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 2012.J01

International Software Testing Qualifications Board

Advanced Level Test Analyst Sub Working Group: Judy McKay (Chair), Mike Smith, Erik Van Veenendaal; 2010-2012.

改訂履歴

◆ ISTQB®

バージョン	日付	摘要
ISEB v1.1	2001 年 9 月 4 日	ISEB Practitione シラバス
ISTQB 1.2E	2003 年 9 月	EOQ-SG による ISTQB Advanced Level シラバス
V2007	2007 年 10 月 12 日	テスト技術者 Advanced Level シラバス、2007 版
D100626	2010 年 6 月 26 日	2009 年に承認された変更箇所の反映と資格種別ごとの各章の分離
D101227	2010 年 12 月 27 日	書式変更と文字校正の反映
D2011	2011 年 10 月 23 日	シラバス分割のための変更、LO に対応する内容変更 BO の追加
Alpha 2012	2012 年 3 月 9 日	D2011 に対して受領したすべての NBs コメントの反映
Beta 2012	2012 年 4 月 7 日	アルファ版に対して受領したすべての NBs コメントの反映
Beta 2012	2012 年 4 月 7 日	GA のためのベータ版リリース
Beta 2012	2012 年 6 月 8 日	NB への文書編集済みバージョンのリリース
Beta 2012	2012 年 6 月 27 日	EWG および用語集コメントの反映
RC 2012	2012 年 8 月 15 日	リリース前バージョン – 最終 NB 編集箇所の反映
GA 2012	2012 年 10 月 19 日	GA リリースのための最終編集

◆ JSTQB®

バージョン	日付	摘要
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 Sub Working Group - Advanced Test Analyst（Advanced Test Analyst 作業部会）のコアメンバである Judy McKay (Chair), Mike Smith, Erik van Veenendaal が執筆した。

本コアチームは、レビューチームおよびすべての国の国際部会のメンバによる提案と意見に感謝したい。

本 Advanced Level シラバス完成時の、Advanced Level Working Group（Advanced Level 作業部会）のメンバは以下のとおりである（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

次のメンバが、本シラバスのレビュー、意見表明および投票に参加した。

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.

このドキュメントは、2012 年 10 月 19 日に開催された ISTQB®の総会で正式に発行された。

0. 本シラバスの紹介

0.1 本書の目的

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、第三者が本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 概要

Advanced Level は、次の 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
- テストアナリスト
- テクニカルテストアナリスト

『Advanced Level シラバス概要』[ISTQB_AL_OVIEW]には、次の情報を記載している。

- 各シラバスのビジネス成果
- 各シラバスの概要
- 各シラバスの関係
- 知識レベルの説明(K レベル)
- 付録

0.3 試験のための学習の目的

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。基本的に本シラバスのすべての箇所は K1 レベルで試験対象となる。つまり、受験志願者は、用語や概念について認識し、記憶し、想起することになる。このため、関連する章の最初には、K2、K3、K4 レベルの学習の目的のみを記載する。

1. テストプロセス – 300 分

用語

具体的テストケース、終了基準、高位レベルテストケース、論理的テストケース、低位レベルテストケース、テストコントロール、テスト設計、テスト実行、テスト実装、テスト計画作業

「テストプロセス」の学習の目的

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 テストの計画作業、モニタリング、およびコントロール

TA-1.3.1 (K2)テストの計画およびコントロールを支援する場合に、テストアナリストが実行する活動を要約する。

1.4 テスト分析

TA-1.4.1 (K4)プロジェクト概要やライフサイクルモデルなどの特定のシナリオを分析して、分析フェーズおよび設計フェーズでのテストアナリストのタスクを決定する。

1.5 テスト設計

TA-1.5.1 (K2)ステークホルダがテスト条件を理解する必要がある理由を説明する。

TA-1.5.2 (K4)プロジェクトシナリオを分析して、使用するのに最も適切な低位レベル(具体的)および高位レベル(論理的)のテストケースを決定する。

1.6 テスト実装

TA-1.6.1 (K2)テスト分析およびテスト設計にとっての典型的な終了基準、およびそれらの基準を満たすことがテスト実装の作業にどのように影響するかを説明する。

1.7 テスト実行

TA-1.7.1 (K3)特定の状況で、テスト実行時に行うべき手順および考慮事項を決定する。

1.8 終了基準の評価とレポート

TA-1.8.1 (K2)テストケースの実行状態に関する正確な情報が重要である理由を説明する。

1.9 テスト終了作業

TA-1.9.1 (K2)テスト終了作業時にテストアナリストが提供する必要のある成果物の例を示す。

1.1 イントロダクション

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

- 計画、モニタリング、およびコントロール
- 分析と設計
- 実装と実行
- 終了基準の評価とレポート
- 終了作業

Advanced Level シラバスでは、プロセスの改良と最適化をさらに行い、ソフトウェア開発のライフサイクルとの適合性を向上させ、テストのモニタリングとコントロールを効率的に推進するために、上に挙げた活動の一部を独立したものとして考えている。Advanced Level では、次の活動を考える。

- 計画、モニタリング、およびコントロール
- 分析
- 設計
- 実装
- 実行
- 終了基準の評価とレポート
- 終了作業

これらの活動はシーケンシャルに、または並行して実行できる。たとえば、設計は実装と並行して実行できる(探索的テストなど)。正しいテストとテストケースの決定、それらの設計および実行は、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、テストプロジェクト期間中の分析、設計、実装、および実行の活動を通して行われる。

上級テスト担当者は、本シラバスで説明するさまざまなテストの局面を自身の組織、チーム、およびタスクに導入する場合に、多くの課題に直面する。ソフトウェア開発ライフサイクルおよびテスト対象システムの種類は、テストのアプローチに影響を及ぼすので、さまざまな側面を検討することが重要である。

1.2 ソフトウェア開発ライフサイクルにおけるテスト

テストに対する長期のライフサイクルアプローチを、テスト戦略の一環として考慮し、定義する必要がある。テストアナリストが関与するタイミングはライフサイクルごとに異なり、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。テストプロセスを独立して行うことはないので、テストアナリストは、次に示すような他の関連する組織領域に情報を提供する可能性のある点を認識する必要がある。

- 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
- プロジェクトマネジメント - スケジュール
- 構成管理および変更管理- ビルド検証テスト、バージョンコントロール
- ソフトウェア開発 - 提供される内容とタイミングの予測
- ソフトウェアメンテナンス - 欠陥マネジメント、ターンアラウンドタイム(欠陥を見つけてから解決するまでの時間)
- テクニカルサポート - 回避策の正確な文書化
- テクニカルドキュメント(たとえば、データベース設計仕様)の生成- 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

テスト活動は、選択したソフトウェア開発ライフサイクルモデル(シーケンシャル、イテレーティブ、またはインクリメンタルのいずれか)と整合している必要がある。たとえば、シーケンシャルな V 字モデルの場合で、ISTQB® が示す基本的な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、システムテストの実行と終了作業が完了するまでテストコントロールを継続する。

- システムテスト分析および設計は、要求仕様から、システムおよびアーキテクチャ設計仕様（上位レベル）を経て、コンポーネント設計仕様（下位レベル）までのプロセスと並行して行う。
- システムテスト環境（たとえば、テストベッド、テストリグなど）の実装は、システム設計時に開始する場合があるが、これらの活動の大部分は通常、コーディングおよびコンポーネントテストと同時に実行する。この場合、システムテスト実装への取り組みは、システムテスト実行の開始直前まで継続してしまいがちである。
- システムテストの実行は、システムテスト開始基準にすべて合致（または条件付き免除）したときに始まる。これは最低でもコンポーネントテストが完了し、コンポーネント統合テストも完了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に合致するまで継続する。
- システムテスト終了基準の評価およびシステムテスト結果のレポートは、システムテスト実行を通じて行う。これは通常、プロジェクトのデッドラインが近付くほど頻繁となり急を要するものとなる。
- システムテスト終了作業は、システムテスト終了基準に合致しシステムテスト実行の終了を宣言してから始まる。ただし場合によっては、受け入れテストが終わり、すべてのプロジェクト活動が終了するまで遅れることがある。

