し、自動化するのが一般的である。イテレーティブ開発およびインクリメンタル開発では、継続的インテグレーションプロセスの一環として行うことが一般的である。
- システム統合テストは、システム、パッケージ、マイクロサービス間の相互処理とインターフェースに焦点をあててテストする。外部組織（Web サービスなど）との相互処理、または外部組織により提供されるインターフェースをカバーすることもある。この場合、開発組織は外部インターフェースを制御できない。このため、テストにさまざまな課題が発生することがある（テストをブロックする欠陥が外部組織のコードに存在する場合の解決されたことの確認、テスト環境の調整など）。システム統合テストは、（シーケンシャル開発とイテレーティブ/インクリメンタル開発の両方で）システムテストの後、または実行中のシステムテスト活動と並列に行う。

テストベース

統合テストでテストベースとして使用できる作業成果物の例：

- ソフトウェア設計とシステム設計
- シーケンス図
- インターフェースと通信プロトコルの仕様
- ユースケース
- コンポーネントレベルまたはシステムレベルのアーキテクチャー
- ワークフロー
- 外部インターフェース定義

テスト対象

統合テストの典型的なテスト対象の例：

- サブシステム
- データベース
- インフラストラクチャー
- インターフェース
- API
- マイクロサービス

典型的な欠陥と故障

コンポーネント統合テストの典型的な欠陥と故障の例：

- 正しくないデータ、データ不足、正しくないデータエンコーディング
- インターフェース呼び出しの正しくない順序またはタイミング
- インターフェースの不整合
- コンポーネント間のコミュニケーション故障
- コンポーネント間のコミュニケーション故障の処理不在、または不適切な処理
- コンポーネント間で渡されるデータの意味、単位、境界の正しくない思い込み

システム統合テストの典型的な欠陥と故障の例：

- システム間で一貫性のないメッセージ構造
- 正しくないデータ、データ不足、正しくないデータエンコーディング
- インターフェースの不整合
- システム間通信による故障
- システム間通信処理不在、または不適切な処理による故障
- システム間で渡されるデータの意味、単位、境界の正しくない思い込み
- 必須のセキュリティ規制への非準拠

アプローチと責務

コンポーネント統合テストとシステム統合テストは、統合だけに集中すべきである。例えば、モジュール A とモジュール B の統合をテストする場合、コンポーネントテストでカバー済みの個々のモジュールの機能ではなく、2 つのモジュール間のコミュニケーションに焦点をあてる必要がある。同様に、システム X とシステム Y の統合をテストする場合、システムテストでカバー済みの個々のシステムの機能ではなく、2 つのシステム間のコミュニケーションに焦点をあてる必要がある。テストタイプのうち、機能テスト、非機能テスト、構造テストは統合テストで行うことができる。

コンポーネント統合テストは開発担当者が実行するのが一般的である。システム統合テストはテスト担当者が実行するのが一般的である。理想的には、システム統合テストを実行するテスト担当者はシステムアーキテクチャーを理解すべきであり、統合計画の作成に関与すべきである。

コンポーネントまたはシステムの構築前に統合テストと統合戦略を計画すると、テストの効率が最大になる順序でコンポーネントとシステムを構築できる。体系的に統合する戦略は、システムアーキテクチャー（トップダウンやボトムアップなど）、機能的タスク、トランザクション処理シーケンス、その他システムやコンポーネントの側面がベースとなる。欠陥の特定を容易にし、早期に検出するために、統合は「ビッグバン（すべてのコンポーネントやシステムに対して一度に実施）」ではなく、インクリメンタル（同時には少数のコンポーネントまたはシステムを追加）するのが普通である。複雑なインターフェースに対するリスク分析は統合テストの焦点をどこにあてるかに役立つ。

統合の範囲が大きくなるほど、欠陥を特定のコンポーネントまたはシステムに絞り込むことが難しくなる。これによりリスクが増え、トラブルシューティングの時間が増大する。これが、ソフトウェアをコンポーネントごとに統合（すなわち機能統合）する継続的インテグレーションが普及してきている理由の 1 つである。このような継続的インテグレーションでは、リグレッションテストの自動化を複数のテストレベルで行うことが理想である。

2.2.3 システムテスト

システムテストの目的

システムテストは、システムが実行するエンドツーエンドのタスクと、タスクの実行時にシステムが示す非機能的振る舞いといったシステムやプロダクト全体の振る舞いや能力に焦点をあてる。システムテストの目的は以下の通りである。

- リスクの軽減
- システムの機能的/非機能的振る舞いが設計および仕様通りであることの検証
- システムが完成し、期待通りに動作することの妥当性確認
- システムの全体的な品質に対する信頼の積み上げ
- 欠陥の検出
- 欠陥がより高いテストレベルまたは運用環境まで見逃されることの防止

信頼できるシステムとなるためには、データ品質の検証も目的とする。コンポーネントテストや統合テストと同様に、システムテストのリグレッションテストを自動化する場合があり、自動化したリグレッションテストにより変更が既存のフィーチャーやエンドツーエンドの能力を破壊していないという信頼を提供できる。ステークホルダーは、システムテストの情報に基づいてリリースに関する意志決定を行うことが多い。システムテストでは、法的もしくは規制的要件、または標準を満たすことを確認する場合もある。 テスト環境は、最終的なターゲットまたは本番環境に準じることが理想である。

テストベース

システムテストでテストベースとして使用できる作業成果物の例：

- システム要求仕様およびソフトウェア要求仕様（機能および非機能）
- リスク分析レポート
- ユースケース
- エピックおよびユーザーストーリー
- システム振る舞いのモデル
- 状態遷移図
- システムマニュアルおよびユーザーマニュアル

テスト対象

システムテストの典型的なテスト対象の例：

- アプリケーション
- ハードウェア/ソフトウェアシステム
- オペレーティングシステム
- テスト対象システム（SUT）
- システム構成および構成データ

典型的な欠陥と故障

システムテストの典型的な欠陥と故障の例：

- 正しくない計算
- 正しくない、または期待通りではないシステムの機能的または非機能的振る舞い
- システム内の正しくない制御フローおよび／またはデータフロー
- エンドツーエンドの機能的タスクを適切かつ完全に実行できない
- 本番環境でシステムが適切に動作しない
- システムマニュアルおよびユーザーマニュアルに記載されている通りにシステムが動作しない

アプローチと責務

システムテストは、システム全体の機能と非機能の両方について、エンドツーエンドの全体的な振る舞いに焦点をあてる。システムテストでは、対象となるシステムにとって最適なテスト技法（第 4 章を参照）を使用する。例えば、デシジョンテーブルを使用して、機能的な振る舞いがビジネスルールで規定されている通りであることを検証する場合がある。

独立したテストチームがシステムテストを行うことが多い。仕様の欠陥（例えば、ユーザーストーリーの欠落、ビジネス要件の正しくない記述など）は、システムの期待される振る舞いに関する正しい理解の欠落や誤解につながる。この状況が偽陽性や偽陰性の結果を招き、時間の浪費や欠陥検出効率の低下となる。テスト担当者がユーザーストーリーを洗練する活動およびレビューなどの静的テストの活動に早期から関与することで、このような状況が起きることを削減できる。

2.2.4 受け入れテスト

受け入れテストの目的

システムテストと同様、一般的に受け入れテストはシステムやプロダクト全体の振る舞いや能力に焦点をあてる。受け入れテストの目的は以下の通りである。

- システムの全体的な品質に対する信頼の積み上げ
- システムが完成し、期待通りに動作することの妥当性確認
- システムの機能的/非機能的振る舞いが仕様通りであることの検証

受け入れテストからの情報に基づいて、システムが導入に向けて準備できているかどうか、および顧客（エンドユーザー）が使用できるかを評価できる。受け入れテストで欠陥が見つかることはあるが、欠陥を見つけることは目的ではない。とても多くの欠陥が受け入れテスト時に検出されるような事態は、重要なプロジェクトリスクと見なすことができる。受け入れテストでは、法的または規制的要件または標準を満たすことを確認する場合もある。

受け入れテストの一般的な形態は以下の通りである。

- ユーザー受け入れテスト
- 運用受け入れテスト
- 契約による受け入れテストおよび規制による受け入れテスト
- アルファテストおよびベータテスト

それぞれの形態について、以降の節で説明する。

ユーザー受け入れテスト

ユーザーが行うシステムの受け入れテストは、意図されているユーザーが本番環境または（模擬の）運用環境にて、想定しているシステムの使用方法と合致していることを確認することに焦点をあてている。ユーザー受け入れテストの主目的は、システムの使用において、システムがユーザーニーズに合致し、要件を満たし、最小限の困難さ、コスト、リスクでビジネスプロセスを実行できるという信頼を積み重ねることである。

運用受け入れテスト

運用担当者またはシステムアドミニストレーターが行うシステムの受け入れテストは、（模擬）本番環境で行うのが一般的である。運用面に焦点を置いている。このテストには、以下の種類がある。

- バックアップ/リストアのテスト
- インストール、アンインストール、アップグレード
- 災害復旧
- ユーザーマネジメント
- メンテナンスタスク
- データのロードと移行のタスク
- セキュリティの脆弱性のチェック
- 性能テスト

運用受け入れテストの主目的は、運用担当者またはシステムアドミニストレーターが、運用環境で（例外的な状況または困難な状況であっても）ユーザー向けにシステムを適切な稼動状態に維持できるという信頼を積み重ねることである。

契約による受け入れテストおよび規制による受け入れテスト

契約による受け入れテストは、カスタムメイドのソフトウェアを開発する場合に、契約書に記述した受け入れ基準に従って行う。受け入れ基準は、契約時に当事者間で合意している。契約による受け入れテストは、ユーザーまたは独立したテスト担当者が行うことが多い。

規制による受け入れテストでは、政府、法律、安全の基準などに合致しているかを確認する。規制による受け入れテストは、ユーザーまたは独立したテスト担当者が行うことが多いが、規制機関が結果を証明または監査する場合もある。

契約による受け入れテストおよび規制による受け入れテストの主目的は、契約または規制に準拠しているという信頼を積み重ねることである。

アルファテストおよびベータテスト

アルファテストおよびベータテストは、市販ソフトウェア（COTS）の開発担当者がソフトウェアプロダクトを市場へリリースする前に、潜在的または既存のユーザー、顧客、および／またはシステムの運用者からフィードバックを受けるために行う。アルファテストは、開発組織内にて開発チームの代わりに潜在的または既存の顧客、および／またはシステムの運用者もしくは独立したテストチームが行う。ベータテストは、潜在的または既存の顧客、および／またはシステムの運用者が彼ら自身の作業場所で行う。ベータテストはアルファテストの後に行うことも、アルファテストなしに行うこともある。

アルファテストおよびベータテストの目的の 1 つは、潜在的または既存の顧客、および／またはシステムの運用担当者が日常的な状況において、運用環境でシステムを使う上での困難、コストおよびリスクが最小限で目的を達成できるという信頼を積み重ねることである。また、システムが使われる環境（特に、開発チームでは再現できない状況や環境）での欠陥を検出することも目的の 1 つである。

テストベース

さまざまな形態の受け入れテストでテストベースとして使用できる作業成果物の例：

- ビジネスプロセス
- ユーザー要件またはビジネス要件
- 規制、法的契約、標準
- ユースケース
- システム要件
- システムドキュメントまたはユーザードキュメント
- インストール手順
- リスク分析レポート

運用受け入れテストでは、テストケースを導くためのテストベースとして、以下の作業成果物も使用できる。

- バックアップ/リストアの手順
- 災害復旧手順
- 非機能要件

- 運用ドキュメント
- デプロイメント指示書およびインストール指示書
- 性能目標
- データベースパッケージ
- セキュリティ標準または規制

典型的なテスト対象

さまざまな形態の受け入れテストで典型的なテスト対象の例：

- テスト対象システム
- システム構成および構成データ
- 完全に統合されたシステムのビジネスプロセス
- リカバリーシステムおよび（ビジネス継続性および災害復旧のテスト用の）ホットサイト
- 運用プロセスおよびメンテナンスプロセス
- 書式
- レポート
- 既存または変換（コンバージョン）した本番データ

典型的な欠陥と故障

さまざまな形態の受け入れテストで典型的な欠陥と故障の例：

- システムのワークフローがビジネス要件またはユーザー要件を満たさない。
- ビジネスルールが正しく実装されていない。
- システムが契約要件または規制要件を満たさない。
- セキュリティの脆弱性、高負荷時の不適切な性能効率、サポート対象プラットフォームでの不適切な操作などの非機能特性の故障。

アプローチと責務

受け入れテストは、顧客、ビジネスユーザー、プロダクトオーナー、またはシステムの運用担当者の責任で行われる。他のステークホルダーも関与することがある。

受け入れテストは、シーケンシャル開発ライフサイクルでは最後のテストレベルであるが、以下のようなタイミングでも実行することがある。

- COTS ソフトウェアプロダクトの受け入れテストは、インストール時または統合時に行う場合がある。
- 機能を新たに強化したときの受け入れテストは、システムテストの前に行うことがある。

イテレーティブ開発において、プロジェクトチームは、新しい機能を受け入れ基準に対して検証し、ユーザーニーズを満たすことの妥当性確認を行うために、各イテレーションの実行中および終了時にさまざまな形態の受け入れテストを利用する。さらに、各イテレーションの最後および完了後、または一連のイテレーションの後に、アルファテストとベータテストを実行することがある。ユーザー受け入れテスト、運用受け入れテスト、契約による受け入れテストおよび規制による受け入れテストも、各イテレーションの最後や完了後、または一連のイテレーションの後に実行する場合がある。

2.3 テストタイプ

テストタイプは、以下に列挙する特定のテストの目的から見たソフトウェアシステム（あるいはシステムの一部分）の特性をテストするための活動を束ねたものである。

- 機能の品質特性、例えば完全、正確および適切であることなどを評価する。
- 非機能の品質特性、例えば信頼性、性能効率性、セキュリティ、互換性、使用性などを評価する。
- コンポーネントまたはシステムの、構造またはアーキテクチャーが正しく完全で仕様通りであることを評価する。
- 欠陥が修正されていることを確認するなどの変更による影響を評価し（確認テスト）、ソフトウェアや環境の変更によって意図しない振る舞いの変化が発生していないかを探す（リグレッションテスト）。

2.3.1 機能テスト

システムの機能テストでは、システムが実行する機能を評価する。機能要件は、ビジネス要件仕様、エピック、ユーザーストーリー、ユースケース、機能仕様などの作業成果物で記述してあったり、まったく文書化していない場合もあったりする。機能とはシステムが「何を」すべきかである。

機能テストは、すべてのテストレベル（例えば、コンポーネント仕様を基にしたコンポーネントテスト）で行うべきである。各テストレベルで焦点は異なる（2.2 節を参照）。

機能テストはソフトウェアの振る舞いが関心事であり、コンポーネントまたはシステムの機能のためのテスト条件やテストケースをブラックボックス技法で導出できる（4.2 節を参照）。

機能テストは、機能カバレッジを用いてテストが十分かを計測できる。機能カバレッジは、テストした機能要素の種類の範囲を示し、カバーした要素の種類の割合で表す。例えば、テストと機能要件の間のトレーサビリティを使用して、テストで対処した要件の割合を計算し、テストが十分でないところを識別できる。

機能テストの設計および実行には、ソフトウェアが解決する特定のビジネス問題（例：石油・ガス業界での地質モデル化ソフトウェア）やソフトウェアが果たす役割（例：対話型エンターテイメントを提供するコンピューターゲーム用ソフトウェア）の知識など、特別なスキルや知識が必要な場合がある。

2.3.2 非機能テスト

システムの非機能テストでは、システムやソフトウェアの使用性、性能効率性、セキュリティといった特性を評価する。ソフトウェアプロダクトの品質特性の分類については、ISO/IEC 25010 を参照されたい。非機能テストは、システムが「どのように上手く」振る舞うかをテストする。

一般的に誤解されているが、非機能テストはすべてのテストレベルで行うことができる。そして可能な限り早期に行うべきである。非機能欠陥の検出が遅れることは、プロジェクトの成功にとってきわめて大きな危険となりえる。

非機能テストではテスト条件やテストケースを抽出するために、ブラックボックス技法（4.2 節を参照）を使う場合がある。例えば、境界値分析で性能テストのストレス条件を定義できる。

非機能テストは、非機能カバレッジを用いてテストが十分かを計測できる。非機能カバレッジは、テストした非機能要素の種類の範囲を示し、カバーした要素の種類の割合で表す。例えば、モバイルアプリケーションの場合に、テストとサポート対象デバイスの間のトレーサビリティを使用して、互換性テストで対処したデバイスの割合を計算し、テストが十分でないところを識別できる。

非機能テストの設計および実行には、設計や技術に特有の弱点（例えば、特定のプログラム言語に付随するセキュリティの脆弱性）や特定のユーザーに関する知識（例えば、医療施設管理システムのユーザーの特性）などの特別なスキルや知識が必要な場合がある。

