テスト技術者資格制度

Foundation Level シラバス

Version 2023V4.0.J01

International Software Testing Qualifications Board

著作権

改訂履歴

ISTQB®

バージョン	日付	摘要
CTFL4.0	2023 年 4 月 12 日	正式公開版
ISTQB® 2018V3.1	2019 年 11 月 11 日	『テスト技術者資格制度 Foundation シラバス Version 2018 V3.1-リリースノート』を参照
ISTQB® 2018	2018 年 6 月 4 日	正式公開版
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®

Version2023V4.0.J01	2023 年 9 月 25 日	CTFL4.0 の日本語翻訳版
Version2018V3.1.J03	2021 年 5 月 12 日	日本語の誤訳を修正 1.1.1　テストの目的の箇条書きの最初の文 2.2.4　運用受け入れテストの最初の段落
Version2018V3.1.J02	2020 年 6 月 28 日	日本語訳を修正 automated test script→自動化テストスクリプト high level test case→ハイレベルテストケース low level test case→ローレベルテストケース
Version2018V3.1.J01	2020 年 6 月 25 日	ISTQB® 2018v3.1 の日本語翻訳版
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
Version2007.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、2023 年 4 月 21 日に開催された ISTQB®の総会で正式に発行された。

これは、ISTQB joint Foundation Level & Agile Working Groups が作成した。Laura Albert, Renzo Cerquozzi (vice chair), Wim Decoutere, Klaudia Dussa-Zieger, Chintaka Indikadahena, Arnika Hryszko, Martin Klonk, Kenji Onishi, Michaël Pilaeten (co-chair), Meile Posthuma, Gandhinee Rajkomar, Stuart Reid, Eric Riou du Cosquer (co-chair), Jean-François Riverin, Adam Roman, Lucjan Stapp, Stephanie Ulrich (vice chair), Eshraka Zakaria.

テクニカルレビューを行った Stuart Reid,Patricia McQuaid,Leanne Howard、提案や意見をくれたレビューチームと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。Adam Roman, Adam Scierski, Ágota Horváth, Ainsley Rood, Ale Rebon Portillo, Alessandro Collino, Alexander Alexandrov, Amanda Logue, Ana Ochoa, André Baumann, André Verschelling, Andreas Spillner, Anna Miazek, Arnd Pehl, Arne Becher, Attila Gyúri, Attila Kovács, Beata Karpinska, Benjamin Timmermans, Blair Mo, Carsten Weise, Chinthaka Indikadahena, Chris Van Bael, Ciaran O'Leary, Claude Zhang, Cristina Sobrero, Dandan Zheng, Dani Almog, Daniel Säther, Daniel van der Zwan, Danilo Magli, Darvay Tamás Béla, Dawn Haynes, Dena Pauletti, Dénes Medzihradszky, Doris Dötzer, Dot Graham, Edward Weller, Erhardt Wunderlich, Eric Riou Du Cosquer, Florian Fieber, Fran O'Hara, François Vaillancourt, Frans Dijkman, Gabriele Haller, Gary Mogyorodi, Georg Sehl, Géza Bujdosó, Giancarlo Tomasig, Giorgio Pisani, Gustavo Márquez Sosa, Helmut Pichler, Hongbao Zhai, Horst Pohlmann, Ignacio Trejos, Ilia Kulakov, Ine Lutterman, Ingvar Nordström, Iosif Itkin, Jamie Mitchell, Jan Giesen, Jean-Francois Riverin, Joanna Kazun, Joanne Tremblay, Joëlle Genois, Johan Klintin, John Kurowski, Jörn Münzel, Judy McKay, Jürgen Beniermann, Karol Frühauf, Katalin Balla, Kevin Kooh, Klaudia Dussa-Zieger, Klaus Erlenbach, Klaus Olsen, Krisztián Miskó, Laura Albert, Liang Ren, Lijuan Wang, Lloyd Roden, Lucjan Stapp, Mahmoud Khalaili, Marek Majernik, Maria Clara Choucair, Mark Rutz, Markus Niehammer, Martin Klonk, Márton Siska, Matthew Gregg, Matthias Hamburg, Mattijs Kemmink, Maud Schlich, May Abu-Sbeit, Meile Posthuma, Mette Bruhn-Pedersen, Michal Tal, Michel Boies, Mike Smith, Miroslav Renda, Mohsen Ekssir, Monika Stocklein Olsen, Murian Song, Nicola De Rosa, Nikita Kalyani, Nishan Portoyan, Nitzan Goldenberg, Ole Chr. Hansen, Patricia McQuaid, Patricia Osorio, Paul Weymouth, Pawel Kwasik, Peter Zimmerer, Petr Neugebauer, Piet de Roo, Radoslaw Smilgin, Ralf Bongard, Ralf Reissing, Randall Rice, Rik Marselis, Rogier Ammerlaan, Sabine Gschwandtner, Sabine Uhde, Salinda Wickramasinghe, Salvatore Reale, Sammy Kolluru, Samuel Ouko, Stephanie Ulrich, Stuart Reid, Surabhi Bellani, Szilard Szell, Tamás Gergely, Tamás Horváth, Tatiana Sergeeva, Tauhida Parveen, Thaer Mustafa, Thomas Eisbrenner, Thomas Harms, Thomas Heller, Tomas Rosenqvist, Werner Lieblang, Yaron Tsubery, Zhenlei Zuo and Zsolt Hargitai.

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）。

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。

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、ソフトウェア開発担当者と開発チームなどが含まれる。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。

0.3 テスト担当者のキャリアパス

ISTQB®のスキームは、キャリアのあらゆる段階にあるテストのプロフェッショナルを支援し、知識の幅と深さの両方を提供する。ISTQB® Foundation 認定資格取得者は、Core Advanced Levels（テストアナリスト、テクニカルテストアナリスト、テストマネージャー）その後、Expert Levels（テストマネジメントまたはテストプロセス改善担当）へと興味を広げられる。アジャイル環境にてテストを実践するスキルの向上を目指す場合は、Agile Technical Tester または Agile Test Leadership at Scale の認定を検討することができる。Specialist コースでは、特定のテストアプローチやテスト活動（テスト自動化、AI テスト、モデルベースドテスト、モバイルアプリテストなど）、特定のテスト分野（性能テスト、使用性テスト、受け入れテスト、セキュリティテストなど）、特定の業界ドメイン（自動車やゲームなど）のテストのノウハウを集約した分野を深く掘り下げることができる。ISTQB 認定テスト担当者制度に関する最新情報は、www.istqb.org を参照のこと。

0.4 ビジネス成果

Foundation Level の認定取得者に求められる 14 のビジネス成果を掲載している。Foundation Level 認定テスト担当者は、次のビジネス成果を達成できる。

FL-BO1 テストとは何か、なぜテストが有効なのかを理解する。

FL-BO2 ソフトウェアテストの基本的な概念を理解する。

FL-BO3 テストのコンテキストに応じて、実施すべきテストアプローチと活動を識別する。

FL-BO4 ドキュメントの品質を評価し、改善する。

FL-BO5 テストの有効性と効率性を向上する。

FL-BO6 テストプロセスとソフトウェア開発ライフサイクルを一致させる。

FL-BO7 テストマネジメントの原則を理解する。

FL-BO8 明確で理解しやすい欠陥レポートを記述して伝える。

FL-BO9 テストに関わる優先度や労力に影響を与える要因を理解する。

FL-BO10 クロスファンクショナルチームの一員として働く。

FL-BO11 テスト自動化に関するリスクと利点を知る。

FL-BO12 テストを行うために求められる必要不可欠なスキルを識別する。

FL-BO13 リスクがテストに与える影響を理解する。

FL-BO14 テスト進捗と品質を効果的にレポートする。

0.5 試験対象の学習の目的と知識レベル

学習の目的はビジネス成果を支援し、テスト技術者資格制度 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 1 章から 6 章のすべての内容は、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A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。

0.6 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に基づく資料の使用が要件となる場合がある。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（第 7 章）、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。「Foundation Level Examination Structures and Rules」を参照のこと。

0.7 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。本シラバスの認定ガイドラインは、プロセスマネジメント・コンプライアンスワーキンググループが発行する一般的な認定ガイドラインに準ずる。

0.8 標準の取り扱い

Foundation シラバスには、参照する標準がある（IEEE や ISO 標準など）。これらの参考文献は、（品質特性に関する ISO 25010 の参考文献のように）枠組みを提供し、読者が望む場合には追加情報のソースを提供するためのものである。標準ドキュメントは試験を目的としたものではない。標準の詳細については、第 7 章を参照のこと。

0.9 最新情報の維持

ソフトウェア業界は急速に変化している。これらの変化に対応し、関係者が関連する最新の情報にアクセスできるように、ISTQB ワーキンググループは、www.istqb.org ウェブサイトにリンクを設け、サポートドキュメントや標準の変更点などを紹介している。これらの情報は、Foundation のシラバスでは試験対象外である。

0.10 詳細レベル

本シラバスの詳細レベルは国際的に一貫した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内容で構成されている。

- Foundation Level の意図について説明する全般的な指導の目的
- 想起することができなければならない用語（キーワード）のリスト
- 達成すべき認知的な学習の成果について説明している、知識領域ごとの学習の目的
- 認められた情報源の参照を含む主要なコンセプトの説明

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。本シラバスは採用したSDLCに関係なく、すべてのソフトウェアプロジェクトに適用できるテストの概念と技法に焦点を置いて説明している。

0.11 本シラバスの構成

6つの章で構成しており、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1135分（18時間55分）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 180 分
  - テストに関わる基本的な原理、テストが必要となる理由、テスト目的について学ぶ。
  - テストプロセス、主なテスト活動、テストウェアを理解する。
  - テストに必要不可欠なスキルを理解する。
- 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 130 分
  - さまざまな開発アプローチへテストをどのように組み込むかを学ぶ。
  - テストファーストアプローチの考え方や、DevOps の考え方を学ぶ。
  - テストレベルの違い、テストタイプの違い、保守テストについて学ぶ。
- 第 3 章：静的テスト-学習時間 80 分
  - 静的テストの基本、フィードバックやレビュープロセスについて学ぶ。
- 第 4 章：テスト分析と設計-学習時間 390 分
  - ブラックボックス、ホワイトボックス、経験ベースのテスト技法を適用し、さまざまなソフトウェア作業成果物からテストケースを導き出す方法を学ぶ。
  - コラボレーションベースのテストアプローチについて学ぶ。
- 第 5 章：テスト活動のマネジメント-学習時間 335 分
  - 一般的なテストの計画方法、テスト工数の見積り方法を学ぶ。
  - リスクがテスト範囲にどのような影響を与えるかを学ぶ。
  - テスト活動をモニタリングし、コントロールする方法を学ぶ。
  - 構成管理がどのようにテストを支援するかを学ぶ。
  - 欠陥を明確に理解しやすく報告する方法を学ぶ。
- 第 6 章：テストツール-学習時間 20 分
  - ツールの分類を学び、テスト自動化のリスクと利点を理解する。

1 テストの基礎 — 180 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、テスト実装、テストモニタリング、テスト対象、テスト目的、テスト計画、テストプロシジャー、テスト結果、テスト、テストウェア、妥当性確認、検証

第 1 章の学習の目的

1.1 テストとは何か？

FL-1.1.1 （K1）典型的なテスト目的を識別する。

FL-1.1.2 （K2）デバッグとテストを区別する。

1.2 なぜテストが必要か？

FL-1.2.1 （K2）テストが必要な理由を例示する。

FL-1.2.2 （K1）テストと品質保証の関係を想起する。

FL-1.2.3 （K2）根本原因、エラー、欠陥、故障を区別する。

1.3 テストの原則

FL-1.3.1 （K2）テストにおける 7 原則を説明する。

1.4 テスト活動、テストウェア、そして役割

FL-1.4.1 （K2）さまざまなテスト活動とタスクを要約する。

FL-1.4.2 （K2）テストを取り巻くコンテキストがテストプロセスに与える影響を説明する。

FL-1.4.3 （K2）テスト活動を支援する多くのテストウェアを区別する。

FL-1.4.4 （K2）トレーサビリティを維持することの価値を説明する。

FL-1.4.5 （K2）テストにおけるさまざまな役割を比較する。

1.5 テストに必要不可欠なスキルとよい実践例

FL-1.5.1 （K2）テストを行うために求められる一般的なスキルについて例を挙げる。

FL-1.5.2 （K1）チーム全体アプローチの利点を想起する。

FL-1.5.3 （K2）テストの独立性の長所と短所を区別する。

1.1 テストとは何か？

ソフトウェアシステムは、日々の生活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、極端な場合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手助けとなる。

ソフトウェアテストは、欠陥を発見し、ソフトウェアアーティファクトの品質を評価するための一連の活動である。これらアーティファクトは、テストをする際のテスト対象である。テストに関するよくある誤解の 1 つは、テストはテスト実行（すなわち、ソフトウェアを実行しテスト結果を確認する）だけだというものである。しかし、ソフトウェアテストには他の活動も含まれる。そして、ソフトウェア開発ライフサイクルと整合させなければばらない（第 2 章参照）。

テストに関するもう 1 つのよくある誤解は、テストはテスト対象の検証だけに重点を置くというものである。テストでは検証、つまり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。

テストは動的な場合、または静的な場合がある。動的テストはソフトウェアの実行を伴うが、静的テストはソフトウェアの実行を伴わない。静的テストはレビュー（第 3 章参照）と静的解析を含む。動的テストでは、さまざまな種類のテスト技法やテストアプローチを用いてテストケースを導出する（第 4 章参照）。

テストは、技術的な活動だけではない。適切に計画、マネジメント、見積り、モニタリング、コントロールすることもまた必要となる（第 5 章参照）。

テスト担当者はツールを使用するが（第 6 章参照）、テストは大部分が知的活動であり、テスト担当者は専門知識を持ち、分析スキルを使い、批判的思考やシステム思考を適用することが求められることを忘れてはならない（Myers 2011, Roman 2018）。

標準である ISO/IEC/IEEE 29119-1 では、ソフトウェアテストの概念についてさらに詳しい情報を提供している。

1.1.1 テスト目的

典型的なテスト目的は以下の通り。

- 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。
- 故障を引き起こし、欠陥を発見する。
- 求められるテスト対象のカバレッジを確保する。
- ソフトウェア品質が不十分な場合のリスクレベルを下げる。
- 仕様化した要件が満たされているかどうかを検証する。
- テスト対象が契約、法律、規制の要件に適合していることを検証する。
- ステークホルダーに根拠ある判断をしてもらうための情報を提供する。
- テスト対象の品質に対する信頼を積み上げる。
- テスト対象が完成し、ステークホルダーの期待通りに動作するかどうかの妥当性確認をする。

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、リスク、従うソフトウェア開発ライフサイクルモデル（SDLC）、および、例えば企業構造、競争上の考慮事項、市場投入までの時間といったビジネスコンテキストに関わる要因などのコンテキストによって異なることがある。

1.1.2 テストとデバッグ

テストとデバッグは別の活動である。テストをすることで、ソフトウェアの欠陥によって引き起こされる故障を発生させたり（動的テスト）、テスト対象の欠陥を直接発見したりすることができる（静的テスト）。

動的テスト（第 4 章参照）が故障を引き起こす一方で、デバッグの関心ごとは、故障の原因（欠陥）を発見し、その原因を解析し、原因を取り除くことにある。典型的なデバッグの流れは以下の通り：

- 故障の再現
- 診断（根本原因を発見すること）
- 原因を修正すること

その後に行う確認テストは、修正によって問題が解決されたかを確認する。確認テストは、最初のテストを行った人と同じ人が行うのがよい。修正によってテスト対象の他の部分に故障が発生していないかを確認するために、続けてリグレッションテストを実施することもできる（確認テストとリグレッションテストの詳細については、2.2.3 項を参照）。

静的テストで欠陥を識別した場合、デバッグすることはその欠陥を取り除くことである。静的テストは欠陥を直接発見するものであり、故障を引き起こすことはできないため、再現や診断の必要はない（第 3 章参照）。

1.2 なぜテストが必要か？

テストをすることは品質コントロールの形式の 1 つであり、設定された範囲、時間、品質、予算の制約の中で、合意されたゴールを達成するために役立つ。成功に向けたテストによる貢献は、テストチームの活動に限定されるべきではない。ステークホルダーであれば、誰でもプロジェクトを成功に近づけるためにテストスキルを活かすことができる。コンポーネント、システム、および関連するドキュメントをテストすることは、ソフトウェアの欠陥を識別するのに役立つ。

1.2.1 成功に対するテストの貢献

テストをすることは、欠陥を検出するためのコスト効果のある手段を提供することになる。この欠陥は、（テスト活動ではないデバッグによって）取り除くことができるため、テストをすることが間接的にテスト対象の品質向上に貢献することになる。

テストをすることは、SDLC のさまざまな段階において、品質を直接評価する手段を提供する。これらの測定結果は、より大きなプロジェクトマネジメント活動の一部として使用し、リリースの判定など、SDLC の次のステージに移行するための判定に貢献する。

テストをすることは、開発プロジェクトにおいて間接的にユーザーが利用した場合の状況を提供することになる。テスト担当者は、開発ライフサイクルを通じて、テスト担当者自身が理解するユーザーのニーズが考慮できていることを確認する。別の方法としては、代表的な複数のユーザーに開発プロジェク

トへ関与してもらう方法があるが、コストが高く、適切なユーザーを確保できないため、通常不可能である。

また、テストは、契約上または法律上の要件を満たすため、あるいは規制基準に準拠するために必要となる場合がある。

1.2.2 テストと品質保証（QA）

「テスト」と「品質保証」（QA）という用語を同じように使う人が多くいる。しかし、テストと品質保証は同じではない。テストは品質コントロール（QC）の形式の 1 つである。

QC は、適切な品質の達成を支援する活動に焦点を当てた、プロダクト指向の是正アプローチである。テストは品質コントロールの主要な形式であり、その他に形式的手法（モデル検査や定理証明）、シミュレーション、プロトタイピングなどがある。

QA は、プロセスの実装と改善に焦点を当てた、プロセス指向の予防的アプローチである。よいプロセスが正しく行われれば、よいプロダクトを作ることができるという考えに基づいている。QA は、開発プロセスとテストプロセスの両方に適用し、プロジェクトに参加するすべての人が責任を持つ。

テスト結果は、QA と QC で使用する。QC では欠陥の修正に使い、QA では開発とテストプロセスがどの程度うまくいっているかについてのフィードバックに使う。

1.2.3 エラー、欠陥、故障、および根本原因

人間はエラー（誤り）を起こす。そのエラーが、欠陥（フォールト、バグ）を生み出し、その欠陥は、故障につながることもある。人間は、時間的なプレッシャー、作業成果物、プロセス、インフラ、相互作用の複雑度、あるいは単に疲れていたり、十分なトレーニングを受けていなかったりなど、さまざまな理由でエラーを起こす。

