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

Version 3.1.1.J02

International Software Testing Qualifications Board

改訂履歴

◆ ISTQB®

バージョン	日付	摘要
V3.1.1	2021 年 5 月 15 日	誤植：著作権表示を現在のものに適合
v3.1.0	2021 年 3 月 3 日	3.2.3 項の書き直しと様々な表現改善によるマイナーアップデート
2019 v3.0	2019 年 10 月 19 日	全体的な改訂とスコープの縮小に伴うメジャーアップデート
2012 v2.0	2012 年 10 月 19 日	AL-TA シラバスとしての初版

◆ JSTQB®

バージョン	日付	摘要
Version 3.1.1.J02	2021 年 8 月 10 日	「2.4.2 将来のテストサイクルに向けたテストの調整」において、記載内容を追加
Version 3.1.1.J01	2021 年 7 月 14 日	ISTQB Advanced Level Syllabus Test Analyst v3.1.1 の日本語翻訳版
Version 2012.J01	2015 年 3 月 25 日	ISTQB Advanced Level Syllabus Test Analyst Version 2012 の日本語翻訳版

目次

謝辞

このドキュメントは、International Software Testing Qualifications Board Advanced Level 作業部会によって発行された。

このドキュメントは、International Software Testing Qualifications Board Advanced Level のテストア
ナリスト作業部会が執筆した: Mette Bruhn-Pedersen (Working Group Chair); Matthias Hamburg (Product Owner); Wim Decoutere, István Forgács, Adam Roman, Jan Sabak, Marc-Florian Wendland (Authors)

本作業部会は、技術編集の Paul Weymouth と Richard Green、用語集適合性チェックの Gary Mogyorodi、および 2019 年版のシラバスへ提案と意見を行ったメンバーボードに感謝したい。
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。Laura Albert, Markus Beck, Henriett Braunné Bokor, Francisca Cano Ortiz, Guo Chaonian, Wim Decoutere, Milena Donato, Klaudia Dussa-Zieger, Melinda Eckrich-Brajer, Péter Földházi Jr, David Frei, Chen Geng, Matthias Hamburg, Zsolt Hargitai, Zhai Hongbao, Tobias Horn, Ágota Horváth, Beata Karpinska, Attila Kovács, József Kreisz, Dietrich Leimsner, Ren Liang, Claire Lohr, Ramit Manohar Kaul, Rik Marselis, Marton Matyas, Don Mills, Blair Mo, Gary Mogyorodi, Ingvar Nordström, Tal Peer, Pálma Polyák, Meile Posthuma, Lloyd Roden, Adam Roman, Abhishek Sharma, Péter Sótér, Lucjan Stapp, Andrea Szabó, Jan te Kock, Benjamin Timmermans, Chris Van Bael, Erik van Veenendaal, Jan Versmissen, Carsten Weise, Robert Werkhoven, Paul Weymouth

このドキュメントは、2021 年 2 月 23 日に ISTQB®によって発行された。

このドキュメントの 2019 年版は、International Software Testing Qualifications Board Advanced Level Working Group のコアチーム（Graham Bath、Judy McKay、Mike Smith）によって作成された。

このシラバスの 2019 年版のレビュー、意見、投票には、次のメンバーが参加した。
Laura Albert, Markus Beck, Henriett Braunné Bokor, Francisca Cano Ortiz, Guo Chaonian, Wim Decoutere, Milena Donato, Klaudia Dussa-Zieger, Melinda Eckrich-Brajer, Péter Földházi Jr, David Frei, Chen Geng, Matthias Hamburg, Zsolt Hargitai, Zhai Hongbao, Tobias Horn, Ágota Horváth, Beata Karpinska, Attila Kovács, József Kreisz, Dietrich Leimsner, Ren Liang, Claire Lohr, Ramit Manohar Kaul, Rik Marselis, Marton Matyas, Don Mills, Blair Mo, Gary Mogyorodi, Ingvar Nordström, Tal Peer, Pálma Polyák, Meile Posthuma, Lloyd Roden, Adam Roman, Abhishek Sharma, Péter Sótér, Lucjan Stapp,

Andrea Szabó, Jan te Kock, Benjamin Timmermans, Chris Van Bael, Erik van Veenendaal, Jan Versmissen, Carsten Weise, Robert Werkhoven, Paul Weymouth.

このドキュメントの 2012 年版は、International Software Testing Qualifications Board Advanced Level Sub Working Group–Advanced Test Analyst のコアチーム（Judy McKay(Chair), Mike Smith, Erik vanVeenendaal　）によって作成された。

Advanced Level のシラバス完成時の Advanced Level Working Group には、次のメンバーが参加していた（アルファベット順）。
Graham Bath, Rex Black, Maria Clara Choucair, Debra Friedenberg, Bernard Homès (Vice Chair), Paul Jorgensen, Judy McKay, Jamie Mitchell, Thomas Mueller, Klaus Olsen, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Jan Sabak, Hans Schaefer, Mike Smith (Chair), Geoff Thompson, Erik van Veenendaal, Tsuyoshi Yumoto.

このシラバスの 2012 年版のレビュー、意見、投票には、次のメンバーが参加した。
Graham Bath, Arne Becher, Rex Black, Piet de Roo, Frans Dijkman, Mats Grindal, Kobi Halperin, Bernard Homès, Maria Jönsson, Junfei Ma, Eli Margolin, Rik Marselis, Don Mills, Gary Mogyorodi, Stefan Mohacsi, Reto Mueller, Thomas Mueller, Ingvar Nordstrom, Tal Pe'er, Raluca Madalina Popescu, Stuart Reid, Jan Sabak, Hans Schaefer, Marco Sogliani, Yaron Tsubery, Hans Weiberg, Paul Weymouth, Chris van Bael, Jurian van der Laar, Stephanie van Dijk, Erik van Veenendaal, Wenqiang Zheng, Debi Zylbermann.

0. 本シラバスの紹介

0.1 本シラバスの目的

本シラバスは、テストアナリスト向けの国際ソフトウェアテスト資格 Advanced Level のベースとなる。ISTQB®は、本シラバスを次の趣旨で提供する。

1. 各国の委員会に対し、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。各国の委員会は、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修正することができる。
2. 試験委員会に対し、各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、試験準備（研修コースの一部、または独立した形）の目的で提供する。
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った場合に限り、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 Advanced Level

Advanced Level Core 資格認定は、次の役割に関連する 3 つの独立したシラバスで構成している。

- テストマネージャー
- テストアナリスト
- テクニカルテストアナリスト

『ISTQB® Advanced Level 概要 2019』が別ドキュメント[ISTQB_AL_OVIEW]として用意されており、次の情報を記載している。

- 各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
- 各シラバスの概要
- 各シラバスの関係

0.3 試験のため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、Advanced Level テストアナリスト認定を取得するための試験問題作成を行うために使用する。

本シラバスでは各章の先頭で、K2 レベル、K3 レベル、および K4 レベルの「学習の目的」を以下の分類にて示している。

- K2:理解
- K3:適用
- K4:分析

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 Advanced Level テストアナリスト試験

Advanced Level テストアナリスト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またがる情報に基づくことがある。本シラバスのすべての節は、「イントロダクション」と付録 A 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。総得点の 65%以上を獲得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 認定資格試験の受験資格

Advanced Level テストアナリストの認定資格試験を受験するには、Certified Tester Foundation Level 認定資格を取得している必要がある。

0.6 前提事項

Advanced テストアナリストの学習の目的には、特定の経験を想定しているものはない。

0.7 コース認定

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。

0.8 シラバスの詳細レベル

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る。

- 全般的な教育内容の目的（Advanced Level テストアナリストの意図について説明）
- アイテムのリスト（想起できる必要のあるアイテム）
- 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全知識領域の説明ではない。詳細レベルは、Advanced 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、アジャイルソフトウェア開発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資料に重点を置いて説明している。本シラバスでは、特定のソフトウェア開発ライフサイクル（SDLC）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ソフトウェア開発、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。

0.9 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、章の最小学習時間および演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 20 時間 30 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:テストプロセスにおけるテストアナリストのタスク(150 分)
- 第 2 章:リスクベースドテストにおけるテストアナリストのタスク(60 分)
- 第 3 章:テスト技法(630 分)
- 第 4 章:ソフトウェア品質特性のテスト(180 分)
- 第 5 章:レビュー(120 分)
- 第 6 章:テストツールおよび自動化(90 分)

1. テストプロセスにおけるテストアナリストのタスク – 150 分

キーワード
終了基準、ハイレベルテストケース、ローレベルテストケース、テスト、テスト分析、テストベース、テスト条件、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、テスト手順、テストスイート

「テストプロセスにおけるテストアナリストのタスク」の学習の目的

1.1 イントロダクション
学習の目的はなし

1.2 ソフトウェア開発ライフサイクルにおけるテスト
TA-1.2.1 (K2) 対応するソフトウェア開発ライフサイクルモデルに応じて、テストアナリストが関与するタイミングとその関わり方がどのように異なるのか、またその理由を説明する。

1.3 テスト分析
TA-1.3.1 (K2) 分析の活動を行う際に、テストアナリストにとって適切なタスクをまとめる。

1.4 テスト設計
TA-1.4.1 (K2) ステークホルダーがテスト条件を理解する必要がある理由を説明する。
TA-1.4.2 (K4) 特定のプロジェクトシナリオに対して、テストケースの適切な設計レベル（ハイレベルまたはローレベル）を選択する 。
TA-1.4.3 (K2) テストケース設計で考慮すべき問題を説明する。

1.5 テスト実装
TA-1.5.1 (K2) テスト実装の活動を行う際に、テストアナリストにとって適切なタスクをまとめる。

1.6 テスト実行
TA-1.6.1 (K2) テスト実行の活動を行う際に、テストアナリストにとって適切なタスクをまとめる。

1.1 イントロダクション

ISTQB® Foundation Level シラバスでは、次の活動を含むテストプロセスを説明している。

- テスト計画
- テストのモニタリング、およびコントロール
- 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

本 Advanced Level Test Analyst シラバスでは、テストアナリストに特に関連する活動を詳細に説明する。本書では、様々なソフトウェア開発ライフサイクル（SDLC）モデルへの適合性を向上させるために、テストプロセスをさらに詳細に解説する。

適切なテストの決定、それらの設計、実装、および実行は、テストアナリストが焦点を当てる主要な領域である。テストプロセスの他の活動を理解することも重要であるが、通常テストアナリストの主要な作業は、次の活動に重点を置く。

- テスト分析
- テスト設計
- テスト実装
- テスト実行

テストプロセスの他の活動については、Foundation Level で適切に説明されており、このレベルでのさらなる詳細は必要ない。

1.2 ソフトウェア開発ライフサイクルにおけるテスト

テスト戦略を定義する際には、SDLC 全体を考慮すべきである。テストアナリストが関与するタイミングは SDLC ごとに異なり、関与の程度、必要な時間、利用可能な情報、および期待も同様に異なる。テストアナリストは、他の関連組織の業務へ提供する情報の種類を把握しておかなければならない。提供先と提供する情報は以下の通り。

- 要求エンジニアリングおよびマネジメント – 要件レビューのフィードバック
- プロジェクトマネジメント – スケジュールに対する入力
- 構成管理および変更管理 – ビルド検証テストの結果、バージョンコントロールの情報
- ソフトウェア開発 – 検出された欠陥の通知
- ソフトウェアメンテナンス – 欠陥レポート、欠陥除去効率、確認テスト
- テクニカルサポート – 正確に文書化した回避策および既知の問題

- テクニカルドキュメント作成（例えば、データベース設計仕様、テスト環境ドキュメント）– これらのドキュメントへの入力とドキュメントのテクニカルレビュー

テストの活動は、選択した SDLC（シーケンシャル、イテレーティブ、インクリメンタル、またはそれらのハイブリッドのいずれか）と整合している必要がある。例えば、シーケンシャルな V 字モデルの場合で、テストプロセスをシステムテストレベルに適用すると、次の内容に沿ったものとなる。

- プロジェクト計画と同時にシステムテスト計画を始め、テスト完了までテストモニタリングとテストコントロールを継続する。これは、プロジェクトをマネジメントする目的でテストアナリストが提供するスケジュールに影響する。
- システムテストのテスト分析およびテスト設計は、システム要件仕様、システムおよびアーキテクチャー（ハイレベル）設計仕様、およびコンポーネント（ローレベル）設計仕様などのドキュメントに沿って行う。
- システムテスト環境の実装は、システム設計時に開始する場合があるものの、その大部分は、コーディングおよびコンポーネントテストと同時に開始するのが常であり、システムテストの実行開始数日前までシステムテスト実装への取り組みが続いてしまうことが多い。
- システムテストの実行は、テストの開始基準を満たした（または必要に応じて免除された）ときに始まる。これは最低でもコンポーネントテストと多くの場合はコンポーネント統合テストも終了基準を満たしていることを意味する。システムテストの実行は、システムテスト終了基準を満たすまで継続する。
- システムテストの完了活動は、システムテスト終了基準を満たした後に実行する。

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、活動を異なる順番で実行したり、一部の活動を除外したりすることがある。例えば、イテレーティブモデルでは、イテレーションごとにテスト活動の縮小セットを使用することがある。テスト分析、設計、実装、および実行をイテレーションごとに実行し、ハイレベルの計画作業をプロジェクトの開始時のみ、完了タスクをプロジェクトの終了時のみに実行することがある。

アジャイルソフトウェア開発では、あまり形式ばらないプロセスを使用し、プロジェクトステークホルダーと密な関係を築いて、プロジェクト内で変更をより簡単に行うことが一般的である。テストアナリストの役割が明確に定義されないことがある。包括的なテストドキュメントを作成することは少なく、日々のコミュニケーションは手短かつ頻繁に行う。

アジャイルソフトウェア開発では、開始時点からテストを行う。つまり、開発者が初期のアーキテクチャーおよび設計の作業を行うプロダクト開発の開始時から、テストを行い始める。レビューは形式的には行わないこともあるが、ソフトウェア開発が進む中で継続的に行われる。テストはプロジェクト全体を通して行われ、テストアナリストのタスクはチームが行うことが期待される。

イテレーティブモデルおよびインクリメンタルモデルは、顧客の要件が進展するにつれ変化することが予測されるアジャイルソフトウェア開発から、V 字モデルのアプローチを組み合わせたイテレーティブ／インクリメンタル開発などのハイブリッドモデルにまで及ぶ。このようなハイブリッドモデルの場合、テストアナリストは計画および設計の一部分に関与し、イテレーティブ／インクリメンタルな活動中に、より相互作用的な役割へと変わっていく必要がある。

使用する SDLC に関係なく、テストアナリストは関与への期待度とタイミングを理解する必要がある。テストアナリストは、事前に定義されたロールモデルに固執することなく、特定の SDLC への活動と関与のタイミングを調整することにより、ソフトウェア品質へ効果的に貢献する。

1.3 テスト分析

テスト計画では、テストプロジェクトの範囲を定義する。テスト分析では、テストアナリストがこの範囲に対して、次のことを行う。

- テストベースを分析する。
- テストベースのさまざまな種類の欠陥を識別する。
- テスト対象のテスト条件とフィーチャーを識別し、優先度を割り当てる。
- 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。
- リスクベースドテストに付随するタスクを実行する（第 2 章を参照）。

テストアナリストはテスト分析を効果的に進めるために、次の開始基準が満たされていることを確認する必要がある。

- （例えば、要件、ユーザーストーリーといった）テスト対象について記述した、テストベースとなりうる知識体系がある（[ISTQB_FL_SYL] 1.4.2 節と 2.2 節を参照、またはテストベースになりうる情報源の一覧）。
- このテストベースは、適切な結果でレビューに合格しており、レビュー後に必要に応じて更新されている。ハイレベルテストケースを定義する作業が予定されている場合（1.4.1 節参照）、テストベースはまだ完全に定義する必要はないことがある。アジャイルソフトウェア開発では、各イテレーションの開始時にユーザーストーリーを洗練していくので、このレビューサイクルは繰り返すこととなる。
- テスト対象に対して、残りのテストタスクを完了するために適切な予算とスケジュールが確保されている。

テスト条件を識別するには、通常、テストベースをテスト目的（テスト計画で定義済み）とともに分析する。ドキュメントが古いままで更新されていないか、存在しない場合、例えばワークショップやイテレーション計画作業の際に、関連するステークホルダーと協議することによりテスト条件を識別できることがある。アジャイルソフトウェア開発では、ユーザーストーリーの

一部として定義されている受け入れ基準が、テスト設計のベースとして使用されることが多い。

テスト条件は、通常、テストされるアイテムごとに具体的にするものであるが、いくつか標準的な考慮事項がテストアナリストにはある。

- 一般的には、テスト条件を様々な詳細度合いで定義することが望ましい。まず、テストをするための大まかな対象を定義するために、「画面 x の機能性」といったハイレベルの条件を識別する、次に、具体的なテストケースのベースとして、「画面 x では、正しい長さよりも1桁足りないアカウント番号を拒否する」といった、より詳細な条件を識別する。このような階層アプローチを使用してテスト条件を定義することにより、ハイレベルのアイテムに対するカバレッジを十分なものにすることができる。このアプローチを使うことで、テストアナリストは、まだ洗練されていないユーザーストーリーに対するハイレベルなテスト条件を定義する作業を開始できる。
- プロダクトリスクが定義済みの場合、各プロダクトリスクに対処するために必要なテスト条件を識別し、リスクアイテムにまで遡ることができる必要がある。