非機能テストの詳細については、『ISTQB テスト技術者資格制度 Advanced Level シラバス 日本語版 テストアナリスト』、『ISTQB テスト技術者資格制度 Advanced Level シラバス 日本語版 テクニカルテストアナリスト』、『 ISTQB-SEC Advanced Level Security Tester Syllabus』、およびその他の ISTQB スペシャリストモジュールを参照されたい。

2.3.3 ホワイトボックステスト

ホワイトボックステストは、システムの内部構造や実装に基づいてテストを導出する。内部構造には、コード、アーキテクチャー、ワークフロー、そして／またはシステム内のデータフローなどがある（4.3 節を参照）。

ホワイトボックステストは、構造カバレッジを用いてテストが十分かを計測できる。構造カバレッジは、テストで実行した構造要素の種類の範囲を示し、カバーした要素の種類の割合で表す。

コンポーネントテストレベルでは、コンポーネント内のテスト済みのコードの割合に基づき、例えばコンポーネント内の実行可能ステートメントや判定結果のうちテストを実行した割合といった、コードのさまざまな側面（カバレッジアイテム）でカバレッジを計測できる。これらのカバレッジはコードカバレッジと呼ぶ。コンポーネント統合テストレベルでは、コンポーネント間のインターフェースなどシステムのアーキテクチャーに基づき、構造カバレッジはテストで実行済みのインターフェースの割合で計測できる。

ホワイトボックステストの設計および実行には、コードのビルド方法（例えば、コードカバレッジツールを使用するため）、データの格納方法（例えば、データベースクエリーを評価するため）、カバレッジツールの使用方法、およびその計測結果を正しく解釈する方法など、特別なスキルや知識が必要な場合がある。

2.3.4 変更部分のテスト

欠陥を修正、または機能の新規追加や変更といったシステムに対する変更をした場合、元の欠陥が修正されたこと、機能が正しく実装されていること、または予測しなかった悪い影響が発生していないことを確認するためにテストをしなければならない。

- 確認テスト：欠陥を修正したら、その欠陥により不合格となった全テストケースを新しいソフトウェアバージョンで再実行しなければならない。また、例えば欠陥が機能不足であった場合は、新しいテストケースでもテストしなければならない。少なくとも、欠陥修正した故障の再現手順は再実行しなければならない。確認テストの目的は、欠陥が確実に修正されたことの確認である。
- リグレッションテスト：修正および変更でコードの一部に対して行った変更が、同一コンポーネント、同一システム内の他コンポーネント、または他システムの振る舞いに意図せず影響を及ぼす場合がある。変更には、オペレーティングシステムやデータベース管理システムの新しいバージョンなど、環境の変更も含まれる。そのような意図しない副作用をリグレッションと呼ぶ。リグレッションテストでは、テストを実行して、そのような意図しない副作用を検出する。

確認テストとリグレッションテストは、すべてのテストレベルで行う。

特にイテレーティブ開発ライフサイクルやインクリメンタル開発ライフサイクル（例えば、アジャイル）では、機能追加、既存の機能への変更およびコードリファクタリングが頻繁なため、変更に関連したテストも常に必要となる。システムは本質的に進化するものであるため、確認テストとリグレッションテストはきわめて重要である。個々の開発対象（例えば、デバイス）が頻繁に更新または置き換えられる IoT システムでは、このことは特に明白である。

リグレッションテストスイートは何度も実行し、通常は少しずつ拡充をしていく。そのため自動化による効果が非常に大きい。リグレッションテストの自動化はプロジェクトの早期に開始すべきである（第6章を参照）。

2.3.5 テストタイプとテストレベル

すべてのテストタイプは、すべてのテストレベルで実行できる。銀行システムを使って、すべてのテストタイプについて、各テストレベルで行うことの例を説明する。最初に、機能テストの例を示す。

- コンポーネントテストでは、コンポーネントがどう複利計算するかに基づきテスト設計をする。
- コンポーネント統合テストでは、ユーザーインターフェースで取得した口座情報をビジネスロジックにどう取り込むかに基づきテスト設計をする。
- システムテストでは、口座保有者が当座預金口座にクレジット限度額をどう適用するかに基づきテスト設計をする。
- システム統合テストでは、システムが外部のマイクロサービスを使用して口座保有者のクレジット利用状況をどう確認するかに基づきテスト設計をする。
- 受け入れテストでは、銀行のクレジット取引承認での判定方法に基づきテスト設計をする。

以下は非機能テストの例である。

- コンポーネントテストでは、複雑な利息総額計算に必要なCPUサイクル数を評価するために性能テストを設計する。
- コンポーネント統合テストでは、ユーザーインターフェースからビジネスロジックにデータを渡す処理に、バッファオーバーフローの脆弱性が存在しないことを確認するためにセキュリティテストを設計する。
- システムテストでは、サポート対象のすべてのブラウザーやモバイルデバイスで表示が正しく動作することをチェックするために移植性テストを設計する。
- システム統合テストでは、クレジット利用状況を取り出すマイクロサービスが応答しない場合のシステムの堅牢性を評価するために信頼性テストを設計する。
- 受け入れテストでは、銀行のクレジット処理インターフェースに対する身体的な制約を持つ人のアクセシビリティを評価するために使用性テストを設計する。

以下はホワイトボックステストの例である。

- コンポーネントテストでは、計算を実行するすべてのコンポーネントに対して、完全なステートメントカバレッジとデシジョンカバレッジ（4.3節を参照）を達成するようテスト設計をする。
- コンポーネント統合テストでは、ブラウザーインターフェースの各画面でデータを次の画面やビジネスロジックに渡す方法をそれぞれ実行するようテストを設計する。
- システムテストでは、クレジット限度額申請におけるWebページの遷移順序をカバーするようテストを設計する。
- システム統合テストでは、マイクロサービスへ送信可能なすべての検索条件を実行するようテストを設計する。
- 受け入れテストでは、銀行間資金移動用のすべてのサポート対象金融データファイル形式と値の範囲をカバーするようテストを設計する。

最後に、変更に関連したテストの例を以下に示す。

- コンポーネントテストでは、自動化したリグレッションテストを各コンポーネント用にビルドし、継続的インテグレーションフレームワークに含める。
- コンポーネント統合テストでは、修正をコードリポジトリにチェックインする際に、インターフェース関連の欠陥に対する修正を確認するテストを設計する。

- システムテストでは、ワークフロー上のいずれかの画面を変更している場合、そのワークフローのすべてのテストを再実行する。
- システム統合テストでは、マイクロサービスの継続的デプロイメントの一環として、マイクロサービスと相互作用しているアプリケーションのテストを毎日再実行する。
- 受け入れテストでは、受け入れテストで検出した欠陥を修正した後に、ステータスが不合格となっているすべてのテストを再実行する。

本節では、全テストタイプにおける各テストレベルの適用例を示したが、すべてのソフトウェアに対して、各テストレベルで示した全テストタイプを適用する必要はない。ただし、各テストレベルで適用可能なテストタイプを行うことが重要である。中でもテストタイプが必要となる最も早いテストレベルで行うことは重要である。

2.4 メンテナンス（保守）テスト

本番環境にデプロイしたソフトウェアやシステムは、メンテナンスを行う必要がある。本番リリース済みのソフトウェアやシステムには、運用で見つかった欠陥の修正、新機能の追加、またはリリース済みの機能の削除や変更を行うために、さまざまな種類の変更が発生する。コンポーネントやシステムを維持している間は、性能効率性、互換性、信頼性、セキュリティ、移植性などの非機能的な品質の特性を確保および改善するためにもメンテナンスは必要である。

メンテナンスの一環として変更が発生した場合は、変更が正しく適用されていることを評価し、システムの変更していない部分（システムの大部分を占める）での副作用（例えば、リグレッション）を確認するために、メンテナンステストを行うべきである。メンテナンステストは、システムの変更に焦点をあてるが、変更の影響を受けた可能性のある変更なしの部分に対しても同様に扱う。メンテナンスは、計画的なリリースと非計画的なリリース（ホットフィックス）の両方の一環として行われる。

メンテナンスリリースでは、その範囲に基づいて、複数のテストレベルでさまざまなテストタイプを使用してメンテナンステストを行う。メンテナンステストの範囲は以下の要因で決定する。

- 変更のリスクの度合い。例えば、ソフトウェアの変更領域が他のコンポーネントまたはシステムとコミュニケーションする度合い
- 既存システムの規模
- 変更の規模

2.4.1 メンテナンスが必要となる理由

計画的なリリースと非計画的なリリースの両方に対して、さまざまな理由によりソフトウェアのメンテナンスとメンテナンステストが必要になる。

ここでは、メンテナンスが必要となる理由を以下のように分類する。

- 変更作業：計画（リリース計画など）に従った拡張、修正や緊急変更、運用環境の変更（計画的な OS の変更やデータベースのアップグレードなど）、COTS ソフトウェアのアップグレード、欠陥や脆弱性に対するパッチ。
- 移行作業：別のプラットフォームへの移行。この場合、新しい環境や変更になったソフトウェアの運用テストを行う。また、別のアプリケーションからのデータをメンテナンス対象のシステムに移行する場合にはデータ変換のテストを行う。
- 廃棄作業：アプリケーションの廃棄。

アプリケーションやシステムを廃棄する際に長期間のデータ保有を要求されている場合、データの移行作業や保管作業のテスト、長期間のデータ保管後のリストア/抽出の手順のテストを行う必要がある。また、残りのサービスが引き続き動作することを確認するためのリグレッションテストも行う必要がある。

IoT システムでのメンテナンステストは、まったく新しいモノまたは変更したモノ（例えば、ハードウェアデバイスやソフトウェアサービス）をシステム全体に導入したときに行う。このようなシステムのメンテナンステストは、さまざまなレベル（例えば、ネットワークレベルやアプリケーションレベル）での統合テストを重要視し、個人データに関連する場合はセキュリティを特に重要視する。

2.4.2 メンテナンスの影響度分析

影響度分析はメンテナンスリリース向けに行った変更を評価し、変更により意図した結果、変更により予想される副作用、および変更が影響するシステムの領域を識別する。影響度分析は、既存のテストに対する修正が必要な箇所も識別する。修正が必要な既存のテストを更新した後に、副作用および影響を受ける領域に対してリグレッションテストを行う必要がある。

影響度分析は、他領域への潜在的な影響度に基づいて変更を行うか判断するため、変更を行う前に実施することもある。

影響度分析は、以下の場合に難しくなる。

- 仕様（例えば、ビジネス要件、ユーザーストーリー、アーキテクチャー）が最新版でない、または存在しない。
- テストケースが文書化されていない、または最新版でない。
- テストとテストベースとの間の双方向のトレーサビリティが維持されていない。
- ツールによる影響度分析のためのサポートが貧弱であるか、まったくない。
- ドメインおよび／またはシステムの知識を持つ担当者がいない。
- 開発時にソフトウェアの保守性が考慮されていない。

3 静的テスト — 135 分

キーワード

アドホックレビュー、チェックリストベースドレビュー、動的テスト、形式的レビュー、非形式的レビュー、インスペクション、パースペクティブベースドリーディング、レビュー、ロールベースドレビュー、シナリオベースドレビュー、静的解析、静的テスト、テクニカルレビュー、ウォークスルー

「静的テスト」の学習の目的

3.1 静的テストの基本

FL-3.1.1 （K1）さまざまな静的テスト技法を使用して検査できるソフトウェア成果物の種類を認識する。

FL-3.1.2 （K2）例を用いて、静的テストの価値を説明する。

FL-3.1.3 （K2）静的技法と動的技法について、目的、識別対象の欠陥の種類、ソフトウェアライフサイクル内でのこれらの技法の役割を考慮して、違いを説明する。

3.2 レビュープロセス

FL-3.2.1 （K2）作業成果物に対するレビュープロセスの活動を要約する。

FL-3.2.2 （K1）形式的レビューにおけるさまざまな役割と責務を認識する。

FL-3.2.3 （K2）さまざまなレビュータイプ（非形式的レビュー、ウォークスルー、テクニカルレビュー、およびインスペクション）の違いを説明する。

FL-3.2.4 （K3）欠陥を検出するために、作業成果物にレビュー技法を適用する。

FL-3.2.5 （K2）レビューの成功に貢献する要因を説明する。

3.1 静的テストの基本

静的テスト技法では、動的テスト技法（テスト対象のソフトウェアの実行が必要）と異なり、作業成果物を人手で調査（すなわち、レビュー）したり、コードや他の作業成果物をツール主導で評価（すなわち、静的解析）したりする。静的テストでは、どちらの種類でもテスト対象のコードやその他の作業成果物を実際に実行することなくアセスメントする。

静的解析はセーフティクリティカルコンピューターシステム（例えば、航空、医療、核関連ソフトウェア）だけでなく、他の分野のシステムでも重要かつ一般的になってきている。例えば、静的解析はセキュリティテストの重要な構成要素である。また、例えばアジャイル開発、継続的デリバリー、継続的デプロイメントでのビルドとリリースの自動システムにも組み込まれることが一般的である。

3.1.1 静的テストで検査可能な作業成果物

ほとんどの作業成果物は静的テスト（レビュー、静的解析、または両方）を使用して調査できる。例を以下に示す。

- 仕様（ビジネス要件、機能要件、セキュリティ要件など）
- エピック、ユーザーストーリー、受け入れ基準
- アーキテクチャーおよび設計仕様
- コード
- テストウェア（テスト計画書、テストケース、テスト手順、自動化テストスクリプトなど）
- ユーザーガイド
- Web ページ
- 契約、プロジェクト計画、スケジュール、予算
- モデルベースドテストで使用するアクティビティ図などのモデル（『ISTQB-MBT Foundation Level Model-Based Tester Extension Syllabus』および『Kramer 2016』を参照）

レビューは、読んで理解できるあらゆる作業成果物に適用できる。静的解析は、形式的な構造を持つあらゆる作業成果物（一般的にはコードやモデルなど）に、適切なツールを使用して効率的に適用できるが、自然言語で記述された作業成果物（要件など）に対しても、誤字脱字、文法、読みやすさを評価するために適用できる。

3.1.2 静的テストの利点

静的テストはさまざまな利点をもたらす。静的テストは、ソフトウェア開発ライフサイクルの初期に適用すると動的テストを実行する前に欠陥を早期に検出できる（例えば、要件/設計仕様レビュー、プロダクトバックログを洗練させる作業など）。早期に検出した欠陥は、本番稼動後のようなライフサイクルの終盤に検出した欠陥よりも、はるかに安価に除去できる。他の作業成果物の更新や確認/リグレッションテストを実行するコストを考慮した場合、動的テスト技法を使用してテスト対象の欠陥を見つけて修正するよりも、静的テスト技法を使用して欠陥を検出し、即座に修正する方がはるかに安価である。

静的テストのその他の利点には、以下のものがある。

- 欠陥の検出と修正をより効率的に、しかも動的テストを実行する前に行うことができる。
- 動的テストでは容易に検出できない欠陥を識別できる。
- 要件の不整合、曖昧性、矛盾、欠落、不正確性、冗長性を明らかにして、設計時またはコーディング時に欠陥が混入するのを防止できる。
- 開発の生産性を向上できる（設計の改善、保守性の高いコードなど）。
- 開発にかかるコストと時間を削減できる。
- テストにかかるコストと時間を削減できる。
- ライフサイクルの終盤または本番リリース後に検出される欠陥数が減少し、ソフトウェアの存続期間にわたる全体的な品質コストを削減できる。
- レビューへの参加過程でチームメンバー間のコミュニケーションを改善できる。

3.1.3 静的テストと動的テストの違い

静的テストと動的テストは、作業成果物の品質を評価すること、および可能な限り早期に欠陥を識別することなど、同じ目的に使える（1.1.1 節を参照）。静的テストと動的テストは、それぞれに異なる種類の欠陥を検出するため、相互に補完する関係にある。

静的テストの主な特徴の 1 つは、ソフトウェア実行時に欠陥により引き起こされた故障を識別するのではなく、作業成果物の欠陥を直接検出することである。欠陥は、故障を引き起こすことなくきわめて長期間にわたって作業成果物に潜んでいることがある。この欠陥が潜んでいるパスはほとんど実行されることがないか、到達することが難しいので、このような欠陥を偶然検出できる動的テストを構築および実行することは容易ではない。静的テストでは、はるかに少ない工数で検出できる。

また、動的テストが外部から見える振る舞いに焦点を置くことが典型的であるのに対し、静的テストは作業成果物の整合性と内部品質を向上させることも特徴である。

静的テストは動的テストに比べて、以下の欠陥を早期かつ安価に検出して修正できる。

- 要件の欠陥（例えば、不整合、曖昧性、矛盾、欠落、不正確性、冗長性）
- 設計の欠陥（例えば、非効率なアルゴリズムやデータベース構造、高い結合度、低い凝集度）
- コードの欠陥（例えば、値が定義されていない変数、宣言されているが使用されることのない変数、到達不能コード、重複したコード）
- 標準からの逸脱（例えば、コーディング標準を遵守していない）
- 正しくないインターフェース仕様（例えば、呼び出し側のシステムと呼び出される側のシステムで異なる単位の使用）
- セキュリティの脆弱性（例えば、バッファオーバーフローが発生する可能性）
- テストベースのトレーサビリティもしくはカバレッジが不十分もしくは不正確（例えば、受け入れ基準に対するテストケースが欠落している）