欠陥は、要件仕様書やテストスクリプトのようなドキュメント、ソースコード、またはビルドファイルのような補助的なアーティファクトの中から発見できる。SDLC の初期に作成されたアーティファクトの欠陥は、もし検出されなければ、しばしばライフサイクルの後半で欠陥のあるアーティファクトにつながる。コードの欠陥が実行されると、システムがすべきことをしない、またはすべきでないことをしてしまうことがあり、故障の原因となる。欠陥の中には、実行すれば必ず故障になるものもあれば、特定の状況下でしか故障にならないもの、絶対に故障にならないものもある。

故障の原因は、エラーや欠陥だけではない。例えば、放射線や電磁波によってファームウェアに欠陥が生じる場合など、環境条件によっても故障が発生することがある。

根本原因とは、問題（例えば、エラーにつながる状況）が発生する根底の理由のことである。根本原因は、根本原因分析によって特定される。根本原因分析は、故障が発生したときや欠陥が確認されたときに行われるのが一般的である。根本原因を取り除くなどの対処をすることで、さらに同様の故障や欠陥を防止したり、その頻度を減らしたりできると考えられている。

1.3 テストの原則

これまで長い間にわたり、さまざまなテストの原則が提唱され、あらゆるテストで適用できる一般的なガイドラインとなってきた。本シラバスでは、そのような 7 つの原則を示す。

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、テスト対象に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（Buxton 1970）。テストにより、テスト対象に

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、テスト対象の正しさを証明できない。

2.全数テストは不可能　すべてをテストすることは、ごく単純なソフトウェア以外では非現実的である（Manna 1978）。全数テストの代わりに、テスト技法（第 4 章参照）、テストケースの優先順位付け（5.1.5 項参照）、リスクベースドテスト（5.2 節参照）を用いて、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　プロセスの早い段階で欠陥を取り除くと、その後の作業成果物では取り除かれた欠陥に起因する欠陥を引き起こすことはない。SDLC の後半に発生する故障が少なくなるため、品質コストは削減される（Boehm 1981）。早い段階で欠陥を見つけるために、静的テスト（第 3 章参照）と動的テスト（第 4 章参照）の両方をなるべく早い時期に開始すべきである。

4.欠陥の偏在　通常、見つかる欠陥や運用時の故障の大部分は、少数のシステムコンポーネントに集中する（Enders 1975）。この現象は、パレートの法則を示している。予測した欠陥の偏在、および実際にテスト中または運用中に観察した欠陥の偏在は、リスクベースドテストの重要なインプットとなる（5.2 節参照）。

5.テストの弱化　同じテストを何回も繰り返すと、新たな欠陥の検出に対する効果は薄れてくる（Beizer 1990）。この影響を克服するため、テストとテストデータを変更したり新規にテストを作成したりすることが必要になる場合がある。しかし、例えば、自動化されたリグレッションテストのように、同じテストを繰り返すことが有益な結果を示すことができる場合がある（2.2.3 項参照）。

6.テストはコンテキスト次第　テストに唯一普遍的に適用できるアプローチは存在しない。テストは、コンテキストによって異なる方法で行われる（Kaner 2011）。

7.「欠陥ゼロ」の落とし穴　ソフトウェアを検証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は誤り（つまり、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、ユーザーのニーズや期待を満たさないシステム、顧客のビジネスゴールの達成に役立たない、および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。検証に加えて、妥当性確認も実施すべきである（Boehm 1981）。

1.4 テスト活動、テストウェア、そして役割

テストはコンテキスト次第である。しかし、ハイレベルでは、テスト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を構成する。テストプロセスは、多くの要因を考慮して特定の状況に対する適切な調整ができる。テストプロセスの中で、どのテスト活動を、どのように、いつ実施するかは、通常、特定の状況に対するテスト計画の一部として決定する（5.1 節参照）。

以降の節では、テストプロセスの一般的な側面を、テスト活動やタスク、コンテキストの影響、テストウェア、テストベースとテストウェア間のトレーサビリティ、テストの役割の観点から説明する。

ISO/IEC/IEEE 29119-2 標準には、テストプロセスのより詳細な情報が記載されている。

1.4.1 テスト活動とタスク

テストプロセスは多くの場合、以下に説明する主な活動のグループから構成される。これらの活動の多くは論理的にはシーケンシャルに行われるように見えるが、イテレーティブにまたは並行して実施されることが多い。これらのテスト活動は、通常、システムやプロジェクトに合わせて調整する必要がある。

テスト計画は、テスト目的を定義することと、その後に全体のコンテキストにより課せられた制約下において目的を最も効果的に達成するアプローチを選択することから構成される。テスト計画については、5.1 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、すべてのテスト活動を継続的にチェックし、実際の進捗をテスト計画と比較することを含む。テストコントロールは、テストの目的を達成するために必要な行動をとることを含む。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。

テスト分析は、テストベースを分析して、テスト可能なフィーチャーを識別し、関連するテスト条件を定義して優先順位を付けるとともに、関連するリスクとリスクレベルを分析することを含む（5.2 節を参照）。テストベースとテスト対象を評価し、それらに含まれている可能性のある欠陥を識別したり、試験性のアセスメントをしたりする。テスト分析では、多くの場合、この活動を支援するためにテスト技法を使用する(第 4 章を参照)。テスト分析では、計測可能なカバレッジ基準から見て「何をテストするか？」という問いに答えている。

テスト設計は、テスト条件をテストケースやその他のテストウェア（テストチャーターなど）に落とし込む作業を含む。この活動には、多くの場合、テストケースの入力を指定するためのガイドとして機能するカバレッジアイテムの識別が含まれる。テスト設計では、この活動を支援するために、テスト技法（第 4 章を参照）が使用できる。テスト設計には、テストデータ要件の定義、テスト環境の設計、およびその他の必要なインフラストラクチャとツールの識別も含まれる。テスト設計では、「どのようにテストするか?」という問いに答えている。

テスト実装は、テスト実行に必要なテストウェア（例えば、テストデータ）の作成または取得を含む。テストケースはテストプロシジャーに編成でき、多くの場合、テストスイートにまとめる。手動および自動のテストスクリプトを作成する。テストプロシジャーは、効率的なテスト実行のために、テスト実行スケジュール内で優先順位を付けて配置する（5.1.5 項を参照）。テスト環境を構築し、正しく設定されていることを検証する。

テスト実行は、テスト実行スケジュールに従ってテストを走らせること（テストラン）を含む。テスト実行は、手動でも自動でもかまわない。テスト実行には、継続的テストやペアテストセッションなど、さまざまな形式がある。実際のテスト結果は、期待結果と比較する。テスト結果を記録する。不正を分析して、考えられる原因を識別する。この分析により、観察された故障に基づいて不正を報告することができる（5.5 節を参照）。

テスト完了の活動は、通常、プロジェクトのマイルストーン（リリース、イテレーションの終了、テストレベルの完了など）で、未解決の欠陥、変更要求、または作成したプロダクトバックログアイテムに対して行う。将来役立つ可能性のあるテストウェアをすべて識別し、保管するか、適切なチームへ引き渡す。テスト環境は合意した状態でシャットダウンする。将来のイテレーション、リリース、またはプロジェクトに向けて、教訓と改善点を識別するために、テスト活動を分析する（2.1.6 項を参照）。テスト完了レポートを作成して、ステークホルダーへ伝える。

1.4.2 コンテキストに応じたテストプロセス

テストは、単独で行うことはない。テスト活動は、組織内で行われる開発プロセスに不可欠な要素である。また、テストはステークホルダーによって資金提供され、その最終ゴールは、ステークホルダーのビジネスニーズの充足を支援することである。したがって、テストを行う方法は、以下のようないくつかのコンテキストに応じた要因に依存する：

- ステークホルダー（ニーズ、期待、要件、協力の意思など）。

- チームメンバー（スキル、知識、経験レベル、空き状況、トレーニングの必要性など）。
- ビジネスドメイン（テスト対象の重要性、識別したリスク、市場ニーズ、特定の法的規制など）。
- 技術的要因（ソフトウェアの種類、プロダクトのアーキテクチャー、利用技術など）。
- プロジェクトの制約（スコープ、時間、予算、リソースなど）。
- 組織的要因（組織構造、現行のポリシー、使用する実践例など）。
- ソフトウェア開発ライフサイクル（エンジニアリングの実践例、開発手法など）。
- ツール（利用可能な状況、使用性、標準適合性など）。

これらの要因は、利用するテスト戦略、テスト技法、テスト自動化の度合い、求められるカバレッジの度合い、テストドキュメントの詳細度合い、レポート作業などを含む、テストに関する多くの事柄に影響を与える。

1.4.3 テストウェア

テストウェアは、1.4.1 項で説明したテスト活動にて出力する作業成果物として作成する。組織によって、作業成果物の作成方法、形式、名称、整理方法や管理方法には大きな違いがある。適切な構成管理（5.4 節参照）により、作業成果物の一貫性と整合性を確保する。以下の作業成果物のリストは、すべてを網羅するものではない：

- **テスト計画の作業成果物**には、テスト計画書、テストスケジュール、リスクレジスター、開始基準と終了基準が含まれる。（5.1 節参照）。リスクレジスターは、リスクの可能性、リスクの影響、リスク軽減に関する情報とともに、リスクを列挙したものである（5.2 節を参照）。テストスケジュール、リスクレジスター、開始基準、終了基準は、多くの場合、テスト計画書の一部である。
- **テストのモニタリングとコントロールの作業成果物**には、テスト進捗レポート(5.3.2 項参照)、コントロールのための指示の文書化（5.3 節参照）、リスク情報（5.2 節参照）が含まれる。
- **テスト分析の作業成果物**には、（優先順位を付けた）テスト条件（例えば、受け入れ基準など、4.5.2 項参照）、およびテストベース内の欠陥についての（直接修正しない場合の）欠陥レポートが含まれる。
- **テスト設計の作業成果物**には、（優先順位を付けた）テストケース、テストチャーター、カバレッジアイテム、テストデータ要件とテスト環境要件が含まれる。
- **テスト実装の作業成果物**には、テストプロシジャー、自動テストスクリプト、テストスイート、テストデータ、テスト実行スケジュール、およびテスト環境が含まれる。テスト環境に含むものの例としては、スタブ、ドライバー、シミュレーター、そしてサービス仮想化がある。
- **テスト実行の作業成果物**には、テスト結果記録および欠陥レポートがある（5.5 節参照）。
- **テスト完了の作業成果物**には、テスト完了レポート（5.3.2 項参照）、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、得られた教訓をドキュメントにしたもの、変更要求（例えば、プロダクトバックログアイテムとして）が含まれる。

1.4.4 テストベースとテストウェアとの間のトレーサビリティ

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、テストベースの各要素と関連するテストウェア（例えば、テスト条件、リスク、テストケース）、テスト結果、検出した欠陥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。

正確なトレーサビリティはカバレッジの評価を支援する。なので、測定可能なカバレッジ基準がテストベースに定義してある場合、非常に有用となる。カバレッジ基準は、テスト目的がどの程度達成しているか示す活動を遂行するための KPI として機能する（1.1.1 項参照）。例えば、以下のようなものがある。

- テストケースと要件のトレーサビリティにより、要件がテストケースでカバーされていることを検証することができる。
- テスト結果とリスクのトレーサビリティは、テスト対象にある残存リスクのレベルを評価するために利用できる。

カバレッジの評価に加えて、優れたトレーサビリティは、変更の影響を判断することができ、テストの監査を容易にし、IT ガバナンスの基準を満たすのに役立つ。また、テストベースの各要素の状況を含めることで、テスト進捗や完了レポートをよりわかりやすくする。テストの技術的な側面をステークホルダーにわかりやすく伝えることにも役立つ。トレーサビリティは、プロダクト品質、プロセス能力、およびビジネスゴールに対するプロジェクトの進捗を評価するための情報を提供する。

1.4.5 テストの役割

本シラバスでは、テストにおける 2 つの主要な役割である、テストをする役割とテストマネジメントをする役割を取り上げる。この 2 つの役割に割り当てられる活動やタスクは、プロジェクトやプロダクトのコンテキスト、各役割を担う人々のスキル、組織などの要因に依存する。

テストマネジメントをする役割は、テストプロセス、テストチーム、テスト活動のリーダーシップに対する全体的な責任を負う。テストマネジメントをする役割は、主にテスト計画、テストモニタリングとコントロール、テスト完了の活動に重点を置いている。テストマネジメントをする役割の実施方法は、状況によって異なる。例えば、アジャイルソフトウェア開発では、テストマネジメントのタスクの一部をアジャイルチームが担当することがある。複数のチームや組織全体にまたがるタスクは、開発チーム以外のテストマネージャーが行うことがある。

テストをする役割は、テストのエンジニアリング（技術的）側面に対する全体的な責任を負う。テストをする役割は、主にテスト分析、テスト設計、テスト実装、テスト実行の活動に重点を置いている。

さまざまな人が、さまざまなタイミングでこれらの役割を担う場合がある。例えば、テストマネジメントをする役割は、チームリーダーが行うこともあれば、テストマネージャーが行うこともあり、開発マネージャーが行うこともある。また、一人がテストをする役割とテストマネジメントをする役割を同時に担うことも可能である。

1.5 テストに必要不可欠なスキルとよい実践例

スキルとは、知識、実践、および適性からくる、何かをうまく行う能力のことである。よいテスト担当者は、以下のような自分の仕事をうまくこなすための重要なスキルを持っているべきである。優れたテスト担当者は効果的なチームプレーヤーであるべきであり、異なるテスト独立性のレベルで、テストを実行できるべきである。

1.5.1 テストに必要な汎用的スキル

汎用的ではあるが、テスト担当者に特に関連するスキルとして以下のようなものがある：

- テスト知識（テスト技法の活用などによりテストの有効性を高めるため）

- 徹底さ、慎重さ、好奇心、細部へのこだわり、理路整然さ（特に発見しにくい欠陥を識別するため）
- 優れたコミュニケーションスキル、積極的な傾聴、チームプレーヤーとなる（すべてのステークホルダーと効果的にやりとりするため、他者に情報を伝えるため、理解してもらうため、欠陥を報告し議論するため）
- 分析的思考、批判的思考、創造性（テストの有効性を高めるため）
- 技術的な知識（適切なテストツールの使用によりテストの効率性を高めるため）
- ドメイン知識（エンドユーザー/ビジネス側代表者を理解し、コミュニケーションできるようになるため）

テスト担当者は、悪い知らせの運び手となることが多い。悪い知らせの運び手を責めるのは、人間の一般的な特性である。そのため、テスト担当者にとってコミュニケーションスキルは非常に重要となる。テスト結果を伝えると、プロダクトやその作成者に対する批判と受け取られる可能性がある。確証バイアスは、現在持っている信念と異なる情報を受け入れることを困難にすることがある。テストはプロジェクトの成功やプロダクトの品質に大きく貢献しているにもかかわらず、破壊的な活動であると認識している人もいるかもしれない。このような見方を改善するために、欠陥や故障に関する情報は建設的な方法で伝えるべきである。

1.5.2 チーム全体アプローチ

テスト担当者にとって重要なスキルの 1 つは、チームの中で効果的に働き、チームのゴールに積極的に貢献する能力である。エクストリームプログラミング（2.1 節参照）に由来するチーム全体アプローチは、このスキルの上に成り立っている。

チーム全体アプローチでは、必要な知識とスキルを持つチームメンバーであれば、どの仕事を行ってもよく、全員が品質に対して責任を持つ。同じ場所で働くことは情報伝達、および相互作用を活性化するため、チームメンバーは同じワークスペース（物理的または仮想的）を共有する。チーム全体アプローチは、チームダイナミクスを向上させ、チーム内の情報伝達とコラボレーションを強化し、チーム内のさまざまなスキルセットをプロジェクトの利益のために活用できるような相乗効果を生み出す。

テスト担当者は、望ましい品質レベルの達成を確かなものにしていくために他のチームメンバーと緊密に連携する。これには、ビジネス側の代表者と協力して適切な受け入れテストを作成することや、開発者と協働してテスト戦略に合意し、テスト自動化アプローチを決定することを含む。テスト担当者は、こうしてテストに関する知識を他のチームメンバーに広めることができ、プロダクトの開発に影響を与えることができる。

コンテキストによっては、チーム全体アプローチが常に適切とは限らない。例えば、セーフティクリティカルのような状況などでは、高いレベルのテストの独立性が必要な場合がある。

1.5.3 テストの独立性

ある程度の独立性があることで、作成者とテスト担当者の認知バイアスの違いにより、テスト担当者がより効果的に欠陥を発見できるようになる（Salman 1995 を参照）。しかし、独立性は、開発者の慣れ親しんだ作業に取って代わるものではない。例えば、開発者は自分のコードにある多くの欠陥を効率的に発見することができる。

作業成果物のテストは、作成者が行う場合（独立性なし）、作成者と同じチームの仲間が行う場合（ある程度の独立性あり）、作成者のチーム外だが組織内のテスト担当者が行う場合（独立性が高い）、組

織外のテスト担当者が行う場合（独立性がとても高い）がある。ほとんどのプロジェクトでは、通常、複数の独立性のレベル（例えば、開発者はコンポーネントテストとコンポーネント統合テストを行い、テストチームはシステムテストとシステム統合テストを行い、ビジネス側代表者は受け入れテストを行う）でテストを実施することが最善となる。

テストの独立性の主な利点は、独立したテスト担当者は、経歴、技術的視点、バイアスが異なるため、開発者と比較して異なる種類の故障や欠陥を認識する可能性が高いことである。さらに、独立したテスト担当者は、システムの仕様策定や実装の際にステークホルダーが設定した仮定を検証し、異議を唱え、あるいは反証することができる。

しかし、欠点もある。独立したテスト担当者が開発チームから孤立してしまい、協調性の欠如、情報伝達の問題、または開発チームとの敵対関係に陥る可能性がある。開発者が品質に対する責任感を失ってしまうことがある。独立したテスト担当者がボトルネックとみなされたり、リリースの遅れを責められたりする可能性がある。

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	130 分

キーワード

受け入れテスト、ブラックボックステスト、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、機能テスト、統合テスト、メンテナンステスト、非機能テスト、リグレッションテスト、シフトレフト、システム統合テスト、システムテスト、テストレベル、テスト対象、テストタイプ、ホワイトボックステスト