テスト戦略そして／またはテスト計画内で識別されているテスト技法を適用することで、テスト分析活動を促進し、次の目的の達成が容易になることがある。

- テスト条件の識別
- 重要なテスト条件が省略される可能性の低減
- より明瞭で正確なテスト条件の定義
- テスト条件を識別して洗練した後に、これらの条件をステークホルダーと一緒にレビューすることで、要件を明確に理解し、テストをプロジェクトのゴールに一致させることができる。

特定の領域（例えば、ある機能仕様）に対するテスト分析の活動が完了したとき、テストアナリストは、その領域に対してどのような仕様のテストケースを設計しなければならないかが分かるようになっているべきである。

1.4 テスト設計

テストプロセスは、テスト計画時に決定した範囲に従い、テストアナリストがテストケースを設計し、それを実装および実行するテストプロセスが続いていく。テスト設計のプロセスは、次の活動を含む。

- ローレベルテストケースまたはハイレベルテストケースがどのテスト領域で最も適切であるかを判断する。
- 必要なカバレッジを確保するテスト技法を決定する。使用する可能性がある一連の技法は、テスト計画時に決めてある。
- 識別したテスト条件をカバーするテストケースおよびテストケースのセットを設計するために、テスト技法を使用する。

- テスト条件とテストケースに準じた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、テストケースなどの間で双方向のトレーサビリティを確立する。

リスク分析およびテスト計画で識別した優先度基準は、分析や設計の段階から実装や実行の段階に至るまで、プロセス全体を通して適用すべきである。

設計するテストケースの種類によっては、設計作業時に使用するツールが利用可能か否かをテスト設計の開始基準の 1 つとすることがある。

テストを設計するときには、テストアナリストは次の点に留意しなければならない。

- 一部のテストアイテムについては、実行手順を指定するテストスクリプトを定義して対処するよりも、テスト条件のみを定義して対処した方がよい場合がある。この場合、テスト条件はテストスクリプトなしにテストをする際のガイドとして使用できるよう定義すべきである。
- 合格／不合格基準を明確に識別すべきである。
- テストケースは、作成者だけでなく、他のテスト担当者が理解できるように定義すべきである。テストケース作成者がテストを実行しない場合、他のテスト担当者はテスト目的およびそのテストケースに関わる重要なことを理解するために、すでに仕様化されたテストケースを参照し、理解することが必要になる。
- テストケースは、(テストケースをレビューする)開発者や(テストケースを承認しなければならない)監査担当者などの他のステークホルダーも理解できるようなテストケースにしなければならない。
- テストケースは、ユーザーインターフェースを通して発生する相互作用だけでなく、テスト対象との相互作用の全種類をカバーすべきである。これには、他のシステムおよび技術的／物理的イベントとの相互作用なども含まれる(詳細については、[IREB_CPRE]参照)。
- テスト対象自体の振る舞いだけでなく、様々なテスト対象間のインターフェースもテストするように、テストケースを設計すべきである。
- テスト設計の労力は、リスクレベルとビジネス価値に合致するように、優先度を割り当て、バランスをとらなければならない。

1.4.1 ローレベルテストケースとハイレベルテストケース

テストアナリストの職務の 1 つは、特定の状況で最適なテストケースの設計レベルを決定することである。ローレベルテストケースとハイレベルテストケースについては、[ISTQB_FL_SYL]を参照されたい。これらの設計レベルを使用することの長所と短所を以下に示す。

ローレベルテストケースには、次の長所がある。

- 経験の少ないテスト担当者は、プロジェクト内で提供される詳細な情報に頼ることができる。ローレベルテストケースは、テスト担当者がテストケース(すべてのデータ要件を含む)を実行し、実際の結果を検証するために必要な特定の情報と手順のすべてを提供する。
- 異なるテスト担当者がテストを再実行しても、同じテスト結果が達成される。
- テストベースに存在する自明ではない欠陥を検出できる。
- 詳細の度合いにより、必要に応じて、テストケースについて監査のような独立した検証が可能になる。
- 自動化に使うテストケースの実装に費やす時間を削減できる。

ローレベルテストケースには、次の短所がある。
- 作成とメンテナンスの両方に非常に多くの労力を必要とする場合がある。
- 実行時にテスト担当者の創造力を制限する傾向にある。
- テストベースを明確に定義する必要がある。
- テスト条件のトレーサビリティを確保するための労力が、ハイレベルテストケースよりも多い場合がある。

ハイレベルテストケースには、次の長所がある。
- テストすべきものに関するガイドラインを提供し、テストアナリストが、テスト実行時に実データや手順さえも変更することを可能にする。
- 実行のたびに少しずつ違うテストをするので、ローレベルテストケースよりも優れたリスクカバレッジを提供することがある。
- 要件プロセスの初期に定義できる。
- テスト実行時に、テストおよびテスト対象の両方に関するテストアナリストの経験を活用できる。
- 詳細かつ形式的なドキュメントが要求されない場合に定義できる。
- 異なるテストデータを使用できる場合に、異なるテストサイクルでの再利用性に優れている。

ハイレベルテストケースには、次の短所がある。
- 再現能力が低く、検証を困難にする。これは、ローレベルテストケースで見られる詳細な説明が不足しているためである。
- テスト実行をするためには、より経験のあるテスト担当者が必要となることがある。
- ハイレベルテストケースに基づいて自動化する場合、詳細な情報が不足していることにより、誤った実際の結果を検証したり、検証すべきアイテムを見逃したりすることがある。

要件がより明確になり安定してきたときに、ハイレベルテストケースはローレベルテストケースの開発に使われることもある。この場合、テストケースの作成は順次行われ、ハイレベルからローレベルとなっていき、実行にはローレベルテストケースのみを使用する。

1.4.2 テストケースの設計

テストケースの設計は、識別されたテスト条件に対して、テスト技法（第 3 章参照）を使って段階的な推敲、洗練をしていく。テストケースは、再利用可能であり、検証可能であり、テストベース（要件など）までのトレーサビリティが確保されているべきである。

テスト設計では、次のアイテムを識別する。

- 目的（観測可能で測定可能なテスト実行の目的）
- プロジェクトまたはローカライズされたテスト環境のどちらかの要件、およびそのテストのリリース計画、テスト実行前のシステムの状態などの事前条件
- テストデータ要件（テストケースを実行するための入力用データとシステム内に必要なデータの両方）
- 明確な合格/不合格基準が定義されている期待結果
- 影響を受けたデータ、テスト実行後のシステムの状態、後続処理のためのトリガーなどの事後条件

テストの期待結果を定義することが、特に課題となることがある。これを手動で計算することは単調で時間のかかる作業で、エラーも発生しやすいので、可能な限り、自動化されたテストオラクルを見つけるか、または作成することを推奨する。テスト担当者は、期待結果を識別する際、画面上の出力だけでなく、データおよび環境的な事後条件も考慮する。テストベースが明確に定義されていると、理論的には、正しい結果を容易に識別できる。ただし、テストベースのドキュメントは、曖昧で矛盾を含み重要な領域をカバーしていないか、もしくはまったく存在しないことがある。このような場合、テストアナリストは、特定分野の専門知識を持っているか、もしくはそれらの情報にアクセスできなければならない。また、テストベースが適切に仕様化してあったとしても、複雑な応答の複合的な相互作用により、期待結果の定義が困難になることがあるので、テストオラクルが必須である。アジャイルソフトウェア開発では、テストオラクルはプロダクトオーナーかもしれない。実際の結果の正しさを判断する方法なしにテストケースを実行すると、付加価値または利点が非常に少なくなり、多くの場合、無効なテストレポートやシステムに対する誤った信頼につながる。

テストベースは異なったとしても、すべてのテストレベルに前述の活動を適用することができる。テストを分析および設計するときには、対象のテストレベルとテスト目的に留意する必要がある。このことは、必要な詳細度合いとツール（例えば、コンポーネントテストレベルでのドライバーおよびスタブ）を決定するのに役立つ。

テスト条件およびテストケースの開発時には、一般的に、テスト作業成果物となるいくつかのドキュメントを作成する。実際には、テスト作業成果物を文書化する範囲は大幅に異なる。次の要因が文書化する範囲に影響を及ぼす。

- プロジェクトリスク（文書化する必要のあるもの、文書化してはいけないもの）
- ドキュメントがプロジェクトに提供する「付加価値」
- 準拠する必要のある標準および満たす必要のある規制
- SDLC または使用するアプローチ（例えば、アジャイルアプローチでは、「必要最小限」の文書化を目標にする）
- テストベースからテスト分析およびテスト設計を通してのトレーサビリティの要件

テストの範囲に応じて、テスト分析およびテスト設計は、テスト対象の品質特性に対応させる。ISO 25010 標準[ISO25010]が、有効な参照情報を提供している。ハードウェア／ソフトウェアシステムをテストする場合には、その他の特性を適用することがある。

テスト分析およびテスト設計の活動は、レビューおよび静的解析とテスト分析およびテスト設計を組み合わせることにより強化できることがある。テスト分析およびテスト設計を実施する活動の中でテストベースの問題を見つけることがあるため、実際に、テスト分析およびテスト設計を行うことが静的テストの一形態となることがしばしばある。要求仕様に基づいたテスト分析およびテスト設計を行うことは、要件レビューミーティングの準備として優れた方法の 1 つである。要件を読み解き、テストを作成するためにそれを使用するには、要件を理解し、要件の達成度を評価する方法を決定できることが求められる。多くの場合、この活動では、不足している要件、明確でない要件、テストできない要件、または受け入れ基準が定義されていない要件を明らかにする。同様に、テストケース、リスク分析、テスト計画書などのテスト作業成果物をレビュー対象にできる。

要求される詳細なテストインフラストラクチャーの要件は、実際にはテスト実装まで完成しない可能性があるが、テスト設計時に定義することがある。テストインフラストラクチャーは、テスト対象およびテストウェア以外のものも含むことに注意する必要がある。例えば、テストインフラストラクチャーの要件は、場所、機器、担当者、ソフトウェア、ツール、周辺機器、通信機器、ユーザー権限、およびテストを実行するために必要な他のすべてのアイテムを含むことがある。

テスト分析およびテスト設計の終了基準はプロジェクトにより異なるが、2 つの節で説明したすべての事項を定義する終了基準に含めるべきである。重要なのは、終了基準が測定可能であり、以降の手順に必要なすべての情報が提供され、必要なすべての準備が実施されていることである。

1.5 テスト実装

テスト実装では、テスト分析と設計に基づいてテスト実行に必要なテストウェアの準備をする。次の活動を含む。

- テスト手順、および、場合によっては自動テストスクリプトを作成する
- 特定のテストランにおいて実行されるテストスイートとして、テスト手順と(もし存在する場合)自動テストスクリプトを整理する
- 実行するテストケースとテストスイートの優先度付けについて、テストマネージャーに相談する
- テストケースの実行を開始するために必要なテスト実行スケジュール(リソースの割り当てを含む)を作成する([ISTQB_FL_SYL] 5.2.4 節参照)
- テストデータおよびテスト環境の準備を完了する
- テストベースと、テスト条件、テストケース、テスト手順、テストスクリプト、テストスイートなどのテストウェアとの間のトレーサビリティを更新する。

テストアナリストはテスト実装時に、テストケースの効率的な実行順序を特定し、テスト手順を作成する。テスト手順を定義する際には、テスト実行の順序に影響を与える可能性のある制約や依存関係を入念に識別することが求められる。テスト手順では、すべての初期事前条件(データリポジトリからのテストデータのロードなど)およびすべての実行後活動(システムステータスのリセットなど)を記述する。

テストアナリストは、グループ化できるテスト手順と自動テストスクリプトを特定し(例えば、それらはすべて特定のハイレベルなビジネスプロセスのテストに関連する)、それらをテストスイートに整理する。これにより、関連するテストケースを一緒に実行できる。

テストアナリストは、効率的なテスト実行が実現できるように、テスト実行スケジュールにテストスイートを配置する。リスクベースドテスト戦略を使用する場合、リスクレベルは、テストケースの実行順序を決定する際に最も優先的に考慮すべき事柄になる。適切な担当者、機器、データ、テスト対象の機能などの可用性のように、他の要因がテストケースの実行順序を決定することもある。

コードが部分ごとにリリースされることは珍しくないので、ソフトウェアがテスト可能になる順序に合わせてテスト工数配分を調整しなければならない。イテレーティブおよびインクリメンタル開発モデルで特に重要なのは、ソフトウェアがテスト可能な順序でテスト用にリリースされるように、テストアナリストが開発者と調整することである。

テスト実装時に行う作業の詳細度合いとそれに関する複雑度は、テスト条件やテストケースの詳細度に影響される場合がある。場合によっては、法規制の適用が求められ、適用する

標準（例えば、米国標準 DO-178C（欧州の ED-12C））への準拠の証跡をテスト作業成果物で示すべきである[RTCA DO-178C/ED-12C]。

前述しているように、大抵の場合、テストではテストデータが必要であり、場合によってはデータセットが非常に大きくなることがある。テストアナリストは実装時に、データベースや他のリポジトリなどにロードするために入力データと環境データを作成する。このデータは欠陥の検出を可能にするために、「目的にかなって」いなければならない。テストアナリストは、手動テストで使用するデータの他に、データ駆動型およびキーワード駆動型のテスト（6.2 節参照）で使用するデータを作成することもある。

テスト実装では、テスト環境についても考慮する。この段階で、テストを実行する前に環境をすべて設定し、検証しておくべきである。「目的にかなった」テスト環境は必要不可欠である。つまり、テスト環境は、テストでコントロールしている間に残っている欠陥を洗い出し、故障が発生しない場合は適切に動作し、より高いテストレベル向けの本番環境またはエンドユーザー環境を必要に応じて適切に再現できなければならない。予期しない仕様変更、テスト結果、または他の考慮事項に応じて、テスト環境をテスト実行時に変更する必要がある。テスト実行時にテスト環境を変更する場合、実行済みのテストケースに対して変更が与える影響を評価することが重要となる。

テストアナリストは、テスト実装時に、テスト環境の作成およびメンテナンスの担当者のサポートを得ることができ、かつすべてのテストウェアやテストサポートツールおよび関連するプロセスが使用できる状態にあることを確認すべきである。これらのプロセスは、構成管理、欠陥マネジメント、およびテスト結果の記録やマネジメントを含む。また、テストアナリストは、終了基準に対する現在のステータスの評価およびテスト結果のレポート作成のためのデータを収集する手順を検証しなければならない。

テスト実装では、テスト計画で決定したバランスのとれたアプローチを使用することを推奨する。例えば、分析的なリスクベースドテスト戦略に対処的テスト戦略を混ぜることがよくある。この場合、テスト実装作業の一部を、事前に決定したスクリプトに従わないテスト（スクリプトがないテスト）に割り当てる。

スクリプトがないテストを行うと、期間とカバレッジの予測が困難になることがあり、欠陥の検出率が低下することがあるので、ランダムまたは目的がないものにするべきではない。むしろ、スクリプトがないテストは、タイムボックスにしたセッションとして行うべきであり、それぞれのセッションでは、テストチャーターが最初の方向性を与えるべきであるが、セッションの中でより生産性の高いテストを行う機会が見つかった場合には、チャーターが与えた方向性から自由に離れてよい。テスト担当者は長い年月をかけて、攻撃[Whittaker03]、エラー推測[Myers11]、探索的テスト[Whittaker09]など、様々な経験ベースのテスト技法を開発してき

た。テスト分析、テスト設計、およびテスト実装は引き続き行われるが、主にテスト実行期間に行われることになる。

このような対処的テスト戦略の場合、各テストの結果は、以降のテストの分析、設計、および実装に影響を及ぼす。これらの戦略は軽量であり、多くの場合、欠陥を見つけるのに有効であるが、以下に示す短所もある。

- テストアナリストに専門知識を要求する
- 期間の予測が困難
- カバレッジの追跡が困難
- テストの再現性を維持するために優れたドキュメントやツールのサポートが必要

1.6 テスト実行

テスト実行はテスト実行スケジュールに従って行い、次のタスクを含む（[ISTQB_FL_SYL]参照）。

- 手動テスト（探索的テストを含む）を実行する
- 自動テストを実行する
- 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
- 故障を観察し、観察に基づいて欠陥を報告する
- テスト実行の実際の結果を記録する
- テスト結果を検討するためテストベースとテストウェアの間でトレーサビリティを更新する
- リグレッションテストを実行する

前述のテスト実行タスクは、テスト担当者またはテストアナリストが行う。

次に示すタスクは、テストアナリストが追加で行う可能性がある典型的なタスクである。

- 欠陥が集中しており、さらなるテストが必要になりうるテスト対象の一部を特定する
- 探索的テストからの発見事項に基づいて、将来の探索的テストに対する提案を作成する
- テスト実行タスクを行う際に取得した情報から新しいリスクを識別する
- テスト実装活動からのあらゆる作業成果物を改善するための提案を作成する（テスト手順に対する改善など）