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、タスクを異なる順番で実行したり、一部のタスクを除外したりすることがある。たとえば、イテレーティブモデルでは、イテレーションごとに標準テストプロセスの縮小セットを使用することがある。分析と設計、実装と実行、および評価とレポートをイテレーションごとに実行し、計画作業をプロジェクトの開始時のみ、終了レポート作成をプロジェクトの終了時のみ実行することがある。アジャイルプロジェクトでは、一般的に、公式度の低いプロセスを使用し、変更が頻繁に発生することを許容する密な関係をプロジェクト内に築く。アジャイルは、「軽量」プロセスであるため、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「スタンドアップミーティング」などを活用してより迅速なコミュニケーション手法を持つ傾向にある（「スタンドアップ」と呼ばれるのは、通常 10～15 分と非常に短いため、誰も着席する必要がなく、全員が関与し続けることに由来する）。

すべてのライフサイクルモデルの中でアジャイルプロジェクトでは、テストアナリストは最も早期に関与する必要がある。テストアナリストはプロジェクトの開始時点から関与し、初期のアーキテクチャおよび設計の作業を開発者と連携して行う必要がある。レビューは公式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。テストアナリストはプロジェクト全体を通して関与し、プロジェクトチームにとって常に利用できる状態である必要がある。これらが浸透することにより、通常、アジャイルチームのメンバは、単一のプロジェクトに専念し、プロジェクトのすべての側面に完全に関与する。

イテレーティブモデルおよびインクリメンタルモデルは、ソフトウェアの開発が進むにつれ変化することが予測されるアジャイルアプローチから、V 字モデル内に存在するイテレーティブ開発モデル（埋め込み型イテレーティブとも呼ばれる）およびインクリメンタル開発モデルまでの多様な形態となる。埋め込み型イテレーティブモデルの場合、テストアナリストは標準的な計画作業および設計の側面に関与することが期待されるが、ソフトウェアの開発、テスト、変更、および展開と段階が進むにつれ、より相互作用的な役割に移行する。

使用するソフトウェア開発ライフサイクルに関係なく、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。たとえば、前述のように V 字モデル内でイテレーティブモデルを使用するなど、多くのハイブリッドモデルが存在する。テストアナリストは多くの場合、関与の適切なタイミングを示すモデルの定義に依存することなく、最も有効な役割を決定し、その役割のために作業する必要がある。

1.3 テストの計画作業、モニタリング、およびコントロール

本節では、テストを計画、モニタリング、およびコントロールするプロセスに焦点を当てる。

1.3.1 テスト計画作業

テスト計画作業は、ほとんどの場合、テスト作業の開始時に発生し、テスト戦略で示される使命と目的を満たすために必要となる活動とリソースのすべてに関する識別と計画を含む。テスト計画作業時には、テストアナリストはテストマネージャと連携して、次の項目を検討し、計画する必要がある。

- テスト計画を、機能テストに限定されないようにする。すべての種類のテストをテスト計画内で考慮し、テスト計画に基づいてスケジュールを立てる。たとえば、テストアナリストは、機能テストだけでなく、使用性テストにも責任を持つことがある。そのテストタイプも、テスト計画ドキュメントでカバーする必要がある。
- テストマネージャと連携してテスト見積りをレビューし、テスト環境の調達と妥当性確認のために適切な時間が計上されていることを確認する。
- 構成テストを計画する。さまざまな種類のプロセッサ、オペレーティングシステム、仮想マシン、ブラウザ、および周辺機器が存在し、それらを組み合わせることで複数の構成が考えられる場合、それら組み合わせの適切なカバレッジを提供するテスト技法を適用することを計画する。
- ドキュメントのテストを計画する。ソフトウェアと一緒にドキュメントをユーザに提供するので、正確で有効なドキュメントを作成する必要がある。テストアナリストは、時間をかけてドキュメントを検証する必要がある。また、テクニカルドキュメント作成スタッフと連携して、スクリーンショットやビデオクリップに使用するデータを準備することが必要になる場合がある。
- インストール手順のテストを計画する。インストール手順、およびバックアップとリストアの手順を、十分にテストする必要がある。ソフトウェアをインストールできないと、そのソフトウェアをまったく使用できないので、これらの手順はソフトウェアよりも重要になる可能性がある。テストアナリストは多くの場合、最終的なインストール手順を利用できない状況で事前に構成されているシステム上で初期テストを行うため、インストール手順のテストを計画することは困難な場合がある。
- ソフトウェアライフサイクルとの整合性がとれるようにテストを計画する。タスクをシーケンシャルに実行すると、ほとんどの場合、要求されるスケジュールでプロジェクトを完了できないので、多くのタスク(少なくともその一部)を並列して実行する必要がある。テストアナリストは、ソフトウェアの設計、開発、および実装の各段階で関与するために、選択されているライフサイクルと自身に期待されていることを認識する必要がある。また、確認テストおよび回帰テストを実行するための時間も計上する必要がある。
- クロスファンクショナルなチームと連携してリスクの識別および分析を行うための適切な時間を計上する。テストアナリストは一般的に、リスクマネジメントセッションを編成する責任を持たないが、これらの活動に積極的に関与する必要がある。

テストベース、テスト条件、およびテストケースとの間に複雑な関連性が存在することがある。たとえば、多対多の関連性がこれらの成果物の間に存在することがある。テスト計画作業とコントロールを効果的な実装をするためにこれらを理解する必要がある。テストアナリストは通常、これらの関係を最も的確に判断し、依存性を最小限にすることができる。

1.3.2 テストのモニタリングとコントロール

テストのモニタリングとコントロールは、通常、テストマネージャの担当であるが、テストアナリストは、コントロールを可能にするための測定を行う。

ソフトウェア開発ライフサイクル全体を通して、さまざまな定量的データを収集する必要がある。たとえば、完了した計画作業の割合、達成したカバレッジの割合、合格および失敗したテストケースの数などを収集する。それぞれのケースについて、ベースライン(参照標準)を定義し、このベースラインに関して進捗を追跡する必要がある。テストマネージャがメトリック情報の要約を編集およびレポートする場合、テストアナリストは各メトリックの情報を収集する。完了した各テストケース、記述した各欠陥レポート、達成した各マイルストンを全体的なプロジェクトメトリクスにまとめる。さまざまな追跡ツールに入力する情報を可能な限り正確にして、メトリクスに現実を反映させることが重要である。

正確なメトリクスを使用することにより、マネージャはプロジェクトをマネジメント(モニタリング)し、必要に応じて変更を開始(コントロール)できる。たとえば、ソフトウェアの特定の領域で非常に多くの欠陥が報告されている場合、その領域に関してさらに多くのテスト作業が必要な可能性がある。要件とリスクカバレッジの情報(トレーサビリティ)を使用することにより、残りの作業に優先度付けを行い、リソースを割り当てることができる。根本原因に関する情報を使用することにより、プロセス改善が可能な領域を判断できる。データを正確に記録することにより、プロジェクトをコントロールすることが可能になり、正確なステータスの情報をステークホルダに報告できる。過去のプロジェクトから収集したデータを計画作業で考慮することにより、プロジェクトをより効果的に計画