第 2 章の学習の目的

2.1 コンテキストに応じたソフトウェア開発ライフサイクルでのテスト

FL-2.1.1 （K2）選択したソフトウェア開発ライフサイクルがテストに与える影響を説明する。

FL-2.1.2 （K1）全ソフトウェア開発ライフサイクルに適用するよいテストの実践例を想起する。

FL-2.1.3 （K1）開発でのテストファーストアプローチの事例を想起する。

FL-2.1.4 （K2）DevOps がテストに与える影響について要約する。

FL-2.1.5 （K2）シフトレフトアプローチを説明する。

FL-2.1.6 （K2）プロセス改善の仕組みとして、ふりかえりをどのように利用できるかを説明する。

2.2 テストレベルとテストタイプ

FL-2.2.1 （K2）さまざまなテストレベルを区別する。

FL-2.2.2 （K2）さまざまなテストタイプを区別する。

FL-2.2.3 （K2）確認テストとリグレッションテストを区別する。

2.3 メンテナンス（保守）テスト

FL-2.3.1 （K2）メンテナンステストとそのきっかけを要約する。

2.1 コンテキストに応じたソフトウェア開発ライフサイクルでのテスト

ソフトウェア開発ライフサイクル（SDLC）モデルとは、ソフトウェア開発プロセスを抽象的かつハイレベルに表現したものである。SDLC モデルは、このプロセスの中で実施するさまざまな開発フェーズや活動の種類が、論理的かつ時系列的にどのように互いに関連しているかを定義する。SDLC モデルの例としては、シーケンシャル開発モデル（例えば、ウォーターフォールモデル、V 字モデル）、イテレーティブ開発モデル（例えば、スパイラルモデル、プロトタイピング）、インクリメンタル開発モデル（例えば、統一プロセス）などがある。

ソフトウェア開発プロセスの中には、より詳細なソフトウェア開発手法やアジャイルプラクティスで説明できる活動もある。例としては、受け入れテスト駆動開発（ATDD）、振る舞い駆動開発（BDD）、ドメイン駆動設計（DDD）、エクストリームプログラミング（XP）、フィーチャー駆動開発（FDD）、カンバン、リーン IT、スクラム、テスト駆動開発（TDD）がある。

2.1.1 ソフトウェア開発ライフサイクルがテストに与える影響

テストの成功のためには、SDLC への適応をしなければならない。SDLC の選択は、以下のことに影響する：

- テスト活動の範囲とタイミング（テストレベルやテストタイプなど）
- テストドキュメントの詳細レベル
- テスト技法とテストアプローチの選択
- テスト自動化の範囲
- テスト担当者の役割と責任

シーケンシャル開発モデルでは、初期段階では、テスト担当者は通常、要件レビュー、テスト分析、およびテスト設計に参加する。実行可能なコードは通常、後のフェーズで作成されるため、通常、動的テストは SDLC の初期には実施できない。

イテレーティブ開発モデルやインクリメンタル開発モデルでは、各イテレーションで作業成果物であるプロトタイプやプロダクトインクリメントを提供することが前提となっているものがある。これは、各イテレーションにおいて、静的テストと動的テストの両方が、すべてのテストレベルで実行される可能性があることを意味する。インクリメントを頻繁に提供するためには、迅速なフィードバックと広範なリグレッションテストが必要となる。

アジャイルソフトウェア開発では、プロジェクトを通じて変更が生じる可能性があることを想定している。そのため、アジャイルプロジェクトでは、作業成果物の文書化を軽量化し、リグレッションテストを容易にするためのテスト自動化を充実させることが好まれる。また、手動テストのほとんどは、事前の大規模なテスト分析やテスト設計を必要としない経験ベースのテスト技法（4.4 節参照）を使用して行われる傾向がある。

2.1.2 ソフトウェア開発ライフサイクルとよい実践例

選択した SDLC モデルとは関係なく、よいテストの実践には、以下のようなものがある。

- 各開発活動に対応してテスト活動がある。そのため、すべての開発活動が品質コントロールの対象となる。
- 異なるテストレベル（2.2.1 項参照）には、冗長性を避けつつ適切に包括したテストをすることができる、そのレベル特有の異なる目的がある。

- 各テストレベルのテスト分析や設計は、対応する SDLC の開発フェーズの間に開始する。そうすることで、テストが早期テストの原則（1.3 節参照）に従うことができる。
- テスト担当者は作業成果物のドラフトができたらすぐにそのレビューに関与する。そうすることで、この早期テストと欠陥検出がシフトレフト戦略（2.1.5 項参照）を支援できる。

2.1.3 テストが主導するソフトウェア開発

TDD、ATDD、BDD は、類似の開発アプローチであり、いずれもテストが開発を導く手段であると定義している。これらのアプローチはいずれも、コードを書く前にテストケースを定義するため、早期テストの原則（1.3 節参照）を実装し、シフトレフトアプローチ（2.1.5 項参照）を採用している。これらは、イテレーティブ開発モデルをサポートする。これらのアプローチの特徴は、以下の通りである：

テスト駆動開発（TDD）：

- （広範なソフトウェア設計の代わりに）テストケースを通じてコーディングを導く（Beck 2003）
- テストを最初に書き、次にテストを満たすようにコードを書き、そしてテストとコードをリファクタリングする

受け入れテスト駆動開発（ATDD）（4.5.3 項参照）：

- システム設計プロセスの一環として、受け入れ基準からテストを導き出す（Gärtner 2011）
- アプリケーションの該当部分がテストを満たすよう開発するために、テストを開発前に書く

振る舞い駆動開発（BDD）：

- アプリケーションの望ましい振る舞いを、ステークホルダーが理解しやすい簡潔な形の自然言語で書いたテストケース-通常は Given/When/Then を使用したフォーマットで表現する。（Chelimsky 2010）
- テストケースは、実行可能なテストへ自動的に変換する。

上記のすべてのアプローチにおいて、将来の適応やリファクタリングにてコード品質を確実にするために、テストケースを自動テストとして持続させることがある。

2.1.4 DevOps とテスト

DevOps は、開発（テストを含む）と運用が共通のゴールを達成するために協力し、相乗効果を生み出すことを目的とした組織的アプローチである。DevOps では、開発（テストを含む）と運用の間のギャップを埋め、両者の機能を同等の価値で扱うために、組織内文化の転換が求められる。DevOps は、チームの自律性、迅速なフィードバック、統合ツールチェーン、そして継続的インテグレーション（CI）と継続的デリバリー（CD）のような技術的な実践を促進する。これにより、チームは、DevOps のデリバリーパイプラインを通じて高品質のコードをより速くビルド、テスト、リリースすることができる（Kim 2016）。

テストをする観点から、DevOps の利点として、以下のようなものがある：

- コード品質や、変更が既存のコードに悪影響を及ぼすかどうかに対する迅速なフィードバック
- CI は、コンポーネントテストや静的解析による高品質なコードのサブミットを開発者に促すことで、テストにおけるシフトレフトアプローチ（2.1.5 項参照）を促進する。
- 安定したテスト環境の構築を容易にする CI/CD などの自動化プロセスを促進する。
- 非機能品質特性（性能、信頼性など）に対する観点が増える。
- デリバリーパイプラインによる自動化により、手動テストを繰り返す必要性を削減する。
- 自動化したリグレッションテストの規模と範囲から、リグレッションのリスクを最小化する。

DevOps には、以下のようなリスクや課題がないわけではない：

- DevOps のデリバリーパイプラインを定義し、確立しなければならない。
- CI/CD ツールの導入と保守をしなければならない。
- テスト自動化は、追加のリソースを必要とし、確立と保守が困難な場合がある。

DevOps にはレベルの高い自動テストが前提となるが、手動テスト、特にユーザーの視点からのテストは依然として必要となる。

2.1.5 シフトレフトアプローチ

早期テストの原則（1.3 節参照）を、シフトレフトと呼ぶことがある。なぜなら、テストが SDLC の早い段階で実行されるアプローチだからである。シフトレフトでは、通常、テストがより早い段階（例えば、コードが実装されるのを待たない、コンポーネントが統合されるのを待たない）で行われるべきだと提言しているが、SDLC の後半でのテストを軽視するべきだということを意味しているわけではない。

テストにおける「シフトレフト」を実現する方法を示すよい実践例は次の通りである：

- テストをする観点から仕様書をレビューする。このような仕様書のレビュー活動では、曖昧さ、不完全さ、矛盾など、潜在的な欠陥を発見することが多い。
- コードを書く前にテストケースを書き、コード実装時にテストハーネスでコードを実行する。
- ソースコードをコードリポジトリへサブミットする際に、付随する自動コンポーネントテスト、そして高速フィードバックが伴うようにするため、CI や CD を用いる。
- 動的テストの前に、または自動化したプロセスの一部として、ソースコードの静的解析を完了する。
- 実施可能であれば、コンポーネントテストレベルで非機能テストの実施を始める。システムが完成し、本番相当のテスト環境が利用可能になる SDLC の後半に実施される傾向がある非機能テストをこのように実施することは、シフトレフトの一形態である。

シフトレフトアプローチは、プロセスの初期には追加のトレーニング、労力、またはコストがかかるかもしれないが、プロセスの後期には労力やコストを削減することが期待される。

シフトレフトアプローチのために、ステークホルダーがこの考え方に納得し、受け入れることが重要である。

2.1.6 ふりかえりとプロセス改善

ふりかえり（「プロジェクト後ミーティング」や「プロジェクトふりかえり」とも呼ばれる）は、プロジェクトやイテレーションの終了時や、リリースのマイルストーンで開催されることが多いが、必要な時に開催されることもある。ふりかえりのタイミングと構成は、それぞれの SDLC モデルに従う。これらのミーティングでは、参加者（テスト担当者だけでなく、開発者、アーキテクト、プロダクトオーナー、ビジネスアナリストなど）が議論する：

- 何が成功したのか、何を残すべきなのか？
- 成功しなかったこと、改善できることは？
- 改善点を取り入れ、成功例を今後どう残していくか？

議論の結果は記録されるべきであり、通常、テスト完了レポートの一部となる。(5.3.2 項参照)。ふりかえりは、継続的改善を成功させるために重大であり、推奨された改善点をフォローアップすることが重要である。

テストにとっての典型的な利点としては以下がある：

- テストの有効性/効率性の向上（例えば、プロセス改善提案の実装）
- テストウェアの品質向上（例えば、テストプロセスを一緒にレビューすることによる）
- チームの絆と学び（例えば、問題提起や改善点の提案の場ができた結果）
- テストベースの品質向上（例えば、要件の品質と不備の度合いを示し、解決することによる）
- 開発とテストの連携が向上（例えば、定期的にコラボレーションを見直し、最適化することによる）

2.2 テストレベルとテストタイプ

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活動のグループである。各テストレベルは、個々のコンポーネント単位から完成したシステムまで、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段階に応じたテストプロセスのインスタンスである。

テストレベルは SDLC 内の他の活動と関連付けられる。シーケンシャルな SDLC モデルでは、テストレベルは、あるレベルの終了基準が次のレベルの開始基準の一部であるように定義されることが多い。いくつかのイテレーティブなモデルにおいては、これは適用されないかもしれない。開発活動は、複数のテストレベルにまたがることがある。テストレベルは、時間的に重なることがある。

テストタイプは、特定の品質特性に関連するテスト活動のグループであり、それらのテスト活動のほとんどは、すべてのテストレベルで実行することができる。

2.2.1 テストレベル

本シラバスでは以下の 5 つのテストレベルを説明する：

コンポーネントテスト（ユニットテストとも呼ばれる）は、コンポーネントを単独でテストすることに焦点を当てる。コンポーネントテストは、テストハーネス、またはユニットテストフレームワークのような特定のサポートを必要とすることが多い。コンポーネントテストは、通常、開発者が開発環境で行う。

コンポーネント統合テスト（ユニット統合テストとも呼ばれる）は、コンポーネント間のインターフェース、および相互処理に焦点を当てる。コンポーネント統合テストは、ボトムアップ、トップダウン、ビッグバンといった統合戦略のアプローチに大きく依存する。

システムテストは、多くの場合、エンドツーエンドのタスクに対する機能テストや品質特性に対する非機能テストといったシステムやプロダクト全体の振る舞いや能力の全般に焦点を当てる。一部の非機能品質特性については、本番相当のテスト環境において、すべて揃ったシステムでテストすることが望ましい（例えば、使用性テスト）。サブシステムのシミュレーションを使用することも可能である。システムテストは、システムの仕様に関連するものであり、独立したテストチームによって実施されることがある。

システム統合テストは、テスト対象システムと他のシステム、外部サービスとのインターフェースのテストに焦点を当てる。システム統合テストには、できれば運用環境に近い適切なテスト環境が必要となる。

受け入れテストは、妥当性確認と、デプロイの準備ができていることの実証に焦点を当てる。これは、システムがユーザーのビジネスニーズを満たしていることを意味する。受け入れテストは、想定ユーザーによって実施をすることが理想的である。受け入れテストの主な形式としては、ユーザー受け入れテ

スト(UAT)、運用受け入れテスト、契約、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がある。

テストレベルは、テスト活動の重複を避けるため、以下の代表的な属性のリストで区別する：

- テスト対象
- テスト目的
- テストベース
- 欠陥、および故障
- アプローチと責務

2.2.2 テストタイプ

テストタイプには多くのものが存在し、プロジェクトに適用することができる。本シラバスでは 4 つのテストタイプを取り上げる。

機能テストでは、コンポーネント、およびシステムが実行する機能を評価する。機能とはテスト対象が「何」をすべきかである。機能テストの主な目的は、機能完全性、機能正確性、および機能適切性をチェックすることである。

非機能テストでは、コンポーネント、およびシステムが実行する機能特性以外の属性を評価する。非機能テストは、システムが「どのようにうまく振る舞うか」をテストする。非機能テストの主な目的は、非機能的なソフトウェア品質特性をチェックすることである。ISO/IEC 25010 標準では、非機能ソフトウェア品質特性について、以下のように分類している：

- 性能効率性
- 互換性
- 使用性
- 信頼性
- セキュリティ
- 保守性
- 移植性

非機能テストは、ライフサイクルの早期（例えば、レビューやコンポーネントテスト、システムテストの一部として）に開始するのが適切な場合もある。多くの非機能テストのテストケースは、同じ機能テストのテストケースを使用するため、機能テストのテストケースから導出する。ただし、機能が動いている際に、非機能制約が満たされることをチェックしている（例えば、ある機能が指定された時間内に動くか、ある機能が新しいプラットフォームに移植できるかをチェックする）。非機能の欠陥の発見が遅れると、プロジェクトの成功に重大な脅威をもたらす可能性がある。非機能テストでは、使用性テスト用のユーザビリティラボのような、非常に特殊なテスト環境が必要な場合がある。

ブラックボックステストは、仕様に基づき、テスト対象の外部にあるドキュメントからテストを導き出す（4.2 節参照）。ブラックボックステストの主な目的は、システムの動作をその仕様に照らしてチェックすることである。

ホワイトボックステストは、構造に基づき、システムの実装または内部構造（コード、アーキテクチャー、ワークフロー、データフローなど）からテストを導き出す（4.3 節参照）。ホワイトボックステストの主な目的は、テストによって基本的な構造を受け入れ可能なレベルまでカバーすることである。

上記の 4 つのテストタイプは、各テストレベルで焦点は異なるが、すべてのテストレベルに適用することができる。さまざまなテスト技法を用いて、前述のすべてのテストタイプのテスト条件とテストケースを導き出すことができる。

2.2.3 確認テストとリグレッションテスト

典型的には、変更は、コンポーネントやシステムに新しい機能を追加して強化したり、欠陥を除去して修正したりするために行われる。テストには確認テストやリグレッションテストも含めるべきである。

確認テストでは、元の欠陥が正常に修正されたことを確認する。リスクに応じて、以下のようないくつかの方法で修正版のソフトウェアのテストを行うことができる：

- 欠陥のために失敗したテストケースをすべて実行する
- 欠陥を修正するために必要な変更をカバーするために新しいテストケースを追加する

しかし、欠陥の修正にかかる時間や費用が限られている場合、確認テストは、単に欠陥によって引き起こされる故障を再現すべきステップを実行し、故障が発生しないことを確認するだけに限定することがある。

リグレッションテストでは、すでに確認テスト済みの修正を含め、変更によって悪影響が生じないことを確認する。これらの悪影響は、変更を加えた同じコンポーネント、同じシステム内の他のコンポーネント、あるいは接続されている他システムにも影響を及ぼす可能性がある。リグレッションテストは、テスト対象そのものに限らず、環境に関連する場合もある。まず、リグレッションテストの拡張を最適化するため、影響度分析を行うことが望ましい。影響度分析は、ソフトウェアのどの部分に影響が及ぶかを示す。

リグレッションテストスイートは何度も実行し、一般的にリグレッションテストケースの数はイテレーションやリリースごとに拡充する。そのため、リグレッションテストは自動化の有力な候補となる。これらのテストケースの自動化は、プロジェクトの早期に開始すべきである。DevOps（2.1.4 項参照）のように CI が使用される場合、自動化されたリグレッションテストも含めることがよい実践例となる。状況によっては、異なるレベルのリグレッションテストを含めることができる。

あるテストレベルで欠陥の修正および/または変更が行われた場合、テスト対象の確認テストおよび/またはリグレッションテストはすべてのテストレベルで必要となる。

2.3 メンテナンス（保守）テスト

メンテナンスにはさまざまなカテゴリーがある。メンテナンスには、修正を伴うもの、環境の変化に適応するもの、パフォーマンスや保守性を改善するもの（詳細は ISO/IEC 14764 を参照）などがある。そのため、計画的なリリース/デプロイと計画外のリリース/デプロイ（ホットフィックス）が含まれる。システムの他の領域への潜在的な影響に基づき変更をするべきかどうかを判断するために、変更をする前に影響度分析を行うことがある。本番稼働中のシステムに対する変更のテストには、変更の実装の成功を評価することと、変更されないシステムの部分（通常はシステムの大部分）で起こりうるリグレッションをチェックすることの両方が含まれる。

メンテナンステストの範囲は、典型的には次に依存する：

- 変更のリスクの度合い
- 既存システムの大きさ
- 変更の大きさ

メンテナンスのきっかけやメンテナンステストのきっかけは、以下のように分類する：

- 計画的な機能拡張（すなわち、リリースベース）、修正を伴う変更またはホットフィックスといった改良。
- あるプラットフォームから別のプラットフォームへといった運用環境の更新、または移行。
  新しい環境や変更されたソフトウェアに関連するテストが必要となる場合があり、別のアプリケーションのデータを保守しているシステムに移行する場合は、データ変換のテストが必要となる。
- アプリケーションの寿命などによる廃棄。
  システムを廃棄する際、長期間のデータ保持が要件となる場合は、データアーカイブのテストが必要となることがある。また、アーカイブ期間中にデータの取得が必要な場合は、アーカイブ後の復元や取得手順のテストも必要になることがある。