2. リスクベースドテストにおけるテストアナリストのタスク – 60 分

キーワード
プロダクトリスク、リスク識別、リスク軽減、リスクベースドテスト

「リスクベースドテストにおけるテストアナリストのタスク」の学習の目的

リスクベースドテストにおけるテストアナリストのタスク
TA-2.1.1　(K3) 特定の状況で、リスク識別に参加し、リスクアセスメントを実行し、適切なリスク軽減策を提案する 。

2.1 イントロダクション

テストマネージャーは、多くの場合、リスクベースドテスト戦略の確立とマネジメントに全体的な責任を持つ。テストマネージャーは、通常、リスクベースドアプローチが適切に実装されるようにするために、テストアナリストの関与を要求する。

テストアナリストは、次のリスクベースドテストのタスクに積極的に関わるべきである。

- リスク識別
- リスクアセスメント
- リスク軽減

リスクの新たな発生および優先度の変更に対処し、定期的にリスクのステータスを評価および共有するために、これらのタスクを SDLC 全体を通して反復的に実行する（詳細については、[vanVeenendaal12]および[Black02]を参照）。アジャイルソフトウェア開発では、これら 3 つのタスクは、イテレーションまたはリリースのいずれかに焦点を置いた、いわゆるリスクセッションに組み入れられる。

テストアナリストは、テストマネージャーがプロジェクトのために確立したリスクベースドテストフレームワーク内で作業すべきである。また、機能安全、ビジネスおよび経済上の懸念、政治的な要因に関連するリスクなど、プロジェクトに内在するビジネスドメインのリスクに関するテストアナリスト自身の知識を提供すべきである。

2.2 リスク識別

リスク識別プロセスでは、可能な限り幅広いステークホルダーを召集することにより、重要なリスクを数多く検出することが可能になる。

テストアナリストは多くの場合、テスト対象システムの特定のビジネスドメインに関する特有の知識を保有するので、次のタスクに非常に適している。

- そのドメインの専門家やユーザーとともに行う専門家へのインタビューの実施
- 独立したアセスメントの実施
- リスクテンプレートの使用
- リスクワークショップへの参加
- 潜在的ユーザーや現在のユーザーとのブレインストーミングセッションへの参加
- テスト用チェックリストの定義
- 類似のシステムまたはプロジェクトにおける過去の経験の活用

特に、テストアナリストは、ユーザーおよび他のドメインの専門家（要件エンジニア、ビジネスアナリストなど）と緊密に連携して、テスト中に対応すべきビジネスリスクの領域を決定するべ

きである。アジャイルソフトウェア開発では、ステークホルダーと緊密に連携することにより、イテレーション計画ミーティングなどの定期的なイベントでリスク識別を実施することが可能になる。

プロジェクトで識別される可能性のあるリスクの例を次に示す。

- 機能の正確性の問題（例えば、誤った計算）
- 使用性の問題（例えば、キーボードショートカットの不足）
- 移植性の問題（例えば、アプリケーションを特定のプラットフォームにインストールできない）

2.3 リスクアセスメント

リスク識別は、可能な限り多くのリスクを識別することを意味するが、リスクアセスメントは、これらの識別されたリスクを調査することを意味する。特に、リスクアセスメントでは、それぞれのリスクを分類して、そのリスクレベルを判定する。

リスクレベルを決定する場合、通常はリスクアイテムごとに、リスクの可能性および影響を評価する。リスクの可能性は、潜在的な問題がテスト対象のシステムに存在し、システムが本番環境に移行した後に観察される可能性と考えることができる。テクニカルテストアナリストは、各リスクアイテムの検出とそれぞれの潜在的な可能性の理解に貢献しなければならないが、テストアナリストは問題が発生した場合にビジネスに対して起こりうる影響の把握に貢献する（アジャイルソフトウェア開発では、この役割ベースの区別は曖昧なことが多い）。

リスクの影響は、多くの場合、ユーザー、顧客、またはその他のステークホルダーに対する影響の重要度と考えることができる。ビジネスリスクに起因するリスク発生の影響とも言える。テストアナリストは、各リスクアイテムがビジネスドメインやユーザーに及ぼす潜在的な影響を識別および評価する必要がある。ビジネスリスクに影響する要因を次に示す。

- 影響を受けるフィーチャーの使用頻度
- ビジネス損失
- 金銭的損失
- 環境保護的または社会的損失、または責任の可能性
- 民事上または刑事上の法的制裁
- 機能安全の課題
- 賠償金、ライセンスの喪失
- さらなる作業を行うことができない場合、妥当な回避策の欠如
- フィーチャーの可視性
- 故障の判明による社会的および潜在的なイメージの悪化
- 顧客の喪失

テストアナリストは、利用可能なリスク情報を活用し、テストマネージャーにより提供されたガイドラインに基づいて、ビジネスリスクのレベルを確立する必要がある。レベルは、使用している順序スケール（数値、または、低、中、高）または信号機の色を用いて分類することがある。リスクの可能性と影響が割り当てられると、テストマネージャーはこれらの値を使用して各リスクアイテムのリスクレベルを決定する。次に、そのリスクレベルを使用して、リスク軽減活動の優先度を決定する[vanVeenendaal12]。

2.4 リスク軽減

プロジェクト期間において、テストアナリストは、次のことに努めるべきである。

- テストが合格したか不合格したかを一意に示す明確なテストケースを設計する、および要件、設計、およびユーザードキュメントなどのソフトウェア作業成果物のレビューに参加することにより、プロダクトリスクを軽減する。
- テスト戦略およびテスト計画で識別される適切なリスク軽減活動を実装する（例えば、特別なテスト技法を使用して、特にリスクの高いビジネスプロセスをテストする）。
- プロジェクトの進み具合に応じて収集した追加情報に基づいて、既知のリスクを再評価する。この際、必要に応じて、リスクの可能性や影響、またはその両方を調整する。
- テスト期間中に収集された情報から、新しいリスクを識別する。

プロダクトリスクが話題として取り上げられる場合、テストはそのようなリスクの軽減に大きく貢献する。テスト担当者は欠陥を見つけることにより、欠陥が認識されるようにし、リリース前に欠陥に対処する機会を提供して、リスクを軽減する。テスト担当者が欠陥をまったく見つけなかった場合、テストされた特定の条件下でシステムは正しく動作するという証拠を提供することによって、テストがリスクを軽減したことになる。テストアナリストは、特に、正確なテストデータ収集のための機会の調査、現実的なユーザーシナリオの作成とテスト、および使用性調査の実践または監督を行うことで、リスク軽減の選択肢を決定することを支援する。

2.4.1 テストの優先度付け

リスクのレベルはテストケースの優先度付けにも使用する。例えば、テストアナリストが、会計システムでトランザクションの正確性の領域に高いリスクを見つけた場合、テスト担当者は、リスクを軽減するために、他のビジネスドメインの専門家と協力して、正確性のために処理および検証できる多数のサンプルデータセットを収集するかもしれない。同様に、テストアナリストが、新しいテスト対象の使用性の問題に大きなリスクを見つけた場合、テストアナリストは、ユーザー受け入れテストで問題が発見されるのを待つのではなく、ユーザー受け入れテストよりも前に、使用性の問題がテストの早期に識別され、解決されるように、プロトタイプに基づいた早期の使用性テストを優先的に実施するかもしれない。必要なテストを必要なタイミングで実施できるようにスケジュールするため、計画段階の可能な限り早期にこの優先度付けを考慮しなければならない。

場合によっては、すべての高リスクのテストケースを、低リスクのテストケースよりも先に実行したり、厳格なリスク順にテストケースを実行したりする。これは「縦型探索」(depth-first)と呼ばれる。また別の場合としては、サンプリングアプローチを使用して、識別したすべてのリスク領域をカバーするテストケースのサンプルを選択する。これは「横型探索」(breadth-first)と呼ばれる。この方法では、リスクレベルに基づいて選択に重みを付け、同時に、すべてのリスクアイテムを少なくとも 1 回はカバーするようにする。

リスクベースドテストを縦型探索と横型探索のどちらで進めても、すべてのテストを実行しないうちにテストに割り当てた時間を消費してしまう可能性がある。リスクベースドテストでは、その時点における残りのリスクレベルについてテスト担当者がマネジメントにレポートし、マネジメントでテストを延長するか、あるいは残りのリスクをユーザー、顧客、ヘルプデスク／テクニカルサポート、運用スタッフに移転するかを決定できる。

2.4.2 将来のテストサイクルに向けたテストの調整

リスクアセスメントは、テスト実装を開始する前に 1 度だけ実行する活動ではなく、継続するプロセスである。将来に計画されている各テストサイクルでは、以下に示す要素を考慮するため、新しくリスク分析をすべきである。

- 新しい、または大幅に変更されたプロダクトリスク
- テスト時に発見された不安定または故障の多い領域
- 修正された欠陥からのリスク
- テスト時に発見された典型的な欠陥
- テストが不十分な(要件カバレッジの低い)領域

3. テスト技法 – 630 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、クラシフィケーションツリー技法、デシジョンテーブルテスト、欠陥分類法、欠陥ベースドテスト技法、同値分割法、エラー推測、経験ベースのテスト、経験ベースのテスト技法、探索的テスト、ペアワイズテスト、状態遷移テスト、テストチャーター、ユースケーステスト

「テスト技法」の学習の目的

3.1 イントロダクション

学習の目的はなし

3.2 ブラックボックステスト技法

TA-3.2.1 (K4) 同値分割法を適用して、特定の仕様アイテムを分析しテストケースを設計する。

TA-3.2.2 (K4) 境界値分析を適用して、特定の仕様アイテムを分析しテストケースを設計する。

TA-3.2.3 (K4) デシジョンテーブルテストを適用して、特定の仕様アイテムを分析しテストケースを設計する。

TA-3.2.4 (K4) 状態遷移テストを適用して、特定の仕様アイテムを分析しテストケースを設計する。

TA-3.2.5 (K2) クラシフィケーションツリー図がテスト技法にどのように役立つかを説明する 。

TA-3.2.6 (K4) ペアワイズテストを適用して、特定の仕様アイテムを分析しテストケースを設計する。

TA-3.2.7 (K4) ユースケーステストを適用して、特定の仕様アイテムを分析しテストケースを設計する 。

TA-3.2.8 (K4) 発見される可能性のある欠陥の種類を判別し、適切なブラックボックステスト技法を選択するためにシステムまたはその要求仕様を分析する。

3.3 経験ベースのテスト技法

TA-3.3.1 (K2) 経験ベースのテスト技法の原則と、ブラックボックステスト技法および欠陥ベースのテスト技法と比較した場合の長所と短所を説明する。

TA-3.3.2 (K3) 与えられたシナリオから探索的テストを識別する。

TA-3.3.3 (K2) 欠陥ベースのテスト技法の適用方法と、使用方法におけるブラックボックステスト技法との違いを説明する。

3.4 最適なテスト技法の適用

TA-3.4.1 (K4) 与えられたプロジェクト状況に対して特定のゴールを達成するためにブラックボックステスト技法または経験ベースのテスト技法のどちらを適用するかを決定する。

3.1 イントロダクション

この章で考慮されるテスト技法は、次のカテゴリーに分類される。

- ブラックボックス
- 経験ベース

これらの技法は補完的であり、実施するテストレベルには関係なく、あらゆるテストの活動に適時使用することができる。

両方のカテゴリーに属する技法は、機能的な品質特性、および非機能的な品質特性をテストするために使用できることを付記しておく。ソフトウェア特性のテストについては、次の章で説明する。

次の節で説明するテスト技法は、最適なテストデータを決定すること(例えば、同値パーティションから)、またはテスト手順を導き出すこと(例えば、状態モデルから)へ第一に重点を置く場合がある。テストケースを完成させるためには、複数の技法を組み合わせることが一般的である。

3.2 ブラックボックステスト技法

ブラックボックステスト技法については、ISTQB® Foundation Level シラバス[ISTQB_FL_SYL]で説明している。

ブラックボックステスト技法に共通する特徴としては以下のものがある。

- 例えば、状態遷移図やデシジョンテーブルなどのモデルは、テスト技法に従ってテスト設計時に作成する
- テスト条件は、これらのモデルから体系的に導き出す

テスト技法は、一般的に、テスト設計およびテスト実行の活動を測定するために使用できるカバレッジ基準を提供する。カバレッジ基準を十分に満たすことは、すべてのテストケースのセットを完了することを意味するのではなく、その技法に基づいてカバレッジを高めるために追加するテストケースがモデルにはこれ以上ないことを意味する。

ブラックボックステストは、通常、システム要件仕様やユーザーストーリーといった、何かしらの仕様ドキュメントに基づく。仕様ドキュメントには、特に機能適合性に関するシステムの振る舞いが記述してあるべきである。そのため、要件からテストケースを導き出すことは、システムの振る舞いに対するテストの一部となることが多い。場合によっては、仕様ドキュメントが存在しないことがあるが、旧システムからの機能適合性の置き換えといった暗黙的な要件は存在する。

数多くのブラックボックステスト技法が存在する。これらの技法はそれぞれ、異なる種類のソフトウェアおよびシナリオを対象にする。以降の節では、各技法の適用性、テストアナリストが経験する可能性のあるいくつかの制限と注意事項、カバレッジを測定する方法、および対象となる検出できる欠陥の種類について説明する。

詳細については、[ISO29119-4]、[Bath14]、[Beizer95]、[Black07]、[Black09]、[Copeland04]、[Craig02]、[Forgács19], [Koomen06]、および[Myers11]を参照されたい。

3.2.1 同値分割法

同値分割法(EP)は、入力、出力、内部値、および時間関連の値の処理を効果的にテストする場合に必要なテストケースの数を少なくするために使用する技法である。同じように処理される必要のある値のセットとして作成される同値パーティション(同値クラスとも呼ばれる)を作成するために分割法を使用する。パーティションから 1 つの代表値を選択することにより、同じパーティション内のすべてのアイテムに対するカバレッジと見なす。

通常、複数のパラメーターがテスト対象の振る舞いを決定する。異なるパラメーターの同値パーティションをテストケースにまとめる際には、様々な手法が適用できる。

適用

この技法は、すべてのテストレベルで適用でき、テストすべき値セットのすべての要素が同じ方法で処理されることが期待され、アプリケーションにより使用される値セット同士が相互作用しない場合に適している。同値パーティションは、順序あり、順序なし、離散、連続、無限、有限、さらには 1 つだけといった何かしらの値のセットとなる。値のセットの選択は、有効パーティションおよび無効パーティション(つまり、テスト対象のソフトウェアに対して無効であると見なされるべき値を含むパーティション)に適用できる。

値は、次に示す分類で考慮しなければならない。

- 連続範囲内の値。例えば、0 < x < 5000 の表記は、変数「x」の連続した値の範囲を表し、有効なパーティションは 1～4999 である。この連続範囲の有効な値は、例えば、3249 である。
- 離散値。例えば、色「赤」、「青」、「緑」、「黄」はすべて有効なパーティションであり、各パーティションで可能な値は 1 つだけである。

EP は、テストする値の範囲をパーティションの境界まで広げる境界値分析と組み合わせて使用すると最も強力になる。有効なパーティションからの値を使用することで、EP は基本的な機能が動作していることを素早く判断できるので、一般的に、新しいビルドまたはリリースをスモークテストする場合に使用する。

制限／注意事項

仮定が正しくなく、パーティション内の値が正確に同じ方法で処理されない場合、この技法は欠陥を見逃す可能性がある。また、パーティションを注意深く選択することも重要である。例えば、正の数と負の数を受け入れる入力欄は処理方法が異なる可能性があるので、正の数と負の数の 2 つの有効パーティションとしてテストする方がよい。ゼロを許容するかどうかによって、パーティションが追加される場合もある（ゼロは正でも負でもない）。テストアナリストは、最適なパーティションを決定するために、裏にある処理を理解することが重要である。このためには、コード設計を理解するための支援が求められる場合がある。

テストアナリストは、異なるパラメーターの同値パーティション間で起こりうる依存関係も考慮すべきである。例えば、フライト予約システムでは、パラメーター「大人の同伴」は、年齢クラス「子ども」との組み合わせでのみ使用することがある。

カバレッジ

カバレッジは、テストされたパーティションの数を、識別されたパーティションの数で除算することにより決定し、パーセンテージで表される。単一のパーティションに含まれる複数の値をテストしても、カバレッジの割合を増やすことにはならない。

テスト対象の振る舞いが単一のパラメーターに依存する場合、各同値パーティションが有効、無効であるかに関わらず、少なくとも一度はカバーすべきである。

複数のパラメーターがある場合、テストアナリストは、リスクに応じて、カバレッジタイプをシンプル、または組み合わせのどちらかを選択すべきである[Offutt16]。そのため、有効なパーティションのみを含む組み合わせと、1 つ以上の無効なパーティションを含む組み合わせを区別することが重要になる。有効な同値パーティションのみが含まれる組み合わせについては、すべてのパラメーターについて、すべての有効なパーティションのシンプルなカバレッジが最低条件である。このようなテストスイートに必要なテストケースの最小数は、パラメーターが互いに独立していると仮定した場合、パラメーターの有効なパーティションの最大数に等しい。組み合わせ技法に関する、より十分なカバレッジタイプには、ペアワイズカバレッジ(後述の 3.2.6 節 を参照)や、有効なパーティションの任意の組み合わせに対する全カバレッジがある。欠陥をマスクしてしまうことを避けるために、少なくとも無効な同値パーティションは個別に、つまり他のパラメーターは有効なパーティションにした組み合わせでテストすべきである。そのため、シンプルなカバレッジのためのテストスイートでは、各無効パーティションに対して 1 つずつのテストケースとなる。リスクが高い場合には、無効なパーティションのみの組み合わせ、あるいは無効なパーティションのペアなど、さらなる組み合わせをテストスイートに追加することがある。