できる。正確なデータの使用方法は無数にある。テストアナリストは、データが正確で、タイムリーであり、客観的であるようにする必要がある。

1.4 テスト分析

テスト計画作業では、テストプロジェクトのスコープを定義する。テストアナリストはこのスコープを使用して、次のことを行う。

- テストベースを分析する。
- テスト条件を識別する。

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。

- テストベースとして機能するテスト対象が記述されたドキュメントが存在する。
- このドキュメントは、レビューに適切な評価で合格しており、レビュー後に必要に応じて更新されている。
- テスト対象に対して、残りのテスト作業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、テストベースとテスト目的を分析する。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、たとえばワークショップやスプリント計画作業の際に、関連するステークホルダと会話することによりテスト条件を識別できることがある。これらのテスト条件は、テスト戦略およびテスト計画内で識別されるテスト設計技法を通して、何をテストするかを決定するために使用する。

テスト条件は、通常、テストされるアイテムに固有であるが、テストアナリストは次の標準的な考慮事項について検討する必要がある。

- 一般的には、テスト条件をさまざまな詳細度合いで定義することが望ましい。まず、高位レベルの条件を識別し、「画面 x の機能性」など、テストの全般的な対象を定義する。次に、特定のテストケースの基準として、より詳細な条件（たとえば、「画面 x では、正しい長さよりも一桁足りないアカウント番号を拒否する」など）を識別する。この種類の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、高位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。
- 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について、リスクアイテムにまで遡って識別する。

テスト分析のまとめとして、テストアナリストは、テストプロジェクト内で割り当てられている領域の要件を満たすために、特定のテストを設計しなければならないということを知っておく必要がある。

1.5 テスト設計

テスト計画作業時に決定されるスコープに従い、テストアナリストが設計するテストを実装および実行して、テストプロセスを継続する。テスト設計のプロセスは、次の活動を含む。

- 低位レベル（具体的）テストケースまたは高位レベル（論理的）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。
- 必要なテストカバレッジを確保するテストケース設計技法を決定する。
- 識別したテスト条件を遂行するテストケースを作成する。

リスク分析およびテスト計画作業で識別した優先度基準を、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る必要がある。

設計するテストの種類によっては、設計作業時に使用するツールの可用性が、テスト設計の開始基準の 1 つとなることがある。

テストを設計するときには、次の点に留意する必要がある。

- 一部のテストアイテムは、手続き化されたテストを定義するよりも、テスト条件のみを定義することで、より適切に対処できる。この場合、手続き化されていないテストに対するガイドとして使用できるように、テスト条件を定義する必要がある。

- 合格／不合格基準を明確に識別する必要がある。
- テストは、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る必要がある。テスト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テストの相対的な重要性を理解するために、以前に指定されたテストを参照し、理解することが必要になる。
- また、テストをレビューする開発者やテストを承認する必要のある監査担当者などの他のステークホルダも理解できるように、テストを作成する必要がある。
- テストは、ユーザに表示される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、アクター（エンドユーザや他のシステムなど）に対するソフトウェアの相互作用をカバーするように設計する必要がある。プロセス間通信、バッチ実行、および他の割り込みもソフトウェアと相互作用し、欠陥を含む可能性があるので、テストアナリストはこれらのリスクを軽減するようにテストを設計する必要がある。
- さまざまなテスト対象間のインターフェースをテストするように、テストを設計する必要がある。

1.5.1 具体的テストケースと論理的テストケース

テストアナリストの職務の 1 つは、特定の状況で最適な種類のテストケースを決定することである。具体的テストケースは、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。また、具体的テストケースは要件が適切に定義されている場合、テスト担当者の経験が少ない場合、および監査などテストの外部検証が必要な場合に役立つ。具体的テストケースは優れた再現性（別のテスト担当者が同じ結果を得る）を提供するが、メンテナンスに非常に多くの労力を必要とし、実行時にテスト担当者の創造力を制限する傾向にある。

論理的テストケースは、テスト対象と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続きさえも変更することを可能にする。論理的テストケースは、各実行時での変更を許容するので、具体的テストケースよりも優れたカバレッジを提供することがある。ただし、再現性が失われる可能性もある。論理的テストケースは、要件が適切に定義されていない場合、テストを実行するテストアナリストがテストとプロダクトの両方に精通している場合、および公式なドキュメントが必要でない（たとえば、監査が行われない）場合に最適に使われる。論理的テストケースは、要件がまだ適切に定義されていない要件プロセスの初期に定義されることがある。要件がさらに定義され、安定したときに、具体的テストケースを開発するためにこれらのテストケースを使用することがある。この場合、テストケースを最初に論理的テストケース、次に具体的テストケースという順に作成し、実行時には具体的テストケースのみを使用する。

1.5.2 テストケースの作成

テストケースは、テスト戦略およびテスト計画内で識別されるテスト設計技法（第 3 章参照）を使用して識別したテスト条件を段階的に作り上げ、詳細化することで設計される。テストケースは、使用するテスト戦略に準拠して、繰り返し使用可能、検証可能、およびテストベース（たとえば要件）まで遡って追跡可能である必要がある。

テストケースを設計する際は、次のアイテムを識別する。

- 目的
- 事前条件、プロジェクトまたはローカライズされたテスト環境要件、それらの受領計画、システムの状況など
- 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）
- 期待結果
- 事後条件、影響を受けるデータ、システムの状態、後続処理のためのトリガーなど

テストケースの詳細度合いは、開発コストおよび実行時の再現性の度合いに影響を及ぼすので、テストケースを実際に作成する前に定義する必要がある。テストケースの詳細度を低くすると、テストアナリストはテストケースを実行する際に自由度を高めることができ、潜在的に関心のある領域を調査する機会を得ることができる。ただし、再現性が低下する可能性もある。

多くの場合、特に課題となるのは、テストの期待結果を定義することである。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を考慮する。テストベースを明確に定義すると、理論的には、正しい結果を容易に識別できる。ただし、多くの場合、テストベースは曖昧であったり、矛盾を含んでいたり、キーエリアのカバレッジが十分でないか、まったく不足したりする。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有しているか、それらの情報にアクセスできる必要がある。また、テストベースを適切に指定している場合でも、複雑な刺激と反応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。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、誤った報告やシステムに対する誤った信頼が生まれる。

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。たとえば、ユーザ受け入れテストは主に要求仕様、ユースケース、および定義したビジネスプロセスに基づき、コンポーネントテストは主に低位レベルの設計仕様、ユーザストーリー、およびコード自体に基づくことがある。これらの活動は、テストの対象が異なる可能性はあるが、すべてのテストレベルを通して発生することに注意する必要がある。たとえば、特定のコンポーネントが、そのコンポーネントの詳細設計で指定した機能性を提供するように、単体レベルの機能テストを設計する。統合レベルの機能テストでは、コンポーネントが相互に作用し、それらの相互作用を通して機能性を提供することを検証する。システムレベルの機能テストでは、エンドツーエンドの機能性を対象にテストする。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。これらは、必要な詳細度合いとツール（たとえば、コンポーネントテストレベルでのドライバおよびスタブ）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
- 使用するライフサイクルモデル（たとえば、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

テストのスコープに応じて、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 25000 標準[ISO25000]（ISO 9126 の改訂版）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。

レビューおよび静的解析と関連付けることにより、テスト分析およびテスト設計のプロセスを強化できる。実際、テスト分析およびテスト設計は、静的テストの形態となることが多い。これは、このプロセスの実行時に問題をベースドキュメントに見つけることができるためである。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、要件レビューを行うための優れた方法の一つである。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、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、テスト計画などのテスト成果物をレビュー対象にする必要がある。