3 静的テスト

80 分

キーワード

不正、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、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

第 3 章の学習の目的

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によって確認できるプロダクトの種類を認識する。

FL-3.1.2 （K2）静的テストの価値を説明する。

FL-3.1.3 （K2）静的テストと動的テストを比較、そして対比する。

3.2 フィードバックとレビュープロセス

FL-3.2.1 （K1）早期かつ頻繁なステークホルダーからのフィードバックの利点を識別する。

FL-3.2.2 （K2）レビュープロセスの活動を要約する。

FL-3.2.3 （K1）レビューの際に、主要な役割にどのような責務が割り当てられているかを想起する。

FL-3.2.4 （K2）さまざまなレビュー種別を比較、そして対比する。

FL-3.2.5 （K1）レビューの成功に貢献する要因を想起する。

3.1 静的テストの基本

静的テストでは、動的テストと異なり、テスト対象のソフトウェアを実行する必要はない。コード、処理仕様、システムアーキテクチャー仕様、または他の作業成果物は、人手による確認（例えば、レビュー）によって、またはツール（例えば、静的解析）の力を借りて評価する。テスト目的は、品質向上、欠陥検出、可読性、完全性、正確性、試験性、一貫性などの特性の評価を含む。静的テストは、検証、および妥当性確認の両方に適用することができる。

テスト担当者、ビジネス側の代表者、開発者は、実例マッピングに一緒に取り組み、ユーザーストーリーを共同で書き、バックログリファインメントセッションにおいて協力し、ユーザーストーリーと関連作業成果物が、定義された基準、例えば「準備完了（Ready）の定義」（5.1.3 項参照）を満たすことを確認する。レビューの技法は、ユーザーストーリーが完全で理解しやすく、テスト可能な受け入れ基準を含むことを確認するために適用できる。テスト担当者は、適切な質問をすることで、提案されたユーザーストーリーを探索し、課題を挙げて、改善を支援する。

静的解析は、テストケースが不要であり、典型的にはツール（第 6 章参照）を使用することにより、動的テストの前に問題を識別できるため、より少ない労力で済む場合が多い。静的解析は、CI フレームワークへ組み込むことが多い（2.1.4 項参照）。静的解析は、主にコードの特定の欠陥を検出するために使用するが、保守性やセキュリティを評価するためにも使用する。スペルチェッカーや可読性を上げるツールも、静的解析ツールの一例である。

3.1.1 静的テストで確認可能な作業成果物

ほとんどの作業成果物は静的テストを使用して確認できる。例えば、要件仕様書、ソースコード、テスト計画書、テストケース、プロダクトバックログアイテム、テストチャーター、プロジェクトドキュメント、契約書、モデルが含まれる。

読んで理解できるあらゆる作業成果物はレビューの対象になりうる。しかし、静的解析では、作業成果物をチェックするための構造（例えば、モデル、コード、および正式な構文を持つテキスト）が必要である。

静的テストに適さない作業成果物は、人間による解釈が困難なもの、ツールで解析してはいけないもの（例えば法規制がある第三者の実行可能コード）を含む。

3.1.2 静的テストの価値

静的テストは SDLC の初期段階で欠陥を検出することができるため、早期テストの原則を満たす（1.3 節参照）。また、動的テストでは検出できない欠陥（例えば、到達できないコード、狙い通りに実装されていないデザインパターン、実行不可能な作業成果物に内在する欠陥）を検出することもできる。

静的テストは、作業成果物の品質を評価し、作業成果物に対する信頼性を高めることができる。文書化した要件を検証することで、ステークホルダーは、要件が実際のニーズを表していることを確認することもできる。静的テストは SDLC の早期に実施できるので、ステークホルダーの間で共通の理解を得ることができる。また、ステークホルダー間のコミュニケーションも改善できる。このため、静的テストには、さまざまなステークホルダーが参加することを推奨する。

レビューを実装するにはコストがかかるが、プロジェクトの後半で欠陥を修正するために費やす時間と労力が少なくて済むため、通常、プロジェクト全体のコストはレビューを実施しない場合よりもはるかに安価となる。

静的解析は動的テストよりも効率的にコードの欠陥を検出することができ、通常、コードの欠陥の減少と開発工数の削減の両方を実現することができる。

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の実践は、互いに補完し合っている。両者の目的は、作業成果物の欠陥検出を支援する（1.1.1 項参照）といったように似てはいるが、以下のような違いもある：

- 静的テストと（故障の分析を含む）動的テストは、どちらも欠陥の発見につながるが、欠陥の種類によっては、静的テスト、および動的テストのどちらかでしか発見できない。
- 静的テストでは、欠陥を直接発見するが、動的テストでは故障を引き起こし、その後の分析によって関連する欠陥を特定する。
- 静的テストでは、動的テストではほとんど実行されない、あるいは到達しにくいコード内のパスに潜む欠陥を、より簡単に検出することがある。
- 静的テストは実行不可能な作業成果物に適用できるが、動的テストは実行可能な作業成果物にしか適用できない。
- 静的テストは、コードの実行に依存しない品質特性（例えば、保守性）を測定するために使用でき、動的テストはコードの実行に依存する品質特性（例えば、性能効率性）を測定するために使用できる。

静的テストは、以下の欠陥を早期および／または安価に検出できる。

- 要件の欠陥（例えば、一貫してないこと、曖昧さ、矛盾、欠落、不正確さ、重複）
- 設計の欠陥（例えば、非効率なデータベース構造、不十分なモジュール化）
- ある種のコードの欠陥（例えば、値が定義されていない変数、宣言されていない変数、到達不能コード、重複したコード、過剰なコード複雑度）
- 標準からの逸脱（例えば、コーディング標準の命名規則を遵守していない）
- 正しくないインターフェース仕様（例えば、パラメーターの数、種類、または順序が不一致）
- 特定の種類のセキュリティ脆弱性（例えば、バッファオーバーフロー）
- テストベースに対する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）

3.2 フィードバックとレビュープロセス

3.2.1 早期かつ頻繁なステークホルダーからのフィードバックの利点

早期かつ頻繁なフィードバックは、潜在的な品質問題を早期に伝えることができる。SDLC の中でステークホルダーの関与が少ない場合、開発中のプロダクトは、ステークホルダーの当初、または現在のビジョンを満たさないかもしれない。ステークホルダーが望むものを提供できない場合、コストのかかる手直し、納期の遅れ、責任のなすり合いが発生し、さらにはプロジェクトの完全な失敗につながる可能性がある。

SDLC を通じてステークホルダーのフィードバックを頻繁に行うことで、要件に関する誤解を防ぎ、要件の変更をより早く理解し実装することができる。これは、開発チームが自分たちで構築しているものに対する理解を深めるのに役立つ。これにより、開発チームは、ステークホルダーに最大の価値を与え、かつ識別したリスクに最もよい影響を与えるフィーチャーに焦点を当てることができる。

3.2.2 作業成果物のレビュープロセス

ISO/IEC 20246 標準は、特定の状況に応じて具体的なレビュープロセスを調整できる、構造化されているが柔軟な枠組みを提供する汎用的なレビュープロセスを定義している。必要となるレビューがより形式的なものである場合、さまざまな活動に対して標準に記載がある、より多くのタスクが必要となる。

多くの作業成果物は、そのサイズが大きすぎるため、1 回のレビューではカバーしきれない。作業成果物全体のレビューを完了させるために、レビュープロセスを 2、3 回行うことがある。

レビュープロセスの活動は以下がある。

- **計画** 計画フェーズでは、目的、レビュー対象となる作業成果物、評価対象の品質特性、重点を置く領域、終了基準、標準などの裏付け情報、レビューの工数と時間枠からなるレビューの範囲を定義する。
- **レビューの立ち上げ** レビューの立ち上げでは、レビュー開始までに関係者や物事が準備できたことの明確化がゴールとなる。これには、すべての参加者がレビュー対象の作業成果物にアクセスできること、自分の役割と責務を理解していること、レビューの実行に必要なすべてのものを受け取っていることを確認することが含まれる。
- **個々人のレビュー** すべてのレビューアは、レビュー対象の作業成果物の品質を精査し、1 つ以上のレビュー技法（チェックリストベースドレビュー、シナリオベースドレビューなど）を適用して、不正、推奨、および質問を識別するために個人のレビューを実施する。ISO/IEC 20246 標準では、さまざまなレビュー技法についてより詳しく説明している。レビューアは、特定されたすべての不正、推奨、および質問を記録する。
- **共有と分析** レビュー中に識別した不正が必ずしも欠陥であるとは限らないため、これらすべての不正を分析して議論する必要がある。すべての不正について、そのステータス、オーナー、必要なアクションを決めるべきである。これは通常、レビューミーティングで行われ、参加者はレビューした作業成果物の品質レベルと必要なフォローアップのアクションも決める。アクションを完了するには、フォローアップのレビューが必要になる場合がある。
- **修正と報告** 是正処置をフォローアップできるように、すべての欠陥に対して欠陥レポートを作成すべきである。終了基準に達すれば、作業成果物を受け入れることができる。レビュー結果はレポートする。

3.2.3 レビューでの役割と責務

レビューにはさまざまなステークホルダーが関与しており、彼らはいくつかの役割を担っている可能性がある。主な役割とその責任は以下の通りである：

- マネージャー-何をレビューするかを決定し、レビューのための人員や時間などのリソースを提供する
- 作成者-レビュー対象の作業成果物を作成し、修正する
- モデレーター（ファシリテーターとも呼ばれる）-調整、時間のマネジメント、誰もが自由に発言できる安全なレビュー環境など、レビューミーティングを効果的に運営する
- 書記（記録係とも呼ばれる）-レビューアからの不正を取りまとめ、決定事項やレビューミーティング中に発見された新たな不正などのレビュー情報を記録する
- レビューア-レビューを行う。レビューアは、プロジェクトに携わっている人、特定分野の専門家、その他のステークホルダーである場合がある
- レビューリーダー-誰が参加するか、いつ、どこでレビューが行われるかを決めるなど、レビューの全体的な責任を負う

ISO/IEC 20246 標準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。

3.2.4 レビュー種別

非形式的なレビューから形式的なレビューまで、さまざまなタイプのレビューが存在する。必要な形式的レベルは、従う SDLC、開発プロセスの成熟度、レビューされる作業成果物の重要性と複雑度、法的または規制上の要件、および監査証跡の必要性などの要因に依存する。同じ作業成果物でも、異なるレビュー種別（例えば、最初は非形式的なレビューを行い、後に形式的なレビューを行う）でレビューすることができる。

適切なレビュー種別を選択することは、要求されるレビュー目的を達成するための鍵となる（3.2.5 項参照）。その選択は目的だけでなく、プロジェクトのニーズ、使用可能なリソース、作業成果物の種類とリスク、ビジネスドメイン、企業文化などの要素にも基づいて行われる。

よく使われるレビュー種別は以下の通りである：

- **非形式的レビュー**　非形式的レビューは、定義されたプロセスに従わず、正式に文書化されたアウトプットを必要としない。主な目的は、不正の検出である。
- **ウォークスルー**　作成者が主導するウォークスルーは、品質評価や作業成果物に対する信頼の積み上げ、レビューアの教育、合意形成、新しいアイデアの創出、作成者のモチベーションの向上と改善、不正の検出など、多くの目的を達成することができる。レビューアはウォークスルーの前に個人のレビューをしてもよいが、必須ではない。
- **テクニカルレビュー**　テクニカルレビューは、技術的に適格なレビューアによって行われ、モデレーターが主導する。テクニカルレビューの目的は、技術的な問題に関して合意を得て判断するだけでなく、不正の検出、品質評価や作業成果物に対する信頼の積み上げ、新しいアイデアの創出、作成者のモチベーションの向上と改善ができるようにすることである。
- **インスペクション**　インスペクションは、最も形式的なレビューであるため、汎用プロセス（3.2.2 項参照）に完全に従う。主な目的は、不正を最大数発見することである。その他の目的は、品質評価や作業成果物に対する信頼の積み上げ、作成者のモチベーションの向上と改善ができるようにすることである。メトリクスを収集して、インスペクションプロセスを含む SDLC を改善するために使用する。インスペクションでは、作成者はレビューリーダーや書記として活動することはできない。

3.2.5 レビューの成功要因

レビューの成功を決めるいくつかの要因があり、それは以下の通りである。

- 明確な目的と測定可能な終了基準を定義する。参加者の評価は、決して目的にしてはならない
- 与えられた目的を達成するために、作業成果物の種類、レビュー参加者、プロジェクトのニーズや状況に合わせて、適切なレビュー種別を選択する
- 個人のレビュー、および/またはレビューミーティング（開催時）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
- レビューからステークホルダーや作成者へフィードバックを提供し、ステークホルダーや作成者がプロダクトおよび自分たちの活動を改善できるようにする（3.2.1 項参照）
- 参加者に十分な準備時間を与える
- レビュープロセスに対してマネジメントから支援をする
- レビューを組織の文化として定着させ、学習とプロセス改善を促進する
- すべての参加者が自分の役割を果たす方法を知ることができるよう、適切なトレーニングを提供する
- ミーティングをファシリテートする

4 テスト分析と設計	390 分

キーワード

受け入れ基準、受け入れテスト駆動開発、ブラックボックステスト技法、境界値分析、ブランチカバレッジ、チェックリストベースドテスト、コラボレーションベースのテストアプローチ、カバレッジ、カバレッジアイテム、デシジョンテーブルテスト、同値分割法、エラー推測、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、ホワイトボックステスト技法

第 4 章の学習の目的

4.1 テスト技法の概要

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を区別する。

4.2 ブラックボックステスト技法

FL-4.2.1 （K3）テストケースを導出するために同値分割法を使用する。

FL-4.2.2 （K3）テストケースを導出するために境界値分析を使用する。

FL-4.2.3 （K3）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使用する。

FL-4.2.4 （K3）テストケースを導出するために状態遷移テストを使用する。

4.3 ホワイトボックステスト技法

FL-4.3.1 （K2）ステートメントテストを説明する。

FL-4.3.2 （K2）ブランチテストを説明する。

FL-4.3.3 （K2）ホワイトボックステストの価値を説明する。

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。

4.5 コラボレーションベースのテストアプローチ

FL-4.5.1 （K2）開発者やビジネス側代表者と共同でユーザーストーリーを作成する方法を説明する。

FL-4.5.2 （K2）受け入れ基準を書くためのさまざまな選択肢を分類する。

FL-4.5.3 （K3）テストケースを導出するために受け入れテスト駆動開発（ATDD）を使用する。

4.1 テスト技法の概要

テスト技法は、テスト分析（何をテストするか）とテスト設計（どのようにテストするか）において、テスト担当者をサポートする。テスト技法は、比較的少ないながらも十分なテストケースのセットを体系的に開発するのに役立つ。また、テスト技法は、テスト担当者がテスト分析やテスト設計の際に、テスト条件を定義し、カバレッジアイテムを識別し、テストデータを識別するのに役立つ。テスト技法と対応する手段に関するさらなる情報は、ISO/IEC/IEEE 29119-4 標準、そして（Beizer 1990, Craig 2002, Copeland 2004, Koomen 2006, Jorgensen 2014, Ammann 2016, Forgács 2019）にて見つけられる。

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、経験ベースに分類している。

ブラックボックステスト技法（仕様ベース技法とも呼ばれる）は、テスト対象の内部構造を参照することなく、仕様上の振る舞いの分析に基づくものである。したがって、テストケースは、ソフトウェアの実装方法とは無関係である。結果的に実装が変わったとしても、求められる振る舞いが同じであれば、テストケースは依然として有用である。

ホワイトボックステスト技法（構造ベース技法とも呼ばれる）は、テスト対象の内部構造や処理の分析に基づくものである。テストケースはソフトウェアの設計方法に依存するため、テスト対象の設計や実装が終わってからでないと作成できない。

経験ベースのテスト技法は、テストケースの設計と実装にテスト担当者の知識と経験を効果的に利用するものである。これらの技法の有効性は、テスト担当者のスキルに大きく依存する。経験ベースのテスト技法は、ブラックボックスやホワイトボックスのテスト技法では見逃してしまうような欠陥も検出することができる。したがって、経験ベースのテスト技法は、ブラックボックスやホワイトボックスのテスト技法を補完するものである。

4.2 ブラックボックステスト技法

本節で説明する一般的に使用するブラックボックステスト技法は以下である：

- 同値分割法
- 境界値分析
- デシジョンテーブルテスト
- 状態遷移テスト

4.2.1 同値分割法

同値分割法(EP)は、ある特定のパーティションのすべての要素がテスト対象によって同等に処理されることを想定して、データをパーティション（これを同値パーティションと呼ぶ）に分割する。この技法の背景にある理論は、同値パーティションから 1 つの値をテストするテストケースが欠陥を検出した場合、この欠陥は同じパーティションから他の値をテストするテストケースでも検出されるはずだというものである。したがって、各パーティションに対して 1 つのテストがあれば十分である。

同値パーティションは、入力、出力、構成アイテム、内部値、時間関連の値、インターフェースパラメーターなど、テスト対象に関連するあらゆるデータ要素について識別できる。パーティションは、連続または離散、順序性ありまたは順序性なし、有限または無限のいずれでもよい。パーティションは、重複してはならず、空でない集合でなければならない。

単純なテスト対象であれば EP は簡単だが、実際には、異なる値をテスト対象がどのように扱うのかを理解することは、困難なことが多い。そのため、パーティションに分割するのは慎重に行うべきである。

有効な値を包含するパーティションを有効パーティションと呼ぶ。無効な値を包含するパーティションは、無効パーティションと呼ぶ。有効な値と無効な値の定義は、チームや組織によって異なる場合がある。例えば、有効な値は、テスト対象が処理すべきもの、また仕様書に処理が定義されているものと解釈できる。無効な値は、テスト対象が無視または拒絶すべきもの、あるいはテスト対象の仕様書に処理が定義されていないものと解釈できる。

EP では、カバレッジアイテムは同値パーティションである。カバレッジ 100%を達成するために、この技法では、テストケースは、識別されたすべてのパーティション（無効パーティションを含む）を、少なくとも 1 回はカバーするように通過しなければならない。カバレッジは、少なくとも 1 つのテストケースによって通したパーティションの数を、識別されたパーティションの総数で割った値として測定し、パーセンテージで表す。