さらに、保守性欠陥のほとんどは、静的テストでのみ検出できる（例えば、不適切なモジュール化、低いコンポーネント再利用性、分析が困難で新しい欠陥の混入なしに修正することが困難なコード）。

3.2 レビュープロセス

レビューには、非形式的なものから、形式的なものまでさまざまな種類がある。非形式的レビューは、定義したプロセスに従わず、レビュー結果を形式的に文書化しないことが特徴である。形式的レビュー

は、チームで参加し、レビューの結果とレビューを行う際の手順を文書化することが特徴である。レビュープロセスの形式は、ソフトウェア開発ライフサイクルモデル、開発プロセスの成熟度、レビュー対象の作業成果物の複雑さ、法律や規制から生じる必要性、そして／または監査証跡の必要性などの要素で決まる。

レビューの焦点は、レビュー目的として合意したこと（例えば、欠陥の発見、内容の理解、テスト担当や新しいチームメンバーなどの参加者に対する教育、協議での判断の合意）により異なる。

ISO 標準（ISO/IEC 20246）には、作業成果物のレビュープロセスの詳細が、役割やレビュー技法を含めて説明されている。

3.2.1 作業成果物のレビュープロセス

レビュープロセスは、以下の主要活動で構成する。

計画

- レビューの範囲を定義する。これには、レビューの目的、レビュー対象のドキュメント（またはその部分）、評価すべき品質特性を含む。
- 工数と時間を見積る。
- レビュー特性を識別する。これには、レビュータイプ、役割、活動、チェックリストなどを含む。
- レビューの参加者を選び、役割を割り当てる。
- 開始基準と終了基準を定義する（インスペクションなど、形式的なレビュータイプほど必要となる）。
- 開始基準が満たされていることをチェックする（形式的なレビュータイプほど必要となる）。

レビューの開始

- 作業成果物もしくは他の資料（例えば、問題記録フォーム、チェックリスト、関連する作業成果物）を（物理的または電子的な手段で）配布する。
- レビューの範囲、目的、プロセス、役割、作業成果物を参加者に説明する。
- レビューについての参加者からの質問に答える。

個々のレビュー（すなわち、個々の準備）

- 作業成果物のすべてまたは一部をレビューする。
- 潜在的な欠陥、提案、質問を書き出す。

懸念事項の共有と分析

- 識別した潜在的な欠陥について、レビューミーティングなどで議論する。
- 潜在的な欠陥を分析し、それらにオーナーとステータスを割り当てる
- 品質特性を評価し、文書化する。
- レビューで見つけた欠陥を終了基準に対して評価し、レビュー判定（却下、大規模な変更が必要、受け入れ可能、小規模な変更が必要など）を行う。

修正と報告

- レビューで見つけた欠陥で変更が必要なものについて欠陥レポートを作成する。
- レビュー対象の作業成果物で見つかった欠陥を修正する（典型的には作成者が修正する）。
- 適切な人またはチームと、欠陥について議論する（レビュー対象の作業成果物に関連する作業成果物で見つかった場合）。

- 欠陥のステータスを更新する（形式的なレビューの場合）。コメント作成者の合意を含むこともある。
- メトリクスを収集する（形式的なレビュータイプほど必要となる）。
- 終了基準が満たされていることをチェックする（形式的なレビュータイプほど必要となる）。
- 終了基準に到達したときに作業成果物を受け入れる。

作業成果物をレビューした結果は、3.2.3 節で記述されているように、レビュータイプや形式的の度合いによりさまざまである。

3.2.2 形式的レビューでの役割と責務

形式的レビューで一般的に含まれる役割は、以下の通りである。

作成者

- レビュー対象の作業成果物を作成する。
- レビュー対象の作業成果物の欠陥を修正する（必要な場合）。

マネージャー

- 責任を持ってレビューの計画を行う。
- レビューの実行を決定する。
- 担当者、予算、時間を割り当てる。
- 費用対効果を継続的にモニタリングする。
- 不適切な結果が発生した状況に対してコントロールをする。

ファシリテーター（モデレーターと一般的に呼ばれる）

- 効果的なレビューミーティングを運営する（開催時）。
- さまざまな意見の調整を行う（必要な場合）。
- レビューの成功を左右する重要な役割を果たす。

レビューリーダー

- レビューに関して全体的な責任を持つ。
- 参加者を人選し、レビューを実施する期間と場所を決定する。

レビューア

- 特定分野の専門家、プロジェクトの従事者、作業成果物に関心のあるステークホルダー、そして／または特定の技術や業務のバックグラウンドを持つ個人などが行う。
- レビュー対象の作業成果物の欠陥を識別する。
- それぞれに異なる観点（例えば、テスト担当者、プログラマー、ユーザー、オペレーター、ビジネスアナリスト、使用性専門家など）でレビューを行う。

書記（記録係）

- 個々のレビュー活動の期間に見つかった潜在的な欠陥を照合する。
- （開催時に）レビューミーティングで見つかった新しい潜在的な欠陥、未決事項、決定事項を記録する。

レビュータイプによっては、1 人で複数の役割を担当することがある。また、各役割に付随する活動もレビュータイプによって異なる。加えて、欠陥、未決事項、判定の記録といったレビュープロセスをサポートするツールが現れていることもあるため、書記を必要としない場合も増えている。

ISO 標準（ISO/IEC 20246）で記載されているように、役割をさらに細かく分割することも可能である。

3.2.3 レビュータイプ

レビューはさまざまな用途で使用するが、主となる目的の 1 つは欠陥を検出することである。すべてのレビュータイプが欠陥の検出に役立つが、レビュータイプの選択は、さまざまな選択基準の中から、プロジェクトのニーズ、利用可能なリソース、プロダクトの種類とリスク、ビジネスドメイン、企業の文化に基づいて行う。

レビューはさまざまな属性で分類できる。レビューの最も一般的な 4 つのタイプと付随する属性を以下に示す。

非形式的レビュー（例えば、バディチェック、ペアリング、ペアレビュー）

- 主な目的：潜在的な欠陥の検出。
- その他の目的：新しいアイデアや解決策の創出、影響度の小さい問題の迅速な解決。
- 形式的な（文書化した）プロセスに基づかない。
- レビューミーティングを行わないことがある。
- 作成者の同僚（バディチェック）やその他の人により実施。
- 結果を文書化することもある。
- レビューアにより、効果はさまざまである。
- チェックリストの使用は任意である。
- アジャイル開発では一般的に行われる。

ウォークスルー

- 主な目的：欠陥の発見、ソフトウェアプロダクトの改善、異なる実装方法の検討、標準や仕様への準拠の評価。
- その他の目的：さまざまな技法やスタイルに関するアイデアの交換、参加者のトレーニング、合意の形成。
- レビューミーティング前の個々のレビューアによる準備は任意である。
- 典型的には作業成果物の作成者がレビューミーティングを主導する。
- 書記は必須である。
- チェックリストの使用は任意である。
- シナリオ、ドライラン、シミュレーションの形態を取る場合がある。
- 潜在的な欠陥の記録やレビューレポートを作成する場合がある。
- 運営により、きわめて非形式的のものから、高度に形式的なものまである。

テクニカルレビュー

- 主な目的：合意の獲得、潜在的な欠陥の検出。
- その他の目的：作業成果物の品質の評価と信頼の積み上げ、新しいアイデアの創出、作成者のモチベーション向上と将来の作業成果物の改善、異なる実装方法の検討。
- レビューアは、作成者の技術領域の同僚、および技術分野が同じ、または別の専門家である。

- レビューミーティング前に個々のレビューアが準備する。
- レビューミーティングの開催は任意である。開催する場合、経験を積んだファシリテーター（作成者ではない）が主導するのが理想である。
- 書記は必須である（作成者でないのが理想）。
- チェックリストの使用は状況による。
- 典型的に、潜在的な欠陥の記録やレビューレポートを作成する。

インスペクション

- 主な目的：潜在的な欠陥の検出、作業成果物の品質の評価と信頼の積み上げ、作成者の学習と根本原因分析による将来の類似欠陥の防止。
- その他の目的：作成者のモチベーション向上と、将来の作業成果物およびソフトウェア開発プロセスの改善、合意の形成。
- ルールやチェックリストに基づいたプロセスで進行し、形式に沿ったドキュメントを作成する。
- 3.2.2 節で指定されているような役割が明確に決まっている。レビューミーティングで作業成果物を読みあげる人を専任で加えたりすることもある。
- レビューミーティング前に個々のレビューアが準備する。
- レビューアは作成者の同僚か、作業成果物に関連する別の分野の専門家である。
- 開始基準と終了基準が指定されている。
- 書記は必須である。
- レビューミーティングは、経験を積んだ進行役（作成者ではなく）が主導する。
- 作成者は、レビューリーダー、ドキュメントを読みあげる担当、書記のどの役割も担わない。
- 潜在的な欠陥の記録やレビューレポートを作成する。
- メトリクスを収集し、ソフトウェア開発プロセス全体（インスペクションプロセスを含む）の改善に使用する。

1 つの作業成果物に対して、複数の種類のレビューを行うことがある。複数の種類のレビューを行う場合の順序はさまざまである。例えば、非形式的レビューを実施して作業成果物の準備が完了したことを確認した後に、テクニカルレビューを行うことがある。

前述の種類のレビューは、ピアレビューとして実行できる。すなわち、組織内の同じ職位の同僚によって実施可能である。

レビューで検出される欠陥の種類は、特にレビュー対象の作業成果物によって異なる。レビュー時にさまざまな作業成果物で検出される欠陥の例については、3.1.3 節を参照されたい。また、形式的インスペクションの詳細については、『Gilb 1993』を参照されたい。

3.2.4 レビュー技法の適用

個々のレビュー（個々の準備）活動で欠陥を見つけるために適用できるさまざまなレビュー技法がある。レビュー技法は、前述のレビュータイプに対して横断的に適用できるが、技法の有効性は、使用するレビュータイプによって異なる。さまざまなレビュータイプに適用できるレビュー技法の例を、以下に示す。

アドホック

アドホックレビューでは、レビューアは、このレビューに関するタスクのガイダンスをほとんどまたはまったく提供されない。一般的に、レビューアは作業成果物を順番に読んで懸念事項を識別するごとに

記録に残す。アドホックなレビューは、準備を必要としない場合によく行う。この技法は、レビューアのスキルに大きく依存し、複数のレビューアから重複した懸念事項が多く報告されてしまう場合がある。

チェックリストベース

チェックリストベースドレビュー技法は体系的に行われ、レビューアはレビューの開始時に（例えばファシリテーターにより）配布されるチェックリストを使用して懸念事項を検出する。レビューチェックリストは、経験から導出した起こりえる欠陥に基づく一連の質問を列挙したものである。レビュー対象の作業成果物ごとにチェックリストを用意し、以前のレビューで見逃された懸念事項の種類をカバーするために定期的にメンテナンスすべきである。チェックリストベースの技法の主な利点は、典型的な懸念事項の種類を体系的にカバーできることである。レビューを行う際には、チェックリストに単純に従うだけでなく、チェックリスト外の欠陥も検出する努力をすることが重要である。

シナリオとドライラン

シナリオベースドレビュー技法では、作業成果物を読むための構造化されたガイドラインをレビューアに提供する。レビューアはシナリオを使用して、（作業成果物がユースケースなど適切な形式で文書化されている場合）作業成果物の期待する使い方に基づいて、作業成果物に対して「ドライラン」を行う。このシナリオは、単純なチェックリスト項目よりも、特定の種類の欠陥を検出するためのよいガイドラインとなる。他の種類の欠陥（例えば、機能の欠落）を見逃さないようにするためには、チェックリストベースドレビュー技法と同様に文書化されたシナリオだけにとらわれないようにしなければならない。

ロールベース

ロールベースドレビュー技法では、レビューアは個々のステークホルダーの役割の観点から作業成果物を評価する。典型的な役割には、特定のエンドユーザーの種類（熟練者、初心者、年配者、子供など）や組織内の特定の役割（ユーザーアドミニストレーター、システムアドミニストレーター、性能テスト担当者など）がある。

パースペクティブベース

パースペクティブベースドリーディングでは、ロールベースのレビュー技法と同じく、レビューアはそれぞれに異なるステークホルダーの観点でレビュー活動を行う。典型的なステークホルダーの観点としては、エンドユーザー、マーケティング担当者、設計者、テスト担当者、運用担当者などがある。レビューアごとに異なる観点を持つことで、より深く掘り下げたレビューが可能になり、検出される問題のレビューア間における重複が少なくなる。

さらに、パースペクティブベースドリーディングでは、レビュー対象の作業成果物から各レビューアの役割で導出する成果物を作成してみる必要がある。例えば、要件仕様についてパースペクティブベースドリーディングを行っているテスト担当者は、暫定的な受け入れテストを作成して必要な情報がすべて含まれていることを確認する。さらに、チェックリストも使用することが前提である。

現場の調査から、要件や技術的な作業成果物のレビューの場合、パースペクティブベースドリーディングが最も効果的な技法であることが判明している。レビュー成功のためには、想定されるリスクに応じて、さまざまなステークホルダーの観点を適切に含めることである。パースペクティブベースドリーディングの詳細については『Shul 2000』を、さまざまな種類のレビューの有効性については『Sauer 2000』を参照されたい。

3.2.5 レビューの成功要因

レビューを成功させるためは、適切な種類のレビューおよび適切な技法の適用を検討することが必要である。これ以外にも、さまざまな要因がレビュー結果に影響する。

レビューを成功させるための組織的な要因を次に示す。

- レビュー計画時に、計測可能な終了基準として使用できる明確な目的を定義する。
- 達成する目的、およびソフトウェア成果物と参加者の種類とレベルに応じてレビュータイプを選択する。
- レビュー対象の作業成果物で欠陥を効果的に識別するために適切なレビュー技法（チェックリストベース技法やロールベース技法など）を 1 つ以上使用する。
- チェックリストは、主要なリスクに言及できるよう最新の状態に保つ。
- 欠陥に関するフィードバックを早期および頻繁に作成者に提供して品質をコントロールするために、大きなドキュメントは小さく分割して記述およびレビューする。
- 参加者に十分な準備時間を与える。
- レビューのスケジュールは適切に通知する。
- マネージャーがレビュープロセスを支援する（例えば、プロジェクトスケジュールでレビューに十分な時間が割り当てられるようにする）。

レビューを成功させるための人的な要因を次に示す。

- レビューの目的に対して適切な人たちに関与させる（例えば、さまざまスキルセットまたはパースペクティブを持ち、対象のドキュメントを使うことがある人たち）。
- テスト担当者は、レビューに貢献するだけでなく、レビュー対象の作業成果物の内容を把握して、有効なテストを早期に準備できると、レビューアとして価値がでる。
- 参加者には適切な時間を割り当て、細心の注意を払って詳細にレビューしてもらう。
- レビューアが個人でのレビュー時、および／またはレビューミーティング時に集中力を維持できるよう、レビューは対象を小さく分割して実施する。
- 見つかった欠陥は客観的な態度で確認、識別、対処をする。
- ミーティングは参加者にとって有意義な時間となるよう適切にマネジメントする。
- レビューは信頼できる雰囲気で行い、レビュー結果を参加者の評価に使用しない。
- 参加者は、自分の言動が他の参加者に対する退屈感、憤り、敵意だと受け取られないように気を付ける。
- 特にインスペクションなど高度に形式的なレビュータイプには、十分なトレーニングを提供する。
- 学習とプロセス改善の文化を醸成する。

レビューの成果を向上させる方法の詳細については、『Gilb 1993』、『Wiegers 2002』、『van Veenendaal 2004』を参照されたい。

4 テスト技法 330 分

キーワード

ブラックボックステスト技法、境界値分析、チェックリストベースドテスト、カバレッジ、デシジョンカバレッジ、デシジョンテーブルテスト、エラー推測、同値分割法、経験ベースのテスト技法、探索的テスト、状態遷移テスト、ステートメントカバレッジ、テスト技法、ユースケーステスト、ホワイトボックステスト技法