アジャイルライフサイクルのような一部のプロジェクトでは、要件の文書化が最小限しか行われていないことがある。これらは、小規模ではあるが機能面の一部を明らかに定義した「ユーザストーリー」の形態をとる場合がある。ユーザストーリーは、受け入れ基準の定義を含む必要がある。ソフトウェアが受け入れ基準を満たしていることを示せる場合、通常、完了している他の機能と統合できる状態にあるか、機能を動かすためにすでに統合されていると考えられる。

要求される詳細なテストインフラの要件は、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。たとえば、テストインフラの要件は、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、この 2 つの節で説明したすべてのアイテムを、定義する終了基準に含める必要がある。測定可能であり、それ以降の手順を実行するために必要なすべての情報と準備が確認できる終了基準が、提供されていることが重要である。

1.6 テスト実装

テスト実装は、テスト設計の実現である。テスト実装は、テストケースの実行を開始するために必要な自動テストの作成、テスト（手動および自動の両方）の実行順序の編成、テストデータおよびテスト環境を完成させること、リソース割り当てなどのテスト実行スケジュールの作成を含む。また、対象のテストレベルの明示的および暗黙的な開始基準のチェック、およびプロセス内の前の手順の終了基準が満たされていることの確認も含む。テストレベルまたはテストプロセスの手順のいずれかの終了基準を省略した場合、実装作業がスケジュール遅延、不十分な品質、および予期しない余分な作業の影響を受ける可能性がある。テスト実装作業を開始する前に、すべての終了基準を満たしていることを確認する必要がある。

実行順序を決定する際には、多くの項目を考慮する必要がある。場合によっては、テストケースをテストスイート（テストケースのグループ）に編成する。これにより、関連するテストケースを一緒に実行するようにテストを編成できる。リスクベースドテスト戦略を使用する場合、リスクの優先順に基づいて、テストケースの実行順序を決定できる。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、他の要因が実行順序を決定することもある。コードがセクションごとにリリースされたり、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作業を調整したりすることが必要になる場合も一般的に発生する。特に、インクリメンタルライフサイクルモデルでは、テストアナリストは開発チームと連携して、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する必要がある。テスト実装時には、テストアナリストは、手動および自動のテストを実行する順序を完成させて確認し、特定の順序でのテストが求められるというような制約を入念に確認する必要がある。依存関係を文書化し、チェックする必要がある。

テスト実装時に実行する作業の詳細度合いとそれに関する複雑度は、テストケースやテスト条件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（たとえば、米国連邦航空局の DO-178B／ED 12B [RTCA DO-178B／ED-12B]）にテストが適合している確証を示す必要がある。

前述しているように、テストデータがテスト用に必要であり、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。また、データ駆動型の自動テストおよび手動テストで使用するデータを作成する。

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完全に設定し、検証する。「目的にかなった」テスト環境を設定することが重要である。つまり、テスト環境は、存在する欠陥をコントロールされたテスト実行時に洗い出し、故障が存在しない場合に適切に動作し、高位レベルテスト向けの本番環境またはエンドユーザ環境を必要に応じて適切に複製できる必要がある。予期しない変更、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ことがある。テスト実行時にテスト環境を変更する場合、実行済みのテストに対して変更が与える影響を評価する必要がある。

テスト担当者は、テスト実行時に、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る必要がある。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作業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続きを検証する必要がある。

テスト実装では、テスト計画作業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。たとえば、分析的なリスクベースドテスト戦略と動的テスト戦略を組み合わせることがよくある。この場合、テスト実装作業の一部を、事前に決定した手続きに従わないテスト（手続き化されていないテスト）に割り当てる。

手続き化されていないテストは、時間を厳しく管理しないと、予測できないほどに期間が延び、テスト範囲が広がることがあるので、スケジュールを立て、明確な目的を持って使用する必要がある。テスト担当者は長い年月

をかけて、攻撃、エラー推測[Myers79]、探索的テストなど、さまざまな経験ベースの技法を開発している。手続き化されていないテストでも、テスト分析、テスト設計、およびテスト実装を行うが、これらは主にテスト実行時に行う。

このような動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、いくつかの短所もある。これらの技法はテストアナリストに専門知識を要求する。また、期間の予測およびテスト範囲の確認が難しく、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になる。

1.7 テスト実行

テスト実行は、テスト対象が受け渡され、テスト実行の開始基準を満たした(または条件付きで免除された)ときに始まる。テストは、テスト実装時に決定した計画に従って実行する必要があるが、テストアナリストは、適切な時間をかけて、テスト時に観察された他の興味を引くテストシナリオおよび振る舞いを確実にカバーする必要がある(そのような逸脱の発生時に検出したすべての故障を、故障を再現するために必要な手続き化されたテストケースからのバリエーションを含めて記述する必要がある)。この手続き化されたテスト技法と手続き化されていないテスト技法(たとえば探索的テスト)の統合は、手続き化した際のテスト範囲の不足が引き起こす見逃しの防止や、殺虫剤のパラドックスの回避を手助けする。

テスト実行の中心となるのは、実行結果を期待結果と比較することである。テストアナリストは、これらのタスクに注意を払い、焦点を当てる必要がある。そうしないと、故障が検出されなかったり(偽陰性結果)、正しい振る舞いが不正として誤って分類されたり(偽陽性結果)して、テストの設計および実装に関するすべての作業が無駄になることがある。期待結果と実行結果の不一致は、インシデントとして扱う。テストアナリストは、インシデントを入念に精査し、その原因(テスト対象の欠陥または欠陥でない可能性のあるもの)を究明し、インシデントの解決を支援するデータを収集する必要がある(欠陥マネジメントの詳細については、第 6 章を参照)。

欠陥を識別したら、テストドキュメント(テスト仕様書、テストケースなど)を入念に評価して、テストドキュメントが正しいことを確認する必要がある。テストドキュメントは、さまざまな理由で不正となる。テストドキュメントが不正な場合、不正箇所を修正して、テストを再実行する必要がある。テストベースとテスト対象の変更は、テストが複数回の実行で正常に終了した後でもテストケースを不正にする可能性があるので、テスト担当者は、観察された結果が不正なテストによりもたらされたものである可能性を認識する必要がある。

テスト実行時には、テスト結果を適切に記録する必要がある。実行結果を記録していないテストは、正しい結果を識別するために繰り返し実行することが必要になる場合があり、非効率と遅延をもたらす可能性がある(適切に記録することにより、探索的テストなどのテスト技法に関連するカバレッジおよびテストの再現性の課題に対応することができる)。テスト対象、テストウェア、およびテスト環境のすべてが変化する可能性があるので、テストした具体的なバージョンだけでなく、具体的な環境構成も記録する必要がある。テスト結果記録作業を行うことにより、テストの実行に関連する詳細情報を時系列に記録できる。

結果記録作業は、個々のテストおよび活動や事象の両方に対して行う。各テストを一意に識別し、テスト実行が進むとともにステータスを記録する。テスト実行に影響を及ぼすすべての事象を記録する。テストカバレッジを測定し、テスト時の遅延および割り込みの理由を文書化できるようにするために、十分な情報を記録する。また、テストコントロール、テスト進捗レポート、終了基準の測定、およびテストプロセス改善をサポートするための情報を記録する必要がある。

記録する内容は、テストレベルおよび戦略により異なる。たとえば、自動コンポーネントテストを実行する場合、自動テストはほとんどのログ情報を記録する必要がある。手動テストを実行する場合、テストアナリストは、テスト実行情報を追跡するテストマネジメントツールに、テスト実行に関する情報を記録することが多い。特定の状況では、テスト実装時と同様に記録するテスト実行情報の量が、規制または監査の要件により影響を受けることがある。