多くのテスト対象は複数のパーティションセットを含んでおり（例えば、複数の入力パラメーターを持つテスト対象）、テストケースはさまざまなパーティションセットからパーティションをカバーすることになる。複数のパーティションセットがある場合の最も単純なカバレッジ基準は、イーチチョイスカバレッジ（Ammann 2016）と呼ぶ。イーチチョイスカバレッジは、テストケースが各パーティションセットから各パーティションを少なくとも 1 回は通すことを要件とする。イーチチョイスカバレッジでは、パーティションの組み合わせは考慮しない。

4.2.2 境界値分析

境界値分析（BVA）は、同値パーティションの境界を確認することに基づいた技法である。したがって、BVA は順序性のあるパーティションにのみ使用できる。1 つのパーティションの最小値と最大値はその境界値となる。BVA の場合、2 つの要素が同じパーティションに属している場合、それらの間にあるすべての要素もそのパーティションに属していなければならない。

BVA では、パーティションの境界値に着目する。これは、開発者が境界値に関するエラーを起こしやすいためである。BVA で発見される欠陥の代表的なものは、実装された境界が意図した位置より上や下にずれていたり、完全に省略されていたりするものである。

本シラバスでは、2 値 BVA と 3 値 BVA の 2 つのバージョンの BVA を扱っている。両者は、100%のカバレッジを達成するために通す必要がある境界ごとのカバレッジアイテムという点で異なっている。

2 値 BVA（Craig 2002、Myers 2011）では、境界値ごとに 2 つのカバレッジアイテムがある。2 つのカバレッジアイテムとは、この境界値と、隣接するパーティションに属するその最も近い隣接値である。2 値 BVA で 100%のカバレッジを達成するには、テストケースですべてのカバレッジアイテム、つまり識別したすべての境界値を通過しなければならない。カバレッジは、通した境界値の数を識別した境界値の総数で割ったものとして測定し、パーセンテージで表す。

3 値 BVA（Koomen 2006、O’Regan 2019）では、境界値ごとに 3 つのカバレッジアイテム（この境界値とその両方の隣接値）がある。したがって、3 値 BVA では、一部のカバレッジアイテムが境界値にならない場合がある。3 値 BVA で 100%のカバレッジを達成するには、テストケースですべてのカバレッジアイテム（つまり、識別された境界値とその隣接値）を通過しなければならない。カバレッジは、通した境界値とその隣接値の数を、識別した境界値とその隣接値の合計数で割って測定し、パーセンテージで表す。

3 値 BVA は、2 値 BVA では見落とされた欠陥を検出する可能性があるため、2 値 BVA よりも厳密である。例えば、「if(x≦10)…」という判定を誤って「if(x=10)…」として実装した場合、2 値 BVA(x=10、x=11)から導出されたテストデータではその欠陥を検出できない。ただし、3 値 BVA から導出される x=9 は、それを検出する可能性がある。

4.2.3 デシジョンテーブルテスト

デシジョンテーブルは、条件のさまざまな組み合わせがどのようにさまざまな結果をもたらすかを仕様化しているシステム要件の実装をテストするために使用する。デシジョンテーブルは、ビジネスルールなどの複雑なロジックを記録する効果的な方法である。

デシジョンテーブルを作成する際には、システムの条件と結果として起きるアクションを決める。条件とアクションは表の行となる。各列は、条件の一意な組み合わせを定義する判定のルールと、関連するアクションに対応する。制限指定デシジョンテーブルでは、条件とアクションの値（無関係なもの、実行不可能なものを除く、下記参照）はすべてブール値（真または偽）で示す。あるいは、拡張指定デシジョンテーブルでは、条件やアクションの一部または全部が複数の値（例えば、範囲のある数値、同値パーティション、離散値）を取ることもある。

条件の表記は以下の通りとなる："T"（真）は、条件が満たされていることを意味する。"F"（偽）は、条件が満たされていないことを意味する。"-"は、その条件の値がアクションの結果に無関係であることを意味する。"N/A"は、その条件が与えられたルールに対して実行不可能であることを意味する。アクションの場合："X"は、そのアクションが発生すべきことを意味する。空白は、そのアクションが発生すべきではないことを意味する。その他の表記を使うこともある。

完全なデシジョンテーブルは、すべての条件の組み合わせをカバーするのに十分な列を持つ。実行不可能な条件の組み合わせを含む列を削除することで、表を簡略化することができる。また、いくつかの条件が結果に影響しない列を 1 つの列に統合することで、表を最小化することができる。デシジョンテーブルの最小化アルゴリズムは本シラバスの範囲外である。

デシジョンテーブルテストでは、カバレッジアイテムは、実行可能な条件の組み合わせを含む列である。カバレッジを 100%にするために、この技法では、テストケースはこれらの列をすべて通過する必要がある。カバレッジは、通した列の数を実行可能な列の総数で割った値で測定し、パーセンテージで表す。

デシジョンテーブルテストの強みは、見逃される可能性がある組み合わせを含めて、条件の組み合わせのすべてを識別するための体系的なアプローチを提供することである。また、要件の相違点や矛盾を発見するのにも役立つ。条件が多い場合、判定ルールの数は条件の数だけ指数関数的に増えるので、すべての判定ルールを確認するのは時間がかかるかもしれない。このような場合、確認する必要のあるルールの数を減らすために、最小化したデシジョンテーブルやリスクベースドアプローチを使用することがある。

4.2.4 状態遷移テスト

状態遷移図は、システムの取り得る状態と有効な遷移を示すことで、システムの振る舞いをモデル化する。遷移は、イベントによって開始され、さらにガード条件によって限定されることがある。遷移は瞬時に起きることを想定しており、遷移によってソフトウェアがアクションを起こす場合がある。一般的な遷移の構文は次の通りラベル付けする：「イベント［ガード条件］／アクション」。ガード条件とアクションは、存在しない場合とテスト担当者にとって無関係な場合は省略することができる。

状態表は、状態遷移図に相当するモデルである。行は状態を表し、列はイベント（存在する場合はガード条件）を表す。テーブルの項目（セル）は遷移を表し、ターゲット状態、ガード条件、および定義さ

れている場合は結果のアクションを含む。状態遷移図とは対照的に、状態表は無効な遷移を明示的に示し、空のセルで表す。

状態遷移図や状態表に基づくテストケースは、通常、一連のイベントとして表現し、一連の状態変化（および必要であればアクション）が結果となる。通常、1 つのテストケースは、状態間のいくつかの遷移をカバーすることができる。

状態遷移テストには多くのカバレッジ基準が存在する。本シラバスでは、そのうちの 3 つについて説明する。

全状態カバレッジでは、カバレッジアイテムは状態である。100%の全状態カバレッジを達成するためには、テストケースはすべての状態を通過することを保証しなければならない。カバレッジは、通過した状態の数を状態の総数で割った値で測定し、パーセンテージで表す。

有効遷移カバレッジ（0 スイッチカバレッジとも呼ばれる）では、カバレッジアイテムは単一の有効な遷移である。100%の有効遷移カバレッジを達成するには、テストケースはすべての有効な遷移を通さなければならない。カバレッジは、通した有効な遷移の数を有効な遷移の総数で割った値で測定し、パーセンテージで表す。

全遷移カバレッジでは、カバレッジアイテムは、状態表に示されるすべての遷移である。100%の全遷移カバレッジを達成するためには、テストケースはすべての有効な遷移を通過し、無効な遷移の実行を試みなければならない。1 つのテストケースで無効な遷移を 1 つだけテストすることで、欠陥のマスキング（ある欠陥が別の欠陥の検出を妨げてしまうこと）を避けることができる。カバレッジは、通した有効および無効な遷移の数、または実行したテストケースでカバーを試みた有効および無効な遷移の数を、有効および無効な遷移の総数で割った値で測定し、パーセンテージで表す。

全状態カバレッジは、すべての遷移を通さなくても達成できることが一般的であるため、有効遷移カバレッジよりも弱い。有効遷移カバレッジは、最も広く使われているカバレッジ基準である。有効遷移カバレッジを完全に達成すると、全状態カバレッジを完全に保証する。全遷移カバレッジを達成することは、全状態カバレッジと有効遷移カバレッジの両方を保証することになる。全遷移カバレッジは、ミッションクリティカルおよびセーフティクリティカルソフトウェアの最低要件とすべきである。

4.3 ホワイトボックステスト技法

知名度とわかりやすさから、この節では、コードに関連する 2 つのホワイトボックステスト技法に焦点を当てる。

- ステートメントテスト
- ブランチテスト

一部のセーフティクリティカル、ミッションクリティカル、または高い完全性が求められる環境で使用する、より徹底したコードカバレッジを実現するためのより厳密な技法が存在する。また、より高いテストレベル（例えば、API テスト）で使用するホワイトボックステスト技法や、コードに関係ないカバレッジ（例えば、ニューラルネットワークテストにおけるニューロンカバレッジなど）もある。これらの技術については、本シラバスでは説明しない。

4.3.1 ステートメントテストとステートメントカバレッジ

ステートメントテストでは、カバレッジアイテムは実行可能なステートメントである。その狙いは、許容できるレベルのカバレッジを達成するまで、コード内のステートメントを通すテストケースを設計することである。カバレッジは、テストにより通過したステートメント数を、コードの実行可能ステートメントの総数で割った値で計測し、パーセンテージで表す。

ステートメントカバレッジが 100%になると、コード内のすべての実行可能なステートメントを少なくとも一度は通過したことになる。これは、特に欠陥のある各ステートメントを実行することを意味し、欠陥の存在を証明する故障が発生する可能性がある。しかし、あるテストケースを用いてステートメントを実行しても、すべてのケースで欠陥が検出されるわけではない。例えば、データに依存する欠陥

（例えば、分母が 0 に設定された場合にのみ失敗するゼロ除算）は検出されないかもしれない。また、ステートメントカバレッジが 100%であっても、判定ロジックをすべてテストしていることは確認できない。例えば、コード内のすべてのブランチ（4.3.2 節参照）を通過していない可能性がある。

4.3.2 ブランチテストとブランチカバレッジ

ブランチとは、制御フローグラフの 2 つのノード間の制御の遷移のことで、テスト対象においてソースコードのステートメントが実行されうるシーケンスを示すものである。各制御の遷移は、無条件（つまり、直列のコード）または条件付き（つまり、判定結果）のいずれかになる。

ブランチテストでは、カバレッジアイテムはブランチであり、その狙いは、許容できるレベルのカバレッジを達成するまで、コード内のブランチを通過するテストケースを設計することである。カバレッジは、テストケースによって通したブランチの数をブランチの総数で割った値で計測し、パーセンテージで表す。

ブランチカバレッジが 100%になると、コード内のすべてのブランチ（無条件および条件付き）をテストケースで通すことになる。条件ブランチは、通常、「if...then」判定による真偽の結果、switch/case ステートメントによる結果、ループ内の終了または継続の判定に対応する。しかし、テストケースでブランチを通過しても、すべてのケースで欠陥が検出されるわけではない。例えば、コード内の特定のパスの実行を必要とする欠陥は検出されない可能性がある。

ブランチカバレッジはステートメントカバレッジを包含する。つまり、ブランチカバレッジ 100％を達成した一連のテストケースは、100%のステートメントカバレッジも達成する（しかし、その逆は成立しない）。

4.3.3 ホワイトボックステストの価値

ホワイトボックステスト技法に共通する基本的な強みは、テスト時にソフトウェアの実装そのものを考慮することで、ソフトウェアの仕様が曖昧であったり、古かったり、不完全であったりしても、欠陥の検出を容易にすることである。一方、弱点としては、ソフトウェアが 1 つまたは複数の要件を実装していない場合、ホワイトボックステストでは、結果として生じる欠落を欠陥として検出できない可能性がある（Watson 1996）。

ホワイトボックステスト技法は、（例えば、コードのドライランの際の）静的テストに使用できる。それらは、制御フローグラフでモデル化できる疑似コードやその他のハイレベルまたはトップダウンのロジックのように、まだ実行可能な状態にないコードのレビューに適している（Hetzel 1988）。

ブラックボックステストだけを実施しても、実際のコードカバレッジを測定することはできない。ホワイトボックステストは、カバレッジを客観的に測定し、テストを追加で作りカバレッジを向上させ、コードの信頼性を高めるために必要な情報を提供するものである。

4.4 経験ベースのテスト技法

以降の項で説明する一般的に使用される経験ベースのテスト技法は次の通り：

- エラー推測
- 探索的テスト
- チェックリストベースドテスト

4.4.1 エラー推測

エラー推測は、エラー、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
- 開発者が起こしやすいエラーの種類と、そのエラーに起因する欠陥の種類
- 他の類似アプリケーションで発生した故障の種類

一般に、エラー、欠陥および故障は、入力（例えば、正しい入力が受け付けられない、パラメーターの誤りまたは不足）、出力（例えば、誤った形式、誤った結果）、論理（例えば、条件の不足、誤った演算子）、計算（例えば、誤ったオペランド、誤った計算）、インターフェース（例えば、パラメーターの不一致、互換性のない型）またはデータ（例えば、誤った初期化、誤った型）に関連することがある。

フォールト攻撃は、エラー推測を実施するための系統的なアプローチである。この技法では、テスト担当者が、想定されるエラー、欠陥、故障のリストを作成または入手し、エラーに関連する欠陥を特定したり、欠陥を露わにしたり、故障を引き起こすようなテストケースを設計する必要がある。これらのリストは、経験、欠陥や故障のデータ、またはソフトウェアが失敗する理由についての一般的な知識に基づいて作成することができる。

エラー推測やフォールト攻撃については、（Whittaker 2002, Whittaker 2003, Andrews 2006）を参照のこと。

4.4.2 探索的テスト

探索的テストは、テスト担当者がテスト対象について学習しながら、テストケースを同時に設計、実行、評価する。探索的テストは、テスト対象についてより深く知り、注力したテストでより深く探索し、まだテストしていない領域のテストを作成するために使う。

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アプローチでは、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。テストセッションの後には、通常、テスト担当者とテストセッションのテスト結果に関心を持つステークホルダーとの間で議論を行うデブリーフィングを行う。このアプローチでは、テスト目的はハイレベルのテスト条件として扱われることがある。カバレッジアイテムはテストセッション中に識別して確認する。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役立つ。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。テスト担当者が経験豊富で、ドメインの知識があり、分析力、好奇心、創造性などの重要なスキルを高いレベルで持っている場合、探索的テストはより効果的になる（1.5.1 項参照）。

探索的テストは、他のテスト技法と併用できる（例えば、同値分割法）。探索的テストに関する詳細な情報は、（Kaner 1999, Whittaker 2009, Hendrickson 2013）に記載されている。

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、チェックリストの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。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、また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ことができ

る。チェックリストには、自動的にチェックできる項目、開始・終了基準に適した項目、一般的すぎる項目は含めない方がよい（Brykczynski 1999）。

チェックリストの項目は、多くの場合、質問の形で表現する。各項目を個別に直接チェックできるようにすべきである。これらの項目は、要件、GUI の設定、品質特性、または他の形式のテスト条件を参照することがある。チェックリストは、機能テストや非機能テストなど、さまざまなテストタイプをサポートするために作成することができる（例：使用性テストのための 10 のヒューリスティック（Nielsen 1994））。

開発者は同じエラーを避けることを学ぶので、チェックリストへ追加した内容は、時間の経過とともに徐々に効果的でなくなるものがある。また、新たに発見した重要度の高い欠陥を反映するために、新しい項目の追加を必要とする場合もある。したがって、チェックリストは欠陥分析に基づき定期的に更新すべきである。ただし、チェックリストが長くなりすぎないように注意すべきである（Gawande 2009）。

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は、テストのガイドラインとある程度の一貫性を与えることができる。チェックリストがハイレベルである場合、実際のテストである程度のばらつきが生じる可能性があり、その結果、カバレッジが大きくなる可能性があるが、再現性は低くなる。

4.5 コラボレーションベースのテストアプローチ

上記の各技術（4.2, 4.3, 4.4 節参照）は、欠陥検出に関して特定の目的を持っている。一方、コラボレーションベースのアプローチは、コラボレーションとコミュニケーションによる欠陥を回避することにも焦点を当てる。

4.5.1 ユーザーストーリーの共同執筆

ユーザーストーリーは、システムやソフトウェアのユーザーや購入者にとって価値のある機能であることを表す。ユーザーストーリーには、3 つの重要な側面があり（Jeffries 2000）、これらを合わせて「3 つの C」と呼ぶ：

- カード（Card）-ユーザーストーリーを記述する媒体（例：インデックスカード、電子なボード上のエントリー）
- 会話（Conversation）-ソフトウェアがどのように使用されるかを説明する（文書化または口頭）
- 確認（Confirmation）-受け入れ基準（4.5.2 項参照）

ユーザーストーリーの最も一般的な形式は、「[役割]として[達成するゴール]を達成したい、それは[役割のための結果としてのビジネス価値]を実現するためだ」、その後に受け入れ基準が続くというものである。

ユーザーストーリーの共同執筆には、ブレーンストーミングやマインドマップなどの技法を用いることができる。コラボレーションによって、ビジネス、開発、テストの 3 つの視点を考慮し、チームが提供すべきものに対する共通のビジョンを得ることができる。

優れたユーザーストーリーは、独立性（Independent）があり、交渉可能（Negotiable）であり、価値があり（Valuable）、見積り可能（Estimable）であり、小さくて（Small）、テスト可能（Testable）であるべきである（INVEST）。ステークホルダーがユーザーストーリーをテストする方法がわからない場合、ユーザーストーリーが十分に明確ではないか、ステークホルダーにとって価値のあるものが反映さ

れていない、またはステークホルダーがテストで助けを必要としているだけであることを示している可能性がある（Wake 2003）。

4.5.2 受け入れ基準

ユーザーストーリーの受け入れ基準とは、ユーザーストーリーの実装をステークホルダーが受け入れるために満たさなければならない条件のことである。この観点から、受け入れ基準は、テストケースによって確認すべきテスト条件とみなすことができる。受け入れ基準は，通常，会話（4.5.1 項参照）の結果である。

受け入れ基準は、以下のように使う：

- ユーザーストーリーのスコープの定義
- ステークホルダー間の合意形成
- ポジティブシナリオとネガティブシナリオの両方に対する説明
- ユーザーストーリーの受け入れテスト（4.5.3 項参照）のベースを提供
- 正確な計画と見積りへの貢献

ユーザーストーリーの受け入れ基準を書くには、いくつかの方法がある。2 つの形式が最も一般的である：

- シナリオ指向（例えば、BDD で使用される Given/When/Then 形式、2.1.3 節参照）
- ルール指向（例えば、箇条書きの検証リストや、入出力マッピングの表形式）