「テスト技法」の学習の目的

4.1 テスト技法のカテゴリ

FL-4.1.1 （K2）ブラックボックステスト技法、ホワイトボックステスト技法、および経験ベースのテスト技法の特徴、共通点、および相違点を説明する。

4.2 ブラックボックステスト技法

FL-4.2.1 （K3）ある特定の要件からテストケースを導出するために同値分割法を適用する。

FL-4.2.2 （K3）ある特定の要件からテストケースを導出するために境界値分析を適用する。

FL-4.2.3 （K3）ある特定の要件からテストケースを導出するためにデシジョンテーブルテストを適用する。

FL-4.2.4 （K3）ある特定の要件からテストケースを導出するために状態遷移テストを適用する。

FL-4.2.5 （K2）ユースケースからテストケースを導出する方法を説明する。

4.3 ホワイトボックステスト技法

FL-4.3.1 （K2）ステートメントカバレッジを説明する。

FL-4.3.2 （K2）デシジョンカバレッジを説明する。

FL-4.3.3 （K2）ステートメントカバレッジとデシジョンカバレッジの価値を説明する。

4.4 経験ベースのテスト技法

FL-4.4.1 （K2）エラー推測を説明する。

FL-4.4.2 （K2）探索的テストを説明する。

FL-4.4.3 （K2）チェックリストベースドテストを説明する。

4.1 テスト技法のカテゴリ

この節で説明されているものを含めて、テスト技法の目的は、テスト条件、テストケース、テストデータを決定することである。

4.1.1 テスト技法の選択

テスト技法を選択する際は、以下を含むさまざまな要因を検討する。

- コンポーネントまたはシステムの種類
- コンポーネントまたはシステムの複雑さ
- 規制や標準
- 顧客または契約上の要件
- リスクレベル
- リスクタイプ
- テスト目的
- 入手可能なドキュメント
- テスト担当者の知識とスキル
- 利用できるツール
- スケジュールと予算
- ソフトウェア開発ライフサイクルモデル
- ソフトウェアの想定される使用方法
- テスト対象のコンポーネントまたはシステムに関してテスト技法を使用した経験
- コンポーネントまたはシステムで想定される欠陥の種類

すべてのテストレベルに適用可能な技法もあれば、特定の状況やテストレベルに対してより適切に利用できる技法もある。テスト担当者はテストケースを作成する際に、テスト技法を組み合わせて使用し、テスト活動で最善の結果を得られるようにする。

テスト分析、テスト設計、テスト実装の活動では、形式に従わない（ドキュメントがほとんどない、もしくはまったくない）活動から形式に従った活動まで、テスト技法を使用できる。形式の適切な度合いは、テストプロセスや開発プロセスの成熟度、時間的制約、安全性要件または規制要件、参加者の知識やスキル、従うべきソフトウェア開発ライフサイクルモデルを含む、テストの状況によって決まる。

4.1.2 テスト技法のカテゴリと特徴

本シラバスでは、テスト技法を、ブラックボックス、ホワイトボックス、経験ベースに分類する。

ブラックボックステスト技法（振る舞い技法または振る舞いベースの技法と呼ぶこともある）は、適切なテストベース（例えば、形式に沿った要件ドキュメント、仕様書、ユースケース、ユーザーストーリー、ビジネスプロセス）の分析に基づく。これらの技法は、機能テストと非機能テストの両方に適用できる。ブラックボックステスト技法は、テスト対象の入力と出力に着目し、その内部構造は参照しない。

ホワイトボックステスト技法（構造技法または構造ベースの技法と呼ぶこともある）は、アーキテクチャー、詳細設計、内部構造、テスト対象のコードの分析に基づく。ホワイトボックステスト技法はブラックボックステスト技法と異なり、テスト対象内の構造と処理に重点を置く。

経験ベースのテスト技法は、テストケースの設計、実装および実行のために、開発担当者、テスト担当者、ユーザーの経験を活用する。この技法では、ブラックボックステスト技法やホワイトボックステスト技法と組み合わせて使用する。

ブラックボックステスト技法に共通する特徴は以下のものを含む。

- テスト条件、テストケース、テストデータは、ソフトウェア要件、仕様、ユースケース、ユーザーストーリーなどのテストベースから導出する。
- テストケースは、要件と実装との間の相違、および要件からの逸脱を検出するために使う。
- カバレッジは、テストベース内のテスト済みアイテムと適用した技法に基づいて計測する。

ホワイトボックステスト技法に共通する特徴は以下のものを含む。

- テスト条件、テストケース、テストデータは、コード、ソフトウェアアーキテクチャー、詳細設計、ソフトウェア構造に関する他の情報などのテストベースから導出する。
- カバレッジは、選択した構造（例えば、コードやインターフェース）内のテスト済みアイテムに基づいて計測する。
- 仕様書は、期待結果を決定するための追加の情報源として使用されることも多い。

経験ベースのテスト技法に共通する特徴は以下のものを含む。

- テスト条件、テストケース、テストデータは、テスト担当者、開発担当者、ユーザー、その他のステークホルダーの知識や経験などのテストベースから導出する。

この知識や経験には、ソフトウェアの想定される使用方法、使用環境、検出される可能性のある欠陥、欠陥の偏在なども含む。

国際標準（ISO/IEC/IEEE 29119-4）には、テスト技法に対応するカバレッジ計測方法の詳細について説明されている（技法の詳細については、『Craig 2002』および『Copeland 2004』を参照されたい）。

4.2 ブラックボックステスト技法

4.2.1 同値分割法

同値分割法は、同等に処理されると想定したデータすべてを同じパーティション（これを同値クラスとも呼ぶ）に振り分ける技法である（『Kaner 2013』および『Jorgensen 2014』を参照）。有効な値と無効な値の両方に対して同値パーティションがある。

- 有効な値は、コンポーネントまたはシステムに受け入れられる値である。有効な値の同値パーティションを「有効同値パーティション」と呼ぶ。
- 無効な値は、コンポーネントまたはシステムに拒否される値である。無効な値の同値パーティションを「無効同値パーティション」と呼ぶ。
- 入力、出力、内部値、時間関連の値（例えば、イベントの前後）などのテスト対象に関連するあらゆるデータ要素、およびインターフェースのパラメーター（例えば、統合テスト時にテスト対象となる統合コンポーネント）に対するパーティションが識別できる。
- パーティションは、必要に応じてさらに細かく分割される場合がある。
- 各値は、必ず 1 つの同値パーティションだけに属する必要がある。
- 無効同値パーティションをテストケースで使用する場合、複数の故障が同時に起き、1 つの故障のみが表面化した場合、他の故障を隠してしまい見逃す恐れがある。故障を隠してしまわないようにするため、他の無効同値パーティションとは組み合わせず単独でテストすべきである。

この技法を使用して 100%カバレッジを達成するには、すべての識別された有効同値パーティションと無効同値パーティションのそれぞれから、少なくとも 1 つの値をテストケースでカバーする。カバレッジは、1 つ以上の値を使用してテストした同値パーティションの数を、識別された同値パーティションの合計数で除算して計測する（パーセンテージで表すことが多い）。同値分割法はすべてのテストレベルで適用できる。

4.2.2 境界値分析

境界値分析（BVA）は同値分割法の拡張であるが、パーティションが数値または順序付け可能な値で構成される場合だけ使用できる。パーティションの最小値および最大値（または最初の値と最後の値）が、境界値である（『Beizer 1990』）。

例えば、ある入力フィールドで 1 桁の整数値が入力できる場合を考える。キーパッドで入力を制限し、整数値以外を入力不可にする。有効な値の範囲は、1～5 である（1 と 5 を含む）。この場合、（小さい）無効、有効、（大きい）無効の 3 つの同値パーティションが存在する。有効同値パーティションの境界値は 1 と 5 である。（大きい）無効同値パーティションの境界値は 6 と 9 である。（小さい）無効同値パーティションの境界値は 0 だけである。

この例では、境界ごとに 2 つの境界値を識別した。（小さい）無効同値パーティションと有効同値パーティションの境界のテスト値は 0 と 1 である。有効同値パーティションと（大きい）無効同値パーティションの境界のテスト値は 5 と 6 である。境界あたり 3 つの境界値（境界上および境界の前後）を識別する考え方もあり、前述の例で 3 ポイント境界値分析を行うと、小さい方の境界値は 0、1、2 で、大きい方の境界値は 4、5、6 となる（『Jorgensen 2014』）。

同値パーティションの境界上での振る舞いは、同値パーティション内での振る舞いよりも正しくないことが多い。仕様上の境界および実装した境界が意図した値の上または下に誤って配置されたり、すべて省略されたり、意図しない他の境界とともに追加されたりする場合があることを忘れてはいけない。境界値分析の結果を用いたテストは、ソフトウェアの境界値が属するパーティションとは別のパーティションの振る舞いを示すような欠陥をほぼすべて見つけ出す。

境界値分析はすべてのテストレベルに適用できる。この技法は、数値（日付や時間を含む）の範囲を必要とする要件に対するテストケースに使用する。あるパーティションに対する境界カバレッジは、テストした境界値の数を識別した境界値の合計数で除算して計測する（パーセンテージで表すことが多い）。

4.2.3 デシジョンテーブルテスト

組み合わせテスト技法は、システム要件から実装した条件のさまざまな組み合わせと組み合わせごとの結果に対する仕様をテストするのに役立つ。組み合わせテストのアプローチの 1 つとしてデシジョンテーブルテストがある。

デシジョンテーブルは、システムが実装すべき複雑なビジネスの規則を表現するのに有効である。デシジョンテーブルを作成する際には、テスト担当者は条件（主に入力）と結果的に起きるシステムのアクション（主に出力）を識別する。条件とアクションは表の行となり、条件を上部、アクションを下部に記述する。各列は判定の規則に対応している。規則は条件の一意な組み合わせとその結果として実行するアクションを表す。条件とアクションの値は、ブール値（真か偽）または離散値（例えば、赤、緑、青）で表現することが多いが、数値や数値の範囲で表現することもある。複数の異なった条件とアクションを同一のデシジョンテーブルに記述するのが一般的である。

デシジョンテーブルの一般的な表記法を以下に示す。

条件：

- Y：条件が真であることを意味する（T または 1 とも記述する）。
- N：条件が偽であることを意味する（F または 0 とも記述する）。
- -：条件の値は判定に影響しないことを意味する（N/A とも記述する）。

アクション：

- X：アクションが発生することを意味する（Y、T、または 1 とも記述する）。
- 空白：アクションが発生しないことを意味する（-、N、F、または 0 とも記述する）。

最大のデシジョンテーブルは、条件のすべての組み合わせをカバーするだけの列を持つ。デシジョンテーブルは、条件の組み合わせが起こりえない列、条件の組み合わせが可能ではあるが現実的に起こりえない列、結果に影響しない条件の組み合わせをテストする列を削除して簡単化できる。デシジョンテーブルを簡単化する方法の詳細については、『ISTQB テスト技術者資格制度 Advanced Level シラバス 日本語版 テストアナリスト』を参照されたい。

通常、デシジョンテーブルテストの標準的なカバレッジの最小単位は、デシジョンテーブル内の判定の規則ごとに 1 つ以上のテストケースを含めることである。これにより、条件のすべての組み合わせがカバーされる。カバレッジは、1 つ以上のテストケースでテストした判定の規則の数を、判定の規則の合計数で除算して計測する（パーセンテージで表すことが多い）。

デシジョンテーブルテストの利点は、見逃される可能性がある組み合わせを含めて、条件の重要な組み合わせのすべてを識別できることである。また、要件の中にあるいくつかの相違点も見つけることができる。デシジョンテーブルテストは、ソフトウェアの振る舞いが条件の組み合わせに依存する状況で適用可能であり、すべてのテストレベルに適用可能である。

4.2.4 状態遷移テスト

コンポーネントやシステムは、現在の条件や過去の履歴（システムの初期化後から起きるイベントなど）によって、イベントに対して異なる振る舞いをする。過去の履歴は、状態の概念を使用してまとめることができる。状態遷移図は、ソフトウェアの取り得る状態、ソフトウェアが開始および終了する方法、ソフトウェアが状態間で遷移する方法を示す。遷移はイベントにより開始される（例えば、ユーザーによるフィールドへの値入力）。イベントは遷移を引き起こす。同じイベントが同じ状態から複数の異なる遷移を引き起こす場合、そのイベントはガード条件で明確にする。状態が変化するとソフトウェアでアクションが実行される（計算結果またはエラーメッセージの表示など）。

状態遷移表は、状態間の有効な遷移と無効だと思われる遷移のすべてと、イベント、ガード条件、有効な遷移の際のアクションを示す。状態遷移図は有効な遷移のみを示し、無効な遷移は除くのが一般的である。

状態の典型的な順序をカバーする、すべての状態をテストする、すべての遷移をテストする、遷移を特定の順序でテストする、無効な遷移をテストする、といったテストケースの設計が考えられる。

状態遷移テストは、メニューベースのアプリケーションのために使われる。そして、組込みソフトウェア業界にて幅広く使われている。この技法は、特定の状態を持つビジネスシナリオのモデル化、および画面遷移のテストをするためにも適している。状態の概念は抽象的であり、数行のコードを表すことも、ビジネスプロセス全体を表すこともある。

カバレッジは、テストをした状態または遷移の数を、テスト対象の識別した状態または遷移の合計数で除算して計測するのが一般的である（パーセンテージで表すことが多い）。状態遷移テストのカバレッジ基準の詳細については、『ISTQB テスト技術者資格制度 Advanced Level シラバス 日本語版 テストアナリスト』を参照されたい。

4.2.5 ユースケーステスト

ユースケースは、ソフトウェアアイテムとの相互作用に表れるソフトウェア機能の要件を取り込むための設計方法であり、テストケースが導出可能である。ユースケースは、サブジェクト（ユースケースが適用されるコンポーネントまたはシステム）とアクター（ユーザー、外部ハードウェア、その他のコンポーネントやシステム）をつないでいる。

各ユースケースは、1 つのサブジェクトと 1 つ以上のアクターとの相互作用による振る舞いを明確にする（UML 2.5.1 2017）。ユースケースは、相互作用とアクティビティ、さらには事前条件、事後条件で記述できる。必要に応じて自然言語でも記述できる。アクターとサブジェクトとの相互作用で、サブジェクトの状態が変化する場合がある。相互作用は、ワークフロー図、アクティビティ図、またはビジネスプロセスモデルを使用して視覚的に記述できる。

ユースケースには、基本的な振る舞いのバリエーションである、例外的な振る舞いおよびエラー処理を含むことができる。エラー処理には、プログラミング、アプリケーションおよびコミュニケーションのエラーに対するエラーメッセージの表示といった、システムの応答や回復の処理がある。テストは、定義した振る舞い（基本、例外または代替処理、およびエラー処理）を実行するように設計する。カバレッジは、テストしたユースケースの振る舞いをユースケースの振る舞いの合計数で除算して計測する（パーセンテージで表すことが多い）。

ユースケーステストのカバレッジ基準の詳細については、『ISTQB テスト技術者資格制度 Advanced Level シラバス 日本語版 テストアナリスト』を参照されたい。

4.3 ホワイトボックステスト技法

ホワイトボックステストは、テスト対象の内部構造を基にしたテストである。ホワイトボックステスト技法は、すべてのテストレベルで使用できるが、本節で説明する 2 つのコード関連の技法は、一般的にコンポーネントテストレベルで使用する。より厳密なカバレッジを達成するために、セーフティクリティカル環境、ミッションクリティカル環境、高い完全性が求められる環境で使用できる高度な技法があるが、ここでは対象としない。このような技法の詳細については、『ISTQB テスト技術者資格制度 Advanced Level シラバス 日本語版 テクニカルテストアナリスト』を参照されたい。

4.3.1 ステートメントテストとカバレッジ

ステートメントテストはコード内の実行可能ステートメントをテストする。カバレッジは、テストにより実行したステートメント数を、テスト対象の実行可能ステートメントの合計数で除算して計測する（パーセンテージで表すことが多い）。

4.3.2 デシジョンテストとカバレッジ

デシジョンテストはコード内の判定をテストし、実行したコードを判定結果に基づいて評価する。このために、テストケースは、判定ポイントを通る制御フローに従う（例えば、IF ステートメントでは、真と偽の結果用に 1 つずつテストケースが必要、CASE ステートメントでは、デフォルトの結果を含むすべての取り得る結果に対してテストケースが必要）。

カバレッジは、テストにより実行した判定結果の数をテスト対象の判定結果の合計数で除算して計測する（パーセンテージで表すことが多い）。

4.3.3 ステートメントテストとデシジョンテストの価値

100%のステートメントカバレッジの達成は、コード内のすべての実行可能ステートメントを 1 回以上テストしたことを保証するが、すべての判定ロジックをテストしたことは保証しない。本シラバスで説明している 2 つのホワイトボックス技法で、ステートメントテストがカバーする度合いはデシジョンテストより少ない。

100%のデシジョンカバレッジの達成は、すべての判定結果を実行したことを意味する。これには、真の結果、偽の結果、さらには明示的な偽のステートメントが存在しない場合（例えば、IF ステートメント内に ELSE のないコードなど）を含む。ステートメントカバレッジは、他のテストでは遂行されないコードの中にある欠陥を見つけるのに役立つ。デシジョンカバレッジは、他のテストでは真の結果と偽の結果の両方が実行されないコードの欠陥を見つけるのに役立つ。