検出できる欠陥の種類

テストアナリストはこの技法を使用して、様々な値の処理に関する欠陥を検出できる。

3.2.2 境界値分析

境界値分析(BVA)は、順序付けられた同値パーティションの境界上に存在する値を適切に処理していることをテストするために使用する。BVA には、一般的に 2 つの値を用いる方法と 3 つの値を用いる方法の 2 つの方法がある。2 つの値を用いる境界のテストでは、境界値(境界上)と境界を少し超えた値(必要な精度に基づいた可能な限り最小の増加分)を使用する。例えば、小数点以下の桁数が 2 である通貨の金額について、パーティションに 1 から 10 までの値が含まれている場合、2 つの値を用いる上限境界のテスト値は 10 と 10.01 になる。下限境界のテスト値は、1 と 0.99 である。境界は、定義された同値パーティションの最大値と最小値によって定義される。

3 つの値を用いる境界のテストでは、境界上、および境界の前後の値を使用する。前述のテストの場合、上限境界のテストは、9.99、10、および 10.01 である。下限境界のテストは、0.99、1、および 1.01 である。2 つの値、または 3 つの値を用いるテストのどちらを使用するかの判断は、テスト対象のアイテムに関連するリスクに基づくべきである。リスクの高いアイテムに対しては 3 つの値を用いる方法を使用する。

適用

この技法は、すべてのテストレベルで適用でき、順序付けられた同値パーティションが存在するときに適している。この理由により、BVA 技法は EP 技法と一緒に使用することが多い。境界上および境界外の概念を考慮するために、順序付けられた同値パーティションが必要である。例えば、数値の範囲は、順序付けられたパーティションである。いくつかのテキスト文字列で構成されているパーティションは、辞書式順序などで順序付けされているかもしれないが、ビジネスの観点から順序付けが重要でない場合は、境界値に注目すべきではない。数字の範囲に加えて、境界値分析を適用できるパーティションには次のようなものがある。

- 非数値変数の数値属性(例えば、文字列長)
- 状態遷移図内のループを含む、ループ実行サイクル数
- 配列などの保存されたデータ構造における反復要素の数
- メモリなどの物理的オブジェクトのサイズ
- アクティビティの継続時間

制限／注意事項

この技法の正確性は、境界を正しく識別するために同値パーティションの正確な識別に依存するので、EP と同じ制限と注意事項の対象となる。また、テストアナリストは、テスト対象の

値を正確に決定できるようにするために、有効値および無効値の精度に注意する必要がある。境界値分析では順序付けられたパーティションのみを使用できるが、有効な入力の範囲に制限されるものではない。例えば、スプレッドシートでサポートされるセルの数をテストする場合、許容される最大数(境界)を含むそれまでのすべてのセルで構成されるパーティションと、最大数を 1 つ超えたセルで始まるパーティション(境界外)が存在する。

カバレッジ

カバレッジは、テスト済みの境界条件の数を、(2 つの値を用いる方法または 3 つの値を用いる方法のいずれかで)識別された境界条件の数で除算することにより決定し、パーセンテージで表される。

検出できる欠陥の種類

境界値分析は、境界のずれ、または欠落を高い信頼性で発見する。さらには、余分な境界を見つけることもある。この技法は、特に「より小さい」および「より大きい」のロジックのエラー(ずれ)など、境界値の処理に関連する欠陥を発見するために使用する。また、例えば、システムは 10,000 人の同時ユーザーをサポートするが 10,001 人の同時ユーザーはサポートしないなどの非機能欠陥を発見するためにもこの技法を使用する。

3.2.3 デシジョンテーブルテスト

デシジョンテーブルは、条件のセットとそれに関連するアクションを表形式で示しており、どの条件値のセットに対してどのアクションが発生するかを示すルールを表現している[OMG-DMN]。テストアナリストは、デシジョンテーブルを使用して、テスト対象のソフトウェアに適用されるルールを分析し、そのルールをカバーするテストを設計することができる。

テスト対象の条件とその結果としてのアクションがデシジョンテーブルの行を構成し、通常、条件が上部に、アクションが下部となる。表の最初の列には、条件とアクションの説明が書かれる。それに続く列はルールと呼ばれ、条件の値とそれに対応するアクションの値を記載する。

条件が「真」と「偽」というシンプルな値を持つブール型のデシジョンテーブルは、制限指定デシジョンテーブルと呼ばれる。条件の例としては、「ユーザーの収入 < 1000」がある。拡張指定デシジョンテーブルでは、条件が個別の要素または要素のセットを表す複数の値を持つことを許容する。例えば、「ユーザーの収入」という条件は、「1000 未満」、「1000 から 2000 の間」、「2000 を超える」という 3 つの値のいずれかをとることができる。

シンプルなアクションは、ブール値「真」と「偽」をとる(例えば、アクション「値引き許容 = 20%」は、アクションが発生すべき「真」の場合は「X」で示され、発生しない「偽」の場合は「-」で示

される)。条件と同様に、アクションも他の領域の値をとることがある。例えば、「値引き許容」というアクションは、0%, 10%, 20%, 35%, 50% の 5 つ値のうちの 1 つをとることができる。

デシジョンテーブルテストは、仕様書に基づいてデシジョンテーブルを設計することから始まる。条件値の実現不可能な組み合わせを含むルールは除外するか、「実現不可能」とマークする。次に、テストアナリストは、他のステークホルダーとデシジョンテーブルをレビューする。テストアナリストは、デシジョンテーブル内のルールが一貫していること(ルールが重複していないこと)、完全であること(実現可能な条件値の組み合わせごとにルールが含まれていること)、正しいこと(意図した振る舞いをモデル化していること)を確認する。

デシジョンテーブルテストの基本原則は、ルールがテスト条件になるということである。

与えられたルールをカバーするテストケースを設計する際、テストアナリストは、テストケースの入力値がデシジョンテーブルの条件値と異なる可能性があることを認識すべきである。例えば、条件「年収 > 100,000(ドル) 」の 「真」となる 値は、直接適用できないかもしれず、テスト担当者の作業として、特定の会計年度に 100,000 を超える入金がある顧客の預金口座を定義することが必要となることがある。同様に、テストケースの期待結果は、デシジョンテーブルのアクションとは異なる場合がある。

デシジョンテーブルの準備ができたら、条件とアクションを満たすテスト入力値(および期待結果)を選択して、ルールをテストケースとして実装する必要がある。

デシジョンテーブルの簡単化

条件に応じて可能なすべての入力の組み合わせをテストしようとすると、デシジョンテーブルは非常に大きくなる。n 個の条件を持つ完全な制限指定デシジョンテーブルは、2^n 個のルールを持つ。組み合わせの数を体系的に減らす技法は、簡単化したデシジョンテーブルテスト [Mosley93] と呼ばれる。この技法を使用すると、同じアクションのセットを持つルールのグループは、いくつかの条件の差異がこのグループ内のアクションに影響を与えず、かつ他のすべての条件が変わらない場合にこのグループを 1 つのルールに縮小(簡単化)できる。このルールでは、関連性のない条件の値は「関係なし」と表記され、通常はダッシュ「-」を付ける。「関係なし」の値を持つ条件については、テストアナリストは、テスト実装のために任意の有効な値を指定することができる。

簡単化できるルールのもう 1 つの例は、ある条件値が他の条件値との組み合わせでは適用できない場合や、2 つ以上の条件が矛盾する値を持つ場合である。例えば、カード決済のデシジョンテーブルでは、「カードが有効」という条件が偽の場合、「PIN コードが正しい」という条件は適用できない。

簡単化したデシジョンテーブルは、完全なデシジョンテーブルに比べてルールの数がはるかに少ない場合があり、結果的にテストケースの数が少なくなり工数が軽減できる。もし、あるルールに「関係なし」という項目があり、そのルールをカバーするテストケースが 1 つしかない場合、そのルールでは、条件の中でテストできる値のうち 1 つしかテストされないため、他の値に含まれる欠陥が検出できないかもしれない。そのため、リスクレベルが高い場合には、テストアナリストはテストマネージャーと連携して、デシジョンテーブルを簡単化するのではなく、単一の条件値の実行可能な組み合わせごとにルールを分けて定義すべきである。

適用

デシジョンテーブルテストは、通常、統合テスト、システムテスト、および受け入れテストで適用する。また、コンポーネントに判定ロジックのセットに対する責務がある場合には、コンポーネントテストにも適用できる。この技法は、テスト対象をフローチャートやビジネスルールの表の形式で仕様化している場合に特に有効になる。

デシジョンテーブルは、要件定義の技法でもあり、要件仕様がすでにこの形式で定義されている場合もある。テストアナリストは、テスト設計を開始する前に、デシジョンテーブルのレビューに参加し、デシジョンテーブルを分析すべきである。

制限／注意事項

条件の組み合わせを考慮する際、特に要件が適切に定義されていない場合、または存在しない場合には、相互作用するすべての条件を見つけることは困難となる。デシジョンテーブルで考慮する条件を選択する際には、それらの条件の組み合わせの数が対処可能であるように注意しなければならない。最悪の場合、ルールの数は指数関数的に増えていくこととなる。

カバレッジ

この技法の一般的なカバレッジ基準は、デシジョンテーブルの各ルールを 1 つのテストケースでカバーすることである。カバレッジは、テストスイートでカバーされたルールの数と、実行可能なルールの総数の割合として測定する。

特に拡張指定デシジョンテーブルの場合には、境界値分析と同値分割法は、デシジョンテーブル技法と組み合わせることができる。順序付けられた同値パーティションが条件にすべて含まれている場合、境界値はルールやテストケースの追加をもたらす追加項目として使用することができる。

検出できる欠陥の種類

典型的な欠陥としては、特定の条件の組み合わせに基づく論理的な処理が正しく行われず、想定外の結果となることが挙げられる。デシジョンテーブルを作成していると、仕様書に欠陥が見つかることがある。準備した条件のセットに含まれる 1 つ以上のルールに対する期待結果が不明確であることは珍しくない。 最も一般的な欠陥は、アクションの欠落（ある状況下で実際に起こるべきことに関する情報がない）と矛盾である。

3.2.4 状態遷移テスト

状態遷移テストは、テスト対象が有効な遷移を介して定義された状態に出入りできるかをテストするために使用する。有効な遷移と同様に、無効な状態に入るのを試す、または無効な遷移を実行することを試すために使用する。イベントの発生により、テスト対象はある状態から別の状態に遷移したり、アクションを実行したりする。イベントは、選択したい遷移パスに影響を与える条件（ガード条件または遷移ガードとも呼ばれる）を考慮して特定することがある。例えば、ログインイベントの場合、有効なユーザー名およびパスワードの組み合わせを使用したものと、無効なパスワードを使用したものとは異なる遷移結果となる。この情報は、状態遷移図または状態遷移表（状態間の潜在的な無効遷移を含む場合もある）で表す[Black09]。

適用

状態遷移テストは、定義済みの状態を持ち、それらの状態間の遷移（例えば、画面の変更）を引き起こすイベントを持つすべてのソフトウェアに適用できる。また、すべてのテストレベルで使用できる。組込みソフトウェア、Web ソフトウェア、および任意の種類のトランザクションソフトウェアが、この種類のテストの適切な適用対象である。信号機制御などの制御システムも同様である。

制限／注意事項

状態を決定することは、状態遷移図または状態遷移表を定義する作業の最も困難な部分である。テスト対象にユーザーインターフェースが含まれる場合、ユーザー向けに表示される様々な画面を状態として表すことが多い。組込みソフトウェアの場合、状態はハードウェアの状態に依存することがある。

状態そのものの他に、状態遷移テストの基本単位として、個々の遷移がある。単純に単一の遷移をすべてテストすれば、いくつかの状態遷移の欠陥を見つけることができるが、遷移のシーケンスをテストすることで、より多くの欠陥が見つかる可能性がある。1 つの遷移を 0 スイッチ、連続した 2 つの遷移のシーケンスを 1 スイッチ、連続した 3 つの遷移のシーケンスを 2 スイッチ、などと呼ぶ。一般に、N スイッチは N+1 個の連続した遷移を表す[Chow1978]。N が増えると、N スイッチの数は急速に伸びていき、妥当な少数のテストケースで N スイッチのカバレッジを達成するのは困難となる。

カバレッジ

他の種類のテスト技法と同様に、カバレッジの度合いには階層がある。許容できる最小限のカバレッジは、すべての状態に滞在し、すべての遷移を少なくとも一度は通過することである。100%の遷移カバレッジ(100%の 0 スイッチカバレッジとも呼ばれる)は、システム設計や状態遷移モデル(図や表)に欠陥がない限り、すべての状態に滞在し、すべての遷移を通過することを保証する。状態と遷移の関係によっては、他の遷移を 1 回実行するために、ある遷移に対する複数回の通過を必要とする場合もある。

用語「N スイッチカバレッジ」は、長さ N+1 でカバーされたスイッチの数を意味し、その長さのスイッチの総数のパーセンテージで表す。例えば、100%の 1 スイッチカバレッジを達成するには、2 つの連続する遷移のすべての有効なシーケンスを、1 回以上テストする必要がある。このテストでは、100%の 0 スイッチカバレッジで見落としがちな故障の種類のいくつかを見つけ出すことができる。

「ラウンドトリップカバレッジ」は、遷移のシーケンスがループを形成するときに適用する。100%のラウンドトリップカバレッジを達成するには、任意の状態から同じ状態に戻るすべてのループを、ループが開始および終了するすべての状態に対してテストする。このループには、開始状態と終了状態を除くすべての特定の状態が複数回含まれていてはならない[Offutt16]。

より高い度合いのカバレッジを達成するには、上記のいずれのやり方だとしても、状態遷移表で識別されるすべての無効な遷移を含めることとなる。状態遷移テストのカバレッジ要件およびテストセットは、無効な遷移を含んでいるかどうかを識別しなければならない。

特定のテスト対象の状態遷移図または状態遷移表は、望ましいカバレッジを達成するためのテストケースの設計を支援する。この情報は、特定の「N」値の N スイッチ遷移を示す表によって表すこともできる[Black09]。

カバー対象のアイテム(遷移、状態、N スイッチなど)を手動で識別できる場合もある。1 つの方法として、状態遷移図および状態遷移表を印刷し、必要なカバレッジが示されるまで、カバー対象のアイテムを筆記用具でマークする[Black09]。この手法は、状態遷移図および状態遷移表の複雑度の度合いが高まると、非常に時間のかかる作業となる。このため、状態遷移テストでは、ツールを使用すべきである。

検出できる欠陥の種類

典型的な欠陥を次に示す([Beizer95]も参照すること)。

- 正しくないイベントタイプまたは値
- 正しくないアクションタイプまたは値

- 正しくない初期状態
- 特定の終了状態に遷移できない
- 必須状態に遷移できない
- 余分な(不要な)状態
- 特定の有効な遷移を正しく実行できない
- 無効な遷移を実行できる
- 間違ったガード条件

状態遷移モデルを作成するときに、仕様ドキュメントの欠陥を発見することがある。欠陥の最も一般的な種類は、欠落(つまり、特定の状況で実際に何が発生するかに関する情報が存在しない)および矛盾である。

3.2.5 クラシフィケーションツリー技法

クラシフィケーションツリーは、テスト対象に適用するために作成するデータ空間を視覚的に表すことができるので、特定のブラックボックステスト技法を支援する。

これらのデータは、次のようにクラシフィケーションとクラスにまとめることができる。

- クラシフィケーション:テスト対象のためのデータ空間内のパラメーターを表すもので、入力パラメーター(さらに環境状態や事前条件を含むこともある)や出力パラメーターなどがある。例えば、多くの異なる方法でアプリケーションを構成できる場合、クラシフィケーションには、クライアント、ブラウザー、言語、およびオペレーティングシステムを含むことがある。
- クラス:各クラシフィケーションは、パラメーターの出現を表す任意の数のクラスとサブクラスを持つことができる。各クラス、または同値パーティションは、クラシフィケーション内の特定の値となる。前述の例では、言語クラシフィケーションは、英語、フランス語、およびスペイン語の同値パーティションを含むことがある。

テストアナリストは、適切だと思われる組み合わせをクラシフィケーションツリーに入力することができる。これには、例えばペアワイズ組み合わせ(3.2.6 節参照)、3 ワイズ組み合わせ、およびシングルワイズが含まれる。

クラシフィケーションツリー技法の使用に関する詳細については、[Bath14]および[Black09]を参照されたい。

適用

テストアナリストはクラシフィケーションツリーを作成することで、関心のあるパラメーター(クラシフィケーション)と同値パーティション(クラス)を識別できる。

クラシフィケーションツリー図をさらに分析することで、可能な境界値を識別できる。そして、特に関心の対象となる、または無視可能となる入力の特定の組み合わせ（例えば、両立しないことによる）を識別できる。作成したクラシフィケーションツリーは、同値分割法、境界値分析、またはペアワイズテスト（3.2.6 節参照）に役立てることができる。