ほとんどの受け入れ基準は、これら 2 つの形式のいずれかで文書化することができる。しかし、受け入れ基準が十分に定義され、曖昧さがない限り、チームは別の独自の形式を使用してもよい。

4.5.3 受け入れテスト駆動開発（ATDD）

ATDD は、テストファーストのアプローチである（2.1.3 項を参照）。テストケースは、ユーザーストーリーを実装する前に作成する。テストケースは、顧客、開発者、テスト担当者など、さまざまな視点を持つチームメンバーが作成する（Adzic 2009）。テストケースは手動で実行することもあれば、自動で実行することもある。

最初のステップは仕様化のワークショップで、チームメンバーによって、ユーザーストーリーと（まだ定義されていない場合は）その受け入れ基準を分析、議論し、記述する。ユーザーストーリーの不完全さ、曖昧さ、または欠陥は、このプロセス中に解決する。次のステップは、テストケースを作成することである。これは、チーム全体で行うことも、テスト担当者が個別に行うこともできる。テストケースは受け入れ基準に基づいており、ソフトウェアがどのように動作するかの例示として見ることができる。これは、チームがユーザーストーリーを正しく実装するのに役立つ。

先の例示とテストケースは同じであるため、これらの用語はしばしば互換的に使用される。テスト設計では、4.2 節、4.3 節、4.4 節に記載しているテスト技法を適用することができる。

典型的には、最初のテストケースはポジティブなもので、例外やエラー条件のない正しい動作を確認し、すべてが期待通りに進んだ場合に実行される一連の活動を構成する。ポジティブなテストケースが終了したら、チームはネガティブなテストを実施すべきである。最後に、チームは非機能的な品質特性もカバーすべきである（例えば、性能効率性、使用性など）。テストケースは、ステークホルダーが理解できるように表現すべきである。典型的に、テストケースは、必要な事前条件（もしあれば）、入力、事後条件を含む自然言語の文章が含まれる。

テストケースはユーザーストーリーの特性をすべてカバーすべきであるが、ストーリーを越えるべきではない。しかし、受け入れ基準では、ユーザーストーリーで示している重要な論点の一部を詳細に記述できる。また、2 つのテストケースにユーザーストーリーの同じ特性を記述すべきではない。

テスト自動化フレームワークでサポートしている形式でキャプチャした場合、ユーザーストーリーで記述しているフィーチャーを実装する際に、開発者がサポートコードを記述することで、テストケースを自動化することができる。このようにして受け入れテストは実行可能な要件となる。

5 テスト活動のマネジメント | 335 分

キーワード

欠陥マネジメント、欠陥レポート、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、リスク分析、リスクアセスメント、リスクコントロール、リスク識別、リスクレベル、リスクマネジメント、リスク軽減、リスクモニタリング、リスクベースドテスト、テストアプローチ、テスト完了レポート、テストコントロール、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、テストピラミッド、テストの四象限

第 5 章の学習の目的

5.1 テスト計画

FL-5.1.1 （K2）テスト計画書の目的と内容を例示する。

FL-5.1.2 （K1）テスト担当者がイテレーション計画やリリース計画にどのような価値を付加するかを認識する。

FL-5.1.3 （K2）開始基準と終了基準を比較、そして対比する。

FL-5.1.4 （K3）見積り技法を用いて、必要なテスト工数を算出する。

FL-5.1.5 （K3）テストケースの優先順位付けを適用する。

FL-5.1.6 （K1）テストピラミッドの概念を想起する。

FL-5.1.7 （K2）テストの四象限とテストレベルやテストタイプとの関係を要約する。

5.2 リスクマネジメント

FL-5.2.1 （K1）リスクの可能性とリスクの影響を用いて、リスクレベルを識別する。

FL-5.2.2 （K2）プロジェクトリスクとプロダクトリスクを区別する。

FL-5.2.3 （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を説明する。

FL-5.2.4 （K2）分析したプロダクトリスクに対して、どのような手段で対応できるかを説明する。

5.3 テストモニタリング、テストコントロールとテスト完了

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。

FL-5.3.3 （K2）テストの状況をどのように伝えるかを例示する。

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 欠陥マネジメント

FL-5.5.1 （K3）欠陥レポートを準備する。

5.1 テスト計画

5.1.1 テスト計画書の目的と内容

テスト計画書は、テストプロジェクトの目的、リソース、プロセスを表す。

- テスト計画書は、テスト目的を達成するための手段やスケジュールを文書化する
- テスト計画書は、実施したテスト活動が、確立した基準を満たすことを保証するのに役立つ
- テスト計画書は、チームメンバーやステークホルダーとのコミュニケーション手段としての役割を担う
- テスト計画書は、テストが既存のテストポリシーやテスト戦略を遵守することを示す（またはテストがそれらから逸脱する理由を説明する）

テスト計画は、テスト担当者の思考をガイドし、リスク、スケジュール、人員、ツール、コスト、労力などに関する将来の課題にテスト担当者が向かい合うよう促す。

テスト計画書を準備するプロセスは、テストプロジェクトの目的を達成するために必要な取り組みを検討するのに有効な方法である。

典型的なテスト計画書の内容は以下の通りである：

- テストの概要（例えば、スコープ、テスト対象、制約、テストベース）
- テストプロジェクトの前提と制約
- ステークホルダー（例えば、役割、責務、テストとの関連性、採用、必要なトレーニング）
- コミュニケーション（例えば、コミュニケーションの形態や頻度、ドキュメントのテンプレートなど）
- リスクレジスター（プロダクトリスク、プロジェクトリスクなど）
- テストアプローチ（例：テストレベル、テストタイプ、テスト技法、テスト成果物、開始基準と終了基準、テストの独立性、収集すべきメトリック、テストデータ要件、テスト環境要件、組織のテストポリシーやテスト戦略からの逸脱など）
- 予算とスケジュール

テスト計画書とその内容の詳細については、ISO/IEC/IEEE 29119-3 標準で確認できる。

5.1.2 イテレーションとリリース計画に対するテスト担当者の貢献

イテレーティブな SDLC では、典型的に、リリース計画とイテレーション計画の 2 種類の計画を行う。

リリース計画では、プロダクトのリリースを見据えて、プロダクトバックログの定義と再定義を行い、大きなユーザーストーリーを一連の小さなユーザーストーリーへと洗練していくこともある。また、すべてのイテレーションにおける、テストアプローチとテスト計画書のベースとなる。リリース計画に関わるテスト担当者は、テスト可能なユーザーストーリーと受け入れ基準の作成（4.5 節参照）、プロジェクトと品質のリスク分析（5.2 節参照）、ユーザーストーリーに関連するテスト工数の見積り（5.1.4 項参照）、テストアプローチの決定と、リリースのためのテストを計画する。

イテレーション計画は、1 回のイテレーションの終わりを見据えて、イテレーションバックログを考慮していく。イテレーション計画に関わるテスト担当者は、ユーザーストーリーの詳細なリスク分析に参加し、ユーザーストーリーの試験性を判断し、ユーザーストーリーをタスク（特にテストタスク）に分解し、すべてのテストタスクのテスト工数を見積り、テスト対象の機能的側面と非機能的側面を識別し、洗練する。

5.1.3 開始基準と終了基準

開始基準では、ある活動を行うための事前条件を定義する。開始基準が満たされていない場合、その活動は結果的により困難で、時間がかかり、コストがかかり、リスクが高くなる可能性がある。終了基準では、活動の完了を宣言するために何を達成しなければならないかを定義する。開始基準と終了基準は、テストレベルごとに定義するべきであり、テスト対象によって異なる。

典型的な開始基準としては、リソースの可用性（例：人、ツール、環境、テストデータ、予算、時間）、テストウェアの可用性（例：テストベース、試験性が保たれた要件、ユーザーストーリー、テストケース）、テスト対象の初期品質レベル（例：すべてのスモークテストが合格）、などがある。

典型的な終了基準には、徹底度の測定（達成されたカバレッジレベル、未解決欠陥の数、欠陥密度、失敗したテストケースの数など）、および完了基準（計画したテストが実行された、静的テストを実行した、発見したすべての欠陥がレポートされた、すべてのリグレッションテストが自動化されたなど）などがある。

時間切れや予算切れも、終了基準として妥当とみなすことがある。他の終了基準が満たされていなくても、ステークホルダーがこれ以上のテストを行わずに本稼働させるリスクをレビューし、受け入れたのであれば、このような状況でテストを終了させることは許容される。

アジャイルソフトウェア開発では、終了基準を完了（done）の定義と呼ぶことが多く、リリース可能なアイテムに対するチームの客観的なメトリックを定義する。ユーザーストーリーが開発および/またはテスト活動を開始するために満たさなければならない開始基準は、準備完了（Ready）の定義と呼ぶ。

5.1.4 見積り技法

テスト工数の見積りは、テストプロジェクトの目的を達成するために必要なテスト関連作業の量を予測することである。見積りは多くの仮定に基づいており、常に見積り誤差の可能性があることを関係者に明示することが重要である。小さなタスクの見積りは、通常、大きなタスクの見積りよりも正確である。したがって、大きなタスクを見積もるときは、それを小さなタスクの集合に分解し、それを順番に見積もることができる。

本シラバスでは、以下の 4 つの見積り技法について説明する。

比率に基づく見積り　このメトリクスベースの手法では、この比率は、組織内の過去のプロジェクトから収集されたものであり、類似プロジェクトの「標準」比率を導き出すことが可能である。一般的に、組織内のプロジェクトの比率（例えば、過去のデータから取得）は、見積りプロセスで使用する最もよいソースである。これらの標準的な比率は、新規プロジェクトのテスト工数を見積もるために使用することができる。例えば、前のプロジェクトで開発工数とテスト工数の比率が 3：2 であり、今回のプロジェクトで開発工数が 600 人日と予想される場合、テスト工数は 400 人日と見積もることができる。

外挿　このメトリクスベースの技術では、測定は、データを収集するために、現在のプロジェクトのできるだけ早い時期に行われる。十分な観測値があれば、残りの作業に必要な工数は、このデータを外挿することで概算することができる（通常は数学モデルを適用する）。この方法は、イテレーティブな SDLC に非常に適している。例えば、チームは、次のイテレーションにおけるテスト工数を、過去 3 回のイテレーションから平均した工数として外挿することができる。

ワイドバンドデルファイ　この繰り返しによる、専門家による手法では、専門家が経験に基づいて推定する。各専門家は、単独で、労力を見積もる。その結果を収集し、合意した境界線から外れた偏差がある場合、専門家は現在の見積りについて議論する。そして、各専門家は、そのフィードバックに基づき、再び単独で新たな見積りを行うよう求められる。このプロセスは、コンセンサスが得られるまで繰

り返される。プランニングポーカーは、アジャイルソフトウェア開発でよく使われるワイドバンドデルファイのバリエーションの 1 つである。プランニングポーカーでは、通常、労力のサイズを表す数字が書かれたカードを使って見積りが行われる。

三点見積り　このエキスパートベースの技術では最も楽観的な推定値（a）、最も可能性の高い推定値（m）、最も悲観的な推定値（b）の 3 つの推定値を専門家が作成する。最終的な推定値(E)は、それらの加重算術平均である。この手法の最も一般的なバージョンでは、推定値は E=(a+4*m+b)/6 として計算される。この手法の利点は、専門家が測定誤差を計算することができることである：SD= (b-a)/6。例えば、見積り（人時）が a=6、m=9、b=18 の場合、E=(6+4*9+18)/6=10、SD=(18-6)/6=2 なので、最終見積りは 10±2 人時（つまり、8 人時～12 人時）であるといえる。

これらのテスト見積り技法やその他多くのテスト見積り技法については、（Kan 2003, Koomen 2006, Westfall 2009）を参照されたい。

5.1.5 テストケースの優先順位付け

テストケースとテストプロシジャーを仕様化し、テストスイートとしてまとめたら、それらのテストスイートは、テスト実行の順番を決めている、テスト実行スケジュールの中に配置することができる。テストケースの優先順位を付ける場合、さまざまな要因が考慮できる。最も一般的に使用されるテストケースの優先順位付け戦略は、以下の通りである：

- リスクに基づく優先順位付け。ここで、テスト実行の順序は、リスク分析の結果（5.2.3 項参照）に基づく。最も重要なリスクをカバーするテストケースは、最初に実行される。
- カバレッジベースの優先順位付け。ここで、テスト実行の順序は、カバレッジ（ステートメントカバレッジなど）に基づいている。最も高いカバレッジを達成するテストケースを最初に実行する。追加カバレッジの優先順位付けと呼ばれる別の方法では、最も高いカバレッジを達成するテストケースが最初に実行され、後続の各テストケースは最も高い追加カバレッジを達成するテストケースとなる。
- 要件に基づく優先順位付け。テスト実行の順序は、対応するテストケースに遡った要件の優先度に基づいている。要件の優先度は、ステークホルダーが定義する。最も重要な要件に関連するテストケースは、最初に実行する。

理想的には、例えば、上記の優先順位付け戦略の 1 つを使用するといったように、テストケースは優先度に基づいて実行する。しかし、テストケース、またはテストする機能に依存関係がある場合、このやり方はうまくいかないかもしれない。優先度の高いテストケースが優先度の低いテストケースに依存している場合、優先度の低いテストケースを先に実行しなければならない。

テスト実行の順序は、リソースの可用性も考慮する必要がある。例えば、必要なテストツール、テスト環境、または特定の時間帯にしか利用できない可能性のある人々などである。

5.1.6 テストピラミッド

テストピラミッドは、異なるテストによって、粒度が異なる可能性があることを示すモデルである。テストピラミッドモデルは、さまざまなゴールをさまざまなレベルのテスト自動化が支援することを示すことで、テスト自動化とテスト工数配分においてチームを支援する。テストピラミッドのレイヤーは、テストのグループを表す。レイヤーが高いほど、テストの粒度が粗くなり、テストの分離が少なくなり、テスト実行時間は遅くなる。最下位レイヤーのテストは、小さく、分離され、速く、機能の一部をチェックするため、通常、妥当なカバレッジを達成するためには多くのテストが必要となる。最上位レイヤーは、複雑でハイレベルのエンドツーエンドテストを表す。このハイレベルのテストは、一般に下位のレイヤーのテストよりも遅く、通常、大きな機能の一部をチェックするため、妥当なカバレッジを達成するためには、通常、ほんの数個のテストのみが必要である。レイヤーの数や名称は異なる場合が

ある。例えば、オリジナルのテストピラミッドモデル（Cohn 2009）では、3 つのレイヤーを定義している：「ユニットテスト」、「サービステスト」、「UI テスト」である。また、ユニット（コンポーネント）テスト、統合（コンポーネント統合）テスト、エンドツーエンドテストを定義する一般的なモデルもある。その他のテストレベル（2.2.1 項参照）も使用可能である。

5.1.7 テストの四象限

Brian Marick（Marick 2003, Crispin 2008）が定義したテストの四象限というモデルは、テストレベルを、アジャイルソフトウェア開発における適切なテストタイプ、活動、テスト技法、作業成果物に分類している。このモデルは、すべての適切なテストタイプとテストレベルが SDLC に含まれることを確実にするために、また、あるテストタイプが他のテストレベルよりも特定のテストレベルに関連していることを理解するために、これらを視覚的に表示するテストマネジメントを支援する。このモデルはまた、開発者、テスト担当者、ビジネス側の代表を含むすべてのステークホルダーに対して、テストのタイプを区別して説明する方法を提供する。

このモデルでは、テストはビジネス向けであったり、技術向けであったりする。また、テストは、チームを支援する（すなわち、開発を導く）ことも、プロダクトを批評する（すなわち、期待に反する動作を測定する）こともできる。これら 2 つの視点の組み合わせにより、4 つの象限を決定する：

- 第一象限（テクノロジー指向、チームを支援する）　この象限には、コンポーネントテストとコンポーネント統合テストを含む。これらのテストは自動化して CI プロセスに含めるべきである。
- 第二象限（ビジネス指向、チームを支援する）　この象限には、機能テスト、実例、ユーザーストーリーテスト、ユーザーエクスペリエンスプロトタイプ、API テスト、シミュレーションを含む。これらのテストは受け入れ基準をチェックするもので、手動または自動のいずれでもよい。
- 第三象限（ビジネス指向、プロダクトを批評する）　この象限は、探索的テスト、使用性テスト、ユーザー受け入れテストを含む。これらのテストはユーザー志向であり、多くの場合手動で行う。
- 第四象限（テクノロジー指向、プロダクトを批評する）　この象限にはスモークテストと非機能テスト（使用性テストは除く）を含む。これらのテストは、しばしば自動化する。

5.2 リスクマネジメント

組織は多くの内的、そして外的要因に直面し、目的を達成できるかどうか、またいつ達成できるかは不確実である（ISO 31000）。リスクマネジメントは、組織が目的を達成する可能性を高め、プロダクトの品質を向上させ、ステークホルダーの信頼と信用を高めることを可能にする。

主なリスクマネジメント活動は以下の通り：

- リスク分析（リスク識別とリスクアセスメントから構成、5.2.3 項参照）
- リスクコントロール（リスク軽減とリスクモニタリングから構成、5.2.4 項参照）

リスク分析とリスクコントロールに基づいてテスト活動を選択し、優先順位を付け、マネジメントしていくテストアプローチをリスクベースドテストという

5.2.1 リスク定義とリスク属性

リスクとは、それらが顕在化することによって悪影響が生じる、潜在的な事象、ハザード、脅威、または状況のことである。リスクは、2 つの要素によって特徴づけられる：

- リスクの可能性-リスクが顕在化する確率（0 より大きく 1 より小さい）

- リスクの影響（損害）-この顕在化がもたらす結果

これら 2 つの要素は、リスクの尺度であるリスクレベルを表す。リスクレベルが高いほど、その処置はより重要である。

5.2.2 プロジェクトリスクとプロダクトリスク

ソフトウェアテストでは、一般的にプロジェクトリスクとプロダクトリスクの2つのタイプのリスクに関心を持つことが多い。
プロジェクトリスクはプロジェクトのマネジメントや統制に関連するものである。プロジェクトリスクには、以下のようなものがある：

- 組織的な問題（作業成果物の納期遅れ、不正確な見積り、コストダウンなど）
- 人の問題（例：スキル不足、対立、コミュニケーションの問題、人手不足など）
- 技術的な問題（スコープクリープ、ツールサポートの不備など）
- サプライヤーの問題（例：第三者による納品の失敗、サポート企業の倒産など）