場合によっては、ユーザまたは顧客がテスト実行に参加することがある。これは、テストで欠陥はほとんど見つからないと想定される場合に、システムに対する信頼感を構築するのに役立つ。このような想定は早期のテストレベルでは意味をなさないが、受け入れテストでは有効なことがある。

次に、テスト実行時に考慮する必要のある特定の領域のいくつかを示す。

- 「関連性のない」異常なものを見つけたり、探索したりしなさい。
  関連性のない観察内容または結果は、多くの場合、氷山のように、水面下に潜む欠陥を示唆している。
- 想定されない振る舞いをプロダクトが行っていないことを確認しなさい。
  テストでは、通常、想定される振る舞いをプロダクトが行っていることに焦点を当てて確認するが、テストアナリストは、行うことが想定されていない何か(たとえば、不要な追加機能)をプロダクトが誤って行っていないことを確認する必要がある。
- テストスイートを構築し、時間と共に成長し変化することを予期しなさい。
  コードは進化するものであり、新しい機能をカバーするために追加テストを実装する必要がある。また、ソフトウェアの他の領域について、回帰テストを実行する必要もある。テスト内のギャップがテスト実行時に見つかることがよくある。テストスイートを構築する活動は、継続的なプロセスである。
- 次のテスト向けにメモを記述しなさい。
  ソフトウェアがユーザに提供されても、また市場に配布されても、テストタスクが終わることはない。ソフトウェアの新しいバージョンまたはリリースが生成される可能性が高いので、次のテストを担当するテスト担当者のために、知識を蓄え、引継ぐ必要がある。
- すべての手動テストを再実行することを期待しない。
  すべての手動テストを再実行することは現実的ではない。疑わしい問題を検出した場合、テストアナリストは、その問題がそのテストケース以降のテスト実行で検出されるであろうと想定するのでなく、問題を調査し、記録する。
- テストケースを追加するため、欠陥追跡ツールを使用してデータを調べなさい。
  手続き化されていないテストまたは探索的テストの実行時に検出された欠陥用にテストケースを作成し、回帰テストスイートに追加することを検討する。
- 回帰テストの前に欠陥を見つけなさい。
  多くの場合、回帰テスト用の時間は制限されており、回帰テスト時に欠陥を見つけることはスケジュールの遅延を発生させる可能性がある。一般的に、回帰テストで検出される欠陥の割合は小さい。これは、回帰テストがすでに実行済みのテストであり(たとえば、同じソフトウェアの以前のバージョンでのテスト)、以前のテスト実行で欠陥が検出されていることが想定されるからである。このことは、回帰テスト全体を排除してもかまわないということは意味しておらず、新しい欠陥を検出する能力の点で、回帰テストの有効性が他のテストよりも低いということのみを意味する。

1.8 終了基準の評価とレポート

テストプロセスの観点では、テスト進捗モニタリングを行うことにより、レポート要件をサポートするための適切な情報を収集できる。モニタリングは、完了に向けての進捗の測定を含む。計画段階で定義した終了基準には、「必須」と「推奨」の基準分類が存在する可能性がある。たとえば、「優先度 1 または優先度 2 のバグはすべて解決されていることが必須である」と「すべてのテストケースで合格率が 95%を超えることが推奨される」の 2 つの基準がある場合、「必須」基準を満たさない故障が存在すると終了基準は満たさないが、合格率が 93%の場合には、プロジェクトを次の段階に進めることができる。終了基準は、客観的に評価できるように、明確に定義する必要がある。

テストアナリストは、終了基準を満たすことに向けて進捗を評価するために、テストマネージャが使用する情報を提供し、データが正確であることを確認する責任がある。たとえば、テストマネジメントシステムが、テストケースの完了時に次のステータスコードを提供する。

- 合格
- 失敗
- 条件付き合格

その場合、テストアナリストは、各ステータスコードが意味することを明確に理解し、一貫性を保ってそのステータスを適用する必要がある。「条件付き合格」は、欠陥が見つかったが、システムの機能性に影響しないことを

意味するのか？ユーザの混乱を引き起こす使用性の欠陥についてはどうか？合格率の達成が「必須」終了基準の場合、「条件付き合格」ではなく「失敗」としてテストケースを計上するかどうかということが重要な要因になる。また、「失敗」とマークされているが、たとえば、テスト環境が適切に構成されていない場合など、失敗の原因が欠陥ではないテストケースについても考慮する必要がある。追跡するメトリクスまたはステータスの使用に混乱がある場合、テストアナリストはテストマネージャと連携してそれらを明確にし、情報がプロジェクト全体を通して正確かつ一貫性を持って追跡できるようにする必要がある。

テストアナリストは一般的に、テストサイクルの期間中にステータスレポートを提供し、テスト終了時に最終レポートの作成に貢献する。この活動では、欠陥およびテストのマネジメントシステムからメトリクスを収集し、全体的なカバレッジと進捗を評価することが必要になる。テストアナリストは、レポートツールを使用でき、テストマネージャが必要な情報を抽出できるように、適切な情報を提供する必要がある。

1.9 テスト終了作業

テスト実行の完了を判断すると、テスト作業から主要な成果物を集めて、関連する担当者に引き渡すか、保管する必要がある。これらの作業全体がテスト終了作業である。テストアナリストは、成果物を必要とする人々へそれを届けることへの関与を期待されている。たとえば、判明している欠陥で改修が延期されているもの、または現状で許容されているものをシステム利用者やサポート担当に通知する。また、テストケースとテスト環境を保守テストの担当者に提供する。その他の成果物には、回帰テストセット（自動または手動）もある。テスト成果物に関する情報を、該当するリンクを含めて明確に文書化する必要がある。また、適切なアクセス許可を付与する必要がある。

テストアナリストは、振り返りミーティング（学習した教訓）に参加する必要がある。このミーティングでは、重要な教訓（テストプロジェクト内からの教訓およびソフトウェア開発ライフサイクル全体からの教訓の両方）を文書化し、さらには、「長所」を強化し、「短所」を排除または少なくともコントロールするための計画を立てる。テストアナリストは、これらのミーティングで要求される情報を豊富に提供することができ、プロセス改善に関する価値ある情報を収集するミーティングには参加する必要がある。テストマネージャのみがミーティングに参加する場合、テストアナリストは関連する情報をテストマネージャに提供して、プロジェクトの実態が正確に提示されるようにする必要がある。

構成管理システムで、結果、ログ、レポート、その他のドキュメント、および成果物を保管する必要がある。このタスクは、多くの場合、テストアナリストの担当であり、特に将来のプロジェクトでこれらの情報を使用する必要がある場合は、重要なテスト終了作業の 1 つである。

通常、保管する情報を決定するのはテストマネージャであるが、テストアナリストも、将来においてプロジェクトを再び開始する場合に必要となる情報について検討する必要がある。この情報をプロジェクトの終了時に検討することにより、プロジェクトを将来または別のチームが再び開始する場合に、数カ月の工数を節約できる。

2. テストマネジメント: テストアナリストの責任- 90 分

用語

プロダクトリスク、リスク分析、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、リスクベースドテスト、テストモニタリング、テスト戦略

「テストマネジメント」の学習の目的: テストアナリストの責任

2.2 テストの進捗モニタリングおよびコントロール

TA-2.2.1 (K2)プロジェクトの適切なモニタリングおよびコントロールを可能にするために、テスト時に追跡する必要のある情報の種類を説明する。

2.3 分散テスト、アウトソーステスト、およびインソーステスト

TA-2.3.1 (K2)24 時間テスト環境でシフト作業する場合の優れたコミュニケーション実践の例を提供する。

2.4 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-2.4.1 (K3)特定のプロジェクト状況で、リスク識別に参加し、リスクアセスメントを実行し、適切なリスク軽減を提案する。