100%のデシジョンカバレッジの達成は、100%のステートメントカバレッジの達成を保証する。逆は保証しない。

4.4 経験ベースのテスト技法

経験ベースのテスト技法を適用する場合、テスト担当者のスキルと直感の他、同様のアプリケーションや技術における経験からテストケースを導出する。この技法は、他の体系的な技法では容易に識別できないテストを識別するのに役に立つ。経験ベースの技法は、テスト担当者のテストの進め方と経験に大きく依存するため、カバレッジや有効性には非常に大きなバラツキがある。この技法のカバレッジの評価は困難であり、計測できない場合がある。

一般的に使用される経験ベースのテスト技法を、以降の節で説明する。

4.4.1 エラー推測

エラー推測は、誤り、欠陥、故障の発生をテスト担当者の以下の知識に基づいて予測する技法である。

- アプリケーションの過去の動作状況
- 開発担当者が犯しやすい誤りの種類
- 他のアプリケーションで発生した故障

エラー推測技法に対する系統的アプローチは、起こりえる誤り、欠陥、故障のリストを作り、それらの故障やそれらを引き起こす欠陥を検出するテストケースを設計する。誤り、欠陥、故障のリストは、テスト担当者の経験、欠陥や故障のデータ、ソフトウェアが不合格となる理由に関する一般的な知識に基づいて作成できる。

4.4.2 探索的テスト

探索的テストは、形式的ではない（事前定義されていない）テストであり、テスト実行時に動的に設計、実行、ログ記録、および評価をする。テスト結果を使用してコンポーネントまたはシステムについての理解を深め、さらにテストを行わなければならない領域のテストケースを作成する。

探索的テストでは、活動を体系的にするためにセッションベースドテストを使用する場合がある。セッションベースドテストでは、探索的テストをあらかじめ決められた時間枠内で行う。テスト担当者はテスト目的を含むテストチャーターに従ってテスト実行をする。テスト担当者はテストセッションシートを使用して、実行した手順や発見した事象を文書化する場合がある。

探索的テストは、仕様がほとんどなかったり、不十分であったり、テストのスケジュールに余裕がなかったりする場合に最も効果が大きい。他の形式的なテスト技法を補完する場合にも効果が大きい。

探索的テストは対処的テスト戦略と密接に関連付けられている（5.2.2 節を参照）。探索的テストは、他のブラックボックス、ホワイトボックス、経験ベースの技法と併用できる。

4.4.3 チェックリストベースドテスト

チェックリストベースドテストでは、チェックリストにあるテスト条件をカバーするように、テストケースを設計、実装、および実行する。テスト担当者は、新しいチェックリストの作成、もしくは既存のチェックリストの拡張をテスト分析の一環として行う。また、既存のチェックリストを変更することなく使用する場合もある。チェックリストは、経験、ユーザーにとって何が重要であるかという知識、もしくはソフトウェアが不合格となる理由と仕組みについての理解に基づいて作成する。

チェックリストは、機能テストや非機能テストを含むさまざまなテストタイプのために作成できる。詳細なテストケースがない場合、チェックリストベースドテストがガイドラインとなり、ある程度の一貫性を実現する。チェックリストは高位レベルのリストであるため、実際にテストをする際には複数の解釈が発生しうる。その結果、広範囲に網羅できるが、記載された内容のまま同じテストを再現するのは困難になる。

5 テストマネジメント

225 分

キーワード

構成管理、欠陥マネジメント、開始基準、終了基準、プロダクトリスク、プロジェクトリスク、リスク、リスクレベル、リスクベースドテスト、テストアプローチ、テストコントロール、テスト見積り、テストマネージャー、テストモニタリング、テスト計画書、テスト計画、テスト進捗レポート、テスト戦略、テストサマリーレポート、テスト担当者

「テストマネジメント」の学習の目的

5.1 テスト組織

FL-5.1.1 （K2）独立したテストの利点と欠点を説明する。

FL-5.1.2 （K1）テストマネージャーとテスト担当者の作業を識別する。

5.2 テストの計画と見積り

FL-5.2.1 （K2）テスト計画書の目的と内容を要約する。

FL-5.2.2 （K2）さまざまなテスト戦略の違いを明確にする。

FL-5.2.3 （K2）可能性のある開始基準と終了基準の例を示す。

FL-5.2.4 （K3）ある特定のテストケースのセットに対するテスト実行のスケジュールを立てるため、優先順位付けの知識、技術的および論理的な依存関係の知識を適用する。

FL-5.2.5 （K1）テストに関連する工数に影響を与える要因を識別する。

FL-5.2.6 （K2）メトリクスを使った見積りと専門家による見積りの違いを説明する。

5.3 テストのモニタリングとコントロール

FL-5.3.1 （K1）テストで使用するメトリクスを想起する。

FL-5.3.2 （K2）テストレポートの目的、内容、および読み手を要約する。

5.4 構成管理

FL-5.4.1 （K2）構成管理がテストをどのように支援するかを要約する。

5.5 リスクとテスト

FL-5.5.1 （K1）可能性と影響を考慮してリスクレベルを定義する。

FL-5.5.2 （K2）プロジェクトリスクとプロダクトリスクの違いを認識する。

FL-5.5.3 （K2）プロダクトリスク分析がテストの十分さ、およびテストの範囲にどのように影響するか例を挙げて説明する。

5.6 欠陥マネジメント

FL-5.6.1 （K3）テスト時に見つかった欠陥を修正するために欠陥レポートを記述する。

5.1 テスト組織

5.1.1 独立したテスト

テストタスクは、テストに特化した役割を持つ人たち、または別の役割を持つ人たち（例えば、顧客）が行う場合がある。ある程度の独立性を確立すると、開発者とテスト担当者の間の認知バイアスの違い（1.5 節を参照）によって、テスト担当者はより効果的に欠陥を発見できる。ただし、独立性は熟知していることの代わりにはならない。開発担当者は自分のコードにある多くの欠陥を効率的に検出できる。

テストにおける独立性の度合いを以下に示す（独立性の低いレベルから高いレベルの順に列挙）。

- 独立したテスト担当者不在（開発担当者が自分のコードをテストするのみである）。
- 開発チーム、またはプロジェクトチーム内に所属する、独立した開発担当者、またはテスト担当者（開発担当者が同僚の成果物をテストすることもある）。
- 組織内にある独立したテストチームまたはグループで、プロジェクトマネージャーや上位管理者の直属組織。
- 顧客またはユーザーコミュニティから派遣された独立したテスト担当者、または、使用性、セキュリティ、性能、規制/標準適合性、移植性など、ある特定のテストタイプを専門に行う独立したテスト担当者。
- 組織外の独立したテスト担当者。オンサイト（インソーシング）またはオフサイト（アウトソーシング）で作業する。

ほぼすべての種類のプロジェクトにおいて、複数のテストレベルのいくつかでは独立したテスト担当者がテストを行うのがよい。開発担当者は自分たちの仕事の成果の品質をコントロールするために、特に低位レベルのテストに参加すべきである。

テストの独立性は、ソフトウェア開発ライフサイクルモデルによって異なる。例えば、アジャイル開発では、テスト担当者を開発チームに参加させる場合がある。アジャイルの手法を使う組織によっては、このようなテスト担当者をより大きな独立したテストチームの一員とみなすこともある。さらに、このような組織では、プロダクトオーナーが各イテレーションの終わりに受け入れテストを実行して、ユーザーストーリーの妥当性確認をすることがある。

独立したテストの利点を以下に示す。

- 独立したテスト担当者は、開発担当者とは異なる背景、技術的視点、バイアスを持つため、開発担当者とは異なる種類の故障を検出する可能性が高い。
- 独立したテスト担当者は、仕様作成時および実装時にステークホルダーが行った仮定について、検証、説明の要求、または反証を行うことができる。

独立したテストの欠点を以下に示す。

- 開発チームから隔絶されると、協力関係の欠落、開発チームへのフィードバック提供の遅延、開発チームとの対立を招くことがある。
- 開発担当者の品質に対する責任感が薄れることがある。
- 独立したテスト担当者は、ボトルネックとして見られたり、リリース遅延で責任を問われたりすることがある。
- 独立したテスト担当者にテスト対象の情報などの重要な情報が伝わらないことがある。

多くの組織が欠点を回避することで、テストの独立性の利点を達成できる。

5.1.2 テストマネージャーとテスト担当者のタスク

本シラバスでは、テストマネージャーとテスト担当者という 2 つの職務を解説している。この 2 つの職務でどのような活動やタスクを行うかは、プロジェクトやソフトウェアの背景、各役割を担当する人のスキルや組織により決まる。

テストマネージャーのタスクは、テストプロセスに対して全体的な責任を持ち、テスト活動においてリーダーシップを適切に発揮することである。テストマネジメントの役割は、プロのテストマネージャーや、プロジェクトマネージャー、開発マネージャー、品質保証マネージャーなどが担う。大きなプロジェクトや組織では、複数のテストチームが 1 人のテストマネージャー、テストコーチ、またはテストコーディネーターの下で作業し、各チームはテストリーダーまたはテスト担当者のリード役が率いることがある。

テストマネージャーの典型的なタスクは以下の通りである。

- 組織のテストポリシーやテスト戦略を開発もしくはレビューする。
- プロジェクトの背景を考慮した上で、テスト目的とリスクを理解してテスト活動を計画する。これにはテストアプローチの選択、テストにかかる時間／工数／コストの見積り、リソースの獲得、テストレベルやテストサイクルの定義、欠陥マネジメントの計画を含む。
- テスト計画書を作成し更新する。
- プロジェクトマネージャー、プロダクトオーナーなどの間でテスト計画書の内容を調整する。
- テスト側の考え方を、統合計画などの他のプロジェクト活動と共有する。
- テストの分析、設計、実装、実行の開始を宣言し、テスト進捗とテスト結果をモニタリングし、終了基準（または完了（done）の定義）のステータスを確認する。
- テスト進捗レポートとテストサマリーレポートを、収集した情報に基づいて準備し配布する。
- （テスト進捗レポートおよび／またはプロジェクトで既に完了しているテストのテストサマリーレポートで報告済みの）テスト結果やテスト進捗に基づいて計画を修正し、テストコントロールのために必要なあらゆる対策を講じる。
- 欠陥マネジメントシステムのセットアップと、テストウェアの適切な構成管理を支援する。
- テスト進捗の計測、およびテストや対象プロダクトの品質の評価のために、適切なメトリクスを導入する。
- テストプロセスで使うツールの選択と実装を支援する。これには、ツール選択（および、購入および／またはサポート）のための予算の提案、パイロットプロジェクトのための時間と工数の割り当て、継続的なツール使用支援の提供を含む。
- 構築するテスト環境を決定する。
- 組織内のテスト担当者、テストチーム、テスト専門職を昇格および支持する。
- テスト担当者のスキルアップとキャリアアップを促す（トレーニング計画、業績評価、コーチングなどを使用）。

テストマネージャーが職務を遂行する方法は、ソフトウェア開発ライフサイクルモデルによって異なる。例えば、アジャイル開発では、前述のタスクのいくつかはアジャイルチームが行う。特にチーム内で行われる日々のテストに関連するタスクは、チーム内のテスト担当者が行うのが一般的である。複数のチームや組織全体にまたがるタスクや、人事に関連するタスクは、開発チーム外のテストマネージャーが行うことが多い。このようなテストマネージャーはテストコーチとも呼ばれる。テストプロセスのマネジメントについてより深く知るためには、『Black 2009』を参照されたい。

テスト担当者の典型的なタスクは以下の通りである。

- テスト計画書のレビューによって貢献する。
- 試験性のために、要件、ユーザーストーリーと受け入れ基準、仕様、モデル（すなわち、テストベース）を分析し、レビューし、評価する。
- テスト条件を識別して文書化し、テストケース、テスト条件、テストベースの間のトレーサビリティを確立する。
- テスト環境（システムアドミニストレーションやネットワークマネジメントと協調する場合が多い）を設計、セットアップし、検証する。
- テストケースとテスト手順を設計し実装する。
- テストデータを作成、および入手する。
- 詳細なテスト実行スケジュールを作成する。
- テストケースを実行し、結果を評価して、期待結果からの逸脱を文書化する。
- 適切なツールを使用して、テストプロセスを円滑にする。
- 必要に応じてテストを自動化する（開発担当者や、テスト自動化の専門家の支援が必要な場合がある）。
- 性能効率性、信頼性、使用性、セキュリティ、互換性、移植性などの非機能特性を評価する。
- 他の人が開発したテストケースをレビューする。

テスト分析、テスト設計、特定のテストタイプ、テストの自動化は、各分野のスペシャリストが担うことがある。プロダクトやプロジェクトのリスク、選択したソフトウェア開発ライフサイクルモデルによるが、テスト担当者の役割を担う人は、テストレベルごとに異なる場合がある。例えば、コンポーネントテストレベルやコンポーネント統合テストレベルでは、開発担当者がテストを担当することが一般的である。受け入れテストレベルでは、ビジネスアナリスト、特定の分野の専門家、ユーザーがテストを担当することが一般的である。また、システムテストレベルやシステム統合テストレベルでは、独立したテストチームがテストを担当することが一般的である。運用受け入れテストレベルでは、運用担当者および／またはシステムアドミニストレーターがテストを担当することが一般的である。

5.2 テストの計画と見積り

5.2.1 テスト計画書の目的と内容

テスト計画書では、開発プロジェクトやメンテナンスプロジェクトのテスト活動を概説する。計画は組織のテストポリシーやテスト戦略、使用する開発ライフサイクルや手法（2.1 節を参照）、テストの範囲、目的、リスク、制約、クリティカルな度合い、試験性（テストを容易にするための考慮）、リソースの調達に基づく。

プロジェクトやテスト計画が進行するに従い、入手できる情報が多くなり、具体的な情報をテスト計画書に含めることができる。テスト計画は継続する活動であり、プロダクトのライフサイクル全体を通して行う（プロダクトのライフサイクルはプロジェクトの範囲を超えて、メンテナンスフェーズまで延長されることに注意する）。テスト活動のフィードバックからリスクの変化を認識し、計画作業を調整する。計画作業は、マスターテスト計画書、テストレベル（システムテストや受け入れテストなど）やテストタイプ（使用性テストや性能テストなど）ごとの個別テスト計画書として文書化できる。テスト計画に含まれる活動には以下があり、そのいくつかはテスト計画書として文書化する場合がある。

- テストの範囲、目的、リスクを決定する。
- テストに対する包括的なアプローチを定義する。

- テスト活動をソフトウェアライフサイクルでの活動に統合し、協調させる。
- 何をテストするか、さまざまなテスト活動でどのような人的リソースとその他のリソースが必要であるか、どのようにテスト活動を進めるかを決める。
- テスト分析、設計、実装、実行、評価の活動を、特定の日付（例えば、シーケンシャル開発）または各イテレーションの状況（例えば、イテレーティブ開発）でスケジューリングする。
- テストのモニタリングとコントロールのためのメトリクスを選ぶ。
- テスト活動の予算を決定する。
- テストドキュメントの詳細レベルと構造を決定する（例えば、テンプレートやサンプルドキュメントを提供する）。

テスト計画書に含まれる内容はさまざまであり、前述以外の活動が含まれることもある。テスト計画書のサンプルは、ISO 標準（ISO/IEC/IEEE 29119-3）にて確認することができる。

5.2.2 テスト戦略とテストアプローチ

テスト戦略は、通常、プロダクトまたは組織のレベルでの、テストプロセスに関する汎用的な考え方を提供する。テスト戦略の一般的な種類は以下の通りである。

- **分析的：**いくつかの要因（要件やリスクなど）の分析に基づく。分析的アプローチの例としては、リスクのレベルに基づいてテストを設計し優先度付けするリスクベースドテストがある。
- **モデルベースド：**プロダクトで必要な特性を表すモデル、例えば、機能、ビジネスプロセス、内部構造、非機能特性（信頼性など）などに基づいて、テストを設計する。そのようなモデルの例としては、ビジネスプロセスモデル、状態モデル、信頼度成長モデルなどがある。
- **系統的：**事前に定義した一連のテストケースまたはテスト条件を体系的に使用する。ここでいうテストケースまたはテスト条件には、一般的または可能性の高い故障を体系的に分類したリスト、重要な品質特性のリスト、モバイルアプリケーションや Web ページに対する企業全体のルックアンドフィール標準などがある。
- **プロセス準拠**（または標準準拠）：外部のルールや標準を使用してテストの分析、設計、実装を行う。使用する外部のルールや標準には、業界固有の標準、プロセスドキュメント、テストベースの厳密な識別や使用、組織によって課せられるプロセスや標準などがある。
- **指導ベース**（コンサルテーションベース）：テストチームの外部または組織自体の外部のステークホルダー、ビジネスドメインの専門家、技術的専門家からの助言、ガイダンス、指示に基づいてテストを行う。
- **リグレッション回避：**既存では実現されていた能力のリグレッションを避けることを目的とする。このテスト戦略には、既存のテストウェア（特にテストケースやテストデータ）、高度に自動化されたリグレッションテスト、および標準テストスイートの再利用が含まれる。
- **対処的：**テスト対象のコンポーネントやシステム、テスト実行時に発生するイベントに対して対処的にテストを行う。他の戦略とは異なり、テストは事前に計画されない。先に実行したテストの結果により得られた知識に応じて、テストを設計および実装し、多くの場合、即座に実行する。対処的戦略では、探索的テストを一般的に使用する。