制限／注意事項

クラシフィケーションそして/またはクラスの量が増加するにつれて、図が大きくなり、使用しづらくなる。また、クラシフィケーションツリー技法では、完全なテストケースは作られず、テストデータの組み合わせのみが作られる。テストアナリストは、完全なテストケースを作成するために各テストの組み合わせの結果を提供しなければならない。

カバレッジ

例えば、最小のクラスカバレッジを達成するようにテストケースを設計する場合がある（つまり、クラシフィケーション内のすべての値を少なくとも 1 回以上テストする）。テストアナリストは、ペアワイズの組み合わせをカバーするか、または 3 ワイズのような他の種類の組み合わせテストを使うことを決める場合もある。

検出できる欠陥の種類

検出できる欠陥の種類は、クラシフィケーションツリーがサポートする技法（つまり、同値分割法、境界値分析、またはペアワイズテスト）によって異なる。

3.2.6 ペアワイズテスト

ペアワイズテストは、可能な値を複数持つ複数の入力パラメーターを組み合わせてソフトウェアをテストしなければならず、組み合わせ数が、許容される時間内にテスト可能な数よりも多く存在するときに使用する。入力パラメーターは、任意の因子における任意のオプション（すなわち、任意の 1 つの入力パラメーターの任意の選択された値）が任意の他の因子における任意のオプションと組み合わせることができるという意味で独立しているかもしれないが、常にそうであるとは限らない（下記のフィーチャーモデルに関する記述を参照）。特定のパラメーター(変数または因子)とそのパラメーターの特定の値の組は、パラメーター－値のペアと呼ばれる(例えば、「色」が「赤」を含む 7 つの許容値を持つパラメーターであれば、「色＝赤」がパラメーター－値のペアとなりうる)。

ペアワイズテストでは、組み合わせの技法を使用して、各パラメーター－値ペアが他の各パラメーターのパラメーター－値ペアそれぞれに対して 1 回はテストをする（つまり、任意の 2 つの異なるパラメーターのパラメーター－値ペアの「オールペア」をテストする)ことで、パラメーター－値ペアのすべての組み合わせをテストすることは避けている。テストアナリストが手動で表を作る方法を使用する場合、行をテストケースとして表現し、各パラメーターに対して

1 つの列となる。テストアナリストは、すべての値のペアが表の中で識別できるように、表に値を入力する ([Black09]を参照)。表の中で空欄になっている項目は、テスト アナリストが自身のドメイン知識を使って値を入力することができる。

テストアナリストがこのタスクを行うにあたり、支援するツールが多数利用可能である (サンプルは www.pairwise.org を参照)。これらのツールは、入力として、パラメーターとその値のリストを必要とする。そして、各パラメーターの値の組み合わせの適切なセットを生成する。このセットは、パラメーター – 値ペアのすべての組をカバーする。このツールの出力は、テストケースの入力として使用できる。テストアナリストはツールによって生成された各組み合わせに対する期待結果を提供しなければならないことに注意する。

クラシフィケーションツリー (3.2.5 節参照) はペアワイズテストと組み合わせて使用することが多い[Bath14]。クラシフィケーションツリーの設計は、ツールで支援されており、パラメーターとそれらの値の組み合わせを視覚化できる (ツールによってはペアワイズの拡張機能を提供しているものもある)。これは、次の情報を識別するのに役立つ。

- ペアワイズテスト技法で使用する入力。
- 関心のある特定の組み合わせ (頻繁に使用する組み合わせ、欠陥の共通の発生源など)。
- 両立しない特定の組み合わせ。これは、組み合わされた因子が相互に影響しないことを想定していない。影響を与える可能性が十分にあり、許容できる範囲で相互に影響するべきである。
- 変数間の論理的関係。例えば、「変数 x が 1 の場合、変数 y が 2 になることはない」。これらの関係を表すクラシフィケーションツリーを、「フィーチャーモデル」と呼ぶ。

適用

パラメーター値に関してあまりに多くの組み合わせを持つことの問題は、テスト時に少なくとも 2 つの異なる局面で現れる。例えば複数の入力欄のある画面のように、一部のテストアイテムには、とり得る値が複数存在するパラメーターが複数存在する。この場合、パラメーター値の組み合わせがテストケースの入力データとなる。さらに、システムによっては、いくつかの次元で設定可能なものがあり、その結果、構成空間が大きくなる可能性がある。いずれの状況でも、ペアワイズテストにより、扱いやすくて実現可能な組み合わせのサブセットを識別することができる。

値の数が非常に多いパラメーターでは、まず、同値分割法や他の抽出の仕組みを各パラメーターに適用して、各パラメーターの値の数を減らしてからペアワイズテストを適用して、結果として得られる組み合わせのセットを減らすことができる。パラメーターとその値をクラシフィケーションツリーに取り込むことがこの活動を支援する。

これらの技法は通常、コンポーネント統合テスト、システムテストおよびシステム統合テストのテストレベルで適用する。

制限／注意事項

これらの技法の主な制限は、少数のテストの結果がすべてのテストの結果を代表し、それらのテストが期待される使用方法を表すと仮定することである。例えば、想定されていない相互作用が特定の変数間に存在する場合、その特定の組み合わせをテストしない限り、その欠陥はこのテスト技法で検出されない可能性がある。これらの技法は、テストを論理的に削減するということが理解されづらいため、技術を知らない関係者への説明が難しい。そのような説明は、経験に基づく研究の結果に言及することによってバランスをとる必要がある[Kuhn16]。医療機器の領域の研究では、故障の 66%は単一の変数によって引き起こされており、97%は 1 つの変数または 2 つの変数の相互作用のいずれかによって引き起こされていることが示されている。3 以上の変数による相互作用が存在する場合、ペアワイズテストでシステム故障を検知できないことがあるというリスクが残る。

パラメーターとそれぞれの値を識別するのが困難な場合がある。このため、このタスクは、可能であればクラシフィケーションツリーの支援を受けて実施するべきである（3.2.5 節参照）。あるレベルのカバレッジを満たす最小の組み合わせのセットを見つけることは、手作業では困難である。可能な限り小さい組み合わせのセットを見つけるために、ツールを使用することができる。ツールの中には、最終的な組み合わせの選択をする際に、いくつかの組み合わせを強制的に含めたり、除外したりする機能をサポートしているものがある。テストアナリストは、この機能を使用することにより、ドメイン知識またはプロダクトの使用情報に基づいて、因子を重要として扱うかどうかを選択できる。

カバレッジ

ペアワイズカバレッジ 100%とするためには、任意の 2 つのパラメーターについて、各値のすべてのペアを 1 つ以上の組み合わせに含める必要がある。

検出できる欠陥の種類

このテスト技法で見つかる最も一般的な欠陥のタイプは、2 つのパラメーターの値の組み合わせに関連する欠陥である。

3.2.7 ユースケーステスト

ユースケーステストは、ユースケースで具体的にした、コンポーネントまたはシステムの意図される使われ方をエミュレートするトランザクションベースおよびシナリオベースのテストを可能にする。何らかのゴールを達成するために、アクターと、コンポーネントまたはシステムとの

間の相互作用の観点でユースケースを定義する。アクターとしては、ユーザー、外部ハードウェア、またはその他のコンポーネントやシステムを挙げることができる。

ユースケースの共通規格は、[OMG-UML]で提供している。

適用

ユースケーステストは、通常、システムテストレベルおよび受け入れテストレベルで適用する。また、コンポーネントまたはシステムの振る舞いがユースケースで指定されている場合には、統合テストにも使用することがある。ユースケースはシステムの実際の使い方を表すので、性能テストのベースとなることが多くある。ユースケースが示すシナリオを仮想ユーザーに割り当てることで、システムに現実的な負荷をかけることがある(負荷および性能要件がユースケース内、またはユースケースのために仕様化されている場合)。

制限／注意事項

有効なユースケースは、現実的なユーザートランザクションを反映する必要がある。ユースケースの仕様は、システム設計の形式の 1 つとなる。ユーザーが達成する必要のある要件は、ユーザーまたはユーザーの代表者から入手し、対応するユースケースを設計する前に組織的な要件と照らし合わせてチェックする必要がある。ユースケースが実際のユーザーか組織の要件を反映していない場合、またはユーザータスクの完了を支援ではなく妨害をする場合、ユースケースの価値は低下する。

カバレッジを十分なものにするには、例外、代替、およびエラー処理に関する振る舞いを正確に定義することが重要となる。ユースケースは、ガイドラインとしてとらえるべきであるが、要件のセット全体を明確に定義するものではない可能性があるため、テストすべき内容を完全に定義するものとはならない。テストの精度を高め、ユースケース自体を検証するためには、ユースケースの記述からフローチャートそして/またはデシジョンテーブルなど他のモデルを作成することも有益なことがある。これによって、他の形式の仕様と同じように、ユースケース仕様に論理的な不正が存在する場合、それらが明らかになる可能性がある。

カバレッジ

ユースケースの受け入れ可能な最小カバレッジレベルは、基本的な振る舞い用の 1 つのテストケースと、代替およびエラー処理の振る舞いのそれぞれをカバーするのに十分な追加テストケースを用意することである。最小テストスイートが必要な場合、複数の代替の振る舞いが相互に互換性があればそれらを 1 つのテストケースに組み込むことができる。診断能力を高める必要がある場合(例えば、欠陥を特定しやすくするため)、代替の振る舞いごとに1つのテストケースを追加で設計することがある。ただし、入れ子になる代替の振る舞いでは、いくつかを単一のテストケースに融合する必要がでてくる(例えば、「リトライ」をする例外の振る舞い内での「終了」と「非終了」の代替の振る舞いなど)。

検出できる欠陥の種類

欠陥には、定義済みの振る舞いの誤った処理、代替の振る舞い欠落、提示された条件の誤った処理、十分に実装されていない、または不適切なエラーメッセージがある。

3.2.8 技法の組み合わせ

テストケースを作成するために技法を組み合わせることがある。例えば、デシジョンテーブルにて識別したある条件に対して、同値分割法にて 1 つの条件を満たす複数の方法を発見することができる。テストケースは、すべての条件の組み合わせをカバーするだけでなく、パーティションで分けられる条件については、同値パーティションをカバーするために追加のテストケースを生成すべきである。テストアナリストは、適用する特定の技法を選択する際に、その技法の適用可能性、制限や困難さ、カバレッジや検出すべき欠陥の観点からのテストのゴールといったことを考慮すべきである。これらの点については、本章で扱う個々の技法にて説明している。ある状況下では、単一の「最善」の技法は存在しないかもしれない。技法を正しく適用するための十分な時間とスキルがあると仮定すれば、技法を組み合わせることで、最も完全なカバレッジが得られる。

3.3 経験ベースのテスト技法

経験ベースのテストは、欠陥検出を向上させるために、テスト担当者のスキルと直感、およびテスト対象に類似のアプリケーションや技術での経験を活用する。これらのテスト技法の範囲は、テスト担当者が実行する活動を事前に計画していない「クイックテスト」から、テストチャーターを使う事前に計画したテストセッション、さらにはスクリプト化されたテストセッションにまで及ぶ。これらは常に有効であり、さらに次の長所一覧に含まれる側面が達成できる状況では特別な価値を提供する。

経験ベースのテストには、次の長所がある。

- システムに関するドキュメントが不足する場合に、構造化されたアプローチに対する優れた代替となることができる。
- テスト時間が厳しく制限されている場合に適用できる。
- ドメインおよび技術で利用可能な専門知識を、テストに適用できる。これには、例えばビジネスアナリスト、顧客、クライアントなど、テストに関与しない人たちからの専門知識も含まれることがある。
- 開発者へ早期にフィードバックを提供できる。
- 作成するソフトウェアに関してチームが精通するのに役立つ。
- 運用上の故障を効果的に分析できる。
- 多様なテスト技法を適用できる。

経験ベースのテストには、次の短所がある。

- 詳細なテストドキュメントを必要とするシステムでは不適切なことがある。

- 高いレベルの再現性を達成することは困難である。
- テストカバレッジを精密に評価できない。
- テストケースがその後の自動化にあまり適さない。

対処的またはヒューリスティックなアプローチでは、テスト担当者は通常、事前に計画するテストよりも、より対処的な経験ベースのテストを使用する。また、実行と評価を同時に行う。経験ベースのテストに対する一部の構造化されたアプローチは、完全に動的であるわけではない。つまり、テスト担当者がテストを実行するときに、すべてのテストを作成するわけではない。これは、例えば、テストを実行する前に、エラー推測を使用してテスト対象の特定の側面をターゲットにする場合が該当する。

ここで説明する技法に関して、カバレッジについていくつかのアイデアを提示するが、経験ベースのテスト技法には、公式なカバレッジ基準が存在しないことに注意する必要がある。

3.3.1 エラー推測

エラー推測技法を使用する場合、テストアナリストは経験を活かして、コードの設計および開発時に発生した可能性のある潜在的なエラーを推測する。テストアナリストは想定されるエラーを識別したら、そのエラーの結果として生じる欠陥を明らかにするために使用する最善の方法を決定する。例えば、テストアナリストが無効なパスワードを入力するとソフトウェアが故障を表示すると想定した場合、エラーが実際に発生し、そのエラーがテストの実行時に故障として認識される欠陥をもたらすかどうかを検証するために、様々に異なる値をパスワード欄に入力するようにテストを実行する。

エラー推測は、テスト技法として使用されるだけでなく、潜在的な故障モードを識別するためのリスク分析時にも役立つ[Myers11]。

適用

エラー推測は、主として、統合テストおよびシステムテスト時に実行するが、すべてのテストレベルで使用できる。この技法は、多くの場合、他の技法と組み合わせて使用する。また、既存のテストケースの範囲を拡大するのに役立つ。エラー推測は、ソフトウェアを新しくリリースするときに、より厳密な手続き化されたテストを開始する前に、一般的な欠陥をテストする場合にも効果的に使用できる。

制限／注意事項

エラー推測には、次の制限と注意事項が適用される。

- カバレッジを評価することは困難であり、テストアナリストの能力と経験に応じて大きく異なる。

- テスト対象のコードの種類に共通して埋め込まれる欠陥の種類に精通している経験を積んだテスト担当者が使用する場合に最適である。
- 一般的に使用されているが、文書化しないことが多く、他の形式のテストに比べて再現性が低いことがある。
- テストケースは文書化される場合があるが、作成者だけが理解して再現できる形である。

カバレッジ

欠陥分類法を使用すると、テスト済みの分類アイテムの数を分類アイテムの総数で割ってカバレッジをパーセンテージで表すことで判定できる。欠陥分類法を使用しない場合は、テスト担当者の経験と知識や利用可能な時間によりカバレッジが制限される。この技法で検出できる欠陥の量は、テスト担当者が問題のある領域を対象にできる能力に依存して異なる。

検出できる欠陥の種類

典型的な欠陥は、通常、特定の欠陥分類法で定義されているか、テストアナリストが「推測」する。これらの欠陥は、ブラックボックステストでは検出されないことがある。

3.3.2 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技法を適用する場合、経験を積んだテストアナリストは、メモ、チェック、または記憶するための項目をハイレベルで汎用化した一覧や、テスト対象に対して検証を行う一連のルールおよび基準を使用する。これらのチェックリストは、一連の標準、経験、および他の考慮事項に基づいて構築する。例えば、ユーザーインターフェースの標準チェックリストをアプリケーションをテストするベースとして使用できる。 アジャイルソフトウェア開発では、チェックリストはユーザーストーリーの受け入れ基準から構築できる。

適用

チェックリストベースドテストは、テスト対象のソフトウェア、またはチェックリストによりカバーされる領域に精通している経験を積んだテストチームがいるプロジェクトで、最も効果的である（例えば、ユーザーインターフェースチェックリストを正しく適用するために、テストアナリストはユーザーインターフェースのテストに精通しているが、テスト対象の特定のソフトウェアには精通していないことがある）。チェックリストはハイレベルであり、テストケースおよびテスト手順で一般的に見られる詳細な手順が不足している傾向にある。このため、テスト担当者の知識を使用してギャップを埋めることになる。詳細な手順が省略されているため、チェックリストはメンテナンスに手間がかからず、複数の類似リリースに適用できる。

チェックリストは、ソフトウェアのリリースおよび変更が頻繁に行われるプロジェクトに非常に適している。これにより、テスト文書の準備およびメンテナンスの両方の時間を削減するのに役

立つ。これらは、すべてのテストレベルで使用でき、リグレッションテストおよびスモークテストにも使用できる。

制限／注意事項

ハイレベルのチェックリストという特性は、テスト結果の再現性に影響を及ぼす可能性がある。テスト担当者によりチェックリストの解釈が異なり、チェックリストの項目を満たすために異なる確認方法をとる可能性がある。これにより、同じチェックリストを使用しても、異なるテスト結果となる可能性がある。これは、より広いカバレッジを可能にするが、再現性が犠牲になることがある。チェックリストを用いると、実際のテストではテスト担当者の判断に依存するため、達成するカバレッジの度合いに関して、過信をもたらすこともある。チェックリストは、より詳細なテストケースまたはテストリストから導き出すことができ、時間とともに増加する傾向にある。テスト対象のソフトウェアの重要な側面をチェックリストが確実にカバーするように、メンテナンスを行う必要がある。