2.1 イントロダクション

多くの領域でテストアナリストはテストマネージャに協力し、データを提供するが、この章では、テストアナリストが主要な役割を果たすテストプロセスの特定の領域に焦点を当てて説明する。テストマネージャは、テストアナリストから必要な情報を入手する必要がある。

2.2 テストの進捗モニタリングおよびコントロール

テスト進捗のモニタリングの対象には、次の 5 つの主要な要素がある。

- プロダクト(品質)リスク
- 欠陥
- テスト
- カバレッジ
- 確信度合い

テストアナリストは多くの場合、開発プロジェクトまたは運用期間を通して、プロダクトリスク、欠陥、テスト、カバレッジを特定の方法で測定し、報告する。確信度合いはこれらを通じて分かるが、主観的な報告になることが一般的である。テストアナリストは日常の業務の一部として、これらのメトリクスをサポートするために必要な情報を収集する。これらのデータの正確性は非常に重要であることを認識しておく必要がある。不正確なデータは不正確な傾向を示してしまい、不正確な結論をもたらす可能性がある。最悪の結果として、不正確なデータが正しくないマネジメントの判断をもたらし、テストチームの信頼度が損なわれる可能性がある。

リスクベースドテストアプローチを使用している場合、テストアナリストは次の項目を追跡する必要がある。

- テストにより軽減されたリスク
- まだ軽減されていないと考えられるリスク

リスク軽減の追跡は、多くの場合、テストの完了も含めて追跡するツール(たとえば、テストマネジメントツール)を使用して行う。これを行うには、識別したリスクをテスト条件に対応付けし、テスト条件をテストケースに対応付ける必要がある。このテストケースを実行して結果が合格になると、リスクを軽減できる。この方法を使用すると、テストケースの更新時にリスク軽減情報が自動的に更新される。テストケースの更新は、手動テストおよび自動テストの両方で行うことができる。

欠陥追跡は、一般的に、欠陥追跡ツールを使用して行う。欠陥を記録するときには、各欠陥に関する特定の分類情報も記録する。この情報を使用することにより、テストの進捗とソフトウェアの品質を示す傾向レポートおよびグラフを生成できる。分類情報については、「欠陥マネジメント」の章で詳しく説明する。使用するライフサイクルが、記録する欠陥ドキュメントの量と、情報を記録する方法に影響を及ぼす。

テストの実行時には、テストケースの情報を記録する必要がある。これは通常、テストマネジメントツールを介して行うが、必要に応じて手動でも行うことができる。テストケースの情報は、次の項目を含む。

- テストケースの作成ステータス(たとえば、設計済み、レビュー済み)
- テストケースの実行ステータス(たとえば、合格、不合格、ブロック、スキップ)
- テストケースの実行情報(たとえば、日付と時刻、テスト担当者の名前、使用したデータ)
- テストケースの実行結果(たとえば、スクリーンショット、関連ログ)

識別したリスクアイテムと同じく、テストケースをテスト対象の要件に対応付ける必要がある。テストケース A のみが要件 A に対応している場合、テストケース A の実行結果が合格になると、要件 A は達成されたとみなすことができることにテストアナリストは注意する必要がある。このことは、正しいとも正しくないとも言える。多くの場合、要件を徹底的にテストするには、より多くのテストケースが必要であるが、時間的な制限により、実際には、それらのテストのサブセットのみを作成する。たとえば、要件の実装を徹底的にテストするのに 20 のテストケースが必要であるが、10 のテストケースのみを作成および実行した場合、要件カバレッジ情報は、実際には 50%の

カバレッジしか達成されていなくとも、100%のカバレッジを示す。カバレッジの正確な追跡を、要件自体のレビュー済みステータスの追跡と同じく、コンフィデンス(確信度合い)の尺度として使用できる。

記録する情報の量(詳細度合い)は、ソフトウェア開発ライフサイクルモデルなど、いくつかの要因により異なる。たとえば、アジャイルプロジェクトでは、チームのメンバが密接に連携し、対面によるコミュニケーションをより多く使用するので、記録するステータス情報はより少ないのが一般的である。

2.3 分散テスト、アウトソーステスト、およびインソーステスト

多くの場合、テスト活動は、プロジェクトチームのさまざまなメンバにより構成される複数のテストチームが、それぞれ異なる複数の地域で行う。複数の地域でテスト活動を行う場合、そのテスト活動を「分散テスト」と呼ぶ。単一の地域でテスト活動を行う場合、そのテスト活動を「集合テスト」と呼ぶ。プロジェクトチームの同僚の従業員ではないメンバが、プロジェクトチームとは異なる 1 つの地域または複数の地域でテスト活動を実行する場合、そのテスト活動を「アウトソーステスト」と呼ぶ。プロジェクトチームと同じ地域に存在するが同僚の従業員でないメンバがテスト活動を実行する場合、そのテスト活動を「インソーステスト」と呼ぶ。

テストチームの一部が複数の場所に分散しているか、または複数の企業に分散しているプロジェクトの場合、テストアナリストは、効果的なコミュニケーションと情報の伝達に特別な注意を払う必要がある。一部の組織は、「24 時間テスト」モデルを使用して作業する。このモデルでは、ある時間帯のチームが次の時間帯のチームに作業を引継ぐことにより、テストを昼夜連続して実行できる。これを実現するには、作業の引継ぎを行うテストアナリストに対応した特別な計画が必要になる。優れた計画を作成するには、責任を理解するというだけでなく、適切な情報が確実に伝達されるようにすることが重要である。

口頭でのコミュニケーションを利用できない場合、ドキュメントによるコミュニケーションを確立する必要がある。これは、電子メール、ステータスレポート、およびテストマネジメントと欠陥追跡ツールの有効な活用が必要であることを意味する。テストマネジメントツールで、テストを個人に割り当てることができる場合、このツールをスケジュールツールとして、また作業を別の担当者へ容易に移動する手段として使用できる。正確に記録された欠陥は、必要に応じてフォローアップのために同僚に再割り当てできる。対面によるコミュニケーションを日常的に行うことのできない組織にとって、これらのコミュニケーションシステムを有効に活用することは、非常に重要である。

2.4 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

2.4.1 概要

テストマネージャは、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与する必要がある。

- 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、これらのタスクをプロジェクトライフサイクル全体を通して反復的に実行する。

テストアナリストは、テストマネージャがプロジェクトのために構築したリスクベースドテストフレームワーク内で作業する必要がある。また、安全性、ビジネス、および経済上の懸念、さらには政治的な要因に関連するリスクなど、プロジェクトに存在するビジネスドメインのリスクに関する知識を提供する必要がある。

2.4.2 リスク識別

リスク識別プロセスでは、ステークホルダのサンプルを広範囲に取り込むことにより、重要なリスクを数多く検出しやすくなる。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ユニークな知識を保有するので、そのドメインエキスパートやユーザと共に行うエキスパートへのインタビュー、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と促進、リスクワークショップの実施、潜在的ユーザや現在のユーザとのブレインストーミングセッションの実施、テスト用チェックリストの定義、および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を活用するのに非常に適している。特に、テストアナリストは、ユーザおよび他のドメインエキスパートと緊密に連携して、テスト中に対応する必要のあるビジネスリスクの領域を決定する必要がある。また、テストアナリストは、ユーザおよびステークホルダに対するリスクの潜在的な影響を識別する際にも特に有用である。