プロジェクトリスクは、顕在化した場合、プロジェクトのスケジュール、予算、スコープに影響を与え、プロジェクトの目的を達成する能力に影響を与える可能性がある。

プロダクトリスクはプロダクトの品質特性（例えば、ISO 25010 品質モデルに記述がある）に関連するものである。プロダクトリスクの例としては、機能の不足や誤り、誤った計算、ランタイムエラー、貧弱なアーキテクチャー、非効率なアルゴリズム、不十分な応答時間、貧弱なユーザーエクスペリエンス、セキュリティの脆弱性などが挙げられる。プロダクトリスクが顕在化した場合、以下のようなさまざまな負の結果をもたらす可能性がある：

- ユーザーの不満足
- 収益、信頼、評判の損失
- 第三者への損害賠償
- メンテナンスコストが高い、ヘルプデスクに負荷がかかる
- 刑事罰
- 極端な場合、身体的な損傷、重傷、または死に至ることもある

5.2.3 プロダクトリスク分析

テストの観点からは、プロダクトリスク分析のゴールは、プロダクトリスクの残存レベルを最小化する方法にテスト工数を集中させるために、プロダクトリスクに対する認識を提供することである。理想的には、プロダクトリスク分析は、SDLC の初期に始まる。

プロダクトリスク分析はリスク識別とリスクアセスメントで構成する。リスク識別は、リスクの包括的なリストを作成することである。ステークホルダーは、ブレーンストーミング、ワークショップ、インタビュー、特性要因図など、さまざまな手法やツールを用いてリスクを識別することができる。リスクアセスメントでは、識別したリスクを分類し、リスクの可能性、リスクの影響、レベルを決定し、優先順位を付け、対処法を提案する。通常、同じカテゴリーに属するリスクは、同様のアプローチで軽減できるため、カテゴリー分けは軽減策の割り当てに役立つ。

リスクアセスメントには、定量的アプローチと定性的アプローチ、あるいはそれらの混合がある。定量的アプローチでは、リスクレベルはリスクの可能性とリスクの影響の乗算として算出される。定性的アプローチでは、リスクレベルはリスクマトリクスを使用して決定できる。

プロダクトリスク分析は、テストの深さと範囲に影響を与える可能性がある。その結果は、以下のように使用する：

- 実施するテスト範囲の決定
- 該当するテストレベルの決定と、実施するテストタイプの提案
- 採用するテスト技法と達成すべきカバレッジの決定
- 各タスクに必要なテスト工数の見積り
- 重要な欠陥をできるだけ早く発見するためのテストの優先順位づけ
- リスクを低減するためにテストに加えて何らかの活動を採用できるかどうかを判断

5.2.4 プロダクトリスクコントロール

プロダクトリスクコントロールは、識別し評価したプロダクトリスクに対応するすべての手段から成る。プロダクトリスクコントロールは、リスク軽減とリスクモニタリングで構成する。リスク軽減には、リスクアセスメントで提案された措置を実施し、リスクレベルを低減させることを含む。リスクモニタリングの目的は、リスク軽減措置が効果的であることを確認し、リスクアセスメントを改善するためにさらなる情報を入手し、新たなリスクを識別することである。

プロダクトリスクコントロールに関しては、リスクを分析すると、テストによるリスク軽減、リスク受け入れ、リスク移転、コンティンジェンシー計画など、リスクに対するいくつかの対応策が考えられる（Veenendaal 2012）。テストによるプロダクトリスクの軽減のために取り得る行動は、以下の通りである：

- リスクタイプに適した経験やスキルのレベルを持つテスト担当者を選ぶ
- 適切なレベルのテストの独立性を適用する
- レビューの実施と静的解析の実施
- 適切なテスト技法とカバレッジレベルの適用
- 影響を受ける品質特性に対応した適切なテストタイプを適用する
- リグレッションテストを含む動的テストの実施

5.3 テストモニタリング、テストコントロールとテスト完了

テストモニタリングでは、テストに関する情報を収集することが関心ごとである。この情報は、テストの進捗を評価し、プロダクトリスク、要件、受け入れ基準などで定めたカバレッジを満たすなど、テスト終了基準または終了基準に関連するテストタスクを満たしているかを測定するために使用する。

テストコントロールは、テストモニタリングからの情報を用いて、最も効果的かつ効率的なテストを達成するためのガイドと必要な補正のアクションを、コントロールのための指示の形で提供する。

コントロールのための指示の例としては、以下のものがある：

- 識別したリスクが問題になった場合のテストの優先度の見直し
- 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
- テスト環境の提供の遅れに対応するためのテストスケジュールの調整
- 必要なときに必要な箇所へ新しいリソースを追加

テスト完了は、完了したテスト活動のデータを収集し、経験、テストウェア、その他の関連情報を統合する。テスト完了活動は、テストレベルの完了、アジャイルイテレーションの終了、テストプロジェクトの完了（またはキャンセル）、ソフトウェアシステムのリリース、メンテナンスリリースの完了など、プロジェクトのマイルストーンで発生する。

5.3.1 テストで使用するメトリクス

テストメトリクスは、計画したスケジュールや予算に対する進捗状況、テスト対象の現在の品質、テスト目的やイテレーションゴールに対するテスト活動の有効性を示すために収集する。テストモニタリングでは、テストコントロールとテスト完了をサポートするために、さまざまなメトリクスを収集する。

代表的なテストメトリクスには、以下のようなものがある：

- プロジェクト進捗メトリクス（例えば、タスク完了、リソース使用量、テスト工数）
- テスト進捗メトリクス（例：テストケースの実装進捗、テスト環境の準備進捗、テストケースの実行/未実行、合格/失敗の数、テスト実行時間など）
- プロダクト品質メトリクス（例：可用性、応答時間、平均故障までの時間など）
- 欠陥メトリクス（例：発見、修正された欠陥の数と優先度、欠陥密度、欠陥検出率）
- リスクメトリクス（例：残留リスクレベル）
- カバレッジメトリクス（例：要件カバレッジ、コードカバレッジ）
- コストメトリクス（例：テストコスト、品質に対する組織的なコスト）

5.3.2 テストレポートの目的、内容、読み手

テストレポート作業は、テスト中およびテスト後のテスト情報を要約し、伝達するものである。テスト進捗レポートは、テストの継続的なコントロールを支援し、テスト計画書からの逸脱や状況の変化により、テストスケジュール、リソース、またはテスト計画の修正が必要な場合に、十分な情報を提供しなければならない。テスト完了レポートは、テストの特定の段階（テストレベル、テストサイクル、イテレーションなど）を要約し、その後のテストのための情報を提供することができる。

テストモニタリング作業とコントロールの間、テストチームは、ステークホルダーに情報を提供するために、テスト進捗レポートを作成する。テスト進捗レポートは通常、定期的に（毎日、毎週など）作成し、以下の内容を含む：

- テスト期間
- テストの進捗状況（例：テストスケジュールの前倒し、遅れ）、顕著な逸脱を含む
- テストに支障をきたすもの、およびその回避策
- テストメトリクス（例については 5.3.1 項参照）
- テスト期間中に新たに発生したリスクと変更されたリスク
- 次の期間でのテスト計画書

テスト完了レポートは、テスト完了時に作成する。ここでいうテスト完了時とは、プロジェクト、テストレベル、またはテストタイプが完了し、それらの終了基準を満たしたときが理想的である。このレポートには、テスト進捗レポートやその他のデータを使用する。典型的なテスト完了レポートには、以下のものがある：

- テスト概要

- 当初のテスト計画書に基づくテストとプロダクト品質の評価（すなわち、テスト目的と終了基準）
- テスト計画書からの逸脱（例：テストスケジュール、期間、労力などの計画との差異）
- テストの阻害要因および回避策
- テスト進捗レポートに基づくテストメトリクス
- 未解決のリスク、修正されなかった欠陥
- テストに関連する学習した教訓

対象者が異なれば、報告書に求める情報も異なり、形式的な度合いや報告の頻度にも影響する。同じチームの他のメンバーへのテスト進捗レポートは、頻繁に行われ、非形式的なものであることが多い。

ISO/IEC/IEEE 29119-3 標準には、テスト進捗レポート（テストステータスレポートと呼ばれる）とテスト完了レポートのテンプレートと例が含まれる。

5.3.3 テストステータスの伝達

テストステータスを伝える最適な手段は、テストマネジメントの懸念、組織のテスト戦略、規制標準、あるいは自己組織化チームの場合はチーム自身（1.5.2 項参照）などによってさまざまである。選択肢としては、以下のようなものがある：

- チームメンバーやその他のステークホルダーとの口頭でのコミュニケーション
- ダッシュボード（例：CI/CD ダッシュボード、タスクボード、バーンダウンチャートなど）
- 電子的なコミュニケーションチャネル（例：電子メール、チャット）
- オンラインドキュメント
- 正式なテストレポート（5.3.2 項参照）

これらの選択肢の 1 つまたは複数を使用することができる。地理的な距離や時差のために直接顔を合わせてのコミュニケーションが必ずしも可能ではない分散型チームでは、より形式的なコミュニケーションが適している場合がある。一般に、ステークホルダーによって関心のある情報は異なるため、コミュニケーションはそれに応じて調整する必要がある。

5.4 構成管理

テストにおいて、構成管理（CM）は、テスト計画書、テスト戦略、テスト条件、テストケース、テストスクリプト、テスト結果記録、テストレポートなどの作業成果物を構成アイテムとして識別、コントロール、トラッキングするための規律を提供する。

複雑な構成アイテム（テスト環境など）については、CM は、構成アイテム、その関係、バージョンなどを記録する。構成アイテムがテスト用に承認された場合、それはベースラインとなり、正式な変更コントロールプロセスを通じてのみ変更することができる。

構成管理では、新しいベースラインを作成すると、変更した構成アイテムの記録を残す。これは、以前のベースラインに戻して、以前のテスト結果を再現することが可能にする。

テストを適切にサポートするために、CM は以下のことを保証する：

- テストアイテム（テスト対象の個々の部分）を含むすべての構成アイテムを一意に識別して、バージョンコントロール行い、変更の追跡を行い、他の構成アイテムと関連付けをして、テストプロセス全体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と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。

継続的インテグレーション、継続的デリバリー、継続的デプロイメント、および関連するテストは、通常、自動化した DevOps パイプライン（2.1.4 項参照）の一部として実装し、その中に自動化している CM が通常含まれる。

5.5 欠陥マネジメント

主要なテスト目的の 1 つは欠陥を検出することであるため、確立した欠陥マネジメントプロセスが不可欠である。ここでは「欠陥」と表現しているが、報告した不正は、本当の欠陥だと判明する場合もあれば、他の何か（例えば、偽陽性、変更要求）の場合もある。不正については、SDLC のどのフェーズでも報告される可能性があり、その形式は SDLC に依存する。プロセスは、関係するすべてのステークホルダーによって遵守されなければならない。最低限、欠陥マネジメントプロセスには、個々の不正を発見から終結まで処理するためのワークフローと、その分類のためのルールが含まれる。ワークフローは通常、報告された不正を記録し、解析し、分類し、修正または現状維持などの適切な対応を決定し、最後に欠陥レポートをクローズする活動で構成している。このプロセスは、関係するすべてのステークホルダーが遵守する必要がある。静的テスト（特に静的解析）の欠陥についても、同様の方法で処理することが望ましい。

代表的な欠陥レポートは、次のような目的がある：

- 報告された欠陥の処理および解決に責任を持つ者に、問題を解決するための十分な情報を提供する
- 作業成果物の品質を追跡する手段を提供する
- 開発・テストプロセス改善のためのアイデアを提供する

動的テスト中に記録される欠陥レポートには、通常、次のような内容を含む：

- 一意な識別子
- 報告される不正の概要を簡潔にまとめたタイトル
- 不正を観測した日付、発行組織、起票者（役割を含む）
- テスト対象およびテスト環境の特定
- 欠陥のコンテキスト（例えば、実行中のテストケース、実行中のテスト活動、SDLC フェーズ、および使用中のテスト技法、チェックリスト、テストデータなどのその他の関連情報）
- 不正を検出したステップ、および/または関連するテスト結果記録、データベースダンプ、スクリーンショット、録画を含む、再現および解決を可能にする故障の説明
- 期待結果と実際の結果
- ステークホルダーの利益や要件に対する欠陥の重要度（影響の度合い）
- 修正する優先度
- 欠陥の状態（例：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、却下）
- 参考文献（テストケースへの言及など）

欠陥マネジメントツールを使用する場合、このデータの一部は自動的に含まれることがある（例えば、識別子、日付、起票者、初期状態など）。欠陥レポートのドキュメントテンプレートと欠陥レポートの

例は、ISO/IEC/IEEE 29119-3 標準で確認できる。この標準では、欠陥レポートはインシデントレポートと呼ぶ。

6 テストツール	20 分

キーワード

テスト自動化

第 6 章の学習の目的

6.1 テストのためのツールによる支援

FL-6.1.1 （K2）さまざまな種類のテストツールがどうテストを支援するかを説明する。

6.2 テスト自動化の利点とリスク

FL-6.2.1 （K1）テスト自動化の利点とリスクを想起する。

6.1 テストのためのツールによる支援

テストツールによる支援は、多くのテスト活動を促進する。例えば、以下のようなものがあるが、これらに限定されるものではない：

- マネジメントツール-SDLC、要件、テスト、欠陥、構成のマネジメントを容易にすることで、テストプロセスの効率性が向上する
- 静的テストツール-レビューや静的解析を行うテスト担当者をサポートする
- テスト設計および実装ツール-テストケース、テストデータ、テストプロシジャーの作成を促進する
- テスト実行とカバレッジツール-テスト実行の自動化とカバレッジを容易にする
- 非機能テストツール-テスト担当者が手動で行うことが困難、または不可能な非機能テストを行うことができる
- DevOps ツール-DevOps デリバリーパイプライン、ワークフロー追跡、自動ビルドプロセス、CI/CD のサポート
- コラボレーションツール-コミュニケーションを円滑にする
- 拡張性/展開の標準化をサポートするツール（仮想マシンなど、コンテナ化ツール）
- その他、テストを補助するツール（例：表計算ソフトはテストのコンテキストではテストツールになる）

6.2 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。新たなツールを導入する際には、現実的かつ持続的な利益を得るための労力が必要となる（例えば、ツールの導入、メンテナンス、トレーニング）。また、分析と軽減が必要なリスクも存在する。

テスト自動化の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、同じテストデータの再入力、期待結果と実際の結果の比較、コーディング標準との適合など）
- 一貫性や再実行性が向上による単純なヒューマンエラーを防止する（例えば、テストは一貫して要件から導き出され、テストデータは体系的な方法で作成され、テストはツールによって同じ順序で同じ頻度で実行される）
- 評価の客観性が向上し（例えば、カバレッジなど）、人間が導き出すには複雑すぎる尺度の提供をする
- テストマネジメントおよびテスト報告を支援するための、テストに関する情報へのアクセスを容易にする（例えば、統計、グラフ、およびテスト進捗、欠陥率、テスト実行期間に関する集計データ）
- テスト実行時間の短縮により、不具合の早期発見、迅速なフィードバック、市場投入までの時間の短縮を実現する
- テスト担当者が、より深く、より効果的な新しいテストを設計する時間を増やす

テスト自動化の潜在的なリスクは以下の通りである。

- テストツールの利点に非現実的な期待をする（例えば、ツールの機能や使いやすさなど）
- ツールの導入、テストスクリプトの保守、既存の手動テストプロセスの変更に必要な時間、コスト、労力の見積りが正確でない
- 手動テストがより適切である場合に、テストツールを使用する

- ツールに過剰な依存をする、例えば人の批判的思考の必要性を無視する
- ツールベンダーがビジネスを廃業したり、別のベンダーにツールを売ったり、サポート（問い合わせへの対応、アップグレード、不具合修正など）が不十分であったりする可能性がある
- オープンソースソフトウェアの使用は、放置された時にそれ以降のアップデートが利用できないことを意味することになるか、その内部コンポーネントが、以降の開発にて、かなり頻繁なアップデートを必要とする可能性がある
- 自動化ツールが開発プラットフォームと互換性がない
- 規制要件や安全基準に適合してない不適切なツールを選択する

7 参考文献

標準ドキュメント

ISO/IEC/IEEE 29119-1 (2022) Software and systems engineering - Software testing - Part 1: Concepts and definitions

ISO/IEC/IEEE 29119-2 (2021) Software and systems engineering - Software testing - Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2021) Software and systems engineering - Software testing - Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2021) Software and systems engineering - Software testing - Part 4: Test techniques

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 (日本では JIS X 20246)

ISO/IEC/IEEE 14764:2022 - Software engineering - Software life cycle processes - Maintenance

ISO 31000 (2018) Risk management - Principles and guidelines

書籍（日本語翻訳が出版されているものは追記している）

Adzic, G. (2009) Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing, Neuri Limited

Ammann, P. and Offutt, J. (2016) Introduction to Software Testing (2e), Cambridge University Press

Andrews, M. and Whittaker, J. (2006) How to Break Web Software: Functional and Security Testing of Web Applications and Web Services, Addison-Wesley Professional

Beck, K. (2003) Test Driven Development: By Example, Addison-Wesley 「テスト駆動開発, オーム社」

Beizer, B. (1990*) Software Testing Techniques (2e)*, Van Nostrand Reinhold: Boston MA 「ソフトウェアテスト技法,日経 BP」

Boehm, B. (1981) Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ

Buxton, J.N. and Randell B., eds (1970), Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27-31 October 1969, p. 16

Chelimsky, D. et al. (2010) The Rspec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends, The Pragmatic Bookshelf: Raleigh, NC 「The RSpec Book, 翔泳社」

Cohn, M. (2009) Succeeding with Agile: Software Development Using Scrum, Addison-Wesley

Copeland, L. (2004) *A Practitioner’s Guide to Software Test Design*, Artech House: Norwood MA 「はじめて学ぶソフトウェアのテスト技法,日経 BP」

Craig, R. and Jaskiel, S. (2002) *Systematic Software Testing*, Artech House: Norwood MA 「体系的ソフトウェアテスト入門,日経 BP」

Crispin, L. and Gregory, J. (2008) *Agile Testing*: A Practical Guide for Testers and Agile Teams, Pearson Education: Boston MA 「実践アジャイルテスト テスターとアジャイルチームのための実践ガイド,翔泳社」

Forgács, I., and Kovács, A. (2019) Practical Test Design: Selection of traditional and automated test design techniques, BCS, The Chartered Institute for IT

Gawande A. (2009) The Checklist Manifesto: How to Get Things Right, New York, NY: Metropolitan Books 「アナタはなぜチェックリストを使わないのか?, 晋遊舎」