適切なテスト戦略は、これらの種類のテスト戦略を複数組み合わせて構築する。例えば、リスクベースドテスト（分析的戦略）を探索的テスト（対処的戦略）と組み合わせることができる。これらは相互に補完的であり、併せて使用することでテストがより効果的になる。

テスト戦略はテストプロセスを汎用的に説明したものであり、テストアプローチは特定のプロジェクトまたはリリース用にテスト戦略をテーラリングしたものである。テストアプローチは、テスト技法、テストレベル、テストタイプの選択や、開始基準と終了基準（またはそれぞれ準備完了（ready）の定義と完了（done）の定義）を決めるための出発点である。テスト戦略のテーラリングは、プロジェクトの複雑さやゴール、開発対象プロダクトの種類、プロダクトリスク分析を考慮して行う。選択されたアプロ

ーチはプロジェクトの状況に依存し、リスク、安全性、リソースとスキル、技術、システムの性質（カスタムメイド、COTS）、テスト目的、および法規制を考慮したものである。

5.2.3 開始基準と終了基準（準備完了（ready）の定義と完了（done）の定義）

ソフトウェアおよびテストの品質を効果的にコントロールするために、特定のテスト活動をいつ開始し、いつ完了するかを定義する基準を設ける。開始基準（アジャイル開発では、通常、準備完了（ready）の定義と呼ぶ）は、特定のテスト活動を開始するための事前条件を定義する。開始基準が満たされないままにテスト活動を開始すると、難易度が上がり、要する時間が増え、コストがかかり、リスクが高まる。終了基準（アジャイル開発では、通常、完了（done）の定義と呼ぶ）は、テストレベルまたは一連のテストの完了を宣言するために達成すべき条件を定義する。開始基準と終了基準は、テスト目的に応じて、テストレベルごと、およびテストタイプごとに定義する。

典型的な開始基準を以下に示す。

- テスト可能な要件、ユーザーストーリー、そして／またはモデル（例えば、モデルベースドテスト戦略に従う場合）が準備できている。
- 前のテストレベルで終了基準を満たしたテストアイテムが準備できている。
- テスト環境が準備できている。
- 必要なテストツールが準備できている。
- テストデータや他の必要なリソースが準備できている。

典型的な終了基準を以下に示す。

- 計画したテスト実行が完了している。
- 定義済みのカバレッジ（要件、ユーザーストーリー、受け入れ基準、リスク、コード）を達成している。
- 未解決の欠陥の件数は合意された制限内である。
- 残存欠陥の想定数が十分に少ない。
- 信頼性、性能効率性、使用性、セキュリティ、他の関連する品質特性を十分に評価している。

終了基準を満たしていない場合でも、消費済みの予算、費やした時間、そして／またはプロダクトを市場に送り出すプレッシャーにより、テスト活動を切り上げることが一般的に行われる。プロジェクトステークホルダーやビジネスオーナーが、テストの追加なしでプロダクトをリリースするリスクを見直しして受け入れた場合には、このような状況でもテストを終了できる。

5.2.4 テスト実行スケジュール

テストケースやテスト手順を作成（テスト手順は可能な限り自動化）し、テストスイートにまとめた後、テスト実行スケジュールを作成してテストを実行する順序を定義する。テスト実行スケジュールは、優先度、依存関係、確認テスト、リグレッションテスト、およびテストの実行に最も効率的な順序を考慮して決める。

テストケースの実行順序は優先度レベルに基づくことが理想的であり、最も高い優先度を持つテストケースを最初に実行するのが一般的である。ただし、テストケースまたはテスト対象の機能に依存関係がある場合はこの限りではない。高い優先度を持つテストケースがそれより低い優先度を持つテストケースに依存している場合、低い優先度を持つテストケースを先に実行する。同様に、テストケース間に依存関係がある場合は、相対的な優先度にとらわれずに、適切な順序で実行する。確認テストやリグレッ

ションテストは、変更に関する迅速なフィードバックの重要性に応じて優先度を割り当てる。ただし、この場合も依存関係を考慮する。

効率が異なるテストケースの実行順序が複数考えられる場合には、テスト実行の効率性と優先度の遵守でトレードオフをしなければならない。

5.2.5 テスト工数に影響する要因

テスト工数の見積りでは、特定のプロジェクト、リリース、イテレーションのテスト目的を達成するために必要な作業工数を予測する。テスト工数に影響する要因には、以下に示すプロダクトの特性、開発プロセスの特性、人の特性、テスト結果などが挙げられる。

プロダクトの特性

- プロダクトに関連するリスク
- テストベースの品質
- プロダクトの規模
- プロダクトドメインの複雑度
- 品質特性の要件（例えば、セキュリティ、信頼性）
- テストドキュメントの詳細度に関する要求レベル
- 法規制への適合性の要件

開発プロセスの特性

- 組織の安定度合いと成熟度合い
- 使用している開発モデル
- テストアプローチ
- 使用するツール
- テストプロセス
- 納期のプレッシャー

人の特性

- 参加メンバーのスキルや経験、特にドメイン知識のような類似プロジェクトやプロダクトのスキルや経験
- チームのまとまりとリーダーシップ

テスト結果

- 検出した欠陥の数と重要度
- 必要な再作業の量

5.2.6 テスト見積りの技術

テストを適切に実行するために必要な工数は、さまざまな技術を使用して見積ることができる。最も一般的に使用する 2 つの技術を以下に示す。

- メトリクスを活用する：以前の類似したプロジェクトのメトリクスや、典型的な値を基にしてテスト工数を見積る。

- 専門家の知識を基にする：テストのタスクの所有者の経験、または専門家による見積りを基にしてテスト工数を見積る。

アジャイル開発の例でいえば、バーンダウンチャートは、実績工数をベロシティとして明らかにして次のイテレーションでチームがこなせる作業量を決めることから、メトリクスを基にしたアプローチの例だといえる。一方、プランニングポーカーは、フィーチャーをリリースするために必要な工数をチームメンバー自身の経験に基づいて見積ることから、専門家によるアプローチの例だといえる。（『ISTQB テスト技術者資格制 Foundation Level Extension シラバス アジャイルテスト担当者 日本語版』を参照）

シーケンシャル開発プロジェクトでは、欠陥除去モデルがメトリクスを基にしたアプローチの例である。欠陥の量、およびその欠陥を除去するためにかけた時間を、類似の特性を持つ将来のプロジェクトを見積る際のベースとする。一方、専門家のグループが彼らの経験に基づいて見積りを行う、ワイドバンドデルファイ見積り技法が専門家によるアプローチの例となる（『ISTQB テスト技術者資格制度 Advanced Level シラバス 日本語版 テストマネージャー』を参照）。

5.3 テストのモニタリングとコントロール

テストモニタリングの目的は、テスト活動に関する情報を収集して可視化し、フィードバックをかけることである。モニタリングする情報は、手動あるいは自動で収集し、テスト進捗を評価するために使う。また、テストの終了基準やアジャイルプロジェクトの完了（done）の定義に関連するテストのタスクで、プロダクトリスク、要件、受け入れ基準のカバレッジのターゲットを満たすかを計測するために使う。

テストコントロールとは、収集、報告されたテストの実施結果を基にガイドや補正のアクションをすることである。補正のアクションは、あらゆるテスト活動をカバーし、他のソフトウェアライフサイクルの活動に影響を与える。

テストコントロールの作業には、例えば以下のものがある。

- 識別したリスクが現実になった場合（例えば、リリース遅延）、テストの優先度を見直す。
- テスト環境や他のリソースの利用可否により、テストスケジュールを変更する。
- 再作業が発生した場合に、テストアイテムが開始基準と終了基準を満たすかを再評価する。

5.3.1 テストで使用するメトリクス

メトリクスは各テスト活動の期間中、および終了時に収集できる、以下の項目を評価する。

- 計画したスケジュールや予算に対する進捗
- テスト対象の現在の品質
- テストアプローチの十分性
- テスト目的に対するテスト活動の効果

代表的なテストメトリクスには以下のものがある。

- 計画したテストケースの準備が完了した割合（または、計画したテストケースを実装した割合）
- 計画したテスト環境の準備が完了した割合
- テストケースの実行（例えば、実行/未実行のテストケース数、合格/不合格のテストケース数、そして／または合格/不合格のテスト条件数）
- 欠陥情報（例えば、欠陥密度、検出および修正した欠陥数、故障率、確認テスト結果）
- 要件、ユーザーストーリー、受け入れ基準、リスク、コードのテストカバレッジ

- タスクの完了、リソースの割り当てと稼働状況、工数
- テストに費やすコスト（現状のコストと、継続して欠陥を見つけていくメリットと比較した場合のコスト、もしくは継続してテストを実行していくメリットと比較した場合のコストなど）

5.3.2 テストレポートの目的、内容、読み手

テストレポートの目的は、例えばテストレベルなどのテスト活動の期間中と終了時の両方の時点でテスト活動に関する情報を要約し周知することである。テスト活動の期間中に作成するテストレポートはテスト進捗レポートと呼び、テスト活動の終了時に作成するテストレポートはテストサマリーレポートと呼ぶことが多い。

テストのモニタリングとコントロールでは、テストマネージャーはステークホルダー向けに定期的にテスト進捗レポートを発行する。テスト進捗レポートとテストサマリーレポートに共通な内容に加えて、典型的なテスト進捗レポートは以下を含む。

- テスト活動の状況とテスト計画書に対する進捗
- 進捗を妨げている要因
- 次のレポートまでの間に計画されているテスト
- テスト対象の品質

終了基準を達成したら、テストマネージャーはテストサマリーレポートを作成する。このレポートでは、最新のテスト進捗レポートや他の関連情報に基づいて、行ったテストについての要約を提供する。

典型的にテスト進捗レポートとテストサマリーレポートに含まれるものは以下の通り。

- 行ったテストの要約
- テスト期間中に発生したこと
- 計画からの逸脱（テスト活動のスケジュール、期間もしくは工数など）
- 終了基準または完了（done）の定義に対するテストとプロダクト品質の状況
- 進捗を妨げた、または引き続き妨げている要因
- 欠陥、テストケース、テストカバレッジ、活動進捗、リソース消費のメトリクス（例えば、5.3.1で示したメトリクス）
- 残存リスク（5.5 節を参照）
- 再利用可能なテスト作業成果物

テストレポートの内容は、プロジェクト、組織の要件、ソフトウェア開発ライフサイクルによって異なる。例えば、多くのステークホルダーが関与する複雑なプロジェクトや法規制を受けるプロジェクトでは、簡易なソフトウェア更新のプロジェクトで作るレポートよりも詳細かつ厳密なレポートが必要になる。また、アジャイル開発では、テスト進捗レポートをタスクボード、欠陥サマリー、バーンダウンチャートに組み込んで、日々のスタンドアップミーティングで討議することがある（『ISTQB テスト技術者資格制度 Foundation Level Extension シラバス アジャイルテスト担当者 日本語版』を参照）。

また、テストレポートの内容は、プロジェクトの状況に応じて、さらにはレポートの読み手に応じてテーラリングする。記述する情報の種類と量は、技術に精通した読み手またはテストチームを対象にする場合と経営者向けのサマリーレポートでは異なる。前者では、欠陥のタイプと傾向に関する詳細な情報が重要である。後者では、高位レベルのレポート（例えば、優先度ごとの欠陥、予算、スケジュール、合格/不合格/テスト未実行のテスト条件などのステータスサマリー）が適切である。

ISO 標準（ISO/IEC/IEEE 29119-3）は、テストレポートの種類として、テスト進捗レポートとテスト完了レポート（本シラバスでの呼称は「テストサマリーレポート」）を定義し、構成と記載例を示している。

5.4 構成管理

構成管理の目的は、プロジェクトやプロダクトのライフサイクルを通じて、コンポーネントやシステムの完全性、テストウェア、およびそれら相互の関係を確立し、維持することである。

テストを適切に支援するために、構成管理では、以下を明確にする。

- 全テストアイテムを一意に識別して、バージョンコントロールを行い、変更履歴を残し、各アイテム間を関連付ける。
- テストウェアの全アイテムを一意に識別して、バージョンコントロールを行い、変更履歴を残し、各アイテム間を関連付ける。また、テストアイテムのバージョンとの関連付けを行い、テストプロセスを通してトレーサビリティを維持できる。
- 識別したすべてのドキュメントやソフトウェアアイテムは、テストドキュメントに明確に記載してある。

テスト計画にて構成管理の手順やインフラストラクチャー（ツール）を識別し、実装すべきである。

5.5 リスクとテスト

5.5.1 リスクの定義

リスクは、将来に否定的な結果となる事象が起きる恐れを含む。リスクのレベルは、そのような事象が起きる可能性とその影響（事象による結果がもたらす損害）で決まる。

5.5.2 プロダクトリスクとプロジェクトリスク

プロダクトリスクは、作業成果物（例えば、仕様、コンポーネント、システム、テストケース）がユーザー、および／またはステークホルダーの正当なニーズを満たすことができない恐れを含む。プロダクトの特定の品質特性（例えば、機能適合性、信頼性、性能効率性、使用性、セキュリティ、互換性、保守性、移植性）に関連するプロダクトリスクは、品質リスクとも呼ばれる。プロダクトリスクの例には以下のものが含まれる。

- ソフトウェアの意図されている機能が仕様通りには動かないかもしれない 。
- ソフトウェアの意図されている機能がユーザー、顧客、および／またはステークホルダーのニーズ通りには動かないかもしれない。
- システムアーキテクチャーが非機能要件を十分にサポートしないことがある。
- 特定の計算結果が状況によって正しくないことがある。
- ループ制御構造が正しくコーディングされていないことがある。
- 高性能トランザクション処理システムで応答時間が適切でないことがある。
- ユーザーエクスペリエンス（UX）のフィードバックがプロダクトの期待と異なるかもしれない。

プロジェクトリスクは、発生した場合にプロジェクトの目的達成に悪影響を与える。プロジェクトリスクの例には以下のものが含まれる。

- プロジェクトの懸念事項：
    - リリース、タスク完了、終了基準または完了（done）の定義の達成が遅延することがある。

- 不正確な見積り、優先度の高いプロジェクトへの資金の再割り当て、組織全体での経費節減により資金が不足することがある。
- プロジェクト終盤での変更により作業の大規模なやり直しが必要な場合がある。

- 組織の懸念事項：
  - 人員不足、および人員のスキルやトレーニング不足の場合がある。
  - 人間関係によって、衝突や問題が発生することがある。
  - ビジネス上の優先度の競合によってユーザー、ビジネススタッフ、特定の分野の専門家の都合がつかないことがある。
- 政治的な懸念事項：
  - テスト担当者が自分たちのニーズおよび／またはテスト結果の十分性を上手く伝えられないことがある。
  - 開発担当者および／またはテスト担当者がテストやレビューで見つかった事項を上手くフォローアップできないことがある（例えば、開発、およびテストが改善しない）。
  - テストから期待できるものを正しく評価しようとしないことがある（例えば、テストで見つかった欠陥の情報を真摯に受け止めようとしない）。
- 技術的な懸念事項：
  - 要件を十分に定義できないことがある。
  - 制約があるために、要件を満たさないことがある。
  - テスト環境が予定した期限までに用意できないことがある。
  - データ変換および移行の計画、それらのツールによる支援が遅れることがある。
  - 開発プロセスの弱点が、設計、コード、構成、テストデータ、テストケースなどのプロジェクトでの作業成果物間の整合性や品質に影響を与えることがある。
  - 不適切な欠陥マネジメントおよび類似の問題によって、欠陥や他の技術的負債が累積することがある。
- 供給者側の懸念事項：
  - サードパーティが、必要なプロダクトまたはサービスを提供できない、もしくは撤退することがある。
  - 契約上の懸念事項がプロジェクトの問題の原因となることがある。

プロジェクトリスクは、開発活動とテスト活動の両方に影響する。プロジェクトマネージャーがすべてのプロジェクトリスクを処理する責任を負うことはあるが、テストマネージャーがテストに関連するプロジェクトリスクの責任を負うことは珍しいことではない。

5.5.3 リスクベースドテストとプロダクト品質

テスト時に必要な工数についてどこに重点を置くかを決定するために、リスクを使用する。また、どのテストをいつから開始するかを決定し、さらなるテストが必要な領域を識別するために使用する。その一方、テストは、期待しない事象が発生するリスクを減らしたり、その影響を少なくしたりするために使用する。また、リスク軽減のための活動として使用し、識別したリスクや残存（未解決）リスクに関するフィードバックを提供する。