カバレッジ

カバレッジはテスト済みのチェックリストアイテムの数をチェックリストアイテムの総数で割ってカバレッジをパーセンテージで表すことで判定できる。カバレッジはチェックリストと同様であるが、チェックリストがハイレベルであるという特性により、結果は、チェックリストを実行するテストアナリストによって異なる可能性がある。

検出できる欠陥の種類

この技法で見つかる典型的な欠陥は、テスト時のデータ、手順の順序、全般的なワークフローなどが異なることにより、故障をもたらす欠陥である。

3.3.3 探索的テスト

探索的テストには、テスト担当者がテスト対象とその欠陥についての学習、完了すべきテスト作業の計画、テストの設計と実行、および結果の報告を同時に行うという特徴がある。テスト担当者は、テスト実行時にテストのゴールを動的に調整し、軽量のドキュメントのみを準備する[Whittaker09]。

適用

優れた探索的テストは、しっかりと計画されており、対話的で創造的である。探索的テストは、テスト対象のシステムに関するドキュメントをほとんど必要としないため、ドキュメントが入手できない、あるいは他のテスト技法では十分ではない状況で使用することが多い。探索的テストは、他のテスト技法に追加して、追加のテストケースを開発するためのベースとして使用することが多い。探索的テストは、アジャイルソフトウェア開発において、最小限のドキュメ

ントのみでユーザーストーリーテストを柔軟かつ迅速に行うために頻繁に使用される。さらに、この技法は、シーケンシャル開発モデルを用いたプロジェクトにも適用可能である。

制限／注意事項

探索的テストのカバレッジはまちまちでまとまりがなく、実行したテストを再現するのが困難になる可能性がある。テストセッションでカバーする領域を指定したテストチャーターや、テストで使える時間を決定するタイムボックスは、探索的テストをマネジメントする技術である。1 回のテストセッション終了時、または複数のテストセッション終了時に、テストマネージャーは状況報告のセッションを開催し、テスト結果をとりまとめ、次のテストセッションのテストチャーターを決定することができる。

その他に、探索的テストセッションは、テストマネジメントシステムで正確に追跡することも困難である。実際には探索的テストセッションになるものをテストケースとして、テストマネジメントシステムで追跡することがある。これにより、探索的テストに割り当てられた時間や計画したカバレッジを別のテスト活動から追跡できる。
探索的テストは再現するのが困難であるため、故障を再現するための手順を思い出す必要がある場合にも問題を引き起こすことがある。ある組織は、テスト自動化ツールのキャプチャ/プレイバック機能を使用して、探索的テストの担当者が実施する手順を記録する。この方法は、探索的テストセッション（またはすべての経験ベースのテストセッション）時のすべての活動を完全に記録する。詳細を分析して、故障の実際の原因を見つけることは、単調で時間のかかる作業であるが、少なくとも関与したすべての手順の記録が残る。

他のツールを使用して探索的テストセッションを記録することは可能であるが、GUI 操作は記録されないので、期待結果を記録できない。この場合、期待結果を書き取り、必要に応じて欠陥を適切に分析できるようにする必要がある。また、探索的テストの実行中にも、必要に応じて再現できるように書き取っておくことが、一般的に推奨される。

カバレッジ

テストチャーターは、特定のタスク、目的、および成果物のために設計することができる。その後、それらの基準を達成するために、探索的テストセッションを計画する。チャーターはまた、テスト工数をどこに集中させるか、テストセッションの範囲内および範囲外はどこか、計画したテストを完了するためにどの程度リソースを投入すべきかも識別することもできる。テストセッションでは、特定の欠陥タイプやその他の潜在的な問題点に焦点を当て、スクリプトテストの形式にとらわれずに対処することができる。

検出できる欠陥の種類

探索的テストで見つかる典型的な欠陥には、スクリプトによる機能適合性テストで見逃されたシナリオベースの問題、機能境界間に存在する問題、およびワークフロー関連の問題がある。性能問題やセキュリティ問題が、探索的テストで見つかることもある。

3.3.4 欠陥ベースのテスト技法

欠陥ベースのテスト技法は、探したい欠陥のタイプをテスト設計のベースとして使用する技法であり、すでに知られている欠陥のタイプからテストを体系的に導き出す。テストケースをテストベースから導き出すブラックボックステストと異なり、欠陥ベースのテストでは、欠陥に着目した一覧からテストを導き出す。この一覧は一般的に、欠陥のタイプ、根本原因、故障の兆候、および欠陥に関連するその他のデータの一覧に整理できる。標準の一覧は、複数のタイプのソフトウェアに適用され、プロダクトに依存しない。この一覧を使用することは、業界標準の知識を活用して特定のテストケースを導き出すのに役立つ。業界固有の一覧に従うことで、プロジェクト全体、さらには組織全体で、欠陥発生に関する指標を追跡することができる。最もよくある欠陥一覧は、組織やプロジェクトごとに作成され、特定の専門知識や経験を利用したものである。

欠陥ベースのテストでは、識別したリスクおよびリスクシナリオの一覧を、テストする対象を絞るためのベースとして使用することもある。テストアナリストは、このテスト技法を使用することにより、特定の欠陥のタイプに対象を絞ったり、特定のタイプの既知および一般的な欠陥の一覧を通して体系的に作業したりできる。また、この情報から、テストアナリストは、欠陥が存在する場合にその欠陥を顕在化させるようなテストケースとテスト条件を作成する。

適用

欠陥ベースのテストは、すべてのテストレベルに適用できるが、一般的に、システムテストに適用する。

制限／注意事項

複数の欠陥分類法が存在し、これらには、使用性など特定のタイプのテストに焦点を当てたものもある。複数の欠陥分類法に利用可能なものがある場合、テスト対象のソフトウェアに適用できる分類法を選択することが重要である。例えば、先進的なソフトウェアに対しては利用できる分類法が存在しないことがある。一部の組織は、発生しやすく、頻繁に見られる欠陥の分類法を独自に作成している。使用する分類法に関係なく、テストを開始する前に期待するカバレッジを定義することが重要である。

カバレッジ

この技法は、すべての有用なテストケースを識別したかどうかを決定するのに使用できるカバレッジ基準を提供する。カバレッジアイテムは、欠陥一覧に応じて、構造要素、仕様要素、使用シナリオ、これらのいずれかの組み合わせである可能性がある。実際問題として、欠陥ベースのテスト技法のカバレッジ基準は、カバレッジのための一般的なルールのみが与えられ、有用なカバレッジの範囲がどの部分までかについての具体的な決定は裁量に任されているという点で、ブラックボックステスト技法に比べてあまり体系的ではない傾向にある。他の

技法と同じく、カバレッジ基準は、すべてのテストセットが完全であることを意味するのではなく、考慮された欠陥からは、この技法に基づいた有用なテストケースがこれ以上ないことを意味する。

検出できる欠陥の種類

検出できる欠陥の種類は、通常、使用する欠陥分類法により異なる。例えば、ユーザーインターフェースの欠陥一覧を使用すると、検出する欠陥の大半はユーザーインターフェースに関連する。しかし、他の欠陥も特定のテストの副産物として検出できることもある。

3.4 最善の技法の適用

ブラックボックステスト技法および経験ベースのテスト技法は、一緒に使用すると最も効果的である。経験ベースの技法は、ブラックボックステスト技法のあらゆる体系的な弱点に起因するカバレッジのギャップを埋める。

すべての状況に対して完璧である技法は存在しない。テストアナリストは、各技法の長所と短所を理解し、プロジェクトの種類、スケジュール、情報へのアクセス、テスト担当者のスキル、および選択に影響する可能性のある要因を考慮して、特定の状況にとって最善の技法または複数の技法を選択できることが重要である。

ブラックボックスおよび経験ベースの各テスト技法の説明において(それぞれ 3.2 節および 3.3 節参照)、「適用」、「制限／注意事項」、および「カバレッジ」で説明されている情報は、適用する最善のテスト技法をテストアナリストが選択するのに役立つ。

4. ソフトウェア品質特性のテスト- 180 分

キーワード
アクセシビリティ、互換性、機能適切性、機能完全性、機能正確性、機能適合性、相互運用性、習得性、運用操作性、使用性測定一覧表（SUMI）、使用性、ユーザーエラー防止性、ユーザーエクスペリエンス、ユーザーインターフェース快美性、Web サイト解析と測定一覧表（WAMMI）

「ソフトウェア品質特性のテスト」の学習の目的

4.1 イントロダクション

学習の目的はなし

4.2 ビジネスドメインテストの品質特性

TA-4.2.1 （K2）機能完全性、機能正確性、および機能適切性の特性をテストする場合に、どのテスト技法が適切であるかを説明する。
TA-4.2.2 （K2）機能完全性、機能正確性、および機能適切性の特性に関して、対象とする典型的な欠陥を定義する。
TA-4.2.3 （K2）機能完全性、機能正確性、および機能適切性の特性に関して、これらの特性を、ソフトウェア開発ライフサイクル内でテストするタイミングを定義する。
TA-4.2.4 （K2）使用性の要件の実装と、ユーザーの期待の達成の両方を検証および妥当性確認するのに適している手法を説明する。
TA-4.2.5 （K2）統合テストでのテストアナリストの役割について、対象となる欠陥を識別する役割を含めて説明する。
TA-4.2.6 （K2）移植性テストでのテストアナリストの役割について、対象となる欠陥を識別する役割を含めて説明する。
TA-4.2.7 （K4）一連の指定された要件について、機能そして/または非機能の品質特性を検証するために必要なテスト条件を、テストアナリストの役割の範疇内で決定する。

4.1 イントロダクション

前述の章では、テスト担当者が使用できる特定の技法について説明したが、この章では、ソフトウェアアプリケーションまたはシステムの品質を特徴付ける特性を評価するために、それらの技法を適用する方法を説明する。

本シラバスでは、テストアナリストが評価することがある品質特性について説明する。テクニカルテストアナリストが評価する属性は、テクニカルテストアナリスト向けの Advanced Level シラバス[CTAL-TTA]で説明する。

本章では、ISO25010 [ISO25010]が定義しているプロダクト品質特性の説明を、特性を説明するためのガイドとして使用する。ISO のソフトウェア品質モデルでは、プロダクト品質は様々なプロダクト品質特性に分類され、各品質特性は副特性を持つ。これらを次の表に示す。次の表では、どの特性／副特性がテストアナリストシラバスおよびテクニカルテストアナリストシラバスで説明されているかを示す。

特性	副特性	テストアナリスト	テクニカルテストアナリスト
機能適合性	機能正確性、機能適切性、機能完全性	○	
信頼性	成熟性、障害許容性、回復性、可用性		○
使用性	適切度認識性、習得性、運用操作性、ユーザーインターフェース快美性、ユーザーエラー防止性、アクセシビリティ	○	
性能効率性	時間効率性、資源効率性、容量満足性		○
保守性	解析性、修正性、試験性、モジュール性、再利用性		○
移植性	適応性、設置性、置換性	○	○
セキュリティ	機密性、インテグリティ、否認防止性、責任追跡性、真正性		○
互換性	共存性		○
	相互運用性	○	

この作業の割り当ては組織によって異なる可能性があるが、関連する ISTQB®シラバスに従っている。

この節で説明するすべての品質特性および副特性について、典型的なリスクを認識して、適切なテスト戦略を形成および文書化する必要がある。品質特性のテストでは、SDLC でのタイミング、必要なツール、ソフトウェアとドキュメントの入手可能性、および技術的な専門知

識に特に注意を払う必要がある。それぞれの特性と固有のテストニーズに対応する戦略がなければ、テスト担当者は、スケジュールに組み込まれた適切な計画、ランプアップ（立ち上げ期間）、テスト実行の時間を確保できない可能性がある[Bath14]。これらのテストの中には、例えば使用性テストのように、特別な人材の配置、広範な計画、専用のラボ、特定のツール、特定のテストスキル、および、ほとんどの場合、膨大な時間が必要になることがある。状況によっては、使用性またはユーザーエクスペリエンスを専門にする別のグループが使用性テストを行うことがある。

テストアナリストは、より技術的な方法を必要とする品質特性のテストに責任を負わないことがあるが、他の特性に留意し、テストで重複する領域を理解することが重要である。例えば、性能テストに不合格となったテスト対象は、ユーザーが効率的に使用するには時間がかかりすぎる場合、使用性テストにも不合格となる可能性が高い。同じく、一部のコンポーネントで相互運用性の問題を抱えるプロダクトは、より基本的な問題が、環境が変化した場合に不明瞭になる傾向があるので、移植性テストを行うための準備が整っていない可能性があると言える。

4.2 ビジネスドメインテストの品質特性

テストアナリストは、主に機能適合性テストに重点を置く。機能適合性テストは、テスト対象が「何をするのか」に重点を置く。機能適合性テストのテストベースは、一般的に、要件、仕様、固有のドメイン知識、または暗黙のニーズである。機能適合性テストは、実施するテストレベルによって様々であり、SDLC の影響を受けることもある。例えば、統合テスト時に実施する機能適合性テストでは、単一の定義済み機能を実装するインターフェースコンポーネントの機能適合性をテストする。システムテストレベルでは、機能適合性テストはシステム全体の機能適合性をテストする。システムオブシステムズの場合、機能適合性テストは主に、統合したシステムを通したエンドツーエンドのテストに重点を置く。機能適合性テストでは、様々なテスト技法を使用する（第 3 章を参照）。

アジャイルソフトウェア開発の機能適合性テストでは、通常、次のテストを含む。

- 特定のイテレーションでの実装が計画されている特定の機能（ユーザーストーリーなど）のテスト
- 変更していないすべての機能に対するリグレッションテスト

この節で説明する機能適合性テストに加えて、テストアナリストの担当範囲の一部であり、（テスト対象が「どのように動作するのか」に重点を置く）非機能テスト領域と見なすことができる品質特性もある。

4.2.1 機能正確性テスト

機能正確性に関しては、明示的または暗黙的な要件にアプリケーションが準拠していることを検証する。また、計算の精度もテストすることがある。機能正確性テストでは、第 3 章で説明した多くのテスト技法を採用し、多くの場合、テストオラクルとして仕様または旧システムを使用する。機能正確性テストは、すべてのテストレベルで実施でき、データや状態の誤った処理を対象にする。

4.2.2 機能適切性テスト

機能適切性テストでは、その意図および指定された任務に対する一連の機能が適切であることを評価および妥当性確認する。このテストは、機能設計（ユースケース、ユーザーストーリーなど）に基づくことができる。機能適切性テストは、通常、システムテスト時に実施するが、統合テストの後半段階でも実施することがある。このテストで見つかる欠陥は、許容できると考えられる方法であってもシステムがユーザーのニーズを満たすことができないことを示している。

4.2.3 機能完全性テスト

機能完全性テストは、実装した機能の指定された任務およびユーザー目的に対するカバレッジを判定するために実行する。仕様アイテム（要件、ユーザーストーリー、ユースケースなど）と実装された機能性（機能、コンポーネント、ワークフローなど）の間のトレーサビリティは、必要となる機能完全性を判定するために不可欠である。機能完全性を測定する方法は、特定のテストレベルそして/または使用する SDLC によって異なる。例えば、アジャイルソフトウェア開発の機能完全性は、実装されたユーザーストーリーおよびフィーチャーに基づくことがある。システム統合テストでの機能完全性は、ハイレベルなビジネスプロセスのカバレッジに重点を置くことがある。

機能完全性の判定作業は一般的に、テストマネジメントツールを使用して行うことができるが、その場合、テストアナリストはテストケースと機能仕様アイテムの間のトレーサビリティを維持する必要がある。期待される度合いの機能完全性を達成できない場合、システムの実装が不完全であることが示唆される。

4.2.4 相互運用性テスト

相互運用性テストは、2 つ以上のシステムまたはコンポーネントが情報を交換できることを検証する。テストケースは、情報を交換し、その後交換した情報を使用できる能力に重点を置く。このテストでは、データ交換が適切に機能することを確認するために、ハードウェア、ソフトウェア、ミドルウェア、オペレーティングシステムなどのバリエーションを含む、すべての意図した対象環境をカバーする必要がある。現実的には、このようなことができるのは、比較的少

数の環境に限られるかもしれない。このため、相互運用性テストを行うのは、選択した代表的な環境グループに限定することがある。相互運用性向けにテストケースを仕様化するには、意図した対象環境の組み合わせを識別し、構成し、テストチームが利用できるようにする必要がある。その後、機能適合性のテストケースから、環境内に存在する様々なデータ交換箇所を動作させるテストケースを選択し、これらの環境をテストする。

相互運用性とは、様々なコンポーネントとソフトウェアシステムが互いにどのように相互作用するかということである。 優れた相互運用性を備えたソフトウェアは、大きな変更や非機能的な振る舞いへの重大な影響を必要とせずに、他の多くのシステムと統合できる。 変更回数と、それらの変更を実装してテストするために必要な工数を相互運用性の尺度として使用できる。