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。
- ソフトウェア機能の正確性の問題。たとえば、誤った計算など。
- 使用性の問題。たとえば、キーボードショートカットの不足など。
- 習得性の問題。たとえば、主要な決定操作のユーザに対するガイダンス不足。

特定の品質特性のテストに関する考慮事項については、本シラバスの第 4 章で説明する。

2.4.3 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、それぞれのリスクに関連する発生確率や影響を判定する。

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、リスク顕在化の発生確率および顕在化した際の影響を評価する。発生確率は、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。言い換えると、テクニカルリスクが対象となる。テクニカルテストアナリストは、各リスクアイテムに潜在するテクニカルリスクレベルの算出と把握に貢献するが、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得る影響の把握に貢献しなければならない。

リスク発生の影響は、多くの場合、ユーザ、顧客、またはその他のステークホルダに対する影響の重要度と考えることができる。言い換えると、ビジネスリスクが対象となる。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の使用頻度
- ビジネス損失
- 財政的、環境保護的、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拘束
- 安全上の課題
- 賠償金、ライセンスの喪失
- 妥当な回避策の欠如
- フィーチャの可視性
- 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、テストマネージャにより確立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、言葉(たとえば、低、中、高)や数値により分類できる。定義された尺度でリスクを客観的に測定する方法でないと、本当の定量的な測定値は求められない。一般的に、確率または可能性、コストまたは重大性を正確に測定することは非常に困難なので、リスクレベルは一般的に定性的に決定する。

数値を定性値に割り当てることはできるが、その数値は本当の定量的な測定値とはならない。たとえば、テストマネージャは、ビジネスリスクを 1 から 10 の 10 段階に分類し、1 が最も高い、つまりビジネスに対する影響度

が最も危険なリスクであるとすることができる。発生確率(テクニカルリスクのアセスメント)および影響度(ビジネスリスクのアセスメント)を割り当てたら、これら 2 つの値を掛け合わせて、各リスクアイテムの全体的なリスクの評価点を決定できる。次に、全体的なリスクの評価点を使用して、リスク軽減活動の優先度を決定する。一部のリスクベースドテストのモデル(PRISMA® [vanVeenendaal12]など)では、リスク値の組み合わせが行われないので、テクニカルリスクとビジネスリスクのそれぞれに対して個別に対応できる。

2.4.4 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、次のことに努める必要がある。

- テストアイテムが合格したか不合格したかを一意に示す、適切に設計されたテストケースを使用する、または要件、設計、およびユーザドキュメントなどのソフトウェア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。
- 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、発生確率や影響度、またはその両方を調整する。
- テスト期間で収集された情報により識別される新しいリスクを認識する。

プロダクト(品質)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に対する軽減策の一つである。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ことが保証され、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、正確なテストデータ収集のための時機の調査、現実的なユーザ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観察を行うことで、リスク軽減のオプションを決定することを支援する。

2.4.4.1 テストの優先度付け

リスクのレベルもテストの優先度付けに使用する。たとえば、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エキスパートと協力して、正確性を確立する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必要がある。同様に、テストアナリストが、新しいプロダクト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、統合レベルのテスト時に実施されるよう早期に使用性テストを優先度付けし、使用性の問題がテストの早期に識別され、解決されるようにする必要がある。この優先度付けは、計画段階の可能な限り早期に考慮され、必要なテストを必要なタイミングで実施できるようにする必要がある。

場合によっては、高リスクのテストをすべて、低リスクのテストよりも先に実行したり、厳格なリスク順にテストを実行したりする。これは「縦型探索」(depth-first)とも呼ばれる。また別の場合は、「横型探索」(breadth-first)とも呼ばれる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からテストのサンプルを選択することもある。この方法では、リスク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、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。

2.4.4.2 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は、新しいリスク分析の対象であり、次の要素を考慮する必要がある。

- 新規に開発した、または大幅に変更したプロダクトのすべてのリスク
- テスト時に発見された不安定または欠陥の多い領域
- 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(テストカバレッジの低い)領域

テストに割り当てる時間を増やせる場合は、リスクカバレッジをより低いリスクの領域に拡張できる。

3. テスト技法 – 825 分

用語

境界値分析(BVA)、原因結果グラフ法、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法、組み合わせテスト、デシジョンテーブルテスト、欠陥分類法、欠陥ベースの技法、ドメイン分析、エラー推測、同値分割法、経験ベースの技法、探索的テスト、直交表、直交表テスト、ペアワイズテスト、要件ベースドテスト、仕様ベースの技法、状態遷移テスト、テストチャータ、ユースケーステスト、ユーザストーリーテスト

「テスト技法」の学習の目的

3.2 仕様ベースの技法

TA-3.2.1 (K2)原因結果グラフの使用方法を説明する。
TA-3.2.2 (K3)定義されたカバレッジを達成するために、同値分割テスト設計技法を適用して、特定の仕様アイテムからテストケースを記述する。
TA-3.2.3 (K3)定義されたカバレッジを達成するために、境界値分析テスト設計技法を適用して、特定の仕様アイテムからテストケースを記述する。
TA-3.2.4 (K3)定義されたカバレッジを達成するために、デシジョンテーブルテスト設計技法を適用して、特定の仕様アイテムからテストケースを記述する。
TA-3.2.5 (K3)定義されたカバレッジを達成するために、状態遷移テスト設計技法を適用して、特定の仕様アイテムからテストケースを記述する。
TA-3.2.6 (K3)定義されたカバレッジを達成するために、ペアワイズテスト設計技法を適用して、特定の仕様アイテムからテストケースを記述する。
TA-3.2.7 (K3)定義されたカバレッジを達成するために、クラシフィケーションツリーテスト設計技法を適用して、特定の仕様アイテムからテストケースを記述する。
TA-3.2.8 (K3)定義されたカバレッジを達成するために、ユースケーステスト設計技法を適用して、特定の仕様アイテムからテストケースを記述する。
TA-3.2.9 (K2)アジャイルプロジェクトでテストをガイドするために、ユーザストーリーを使用する方法を説明する。
TA-3.2.10 (K3)定義されたカバレッジを達成するために、ドメイン分析テスト設計技法を適用して、特定の仕様アイテムからテストケースを記述する。
TA-3.2.11 (K4)発見される可能性のある欠陥の種類を判別し、適切な仕様ベースの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 欠陥ベースの技法

TA-3.3.1 (K2)欠陥ベースの技法の適用方法を説明し、仕様ベースの技法と使い方を区別する。
TA-3.3.2 (K4)優れた分類法の基準を使用して、特定の状況で適用できるよう欠陥分類法を分析する。

3.4 経験ベースの技法

TA-3.4.1 (K2)経験ベースの技法の原則と、仕様ベースおよび欠陥ベースの技法と比較した場合の長所と短所を説明する。
TA-3.4.2 (K3)与えられたシナリオに対して探索的テストを指定し、結果がレポートされる方法を説明する。
TA-3.4.3 (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仕様ベース、欠陥ベース、または経験ベースの技法のどれ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設計技法は、次のカテゴリに分類される。
- 仕様ベース(または動作ベース、ブラックボックス)
- 欠陥ベース
- 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関係なく、どのようなテスト活動にも使用できる。

技法に関するこれら 3 つのカテゴリすべてが、機能または非機能の品質特性の両方をテストするために使用できる。非機能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設計技法は、主に、最適なテストデータの決定(たとえば、同値分割)、またはテスト順序の導き出すこと(たとえば、状態モデル)に重点を置いている。一般的に、技法を組み合わせて、テストケースを完成させる。