Gärtner, M. (2011), ATDD by Example: A Practical Guide to Acceptance Test-Driven Development, Pearson Education: Boston MA

Gilb, T. and Graham, D. (1993) *Software Inspection*, Addison Wesley: Reading MA 「ソフトウェアインスペクション,構造計画研究所」

Hendrickson, E. (2013) Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing, The Pragmatic Programmers

Hetzel, B. (1988) The Complete Guide to Software Testing, 2nd ed., John Wiley and Sons

Jeffries, R., Anderson, A., Hendrickson, C. (2000) Extreme Programming Installed, Addison-Wesley Professional 「XP エクストリーム・プログラミング導入編, ピアソン・エデュケーション」

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

Kan, S. (2003) Metrics and Models in Software Quality Engineering, 2nd ed., Addison-Wesley 「ソフトウェア品質工学の尺度とモデル, 共立出版」

Kaner, C., Falk, J., and Nguyen, H.Q. (1999) Testing Computer Software, 2nd ed., Wiley 「基本から学ぶソフトウェアテスト, 日経 BP」

Kaner, C., Bach, J. and Pettichord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons: New York NY 「ソフトウェアテスト 293 の鉄則,日経 BP」

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

Kim, G., Humble, J., Debois, P. and Willis, J. (2016) The DevOps Handbook, Portland, OR 「The DevOps ハンドブック 理論・原則・実践のすべて, 日経 BP」

Koomen, T., van der Aalst, L., Broekman, B. and Vroon, M. (2006) TMap Next for result-driven testing, UTN Publishers, The Netherlands

Myers, G. (2011) *The Art of Software Testing*, (3e), John Wiley & Sons: New York NY 「ソフトウェア・テストの技法 第 2 版,近代科学社」

O'Regan, G. (2019) Concise Guide to Software Testing, Springer Nature Switzerland

Pressman, R.S. (2019) Software Engineering. A Practitioner's Approach, 9th ed., McGraw Hill 「実践ソフトウェアエンジニアリング(第 9 版), オーム社」

Roman, A. (2018) Thinking-Driven Testing. The Most Reasonable Approach to Quality Control, Springer Nature Switzerland

Van Veenendaal, E (ed.) (2012) Practical Risk-Based Testing, The PRISMA Approach, UTN Publishers: The Netherlands

Watson, A.H., Wallace, D.R. and McCabe, T.J. (1996) Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, U.S. Dept. of Commerce, Technology Administration, NIST

Westfall, L. (2009) The Certified Software Quality Engineer Handbook, ASQ Quality Press

Whittaker, J. (2002) How to Break Software: A Practical Guide to Testing, Pearson

Whittaker, J. (2009) Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design, Addison Wesley

Wiegers, K. (2002) *Peer Reviews in Software*, Pearson Education: Boston MA 「ピアレビュー,日経BP」

記事と Web ページ他の資料（本シラバスでの直接の参照はない）

Brykczynski, B. (1999) “A survey of software inspection checklists,” *ACM SIGSOFT Software Engineering Notes, 24(1), pp. 82-89*

Enders, A. (1975) “An Analysis of Errors and Their Causes in System Programs,” *IEEE Transactions on Software Engineering 1(2), pp. 140-149*

Manna, Z., Waldinger, R. (1978) “The logic of computer programming,” *IEEE Transactions on Software Engineering 4(3), pp. 199-229*

Marick, B. (2003) Exploration through Example, http://www.exampler.com/old-blog/2003/08/21.1.html#agile-testing-project-1

Nielsen, J. (1994) “Enhancing the explanatory power of usability heuristics,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, ACM Press, pp. 152-158*

Salman. I. (2016) “Cognitive biases in software quality and testing,” *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16), ACM, pp. 823-826.*

Wake, B. (2003) “INVEST in Good Stories, and SMART Tasks,” https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

8 付録 A - 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トピックを試験する。学習の目的は、以下にリストされているように、知識の認知レベルに対応する動詞で記載する。

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。

動作動詞：識別する（identify）、記憶する（remember）、想起する（recall）、認識する（recognize）

例：

- 「典型的なテスト目的を識別する。」
- 「テストピラミッドの概念を想起する。」
- 「テスト担当者がイテレーション計画やリリース計画にどのような価値を付加するかを認識する。」

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、例の提示を行うことができる。

動作動詞：分類する（classify）、比較する（compare）、対比する（contrast）、区別する（differentiate）、区別する（distinguish）、例示する（exemplify）、説明する（explain）、例を挙げる（give examples）、解釈する（interpret）、要約する（summarize）

例：

- 受け入れ基準を書くためのさまざまな選択肢を分類する。
- テストにおけるさまざまな役割を比較する。（類似点、相違点、または両方を探す）
- プロジェクトリスクとプロダクトリスクを区別する。（概念を区別することができる）
- テスト計画書の目的と内容を例示する。
- テストを取り巻くコンテキストがテストプロセスに与える影響を説明する。
- レビュープロセスの活動を要約する。

レベル 3：適用レベル（K3）

精通したタスクに直面した際にプロシジャーに沿って実施、もしくはプロシジャー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。

動作動詞：適用する（apply）、実装する（implement）、準備する（prepare）、使用する（use）

例：

- テストケースの優先順位付けを適用する。（プロシジャー、技法、プロセス、アルゴリズムなどを参照すべきである）
- 欠陥レポートを準備する。
- テストケースを導出するために境界値分析を使用する。

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

9 付録 B ビジネス成果と学習の目的のトレーサビリティマトリクス

この節では、ビジネス成果に関連する Foundation Level の学習の目的の数、および Foundation Level のビジネス成果と Foundation Level の学習の目的の間のトレーサビリティを列挙する。

ビジネス成果：Foundation Level		FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
BO1	テストとは何か、なぜテストが有効なのかを理解する。	6													
BO2	ソフトウェアテストの基本的な概念を理解する。		22												
BO3	テストのコンテキストに応じて、実施すべきテストアプローチと活動を識別する。			6											
BO4	ドキュメントの品質を評価し、改善する。				9										
BO5	テストの有効性と効率性を向上する。					20									
BO6	テストプロセスとソフトウェア開発ライフサイクルを一致させる。						6								
BO7	テストマネジメントの原則を理解する。							6							
BO8	明確で理解しやすい欠陥レポートを記述して伝える。								1						
BO9	テストに関わる優先度や労力に影響を与える要因を理解する。									7					
BO10	クロスファンクショナルチームの一員として働く。										8				
BO11	テスト自動化に関するリスクと利点を知る。											1			
BO12	テストを行うために求められる必要不可欠なスキルを識別する。												5		
BO13	リスクがテストに与える影響を理解する。													4	
BO14	テスト進捗と品質を効果的にレポートする。														4

章/節/項	学習の目的	K レベル	ビジネス成果													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
第 1 章	**テストの基礎**															
1.1	**テストとは何か？**															
1.1.1	典型的なテスト目的を識別する。	K1	X													
1.1.2	デバッグとテストを区別する。	K2		X												
1.2	**なぜテストが必要か？**															
1.2.1	テストが必要な理由を例示する。	K2	X													
1.2.2	テストと品質保証の関係を想起する。	K1		X												
1.2.3	根本原因、エラー、欠陥、故障を区別する。	K2		X												
1.3	**テストの原則**															
1.3.1	テストにおける 7 原則を説明する。	K2		X												
1.4	**テスト活動、テストウェア、そして役割**															
1.4.1	さまざまなテスト活動とタスクを要約する。	K2			X											
1.4.2	テストを取り巻くコンテキストがテストプロセスに与える影響を説明する。	K2			X			X								
1.4.3	テスト活動を支援する多くのテストウェアを区別する。	K2			X											
1.4.4	トレーサビリティを維持することの価値を説明する。	K2				X	X									
1.4.5	テストにおけるさまざまな役割を比較する。	K2										X				
1.5	**テストに必要不可欠なスキルとよい実践例**															

1.5.1	テストを行うために求められる一般的なスキルについて例を挙げる。	K2											X		
1.5.2	チーム全体アプローチの利点を想起する。	K1									X				
1.5.3	テストの独立性の長所と短所を区別する。	K2			X										
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**														
2.1	**コンテキストに応じたソフトウェア開発ライフサイクルでのテスト**														
2.1.1	選択したソフトウェア開発ライフサイクルがテストに与える影響を説明する。	K2					X								
2.1.2	全ソフトウェア開発ライフサイクルに適用するよいテストの実践例を想起する。	K1					X								
2.1.3	開発でのテストファーストアプローチの事例を想起する。	K1				X									
2.1.4	DevOps がテストに与える影響について要約する。	K2				X	X			X	X				
2.1.5	シフトレフトアプローチを説明する。	K2				X	X								
2.1.6	プロセス改善の仕組みとして、ふりかえりをどのように利用できるかを説明する。	K2				X					X				
2.2	**テストレベルとテストタイプ**														
2.2.1	さまざまなテストレベルを区別する。	K2		X	X										
2.2.2	さまざまなテストタイプを区別する。	K2		X											
2.2.3	確認テストとリグレッションテストを区別する。	K2		X											
2.3	**メンテナンス（保守）テスト**														
2.3.1	メンテナンステストとそのきっかけを要約する。	K2		X				X							
第 3 章	**静的テスト**														
3.1	**静的テストの基本**														

3.1.1	さまざまな静的テスト技法によって確認できるプロダクトの種類を認識する。	K1				X	X									
3.1.2	静的テストの価値を説明する。	K2	X			X	X									
3.1.3	静的テストと動的テストを比較、そして対比する。	K2				X	X									
3.2	**フィードバックとレビュープロセス**															
3.2.1	早期かつ頻繁なステークホルダーからのフィードバックの利点を識別する。	K1	X			X						X				
3.2.2	レビュープロセスの活動を要約する。	K2			X	X										
3.2.3	レビューの際に、主要な役割にどのような責務が割り当てられているかを想起する。	K1				X								X		
3.2.4	さまざまなレビュー種別を比較、そして対比する。	K2		X												
3.2.5	レビューの成功に貢献する要因を想起する。	K1					X							X		
第 4 章	**テスト分析と設計**															
4.1	**テスト技法の概要**															
4.1.1	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を区別する。	K2		X												
4.2	**ブラックボックステスト技法**															
4.2.1	テストケースを導出するために同値分割法を使用する。	K3					X									
4.2.2	テストケースを導出するために境界値分析を使用する。	K3					X									
4.2.3	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使用する。	K3					X									
4.2.4	テストケースを導出するために状態遷移テストを使用する。	K3					X									
4.3	**ホワイトボックステスト技法**															
4.3.1	ステートメントテストを説明する。	K2		X												
4.3.2	ブランチテストを説明する。	K2		X												

4.3.3	ホワイトボックステストの価値を説明する。	K2	X	X												
4.4	**経験ベースのテスト技法**															
4.4.1	エラー推測を説明する。	K2		X												
4.4.2	探索的テストを説明する	K2		X												
4.4.3	チェックリストベースドテストを説明する。	K2		X												
4.5	**コラボレーションベースのテストアプローチ**															
4.5.1	開発者やビジネス側代表者と共同でユーザーストーリーを作成する方法を説明する。	K2				X						X				
4.5.2	受け入れ基準を書くためのさまざまな選択肢を分類する。	K2										X				
4.5.3	テストケースを導出するために受け入れテスト駆動開発（ATDD）を使用する。	K3					X									
第 5 章	**テスト活動のマネジメント**															
5.1	**テスト計画**															
5.1.1	テスト計画書の目的と内容を例示する。	K2		X					X							
5.1.2	テスト担当者がイテレーション計画やリリース計画にどのような価値を付加するかを認識する。	K1	X									X		X		
5.1.3	開始基準と終了基準を比較、そして対比する。	K2				X		X								X
5.1.4	見積り技法を用いて、必要なテスト工数を算出する。	K3							X		X					
5.1.5	テストケースの優先順位付けを適用する。	K3							X		X					
5.1.6	テストピラミッドの概念を想起する。	K1		X												
5.1.7	テストの四象限とテストレベルやテストタイプとの関係を要約する。	K2		X							X					
5.2	**リスクマネジメント**															
5.2.1	リスクの可能性とリスクの影響を用いて、リスクレベルを識別する。	K1							X						X	

5.2.2	プロジェクトリスクとプロダクトリスクを区別する。	K2		X											X	
5.2.3	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を説明する。	K2					X				X				X	
5.2.4	分析したプロダクトリスクに対して、どのような手段で対応できるかを説明する。	K2		X			X								X	
5.3	**テストモニタリング、テストコントロールとテスト完了**															
5.3.1	テストで使用するメトリクスを想起する。	K1									X					X
5.3.2	テストレポートの目的、内容、および読み手を要約する。	K2					X				X					X
5.3.3	テストの状況をどのように伝えるかを例示する。	K2												X		X
5.4	**構成管理**															
5.4.1	構成管理がテストをどのように支援するかを要約する。	K2					X		X							
5.5	**欠陥マネジメント**															
5.5.1	欠陥レポートを準備する。	K3		X						X						
第 6 章	**テストツール**															
6.1	**テストのためのツールによる支援**															
6.1.1	さまざまな種類のテストツールがどうテストを支援するかを説明する。	K2					X									
6.2	**テスト自動化の利点とリスク**															
6.2.1	テスト自動化の利点とリスクを想起する	K1					X						X			

10 付録 C - リリースノート

ISTQB® Foundation Syllabus v4.0は、Foundationのシラバス（v3.1.1）およびアジャイルテスト担当者2014のシラバスに基づくメジャーアップデート版である。そのため、章や節ごとの詳細なリリースノートはないが、主要な変更点の要約は以下に示す。また、ISTQB®は、別のリリースノートドキュメントにおいて、3.1.1のFoundation Levelシラバス、2014アジャイルテスト担当者シラバスの学習の目的（LO）と新しいFoundation Level v4.0シラバスの学習の目的の間のトレーサビリティを示し、どのLOが追加・更新・削除されたかを示す。

シラバスが書かれた時点（2022-2023年）では、100カ国以上で100万人以上がFoundation Levelの試験を受験し、世界中で80万人以上のテスト技術者が認定を受けている。その全員がFoundation Levelシラバスを読んで試験に合格していることが予想されるため、Foundation Levelシラバスはこれまでで最も読まれたソフトウェアテストドキュメントとなる可能性が高い！このメジャーアップデートは、これまでの財産を受け継ぎながら、ISTQB®が世界のテストコミュニティへ提供する品質のレベルに関する何十万人もの人々の見解を向上するためのものである。

本バージョンでは、すべてのLO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、LOとシラバスの節の間に1対1のトレーサビリティを確立し、LOが無ければコンテンツも無いようにしている。また、より読みやすく、理解しやすく、学びやすく、翻訳しやすくすることを目指し、実践的な有用性を高め、知識と技能のバランスを重視している。

今回のメジャーリリースの変更は以下の通り：

- シラバス全体のサイズダウンをした。シラバスは教科書ではなく、ソフトウェアテストの入門コースにおいて、どのようなトピックをどのようなレベルで取り上げるべきかなど、基本的な要素をまとめたドキュメントである。したがって、特に次のように変更した。
  - 研修中に例題や練習問題を提供するのは、研修提供者の仕事であるため、ほとんどの例題をテキストから除外した。
  - K レベルごとに LO の最大ドキュメント量を提示している、「シラバス作成チェックリスト」に従った。（K1＝最大 10 行、K2＝最大 15 行、K3＝最大 25 行）
- Foundation v3.1.1、Agile v2014 のシラバスと比較して LO の数を削減した。
  - K1 の LO は、FLv3.1.1（15）、AT2014（6）の 21 に対し、14 となる。
  - K2 の LO は、FLv3.1.1（40）、AT2014（13）の 53 に対し、42 となる。
  - K3 の LO は、FLv3.1.1（7）、AT2014（8）の 15 に対し、8 となる。
- ソフトウェアテストと関連するトピックに関する古典的な、そして/または評判のよい書籍や記事へのより広範な参照を提供している。
- 第 1 章（テストの基礎）の主な変更点
  - テストスキルの節を拡大、改善した。
  - チーム全体アプローチ（K1）の項を追加した。
  - テストの独立性に関する節を、第 5 章から第 1 章に移行した。
- 第 2 章（SDLC 全体を通してのテスト）の主な変更点
  - 2.1.1 項、2.1.2 項の書き換えと改善、および対応する LO を変更した。
  - テストファーストアプローチ（K1）、シフトレフト（K2）、ふりかえり（K2）といった実践例により焦点を当てた。

- DevOps のコンテキストにおけるテストに関する新しい項を追加した（K2）。
- 統合テストレベルをコンポーネント統合テストとシステム統合テストという 2 つのテストレベルに分割した。

- 第 3 章（静的テスト）の主な変更点
  - レビュー技法に関する節は、K3 の LO（レビュー技法を適用する）とともに削除した。
- 第 4 章（テスト分析と設計）の主な変更点
  - ユースケーステストを削除した（ただし、Advanced Test Analyst のシラバスには掲載されている）。
  - テストケース導出に ATDD を使用することに関する新しい K3 の LO と、ユーザーストーリーと受け入れ基準に関する 2 つの新しい K2 の LO といった、テストのためのコラボレーションベースドアプローチにより焦点を当てた。
  - デシジョンテストとカバレッジをブランチテストとカバレッジに置き換えた（最初に、ブランチカバレッジの方が実務でよく使われる；2 番目に、標準によって、「ブランチ」に対する「デシジョン」が異なっている；3 番目に、旧 FL2018 の細かいが深刻な欠陥である「100%デシジョンカバレッジは 100%ステートメントカバレッジを意味する」-この文は、判定がないプログラムでは真実ではない、と主張していることを解決できる）。
  - ホワイトボックステストの価値に関する項を改善した。
- 第 5 章（テスト活動のマネジメント）の主な変更点
  - テスト戦略/アプローチに関する節を削除した。
  - テスト工数を見積もるための見積り技法に関する新しい K3 の LO を追加した。
  - イテレーションとリリース計画（K1）、テストピラミッド（K1）、テスト四象限限（K2）といったテストマネジメントにおけるよく知られたアジャイル関連の概念とツールにより焦点を当てた。
  - リスクマネジメントの節は、リスク識別、リスクアセスメント、リスク軽減、リスクモニタリングの 4 つの主要な活動を説明することで、よりよい構成とした。
- 第 6 章（テストツール）の主な変更点
  - いくつかのテスト自動化に関する論点の内容は、foundation level では高度すぎるため除外となり、ツールの選択、パイロットプロジェクトの実行、組織へのツールの導入といった節を削除した。