ソフトウェアの相互運用性をテストするには、例えば、次の設計上の特徴に重点を置くことができる。

- XML など業界標準の通信規格の使用
- 相互作用するシステムが通信上必要なものを自動的に検出し、それに従って調整する能力

相互運用性テストは、次に対して特に重要なことが多い。

- 市販の既成ソフトウェアプロダクトやツール
- システムオブシステムズに基づくアプリケーション
- モノのインターネット(Internet of Things(IoT))に基づくシステム
- 他のシステムとの接続を有する Web サービス

この種類のテストは、コンポーネント統合テスト時およびシステム統合テスト時に行う。システム統合レベルでは、この種類のテストは、開発完了したシステムが他のシステムとどの程度上手く相互作用するかを判定するために実施する。システムは、複数のレベルで相互運用を行う可能性があるので、テストアナリストはこれらの相互作用を理解し、様々な相互作用を動作させる条件を作成できなければならない。例えば、2 つのシステムがデータを交換する場合、テストアナリストはデータ交換を動かすために必要なデータとトランザクションを作成できなければならない。すべての相互作用が要件ドキュメントで明確に記述されているとは限らないことを覚えておくことが重要である。むしろ、これらの相互作用の多くは、システムアーキテクチャーおよび設計ドキュメントでのみ定義している。テストアナリストはそれらのドキュメントを調査して、システム間およびシステムとその環境との間の情報交換をしている箇所を特定し、すべてのテストを確実に行うことができるように準備しなければならない。同値分割法、境界値分析、デシジョンテーブル、状態遷移図、ユースケース、およびペアワイズテストなどの技法はすべて、相互運用性テストに適用できる。相互運用性テストで典型的に見つかる欠陥には、相互作用するコンポーネント間の不適切なデータ交換がある。

4.2.5 使用性評価

テストアナリストは、使用性の評価作業を調整および支援する立場にいることが多い。この役割には、使用性のテストケースを仕様化するか、ユーザーと一緒にテストケースを実施するモデレーターとしての役割が含まれる。テストアナリストはこれを効果的に行うために、これらの種類のテストに関連する主要な側面、ゴール、および手法を理解する必要がある。本節で説明されている内容以上の詳細については、ISTQB® Specialist Syllabus in Usability Testing[ISTQB_UT_SYL]を参照されたい。
ユーザーがシステムを使いにくいと感じる、または(エンターテイメントのためにソフトウェアを使用する場合など)好ましいユーザーエクスペリエンス(UX)を得られていない理由を理解することが重要である。この理由を理解するには、まず、「ユーザー」という用語が、IT の専門家から子ども、障害を持つ人に至るまで、広い範囲の様々な種類のペルソナを表すことを認識する必要がある。

4.2.5.1 使用性の側面

本節では、次の 3 つの側面について説明する。

- 使用性
- ユーザーエクスペリエンス(UX)
- アクセシビリティ

使用性

使用性テストでは、ユーザーインターフェースを介して任務をこなすユーザーの操作のしやすさに影響を与えるソフトウェア欠陥を対象としている。このような欠陥が存在すると、ユーザーは効果的に、効率的に、または満足感を持って意図したゴールに到達できない可能性がある。使用性の問題は、ユーザー側の混乱、エラー、遅延、またはユーザー側の任務が完了できないことにつながる。

以下に、使用性の副特性[ISO 25010]を示す。それぞれの定義については[ISTQB_GLOSSARY]を参照されたい。

- 適切度認識性(つまり、理解性)
- 習得性
- 運用性
- ユーザーインターフェース快美性(つまり、魅力性)
- ユーザーエラー防止性
- アクセシビリティ(以下を参照)

ユーザーエクスペリエンス(UX)

ユーザーエクスペリエンス評価が対象にするのは、直接操作だけでなく、テスト対象のユーザーエクスペリエンス全体である。これは、テスト対象を使用することで楽しみや満足感を得られるなどの要因がビジネスの成功にとって必須である場合などに特に重要である。

ユーザーエクスペリエンスに影響する典型的な要因を次に示す。

- ブランドイメージ(つまり、製造元に対するユーザーの信頼)
- 相互に作用する振る舞い
- テスト対象の使いやすさ(ヘルプシステム、サポート、およびトレーニングを含む)

アクセシビリティ

ソフトウェアを使用する際に、特定のニーズまたは制限を持つユーザーのために、ソフトウェアへのアクセシビリティを考慮することが重要である。これらのユーザーには、障害を持つ人が含まれる。アクセシビリティテストでは、ウェブコンテンツ・アクセシビリティ・ガイドライン(WCAG)などの関連する標準、および障害者差別禁止法(北アイルランド、オーストラリア)、2010 年平等法(イングランド、スコットランド、ウェールズ)、第 508 条(米国)などの法律を考慮する必要がある。アクセシビリティは、使用性と同じように、設計活動時に考慮しなければならない。テストは、多くの場合、統合テストレベルで行い始めて、システムテストおよび受け入れテストのテストレベルまで継続する。アクセシビリティを欠陥として判断するのは、通常、指定した規制またはソフトウェアに対して定義した標準をソフトウェアが満たさない場合である。

アクセシビリティを向上させるための典型的な手段は、障害を持つユーザーがアプリケーションを操作できるようにするために提供されている機会に焦点を当てる。これらには、次のようなものがある。

- 音声認識による入力
- ユーザーに提示されるテキスト以外のコンテンツに、同等の代替テキストがあることを保証すること
- コンテンツや機能を損なうことなく、テキストのサイズを変更可能にすること

アクセシビリティガイドラインは、テストで使用できる情報やチェックリストを提供しており、テストアナリストの活動に役立つ(アクセシビリティガイドラインのサンプルについては、[ISTQB_UT_SYL]を参照されたい)。さらに、Web ページで色覚異常者に対するガイドラインに違反する不適切な色の選択などのアクセシビリティの問題を、テスト担当者が識別するのに役立つツールやブラウザープラグインも利用できる。

4.2.5.2 使用性評価手法

使用性、ユーザーエクスペリエンス、およびアクセシビリティは、以下の手法の 1 つ以上を使用することによってテストできる場合がある。

- 使用性テスト
- 使用性レビュー
- 使用性の調査およびアンケート

使用性テスト

使用性テストは、ユーザーがシステムを使用して、または使用する方法を習得して、特定の状況で特定のゴールに到達できるという、使いやすさをテストする。使用性テストでは、次の項目を測定することに注力する。

- 有効性 – 利用者が指定された利用の状況で、正確かつ安全に、指定されたゴールを達成できるテスト対象の能力
- 効率性 – 特定の使用条件下で、有効性を達成することに関連し、ユーザーに適切な度合いの資源を使用させるテスト対象の能力
- 満足性 – 指定された利用の状況で、利用者を満足させるテスト対象の能力

使用性テストの設計と仕様化の作業は、テストアナリストが、使用性テストの専門スキルを持つテスト担当者、および人間中心の設計プロセスを理解している使用性設計エンジニアと協力して行うことが多い点に気を付けることが重要である。(詳細については、[ISTQB_UT_SYL]参照)。

使用性レビュー

ユーザーの利用度合いを増加させるのに役立つ使用性の観点から実施するテストの種類としては、インスペクションとレビューが挙げられる。これは、使用性の問題を SDLC の早期に要件仕様および設計で見つけることで、費用対効果を高める可能性がある。ヒューリスティック評価(使用性に関するユーザーインターフェース設計の体系的なインスペクション)は、設計における使用性の問題を発見するために使用でき、イテレーティブな設計プロセスの一部として取り組むことができる。この評価では、少数の評価者によりインターフェースを検証し、認識済みの使用性原則(「ヒューリスティック」)に適合していることを判断する。ユーザーインターフェースがより視覚的である場合は、レビューがより有効である。例えば、スクリーンショットのサンプルの方が、特定の画面で提供される機能の説明よりも、通常、理解および解釈しやすい。ドキュメントの使用性を適切にレビューするために視覚化が重要である。

使用性の調査およびアンケート

調査およびアンケートの技法は、システムを操作するときのユーザーの振る舞いに関する所見やフィードバックを収集するために適用する場合がある。SUMI(ソフトウェア使用性測定一覧表)や WAMMI(Web サイト解析と測定一覧表)など、公開されている標準的な調査方法を使用することにより、過去に行われた使用性測定のデータベースと比較してベンチマークを実施できる。さらに、SUMI は使用性に関する明確な測定値を提供するので、これらの測定値は、一連の完了基準／受け入れ基準を提供することができる。

4.2.6 移植性テスト

移植性テストは、ソフトウェアコンポーネントまたはシステムを新規インストールとして、もしくは既存環境から意図した環境へ移植できる度合いに関連する。

プロダクト品質特性の分類である ISO 25010 は、移植性に関する次の副特性を定義している。

- 設置性
- 環境適応性
- 置換性

移植性の特性に関してリスクを識別しテストケースを設計するタスクは、テストアナリストとテクニカルテストアナリストが協力して作業する（[ISTQB_ALTTA_SYL] 4.7 節参照）。

4.2.6.1 設置性テスト

設置性テストは、対象とする環境にソフトウェアをインストールおよびアンインストールするために使用するソフトウェアと、文書化された手順に対して行う。

テストアナリストが焦点を当てる典型的なテスト目的を以下に示す。

- ソフトウェアを様々な構成で正常にインストールできることを確認する。構成可能なパラメーターが大量に存在する場合、テストアナリストはペアワイズ技法を使用してテストを設計し、テストするパラメーターの組み合わせの数を削減して、関心のある特定の構成（頻繁に使用する組み合わせなど）に重点を置く。
- インストールおよびアンインストールの手順の機能正確性をテストする。
- インストールまたはアンインストールの手順の後に機能適合性テストを実行し、それらの手順で発生したあらゆる欠陥（正確ではない構成、機能が利用不能など）を検出する。
- インストールおよびアンインストールの手順における使用性の問題を識別する（分かりやすい手順説明やフィードバック/エラーメッセージが、手順の実行時にユーザーに提供されることを検証するなど）。
-

4.2.6.2 環境適応性テスト

環境適応性テストでは、意図したすべての対象となる環境（ハードウェア、ソフトウェア、ミドルウェア、オペレーティングシステム、クラウドなど）で所定のアプリケーションが正しく機能するように、効果的、かつ効率的に適応ができているかどうかを確認する。テストアナリストは、意図する対象の環境（様々なモバイルオペレーティングシステムのサポート対象バージョン、使用可能な様々なブラウザーの様々なバージョンなど）を識別し、これらの環境の組み合わせをカバーするテストを設計することで環境適応性テストをサポートする。その後、環境

内に存在する様々なコンポーネントを動かす機能適合性のテストケースを選択して、対象となる環境をテストする。

4.2.6.3 置換性テスト

置換性テストは、システム内のソフトウェアのコンポーネントまたはバージョンを他のものに置換できる能力に焦点を当てている。これは、様々なハードウェアデバイスそして/またはソフトウェアのインストール環境の置換が頻繁に発生する、モノのインターネット(Internet of Things(IoT))に基づくシステムアーキテクチャーに特に関係してくる可能性がある。例えば、倉庫で在庫量を登録および管理するために使用しているハードウェアデバイスを、(改良版のスキャナーを搭載しているなどの)最新のハードウェアデバイスに置き換えることがある。また、在庫交換の注文を供給元のシステムに自動発行する新しいバージョンにインストール済みのソフトウェアをアップグレードすることもある。

置換性テストは、完成システムに統合するための複数の代替コンポーネントを使用できる場合、機能統合テストと並行してテストアナリストが実行することができる。

5. レビュー – 120 分

キーワード
チェックリストベースドレビュー

「レビュー」の学習の目的

5.1 イントロダクション

学習の目的はなし

5.2 レビューでのチェックリストの使用

TA-5.2.1 (K3)シラバスが提供するチェックリストの情報に従って、要件仕様に存在する問題を識別する。

TA-5.2.2 (K3)シラバスが提供するチェックリストの情報に従って、ユーザーストーリーに存在する問題を識別する。

5.1 イントロダクション

テストアナリストは、レビュープロセスに積極的に参加して特有の見解を提供しなければならない。適切に行われれば、全体的な納品時の品質に最大限寄与する最も費用対効果の高い手段となることができる。

5.2 レビューでのチェックリストの使用

チェックリストベースドレビューは、テストアナリストがテストベースをレビューする際に最もよく使用する技法である。レビューでチェックリストを使用することにより、参加者はレビュー時に検証する具体的なポイントを認識できる。また、チェックリストは、属人的なレビューを避けるのにも役立つ（例えば、「このチェックリストはすべてのレビューで使用しているものと同じなので、あなたの作業成果物のみを対象としているのではない」と示すことができる）。

チェックリストベースドレビューは、汎用化してすべてのレビューで使用することも、または特定の品質特性、分野、ドキュメントの種類に焦点を絞ることもできる。例えば、汎用チェックリストは、一意の識別子を持つこと、「未定」と記されている参照がないこと、適切な書式であること、および同様の適合項目であることなど、一般的なドキュメントの特性を検証することがある。要件ドキュメント向けの特定のチェックリストは、「必須(shall)」と「推奨(should)」の適切に使われていることの確認、記述されている要件の試験性の確認などを含んでいることがある。

また、要件の書式によっても、使用するチェックリストの種類が異なることもある。文章で記述されている要件ドキュメントには、図に基づくドキュメントとは異なるレビュー基準が存在する。

チェックリストは、次のような特定の側面を対象にすることもある。

- プログラマー／アーキテクトのスキルセットまたはテスト担当者のスキルセット –テストアナリストの場合、テスト担当者のスキルセットに対するチェックリストが最も適している
- 特定のリスクレベル（セーフティクリティカルシステムの場合など） – チェックリストはそのリスクレベルで必要な固有の情報を含むことが多い
- 特定のテスト技法 – チェックリストは特定の技法で必要な情報に重点を置く（デシジョンテーブルで提示される規則など）
- 特定の仕様アイテム（要件、ユースケース、ユーザーストーリーなど） – これらについては、以下の節で説明する。コードまたはアーキテクチャーのレビュー向けにテクニカルテストアナリストが使用するものとは、異なる側面に重点を置くことが多い。

5.2.1 要件レビュー

要件指向のチェックリストが含むアイテムの例を以下に示す。

- 要件の出どころ(人、部署など)
- 各要件の試験性
- 各要件の優先順位
- 各要件の受け入れ基準
- ユースケースの呼び出し構造の可用性(該当する場合)
- 各要件/ユースケース/ユーザーストーリーの一意な識別子
- 各要件/ユースケース/ユーザーストーリーのバージョン設定
- ビジネス/マーケティング要件から各要件へのトレーサビリティ
- 要件そして/またはユースケースの間のトレーサビリティ(該当する場合)
- 一貫性のある用語の使用(例えば、用語集の使用など)

要件がテストできない場合、つまりテストアナリストがどうテストをするか決められないように要件定義をしている場合、その要件には欠陥があると認識することが重要である。例えば、「ソフトウェアは非常にユーザーフレンドリーであるべきだ」と記述されている要件はテストできない。どのようにしたら、テストアナリストは、ソフトウェアがユーザーフレンドリーであるか、または非常にユーザーフレンドリーなのかを判断できるだろうか？ 代わりに、要件が「ソフトウェアは、使用性の標準ドキュメント、バージョン xxx で規定されている使用性の標準に適合していなければならない」と記述されており、使用性の標準ドキュメントが存在する場合、これは、テストが可能な要件である。この 1 つの要件はインターフェースのすべての項目に適用するので、包括的な要件でもある。この場合、この 1 つの要件で、重要なアプリケーションの個別のテストケースを容易に多数作り出せることもある。また、この要件、または使用性の標準ドキュメントからテストケースまでのトレーサビリティも重要である。なぜなら、参照する使用性の仕様の変更が発生した場合、すべてのテストケースをレビューして、必要に応じて更新する必要があるからである。

テスト担当者がテストの合格/不合格を判断できない場合、または合格/不合格にするテストを構築できない場合も、その要件はテストできない。例えば、「システムは時間の 100%、つまり、1 日 24 時間、週 7 日、年 365 日(または 366 日)の可用性を提供しなければならない」という要件はテストできない。

ユースケースレビューの単純なチェックリスト[1]は、次の質問を含む場合がある。

- メインの振る舞い(パス)は明確に定義されているか？
- すべての代替の振る舞い(パス)は識別されており、エラー処理は完全に備わっているか？

[1] 試験問題では、質問に答える際に使用するユースケースチェックリストの一部分を提供する

- ユーザーインターフェースメッセージは定義されているか？
- メインの振る舞いは 1 つだけか（1 つであるべきであり、そうでない場合、複数のユースケースとなる）？
- それぞれの振る舞いはテスト可能か？

5.2.2 ユーザーストーリーレビュー

アジャイルソフトウェア開発では、一般的に、要件をユーザーストーリーの形式で表す。これらのストーリーは、実証可能な機能の小さな単位を表す。ユースケースは、機能の複数の領域を横断するユーザートランザクションであるが、ユーザーストーリーはより独立したフィーチャーであり、一般的に開発にかかる時間によりその範囲が決まる。ユーザーストーリー向けのチェックリスト[1]は次に示す質問を含めることができる。