リスクベースのアプローチを採用すると、プロダクトリスクを軽減する予防的措置を講じられる。このアプローチではリスク分析を行い、プロダクトリスクを識別し、各リスクの発生する可能性と発生した場合の影響を評価する。この活動を通して得られるプロダクトリスクの情報を、テスト計画、仕様化、テストケースの準備と実行、テストのモニタリングとコントロールに役立てる。プロダクトリスクを早期に分析することは、プロジェクトの成功に貢献する。

リスクベースのアプローチでは、プロダクトリスク分析の結果を使用して以下を行う。

- 適用するテスト技法を決める。
- 実施するテストレベルおよびテストタイプを決める（例えば、セキュリティテストやアクセシビリティテスト）。
- テストを実行する範囲を決める。
- 重大な欠陥をなるべく早い時期に検出するため、テストの優先順位を決める。
- テスト以外の活動でリスクを減らす方法があるか検討する（経験の少ない設計者に教育を実施するなど）。

リスクベースドテストでは、プロダクトリスクの分析を行うために、プロジェクトのステークホルダーの総合的な知識や洞察力を必要とする。プロダクトが故障を起こす可能性を最小にするため、リスクマネジメントでは、以下のように、きちんと決まったアプローチを備えている必要がある。

- どこが上手くいかないか（リスク）を分析する（定期的に再評価する）。
- リスクの重要度を決める。
- 重大なリスクを処理する方法を実装する。
- リスクが実際に事象として発生した場合に備えてコンティンジェンシープランを作る。

さらに、テストにより、新しいリスクを洗い出したり、どのリスクを軽減すべきかを決めたり、リスクの不明確さを少なくしたりすることができる。

5.6 欠陥マネジメント

テストの目的の 1 つは欠陥を検出することであるため、テスト時に見つかった欠陥は記録を残す必要がある。欠陥を記録する方法は、テスト対象のコンポーネントまたはシステム、テストレベル、ソフトウェア開発ライフサイクルモデルなどの要因により異なる。検出したすべての欠陥を調査し、発見や分類から解決にいたるまで追跡する必要がある（例えば、欠陥修正や解決策の確認テストでの合格、後続のリリースへ延期、永続的なプロダクト制約としての受け入れなど）。すべての欠陥を解決にいたるまでマネジメントするために、組織で欠陥マネジメントのプロセスを定め、ワークフローと分類の規則を設ける必要がある。このプロセスは、欠陥マネジメントに参加するすべての関係者（設計者、開発担当者、テスト担当者、プロダクトオーナーなど）によって合意される必要がある。欠陥の記録や追跡を非形式的に行う組織もある。

欠陥マネジメントのプロセスで、一部のレポートが偽陽性で、実際には、欠陥による故障ではないと判明することがある。例えば、ネットワーク接続の切断やタイムアウトによってテストが不合格となることがある。この振る舞いはテスト対象に存在する欠陥が引き起こしたわけではないが、不正として調査する必要がある。テスト担当者は、欠陥として報告する偽陽性の数を最小限にする必要がある。

欠陥は、コーディング、静的解析、レビュー、動的テストの間、もしくはソフトウェアプロダクトを使用している際に報告されることがある。また、コードや稼動システムの懸念事項、もしくは要件、ユーザーストーリーと受け入れ基準、開発ドキュメント、テストドキュメント、ユーザーマニュアル、インストールガイドなどのあらゆる種類のドキュメントの懸念事項に対して報告されることもある。効果的で効率的な欠陥マネジメントプロセスのために、欠陥の属性、分類、ワークフローを定義する。

典型的な欠陥レポートには、以下のような目的がある。

- 開発担当者や他の関係者に対して、発生したあらゆる期待に反する事象についての情報を提供する。また、必要に応じて、もしくは問題を解決するために、具体的な影響を識別して、最小の再現テストで問題の切り分けを行い、欠陥の修正ができるようにする。
- テストマネージャーに対し、作業成果物の品質や、テストへの影響を追跡する手段を提供する（欠陥の報告数が多いと、テスト担当者は多くの時間をテスト実行ではなく報告作業に費やす必要があり、さらに多くの確認テストが必要になる）。
- 開発プロセスとテストプロセスを改善するためのヒントを提供する。

動的テスト時の欠陥レポートには、一般的に以下の情報を記入する。

- 識別子
- レポート対象の欠陥の件名と概要
- 欠陥レポートの作成日付、作成した組織、作成者
- テストアイテム（テスト対象の構成アイテム）および環境を識別する情報
- 欠陥を観察した開発ライフサイクルのフェーズ
- ログ、データベースのダンプ、スクリーンショット、（テスト実行中に検出された場合は）実行状況などの欠陥の再現と解決を可能にする詳細な説明資料
- 期待結果と実際の結果
- ステークホルダーに与えるインパクトの範囲や程度（重要度）
- 修正の緊急度/優先度
- 欠陥レポートのステータス（例えば、オープン、延期、重複、修正待ち、確認テスト待ち、再オープン、クローズなど）
- 結論、アドバイス、承認
- 欠陥の修正が他の領域への影響を与えるといった、広範囲にわたる懸念事項
- プロジェクトチームのメンバーによる欠陥の切り分け、修正、確認といった一連の修正履歴
- 問題を明らかにしたテストケースを含む参照情報

これらの詳細情報の一部は、欠陥マネジメントツールを使用することで自動的にレポートに含める、および／または取り扱うことができる。例えば、識別子の割り当て、ワークフローでの欠陥レポートのステータスの割り当てや更新を自動的に行うことができる。静的テスト、特にレビューで検出した欠陥は、例えばレビューミーティング資料など別の方法で文書化するのが一般的である。

欠陥レポートの内容の例については、ISO（ISO/IEC/IEEE 29119-3）にて参照できる（この標準では、「欠陥レポート」を「インシデントレポート」と呼ぶ）。

6 テスト支援ツール

40 分

キーワード

データ駆動テスト、キーワード駆動テスト、性能テストツール、テスト自動化、テスト実行ツール、テストマネジメントツール

「テストツール」の学習の目的

6.1 テストツールの考慮事項

FL-6.1.1 （K2）テストツールをその目的と支援するテスト活動に従って分類する。

FL-6.1.2 （K1）テスト自動化の利点とリスクを識別する。

FL-6.1.3 （K1）テスト実行ツールとテストマネジメントツールの特別な考慮事項を記憶する。

6.2 ツールの効果的な使い方

FL-6.2.1 （K1）ツールを選択する際の主な原則を識別する。

FL-6.2.2 （K1）ツールを導入する際にパイロットプロジェクトを使用する目的を想起する。

FL-6.2.3 （K1）組織内でテストツールの評価、実装、導入、継続的なサポートを成功させる要因を識別する。

6.1 テストツールの考慮事項

テストツールを使用して、1 つ以上のテスト活動を支援できる。以下のようなツールを利用できる。

- テストに直接利用できるツール（例えば、テスト実行ツールやテストデータ準備ツールなど）
- 要件、テストケース、テスト手順、自動化テストスクリプト、テスト結果、テストデータ、欠陥などを上手く取り扱うツール、およびテスト実行のモニタリングとレポートのためのツール
- 調査と評価に利用できるツール
- テストに役立つあらゆるツール（この意味では表計算ソフトウェアもテストツールである）

6.1.1 テストツールの分類

テストツールは、状況に応じ、以下に示すもののうちの 1 つ以上を目的とする。

- 手動で行うと大量のリソースを必要とする反復作業を自動化することによって、テスト活動の効率を改善する（テスト実行、リグレッションテスト）。
- テストプロセス全体を通して、手動で行うテスト活動を支援することによって、テスト活動の効率を改善する（1.4 節を参照）。
- テストの一貫性や欠陥の再現性を高めることで、テスト活動の品質を改善する。
- 手動では実行できない活動（例えば、大規模な性能テスト）を自動化する。
- （例えば、大量のデータの比較を自動化、または動作のシミュレーションをすることにより、）テストの信頼度を向上させる。

ツールは、目的、価格、ライセンスモデル（有料、オープンソースなど）、使用している技術といったさまざまな基準で分類できる。本シラバスでは、テストツールが支援するテスト活動によって、ツールを分類する。

ツールには、明確に 1 つの活動しか支援しないツールだけでなく、複数の活動を支援するツールがあるが、それらは代表的な活動で分類する。また、単一のプロバイダーによるツールには、連携するように設計された複数のツールが 1 つのパッケージとして提供されることもある。

テストツールの中には、テストの実行結果に影響を及ぼすものがある。例えば、性能テストツールによって実行される余分な命令のせいで、アプリケーションの応答時間が実際とは異なってしまうかもしれないし、あるいは、達成したコードカバレッジの量がカバレッジツールを使用したせいで変わってしまうかもしれない。このようにツールが結果に影響を及ぼすことを、プローブ効果と呼ぶ。

テストツールの中には、一般的に、テスト担当者より開発担当者に効果の高いものもある（例えば、コンポーネントテストやコンポーネント統合テスト時に使用されるツールなど）。その種のツールは、（D）の印を付けて以下に列挙した。

テストとテストウェアのマネジメントの支援ツール

マネジメント用ツールによっては、ソフトウェア開発ライフサイクル全体であらゆるテスト活動に適用できる。テストとテストウェアのマネジメントを支援するツールには、以下のものがある。

- テストマネジメントツールとアプリケーションライフサイクルマネジメントツール（ALM）
- 要件マネジメントツール（テスト対象へのトレーサビリティなど）
- 欠陥マネジメントツール

- 構成管理ツール
- 継続的インテグレーションツール（D）

静的テストの支援ツール

静的テストツールの対象となる活動や利点は、第 3 章で説明している。これらのツールには、以下のものがある。

- レビューを支援するツール
- 静的解析ツール（D）

テスト設計とテスト実装の支援ツール

テスト設計ツールは、テストの設計と実装で保守性の高い作業成果物（例えば、テストケース、テスト手順、テストデータ）を作成できるように支援する。これらのツールには、以下のものがある。

- テスト設計ツール
- モデルベースドテストツール
- テストデータ準備ツール
- 受け入れテスト駆動開発（ATDD）ツールや振る舞い駆動開発（BDD）ツール
- テスト駆動開発（TDD）ツール（D）

ツールによっては、テストの設計と実装を支援するだけでなく、テスト実行や結果記録を支援したり、テスト実行や結果記録を支援する他のツールに出力結果を提供したりすることができる。

テスト実行と結果記録の支援ツール

テスト実行と結果記録の活動を支援する多くのツールがある。これらのツールには、以下のものがある。

- テスト実行ツール（例えば、リグレッションテストの実行）
- カバレッジツール（例えば、要件カバレッジ、コードカバレッジ（D））
- テストハーネス（D）
- ユニットテストフレームワークツール（D）

性能計測と動的解析の支援ツール

性能テストとロードテストは手動では効果的に実行できないため、これらのテスト活動では性能計測ツールと動的解析ツールが必須になる。これらのツールには、以下のものがある。

- 性能テストツール
- モニタリングツール
- 動的解析ツール（D）

特定のテストに対する支援ツール

全般的なテストプロセスを支援するツールに加えて、特定のテストの問題を支援するツールも存在する。このようなツールは、以下の目的に使用される。

- データ品質の評価
- データのコンバージョンとマイグレーション
- 使用性テスト
- アクセシビリティテスト
- ローカライゼーションテスト
- セキュリティテスト
- 移植性テスト（例えば、複数のサポート対象プラットフォームにまたがるソフトウェアのテスト）

6.1.2 テスト自動化の利点とリスク

単にツールを取得しても、成功は保証されない。組織に対して新たに導入した各ツールの本来の効果を継続的に上げるには、相応の工数が必要になる。テストでツールを使うことは潜在的に利点があり、使うのに適した機会も多いが、リスクもある。このことは特にテスト実行ツールに当てはまる（テスト自動化と呼ばれる）。

テスト実行を支援するツールを使う潜在的な利点は以下の通りである。

- 反復する手動作業の削減と時間の節約ができる（例えば、リグレッションテストの実行、環境の準備/復旧タスク、同じテストデータの再入力、コーディング標準準拠のチェックなど）。
- 一貫性や再実行性が向上する（例えば、整合性のあるテストデータ、同じ頻度と順序でのテスト実行、要件からの一貫したテストケースの抽出など）。
- 評価の客観性が向上する（例えば、静的な計測、カバレッジなど）。
- テストに関する情報へのアクセスの容易性が向上する（例えば、テスト進捗、欠陥率や性能計測結果の集計、およびグラフの作成など）。

テストを支援するツールを使う潜在的なリスクは以下の通りである。

- テストツールの効果を過大に期待する（例えば、ツールの機能や使いやすさなど）。
- テストツールを初めて導入する場合に要する時間、コスト、工数を過小評価する（例えば、教育や、外部の専門家の支援など）。
- 大きな効果を継続的に上げるために必要な時間や工数を過小評価する（例えば、テストプロセスの変更、ツールの使用法の継続的な改善など）。
- ツールが生成するテスト資産をメンテナンスするために必要な工数を過小評価する。
- ツールに過剰な依存をする（テスト設計またはテスト実行と置き換えられると考える、または手動テストの方が適したケースで自動テストを利用できると考えるなど）。
- ツール内にあるテスト資産のバージョン管理を怠る。
- 重要なツール間での関係性と相互運用性の問題を無視する。例えば、要件マネジメントツール、構成管理ツール、欠陥マネジメントツール、および複数のベンダーから提供されるツールなど。
- ツールベンダーがビジネスを廃業したり、ツールの販売から撤退したり、別のベンダーにツールを売ったりするリスクがある。
- ツールのサポート、アップグレード、欠陥修正に対するベンダーの対応が悪い。
- オープンソースプロジェクトが停止する。

- 新しいプラットフォームや新規技術をサポートできない。
- ツールに対する当事者意識が明確でない（例えば、助言や手助け、および更新など）。

6.1.3 テスト実行ツールとテストマネジメントツールの特別な考慮事項

組織にてテスト実行ツールとテストマネジメントツールを選択および統合する際には、実装を支障なく正常に完了するために、多くの事項を考慮する必要がある。

テスト実行ツール

テスト実行ツールは、自動テストスクリプトを使ってテスト対象を実行する。この種類のツールで大きな効果を出すには、大量の工数をかけねばならないことが多い。

手動テストの担当者の操作をそのまま記録することでテストの実行手順をキャプチャーするのは魅力的である。しかし、このやり方は大量のテストスクリプトを活用するよう拡張できない。テストをキャプチャーしたときのスクリプトは、特定のデータや動作で線形表現した、各スクリプトの一部にすぎない。この種のスクリプトは、予期しない事象が起きると、動作が不安定になる。システムのユーザーインターフェースには常に更新がかかるため継続的なメンテナンス作業が必要であるが、最新世代のテスト実行ツールは「スマート」イメージキャプチャー技術を利用しており、この種類のツールの有用性を向上させている。

データ駆動によるテストアプローチでは、テスト入力と期待結果をスプレッドシートに記述してスクリプトから分離する。そして、入力データを読み込み、異なるテストデータで同じテストスクリプトを実行する汎用的なテストスクリプトを使う。スクリプト言語に慣れていないテスト担当者は、あらかじめ定義済みのスクリプトに対し、新しいテストデータを作ればよい。

キーワード駆動によるテストアプローチでは、実行する動作を定義するキーワード（アクションワードとも呼ぶ）を汎用スクリプトが処理し、その後、キーワードスクリプトを呼び出して付随するテストデータを処理する。スクリプト言語に慣れていないテスト担当者でも、テスト対象のアプリケーション用に調整したキーワードと付随するデータを使ってテストケースを定義できる。データ駆動によるテストアプローチとキーワード駆動によるテストアプローチの詳細および例については、『ISTQB-TAE Advanced Level Test Automation Engineer Syllabus』、『Fewster 1999』、『Buwalda 2001』に記載されている。

上記のアプローチでは、テスト担当者、開発担当者、テスト自動化のスペシャリストの誰かがスクリプト言語に精通している必要がある。どのようなスクリプト技術を使用したとしても、各テストの「期待結果」は、テストの実行結果と動的に比較するか、蓄積しておいて実行後に比較する必要がある。

モデルベースドテスト（MBT）ツールでは、機能仕様をアクティビティ図などにモデル化できる。この作業は主にシステム設計担当者が行う。MBT ツールはモデルを解釈してテストケース仕様を作成する。これらのテストケースは、テストマネジメントツールに格納し、そして／またはテスト実行ツールを使用して実行する（『ISTQB-MBT Foundation Level Model-Based Tester Extension Syllabus』を参照）。