- ストーリーは対象のイテレーション／スプリントにとって適切か？
- ストーリーは要求者の観点で記述されているか？
- 受け入れ基準は定義されており、テスト可能か？
- フィーチャーは明確に定義されており、他と区別できるか？
- ストーリーは他のいずれのストーリーから独立しているか？
- ストーリーに優先度が割り当てられているか？
- ストーリーは一般的に使用される形式に従っているか？
  （<ユーザーの種類>として、<あるゴール>をしたい、なぜなら<ある理由>だから [Cohn04]）

ストーリーが新しいインターフェースを定義する場合は、前述のような汎用的なストーリーチェックリストおよび詳細なユーザーインターフェースチェックリストを使用することが適切である。

5.2.3 チェックリストの調整

チェックリストは、次の項目に基づいて調整できる。

- 組織（例：企業ポリシー、標準、慣習、および法的制約の考慮）
- プロジェクト／開発の取り組み（例：重点項目、技術的標準、リスク）
- レビュー対象の作業成果物の種類（例：コードレビューは特定のプログラム言語向けに調整することがある）
- レビュー対象の作業成果物のリスクレベル
- 使用するテスト技法

優れたチェックリストは、問題を発見しやすくし、チェックリストで特別に参照されていない可能性のある他のアイテムに関する議論を開始するのに役立つ。チェックリストを組み合わせて使用すると、レビューを通して作業成果物の品質を最も高くすることができる。Foundation

Level シラバスで参照しているような標準チェックリストとともにチェックリストベースドレビューを使用し、前述のような組織固有のチェックリストを開発すると、テストアナリストは効果的にレビューを実行できる。

レビューとインスペクションの詳細については、[Gilb93]および[Wiegers03]を参照されたい。7.4 節で参照されている文献で、チェックリストのその他の例が提供されている。

6. テストツールおよび自動化 – 90 分

キーワード
キーワード駆動テスト、テストデータ準備、テスト設計、テスト実行、テストスクリプト

「テストツールおよび自動化」の学習の目的

6.1 イントロダクション
学習の目的はなし

6.2 キーワード駆動自動化
TA-6.2.1 (K3)特定のシナリオで、キーワード駆動テストプロジェクトでのテストアナリストの適切な活動を判断する 。

6.3 テストツールの種類
TA-6.3.1 (K2)テスト設計、テストデータ準備、テスト実行で適用するテストツールの使用方法と種類を説明する。

6.1 イントロダクション

テストツールを使用すると、テストの効率性と正確性を大幅に改善できる。本章では、テストアナリストが使用するテストツールと自動化手法について説明する。テストアナリストは、開発者、テスト自動化エンジニア、およびテクニカルテストアナリストと連携してテスト自動化ソリューションを構築する点に気を付けるべきである。キーワード駆動自動化では特に、テストアナリストの貢献度が高く、ビジネスおよびシステム機能性に関する経験が活かされる。

テスト自動化とテスト自動化エンジニアの役割に関する詳細については、ISTQB® Advanced Level Test Automation Engineer シラバス[ISTQB_TAE_SYL]を参照されたい。

6.2 キーワード駆動テスト

キーワード駆動テストは、主要なテスト自動化手法の 1 つで、テストアナリストは主となる入力であるキーワードとデータを提供する。

キーワード（アクションワードとも呼ばれる）は、例外はあるが、ほとんどの場合、ビジネスとシステムのハイレベルな相互作用（例えば、注文の取り消し）を表すために使用する。各キーワードは、一般的に、アクターとテスト対象のシステムとの間にあるいくつかの詳細な相互作用を表すために使用する。キーワード（関連するテストデータを含む）の配列は、テストケースを特定するために使用する[Buwalda02]。

テスト自動化では、キーワードを 1 つ以上の実行可能テストスクリプトとして実装する。ツールはキーワードの配列として記述したテストケースを読み込む。キーワードの列は、キーワードの機能を実装する適切なテストスクリプトを呼び出す。スクリプトは特定のキーワードに容易にマッピングできるように、高度にモジュール化して実装する。これらのモジュール化したスクリプトを実装するには、プログラミングスキルが必要になる。

キーワード駆動テストの主な利点を次に示す。

- 特定のアプリケーションまたはビジネスドメインに関連するキーワードは、ドメインの専門家が定義できる。これにより、テストケース仕様作成のタスクがより効率的になる。
- 主要なドメインの専門知識を持つ人が、基になる自動化スクリプトのコードを理解する必要なく、自動化テストケース実行の利点を受けられる（キーワードをスクリプトとして実装した場合）。
- モジュラー型記述技法を使用すると、テスト対象のソフトウェアの機能およびインターフェースに対する変更が発生した場合に、テスト自動化エンジニアはテストケースを効率的にメンテナンスできる[Bath14]。
- テストケース仕様は、その実装から独立している。

テストアナリストは通常、キーワード/アクションワードデータを作成およびメンテナンスする。テストアナリストは、キーワードを実装するために、スクリプト開発のタスクが引き続き必要であることを認識すべきである。使用するキーワードとデータを定義した後は、テスト自動化担当者（テクニカルテストアナリストまたはテスト自動化エンジニアなど）が、ビジネスプロセスのキーワードとローレベルのアクションを自動化テストスクリプトに変換する。

キーワード駆動テストは、一般的に、システムテストで実行するが、スクリプトのコード開発はテスト設計のできる限り早い時期に始まることがある。イテレーティブ環境では、特に継続的インテグレーション/継続的デプロイを使用する場合、テスト自動化開発は継続して行うプロセスである。

入力キーワードおよびデータの作成が完了したら、テストアナリストは、キーワード駆動テストケースを実行し、故障が発生した場合にはその分析を行う責任を担う。

不正を検出したら、テストアナリストは故障の原因の調査をすることで、欠陥がキーワード、入力データ、テスト自動化スクリプト自体にあるのか、またはテスト対象のアプリケーションにあるのかを判断する支援をすべきである。通常、トラブルシューティングの最初の手順では、同じデータを使用して同じテストケースを手動で実行し、故障がアプリケーション自体に存在するかどうかを確認する。この手順で故障が発生しない場合、テストアナリストは、故障につながるテストケースの順序をレビューし、問題は先行する手順で発生している（正しくない入力データが投入されているかもしれない）が、後続の処理まで表面化しないのかどうかを判定する。テストアナリストが故障の原因を判定できない場合、さらなる分析を行うために、トラブルシューティング情報をテクニカルテストアナリストまたは開発者に渡すべきである。

6.3 テストツールの種類

多くのテストアナリストが担当する作業では、ツールを有効に使用する必要がある。この際、以下によってツールの有効性を高めることができる。

- 使用すべきツールを知っている
- ツールがテスト作業の効率を高められる（例えば、許容される時間内によりよいカバレッジを提供する一助になることによる）ことを知っている

6.3.1 テスト設計ツール

テスト設計ツールは、テストに適用するテストケースおよびテストデータの作成を支援するために使用する。これらのツールは、特定の要件ドキュメント形式、モデル（UML など）、またはテストアナリストが提供する入力に基づいて動作する。テスト設計ツールは、多くの場合、特定の形式および特定のツール（特定の要件マネジメントツールなど）と連携して動作するように設計および構築する。

テスト設計ツールは、特定の狙ったレベルのカバレッジ、システムへの確証、またはプロダクトリスク軽減アクションを達成するために必要なテストケースの種類を決定するための情報をテストアナリストに提供する。例えば、クラシフィケーションツリーツールは、選択したカバレッジ基準に基づくカバレッジを完全に達成するために必要な組み合わせセットを生成(および表示)する。テストアナリストはこの情報を使用して、実行する必要のあるテストケースを決定できる。

6.3.2 テストデータ準備ツール

テストデータ準備ツールには、次の利点がある。

- 要件ドキュメント、さらにはソースコードなどのドキュメントも分析して、カバレッジのレベルを達成するためにテストする中で必要なデータを決定する。
- データセットを本番システムから取得し、データの内部的な整合性を維持しながら、「選別(scrub)」または匿名化を実行して個人情報を削除できる。選別したデータは、個人情報の漏えい、または誤用のリスクを発生させることなく、テストに使用できる。この機能は、現実的なデータを大量に必要とする場合、およびセキュリティとデータプライバシーのリスクが危惧される場合に、特に重要となる。
- 特定の入力パラメーターのセットから合成テストデータを生成できる(例えば、ランダムテスト用)。これらのツールのいくつかは、データベース構造を分析して、テストアナリストから要求される入力が何かを判断する。

6.3.3 テスト自動実行ツール

テスト実行ツールは、自動化したテストを実行し、実際の結果を確認するためにすべてのテストレベルでテストアナリストが使用する。テスト実行ツールを使用する目的は、通常、次の 1 つまたは複数の項目に該当する。

- コスト削減のため(工数、および/または時間の観点から)
- より多くのテストケースを実行するため
- 同じテストケースを多くの環境で実行するため
- テスト実行の再現性を向上するため
- 手動で実行できないテストケースを実行するため(例:大規模なデータ妥当性確認テスト)

これらの目的は、多くの場合、コスト削減しながらカバレッジを上げるという主要な目的と重なる。

テスト実行ツールの投資効果は、通常リグレッションテストを自動化したときに最高となる。これは、少ないメンテナンス工数が期待され、かつテストを繰り返し実行するためである。スモークテストの自動化でも、頻繁なテストケースの利用、迅速なテスト結果の取得、および継続的インテグレーション環境で新しいビルドを評価するための自動化ができることにより、自動化ツールを有効に使用できる。ただし、継続的インテグレーション環境で新しいビルドを評価するための自動化は、メンテナンスコストが高くなることもある。
テスト実行ツールは、一般的にシステムテストおよび統合テストで使用する。特に API テストツールなどの一部のツールは、コンポーネントテストでも使用することがある。最も適した状況でツールを活用することにより、投資効果が向上する。

7. 参考文献

7.1 標準

[ISO25010] ISO/IEC 25010 (2011) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models., Chapter 4

[ISO29119-4] ISO/IEC/IEEE 29119-4 Software and Systems Engineering – Software Testing - Part 4, Test Techniques, 2015

[OMG-DMN] Object Management Group: OMG® Decision Model and Notation™, Version 1.3, December 2019; url: www.omg.org/spec/DMN/, Chapter 8

[OMG-UML] Object Management Group: OMG® Unified Modeling Language®, Version 2.5.1, December 2017; url: www.omg.org/spec/UML/

[RTCA DO-178C/ED-12C] Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE ED12C, 2013., Chapter 1

7.2 ISTQB® と IREB のドキュメント

[IREB_CPRE] IREB Certified Professional for Requirements Engineering Foundation Level Syllabus, Version 2.2.2, 2017

[ISTQB_AL_OVIEW] ISTQB® Advanced Level Overview, Version 2.0

[ISTQB_ALTTA_SYL] ISTQB® Advanced Level Technical Test Analyst Syllabus, Version 2019

[ISTQB_FL_SYL] ISTQB® Foundation Level Syllabus, Version 2018

[ISTQB_GLOSSARY] Standard glossary of terms used in Software Testing, url: https://glossary.istqb.org/

[ISTQB_TAE_SYL] ISTQB® Advanced Level Test Automation Engineer Syllabus, Version 2017

[ISTQB_UT_SYL] ISTQB® Foundation Level Specialist Syllabus Usability Testing, Version 2018

7.3 書籍と記事

[Bath14] Graham Bath, Judy McKay, "The Software Test Engineer's Handbook (2nd Edition)", Rocky Nook, 2014, ISBN 978-1-933952-24-6

[Beizer95] Boris Beizer, "Black-box Testing", John Wiley & Sons, 1995, ISBN 0-471-12094-4　JSTQB 訳注) 日本では「実践的プログラムテスト入門」(日経 BP 社, 1997 年)として発行されている。

[Black02]:Rex Black, "Managing the Testing Process (2nd edition)", John Wiley & Sons: New York, 2002, ISBN 0-471-22398-0、JSTQB 訳注) 日本では「基本から学ぶテストプロセス管理」(日経 BP 社, 2004 年)として発行されている。

[Black07]:Rex Black, "Pragmatic software testing: Becoming an effective and efficient test professional", John Wiley and Sons, 2007, ISBN 978-0-470-12790-2　JSTQB 訳注) 日本では「ソフトウェアテスト実践ワークブック」(日経 BP 社, 2007 年)として発行されている。

[Black09]:Rex Black, "Advanced Software Testing, Volume 1", Rocky Nook, 2009, ISBN 978-1-933-952-19-2

[Buwalda02]:Hans Buwalda, "Integrated Test Design and Automation: Using the Test Frame Method", Addison-Wesley Longman, 2002, ISBN 0-201-73725-6

[Chow1978]　T.S. Chow, "Testing Software Design Modeled by Finite-State Machines", IEEE Transactions on Software Engineering vol. SE-4, issue 3, May 1978, pp. 178-187

[Cohn04]:Mike Cohn, "User Stories Applied: For Agile Software Development", Addison-Wesley Professional, 2004, ISBN 0-321-20568-5

[Copeland04]:Lee Copeland, "A Practitioner's Guide to Software Test Design", Artech House, 2004, ISBN 1-58053-791-X　JSTQB 訳注) 日本では「はじめて学ぶソフトウェアテスト技法」(日経 BP 社, 2005 年)として発行されている。

[Craig02]:Rick David Craig, Stefan P. Jaskiel, "Systematic Software Testing", Artech House, 2002, ISBN 1-580-53508-9、JSTQB 訳注) 日本では「体系的ソフトウェアテスト入門」(日経 BP 社, 2004 年) として発行されている。

[Forgács19]　István Forgács, Attila Kovács, “Practical Test Design”, BCS, 2019, ISBN 978-1-780-1747-23

[Gilb93]：Tom Gilb, Dorothy Graham, "Software Inspection", Addison-Wesley, 1993, ISBN 0-201-63181-4　JSTQB 訳注）日本では「ソフトウェアインスペクション」（構造計画研究所, 1999 年)として発行されている。

[Koomen06]：Tim Koomen, Leo van der Aalst, Bart Broekman, Michiel Vroon "TMap NEXT, for result driven testing", UTN Publishers, 2006, ISBN 90-72194-80-2

[Kuhn16]：D. Richard Kuhn et al, "Introduction to Combinatorial Testing", CRC Press, 2016, ISBN 978-0-429-18515-1

[Myers11]：Glenford J. Myers, "The Art of Software Testing 3rd Edition", John Wiley & Sons, 2011, ISBN: 978-1-118-03196-4　JSTQB 訳注）日本では「ソフトウェアテストの技法」（近代化学社, 2006 年)として発行されている。

[Offutt16]：Jeff Offutt, Paul Ammann, “Introduction to Software Testing – 2nd Edition”, Cambridge University Press, 2016, ISBN 13: 9781107172012,

[vanVeenendaal12]：Erik van Veenendaal, "Practical risk-based testing." Product Risk Management: The PRISMA Method ", UTN Publishers, 2012, ISBN 9789490986070

[Wiegers03]：Karl Wiegers, "Software Requirements 2", Microsoft Press, 2003, ISBN 0-735-61879-8、JSTQB 訳注）日本では「ソフトウェア要求」（日経 BP 社, 2003 年)として発行されている。

[Whittaker03]：James Whittaker, "How to Break Software", Addison-Wesley, 2003, ISBN 0-201-79619-8

[Whittaker09]：James Whittaker, "Exploratory software testing: tips, tricks, tours, and techniques to guide test design", Addison-Wesley, 2009, ISBN 0-321-63641-4

7.4 その他の参照元

以下は、インターネットなどで参照できる情報を示している。これらの参照については、本 Advanced Level シラバス発行時にチェックしているが、リファレンスがすでに参照できなくなっていても、ISTQB®はその責を負わない。

- 第 3 章
  - Czerwonka, Jacek：www.pairwise.org
  - 欠陥分類法：www.testingeducation.org/a/bsct2.pdf
  - Boris Beizer の著作に基づくサンプル欠陥分類法：
    inet.uni2.dk/~vinter/bugtaxst.doc
  - 様々な分類法の優れた概説：testingeducation.org/a/bugtax.pdf
  - James Bach, "Heuristic Risk-Based Testing"
  - Exploring Exploratory Testing, Cem Kaner and Andy Tinkham,
    www.kaner.com/pdfs/ExploringExploratoryTesting.pdf
  - Pettichord, Bret, "An Exploratory Testing Workshop Report",
    www.testingcraft.com/exploratorypettichord
- 第 5 章
  http://www.tmap.net/checklists-and-templates

8. 付録 A:

次の表は、ISO 25010 で提供されている完全な表の抜粋である。Test Analyst シラバスで説明されている品質特性のみを取り上げており、ISO 9126 で使用されている用語(本シラバスの 2012 バージョンで使用)とより新しい ISO 25010 で使用されている用語(本バージョンで使用)を比較している。

ISO/IEC 25010	ISO/IEC 9126-1	注
機能適合性	**機能性**	
機能完全性		
機能正確性	正確性	
機能適切性	合目的性	
	相互運用性	「互換性」に変更
使用性		
適切度認識性	理解性	新しい用語
習得性	習得性	
運用操作性	運用性	(訳注:日本語訳のみ変更)
ユーザーエラー防止性		新しい副特性
ユーザーインターフェース快美性	魅力性	新しい用語
アクセシビリティ		新しい副特性
互換性		新しい定義
相互運用性		
共存性		Technical Test Analyst シラバスで説明