テストマネジメントツール

テストマネジメントツールは、以下に示すさまざまな理由により、他のツールやスプレッドシートとのインターフェースが必要である。

- 組織が必要とするフォーマットで利用できる情報を作成する
- 要件マネジメントツールで要件に対する一貫したトレーサビリティを維持する
- 構成管理ツールでテスト対象のバージョン情報と同期する

これは、テストマネジメントモジュール（および欠陥マネジメントシステム）に加えて、組織内の異なるグループが使用する他のモジュール（例えば、プロジェクトスケジュールや予算情報）を含む統合ツール（例えば、アプリケーションライフサイクルマネジメント）を使用する場合に特に重要になる。

6.2 ツールの効果的な使い方

6.2.1 ツールを選択する際の基本原則

組織向けにツールを選択する際の基本原則は、以下の通りである。

- 組織の成熟度、長所と短所を評価する。
- ツールを活用するためにテストプロセスを改善する機会を識別する。
- テスト対象で使用する技術を理解して、その技術と互換性のあるツールを選択する。
- 組織内で既に使用しているビルドツールや継続的インテグレーションツールとの互換性と統合の可否を明らかにする。
- 明確な要件と客観的な基準を背景にツールを評価する。
- 無料試行期間があるかどうか（およびその期間）を確認する。
- ツールベンダー（トレーニングや、サポートメニュー、ビジネス的な要素も含む）、もしくは無料ツール（オープンソースツールなど）のサポートを評価する。
- ツールを使用するためのコーチングおよびメンタリングに関する組織内での要件を識別する。
- ツールに直接携わる担当者のテスト（およびテスト自動化）スキルを考慮して、トレーニングの必要性を評価する。
- さまざまなライセンスモデル（有料、オープンソースなど）の長所と短所を考慮する。
- 具体的なビジネスケースに基づいて、費用対効果を見積る（必要に応じて）。

最後に、「コンセプトの証明（proof-of-concept）」を行う。つまり、テスト対象のソフトウェアに対して、現在のインフラストラクチャーの中で効果的に使用できるツールなのかを立証する。必要に応じて、ツールを効果的に使うために必要となるインフラストラクチャーの変更点を識別する。

6.2.2 ツールを組織に導入するためのパイロットプロジェクト

コンセプトの証明が成功し、ツールの選定が完了したした後、一般的には、組織へのツール導入をパイロットプロジェクトから始める。パイロットプロジェクトには以下のような目的がある。

- ツールに関する知識を深め、強みと弱みを理解する。
- 現状のプロセスや実践しているやり方にツールをどのように適用するかを評価する。そして何を変更する必要があるかを特定する。
- ツールやテスト資産の標準的な使用方法、管理方法、格納方法、メンテナンス方法を決める（例えば、ファイルやテストケースの命名規約の決定、コーディング規約の選択、ライブラリーの作成、およびテストスイートをモジュール化した際の分割度合いの良し悪しの定義など）。
- 期待する効果が妥当なコストで実現可能かどうかを見極める。
- ツールによって収集およびレポートをさせたいメトリクスを理解し、メトリクスを確実に記録しレポートするようにツールを設定する。

6.2.3 ツール導入の成功要因

組織内でツールの評価、実装、導入、日々のサポートを成功させるには、以下が必要になる。

- ツール未使用の部署にツールを順々に展開する。

- ツールが適用できるよう、プロセスを調整、改善する。
- ツールのユーザーに対し、トレーニング、コーチング、メンタリングを行う。
- 利用ガイドを定める（例えば、組織内の自動化標準）。
- ツールを実際に使用する中で得られる情報の集約方法を実装する。
- ツールの利用状況や効果をモニタリングする。
- ツールのユーザーサポートを提供する。
- すべてのユーザーから、得られた教訓を集める。

開発とは別の運用に責任を持つ組織および／またはサードパーティのサプライヤーなども巻き込んで、ツールが技術的にも組織的にもソフトウェア開発ライフサイクルに統合できることを確認することも重要である。

テスト実行ツールの使用に関する体験談やアドバイスについては、『Graham 2012』を参照されたい。

7 参考文献

標準ドキュメント

ISO/IEC/IEEE 29119-1 (2013) Software and systems engineering - Software testing - Part 1: Concepts and definitions

ISO/IEC/IEEE 29119-2 (2013) Software and systems engineering - Software testing - Part 2: テストプロセス

ISO/IEC/IEEE 29119-3 (2013) Software and systems engineering - Software testing - Part 3: テストドキュメント

ISO/IEC/IEEE 29119-4 (2015) Software and systems engineering - Software testing - Part 4: テスト技法

ISO/IEC 25010, (2011) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models (日本では JIS X 25010)

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

UML 2.5, Unified Modeling Language Reference Manual, http://www.omg.org/spec/UML/2.5.1/, 2017

ISTQB ドキュメント

ISTQB 用語集

ISTQB Foundation Level シラバス 日本語版 概要

ISTQB-MBT Foundation Level Extension Model-Based Tester Extension Syllabus

ISTQB テスト技術者資格制度 Foundation Level Extension シラバス アジャイルテスト担当者 日本語版

ISTQB テスト技術者資格制度 Advanced Level シラバス 日本語版 テストアナリスト

ISTQB テスト技術者資格制度 Advanced Level シラバス 日本語版 テストマネージャー

ISTQB テスト技術者資格制度 Advanced Level シラバス 日本語版 テクニカルテストアナリスト

ISTQB-SEC Advanced Level Security Tester Syllabus

ISTQB-TAE Advanced Level Test Automation Engineer Syllabus

ISTQB-ETM Expert Level Test Management Syllabus

ISTQB-EITP Expert Level Improving the Test Process Syllabus

書籍と記事

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

Black, R. (2017) *Agile Testing Foundations*, BCS Learning & Development Ltd: Swindon UK

Black, R. (2009) *Managing the Testing Process (3e)*, John Wiley & Sons: New York NY

Buwalda, H. et al.(2001) *Integrated Test Design and Automation*, Addison Wesley: Reading MA

Copeland, L. (2004) *A Practitioner's Guide to Software Test Design*, Artech House: Norwood MA

Craig, R. and Jaskiel, S. (2002) *Systematic Software Testing*, Artech House: Norwood MA

Crispin, L. and Gregory, J. (2008) *Agile Testing*, Pearson Education: Boston MA

Fewster, M. and Graham, D. (1999) *Software Test Automation*, Addison Wesley: Harlow UK

Gilb, T. and Graham, D. (1993) *Software Inspection*, Addison Wesley: Reading MA

Graham, D. and Fewster, M. (2012) *Experiences of Test Automation*, Pearson Education: Boston MA

Gregory, J. and Crispin, L. (2015) *More Agile Testing*, Pearson Education: Boston MA

Jorgensen, P. (2014) *Software Testing, A Craftsman's Approach (4e)*, CRC Press: Boca Raton FL

Kaner, C., Bach, J. and Pettichord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons: New York NY

Kaner, C., Padmanabhan, S. and Hoffman, D. (2013) *The Domain Testing Workbook*, Context-Driven Press: New York NY

Kramer, A., Legeard, B. (2016) *Model-Based Testing Essentials: Guide to the ISTQB Certified Model-Based Tester: Foundation Level*, John Wiley & Sons: New York NY

Myers, G. (2011) *The Art of Software Testing*, (3e), John Wiley & Sons: New York NY

Sauer, C. (2000) "The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research," *IEEE Transactions on Software Engineering*, *Volume 26, Issue 1, pp 1-*

Shull, F., Rus, I., Basili, V. July 2000."How Perspective-Based Reading can Improve Requirement Inspections."*IEEE Computer, Volume 33, Issue 7, pp 73-79*

van Veenendaal, E. (ed.) (2004) *The Testing Practitioner* (第 8 章～第 10 章), UTN Publishers: The Netherlands

Wiegers, K. (2002) *Peer Reviews in Software*, Pearson Education: Boston MA

Weinberg, G. (2008) *Perfect Software and Other Illusions about Testing*, Dorset House: New York NY

他の資料（本シラバスでの直接の参照はない）

Black, R., van Veenendaal, E. and Graham, D. (2012) *Foundations of Software Testing: ISTQB Certification (3e)*, Cengage Learning: London UK

Hetzel, W. (1993) *Complete Guide to Software Testing (2e)*, QED Information Sciences: Wellesley MA

Spillner, A., Linz, T., and Schaefer, H. (2014) *Software Testing Foundations (4e)*, Rocky Nook: San Rafael CA

8 付録 A - 本シラバスの背景

このドキュメントができるまでの経緯

本ドキュメントは、ISTQB（www.istqb.org）によって承認された、国際的資格の初級レベルにあたる、ISTQBテスト技術者資格制度Foundation Levelのシラバスである。

本ドキュメントは、国際ソフトウェアテスト資格認定委員会（ISTQB）によって選任されたメンバーにより構成されるWorking Groupによって、2014年から2018年の間に作成された。2018年度版は、まず、すべてのISTQBメンバー委員会の代表によってレビューされ、その後、国際ソフトウェアテストコミュニティから選ばれた代表によってレビューされた。

Foundation Level 資格認定の目的

- テストがきわめて重要で専門的なソフトウェアエンジニアリングの一領域であるという認識を得るため。
- テスト技術者のキャリア形成のための基準を提供するため。
- テストのプロフェッショナルとして資格認定を受けていることを、上層部、顧客、そして職場内にて認知してもらうようにし、テスト技術者の評価を上げるため。
- 一貫して、よいテストのプラクティスをすべてのソフトウェアエンジニアリング領域へ広めるため。
- テストのトピックが産業にて適切で価値のあるものだという認識を得るため。
- ソフトウェア供給者側が資格認定を受けたテスト技術者を雇うことができるようになり、さらに、ソフトウェア供給者側にとってテスト有資格者の人材採用方針が、他の競争相手より商業的に有利に立てるようにするため。
- テスト技術者やテストに興味がある人に国際的に認められた資格認定を身につける機会を与えるため。

国際資格認定の目的

- 国をまたいでテストの知識を同等の基準で判断できるため。
- テスト技術者がさらに簡単に国をまたいで仕事をできるようにするため。
- 多国籍/国際的なプロジェクトがテストに関して共通の認識を持つようにするため。
- 世界的にテストの資格認定者数を増やすため。
- 一国だけでアプローチするより、国際的な基準を背景にすることでさらに価値を持たせるため。
- シラバスや用語集などを通して、国際的にテストへの理解と知識の体系を普及させ、すべての本資格認定に関わる人たちの知識レベルを上げるため。
- テストがプロフェッショナルな能力を必要とすることを多くの国々で普及させるため。
- テスト技術者が各国の言葉を用いて資格認定を取得できるようにするため。
- 知識とリソースを各国で分かち合うことができるようにするため。
- 多くの国が本資格認定に関わることで、テストの人材と本資格の国際的認知度を高めるため。

本資格認定試験の受験要件

ISTQBテスト技術者資格制度Foundation Level試験へ申し込むための基準は、志願者がソフトウェアテストに興味を持っているか、ということであるが、志願者が以下のことについても満たしていることが強く求められる。

- システムテストやユーザー受け入れテストなどのテスト担当、もしくはソフトウェア開発担当など、最低 6 か月程度、技術者としての経歴があること。
- ISTQB に加盟している各国の委員会（日本では JSTQB が該当する）によって認定された教育コースを受けていること。

ソフトウェアテスト Foundation Level の資格認定の経緯と歴史

独立したソフトウェアテストの資格認定は、1998年にイギリスのブリティッシュコンピューターソサエティ情報システム研究委員会（ISEB）によってソフトウェアテスト委員会が設置されたときからスタートした（www.bcs.org.uk/iseb）。2002年には、ドイツのASQFがドイツのテスト技術者資格認定のスキームを支援し始めた（www.asqf.de）。本シラバスは、ISEBとASQFのシラバスが基となっている。これは再編成され、内容を更新、追加したものとなっている。この際最も重要視したことは、テスト技術者にとってより実用的になるように心がけたことである。

この国際資格がスタートする前に認定されたソフトウェアテストFoundation Levelの資格（例えば、ISEB、ASQF、またはISTQBが認可した各国の委員会によるもの）は国際資格と同等のものだと見なす。Foundation Leveの資格認定は、期限が切れることはなく、更新の必要もない。資格が与えられた日付が認定書に記されている。

各々の参加国において、そこでの見解はISTQBに加盟している各国の委員会（日本ではJSTQBが該当する）にて管理される。各国の委員会の任務はISTQBによって規定されているが、各国内の小委員会によって履行される。各国内の小委員会の任務は、教育機関の認定と資格試験の設定も含まれる。

9 付録 B - 学習している知識の目的と認知レベル

本シラバスに当てはまるものとして、以下の学習の目的が定義されている。学習の目的に従ってシラバスのそれぞれの課題を試験する。

レベル 1：記憶レベル（K1）

用語または概念を認識し、記憶して、想起することができる。

キーワード：識別（identify）、記憶（remember）、検索（retrieve）、想起（recall）、認識（recognize）、知識（know）

例：

「故障」の定義を以下のように認識できる。

- 「エンドユーザーまたは他の関係者にサービスを引き渡しできないこと」、または
- 「コンポーネントやシステムが、期待した機能、サービス、結果から逸脱すること」

レベル 2：理解レベル（K2）

課題に関連する記述について理由または説明を選択することができ、テスト概念に関して要約、比較、分類、類別、例の提示を行うことができる。

キーワード：要約（summarize）、一般化（generalize）、抽象（abstract）、分類（classify）、比較（compare）、配置（map）、対照（contrast）、例示（exemplify）、解釈（interpret）、変換（translate）、表現（represent）、推察（infer）、結論（conclude）、類別（categorize）、構造モデル（construct models）

例：

できるだけ早期にテストの分析と設計を行わなければならない理由を説明することができる。

- 早期に欠陥を検出すれば除去コストが低くなるため。
- 重要な欠陥をより早く見つけるため。

統合テストとシステムテストの類似点と相違点を説明することができる。

- 類似点：複数のコンポーネントをテストし、非機能面をテストする。
- 相違点：統合テストではインターフェースと相互作用に着目し、システムテストでは、システム全体（例えばエンドツーエンド処理）に着目する。

レベル 3：適用レベル（K3）

概念または技法を正しく選択することができ、それを特定の事例に適用することができる。

キーワード：実装（implement）、実行（execute）、使用（use）、手順の実施（follow a procedure）、手順の適用（apply a procedure）

例：

- 有効/無効に分ける境界値を見分けることができる。
- 特定の状態遷移図からすべての遷移をカバーするテストケースを選択できる。

参考資料（学習の目的の認知レベル用）

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: Boston MA

10 付録 C - リリースノート

ISTQB Foundationシラバス2018は、リリース2011を書き直したメジャーアップデートである。このため、章や節ごとの詳細なリリースノートはないが、主要な変更の要約を以下に示す。また、ISTQBは別のリリースノートドキュメントで、Foundation Levelシラバスの学習の目的について、2011年度版から2018年度版への追加、更新、削除に関する情報を提供している。

2017年初頭までに、100を超える国で55万人を超える人がFoundation Levelの試験を受験し、世界で50万人を超えるテスト技術者が認定を受けている。これらの受験者のすべてが試験に合格するためにFoundation Levelシラバスを読んでいることが想定されるため、本シラバスは最も多く読まれたソフトウェアテストのドキュメントである。

このメジャーアップデートでは、これまでの財産を受け継ぎながら、世界中のテストコミュニティの次世代の50万人にISTQBが優れた価値を提供できるよう改善が行われている。

本バージョンでは、すべての学習の目的はそれだけで1つのまとまりとなるように編集されており、各学習の目的からそれらに関連する内容（および試験の質問）へ、およびその逆方向に明確なトレーサビリティを確立している。さらに、各章の学習時間は2011年度版に比べて現実的なものになっている。学習時間は、他のISTQBシラバスで使用されている実績のある経験則と計算を使用して、各章でカバーする学習の目的を分析することによって割り当てている。

本書はFoundation Levelシラバスであり、長年にわたり使用されている最良の実践事例と技法を説明しているが、特にソフトウェア開発方法（例えば、スクラムや継続的デプロイメント）や技術（例えば、IoT）の観点で時代に合わせて、内容を新たにしている。参照している標準についても、以下のようにより新しいものに更新している。

1. ISO/IEC/IEEE 29119 で IEEE Standard 829 を置き換えた。
2. ISO/IEC 25010 で ISO 9126 を置き換えた。
3. ISO/IEC 20246 で IEEE 1028 を置き換えた。

また、この10年でISTQBのポートフォリオは大きく成長したため、他のISTQBシラバスの関連する資料に大幅な相互参照を追加した。これに関連して、すべてのシラバスとISTQB用語集の間で整合性が取れるように入念にレビューした。ゴールは、本バージョンを、読みやすく、理解しやすく、学習しやすく、翻訳しやすくすることであり、実際の有用性向上や、知識とスキルのバランスに重点を置いた。

本リリースの変更点の詳細については、『ISTQB Foundation Levelシラバス 日本語版 概要』を参照されたい。