3.2 仕様ベースの技法

仕様ベースの技法は、コンポーネントまたはシステムの内部構造を参照することなく、テストベースの分析に基づいてテストケースを導き出すためのテスト条件に適用する。

仕様ベースの技法の特徴を次に示す。
- たと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。

一部の技法は、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も提供する。カバレッジ基準を完全に満たすことは、テストセット全体が完全であることではなく、その技法に基づくカバレッジを増やすためにこのモデルが示すテストはこれ以上ないことを意味する。

仕様ベースドテストは、通常、システム要件ドキュメントに基づく。要求仕様は特に機能性の領域でシステムの振る舞いを定めるので、要件からテストケースを導き出す作業は、多くの場合、システム動作のテストの一部となる。場合によっては、文書化された要件が存在しないことがあるが、旧システムからの機能の置き換えといった暗黙的な要件は存在する。

数多くの仕様ベースド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、テスト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。

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率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。同じ方法で処理される値のセットとして作成される同値クラス(同値分割とも呼ばれる)を作成するために分割を使用する。同値クラスから 1 つの代表値を選択することにより、同じ同値クラス内のすべてのアイテムに対するカバレッジとみなす。

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メンバ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が相互作用しない場合に適している。値セットの選択は、有効同値クラスおよび無効同値クラス(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同値クラス)に適用できる。この技法は、テストする値の範囲を同値クラス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。この技法は、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、同値クラス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、分割点を注意深く選択することも重要である。たとえば、正の数と負の数を受け入れる入力欄に対しては、正負で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 つの有効同値クラスとしてテストする必要がある。同様に、ゼロを許容するかどうかについても、別の同値クラスとなる。テストアナリストは、最適な分割を決定するために、基になる処理を理解することが重要である。

カバレッジ
カバレッジは、テストされた同値クラスの数を、識別された同値クラスの数で除算することにより決定する。単一の同値クラス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。

検出できる欠陥の種類
この技法では、さまざまな値の処理に関する機能的な欠陥を検出できる。

3.2.2 境界値分析

境界値分析（BVA）は、順序付けられた同値分割の境界上に存在する値をテストするために使用する。BVA には、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 つの値を用いる方法では、境界値（境界上）と境界を少し超えた値（最小の増加分）を使用する。たとえば、同値クラスに 1 から 10 の範囲の値が 0.5 刻みで存在する場合、2 つの値を用いる方法の上位の境界値は、10 と 10.5 である。下位の境界値は、1 と 0.5 である。境界は、定義されている同値クラスの最大値と最小値に基づいて定義する。

3 つの値を用いる方法では、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、上位の境界値テストは、9.5、10、および 10.5 である。下位の境界値テストは、1.5、1、および 0.5 である。2 つの値を用いる方法または 3 つの値を用いる方法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必要がある。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クラスが存在するときに適している。境界上および境界外の概念を考慮するために、順序付けが必要である。たとえば、数値の範囲は、順序付けられた同値クラスである。すべて矩形オブジェクトで構成される同値クラスは、順序付けられた同値クラスではなく、境界値も持たない。境界値分析は、数値の範囲以外にも、次の項目に適用できる。

- 数値属性を持つ非数値変数（たとえば、A3、B5 といった用紙サイズなど）
- ループ（ユースケース内のループを含む）
- 格納されているデータ構造
- 物理オブジェクト（メモリを含む）
- 時間により判定される活動

制限／注意事項
この技法の正確性は、同値分割の正確な識別に依存するので、同値分割法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、有効同値および無効同値の増分に注意する必要がある。境界値分析では順序付けられた同値クラス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。たとえば、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分割と、最大数を 1 つ超えたセルで始まる同値クラス（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、識別された境界条件（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）の数で除算することにより決定する。これが、境界値テストのカバレッジとなる。

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、追加の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、負荷の耐久性（たとえば、システムは 10,000 人の同時ユーザをサポートする）などにおける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

デシジョンテーブルは、条件の組み合わせ間の相互作用をテストするために使用する。デシジョンテーブルは、条件に関連するすべての組み合わせのテストを確認し、すべての可能な組み合わせがテスト対象のソフトウェアにより処理されることを検証する明確な方法を提供する。デシジョンテーブルテストのゴールは、条件、関係、および制約のすべての組み合わせを確実にテストすることである。可能な組み合わせのすべてをテストしようとすると、デシジョンテーブルが非常に大きくなることがある。すべての可能な組み合わせの数から、「関心のある」組み合わせの数に理にかなって削減する方法を、簡単化したデシジョンテーブルのテストと呼ぶ。この技法を使用すると、出力に関連しない条件のセットを取り除くことができ、異なる出力を生成するものだけに組み合わせを削減できる。重複するテストまたは条件の組み合わせが可能ではないテストは取り除く。完全なデシジョンテーブルと簡単化したデシジョンテーブルのどちらを使用するかの決定は、通常、リスクに基づいて行う [Copeland03]。

適用
この技法は、一般的に、統合テスト、システムテスト、および受け入れテストのテストレベルで適用する。コードにもよるが、コンポーネントに判定ロジックのセットが含まれている場合に、コンポーネントテストに適用することもある。この技法は、要件がフローチャートまたはビジネスルールのテーブルの形式で提供される場合に、特に役立つ。デシジョンテーブルはまた、要件定義の技法であり、一部の要求仕様を、この形式ですでに提供していることがある。要件がテーブル形式またはフローチャート形式で提供されていない場合、条件の組み合わせは、通常、説明文で提供する。デシジョンテーブルを設計する場合、定義された条件の組み合わせ、および明確には定義されていないが存在する可能性のある組み合わせを考慮する必要がある。有効なデシジョンテーブルを設計するには、テスト担当者は、すべての条件の組み合わせに対する期待結果を仕様またはテストオラクルから導き出すことができなければならない。すべての相互作用条件が考慮された場合のみ、デシジョンテーブルを優れたテスト設計ツールとして使用できる。

制限／注意事項
すべての相互作用条件を見つけることは、特に要件が適切に定義されていない、または存在しない場合に、困難な作業となる。それでも、条件のセットを準備すること、不明である期待結果を決定することは珍しくない。

カバレッジ
デシジョンテーブルに関する最小限のテストカバレッジでは、各列に 1 つのテストケースを持つ。これは、複合条件が存在せず、すべての可能な条件組み合わせが 1 つの列に記録されていることを想定する。テストをデシジョンテーブルから決定する場合、テスト対象のすべての境界条件も考慮する必要がある。これらの境界条件を考慮することにより、ソフトウェアを適切にテストするために必要なテストケースの数が増える可能性がある。境界値分析と同値分割法は、デシジョンテーブル技法を補完する。

検出できる欠陥の種類
欠陥の種類は、予期しない結果をもたらす特定の条件の組み合わせに基づく不正な処理を含む。デシジョンテーブルを作成するときに、欠陥を仕様ドキュメントで発見することがある。欠陥の最も一般的な種類は、欠落（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。テストでも、処理されない、または適切に処理されない条件組み合わせの問題を発見することがある。

3.2.4 原因結果グラフ法

原因結果グラフは、ユーザストーリーやフローチャートなど、プログラムの機能ロジック（つまり「ルール」）を記述するすべてのソースから生成できる。これらのグラフは、プログラムの論理構造を視覚的に把握するのに役立ち、一般的に、デシジョンテーブルを作成するための基礎として使用する。原因結果グラフまたはデシジョンテーブルとして判定を把握することにより、プログラムロジックの体系的なテストカバレッジを達成できる。

適用
原因結果グラフは、デシジョンテーブルと同じ状況、および同じテストレベルに適用できる。特に、原因結果グラフは、結果を発生させる条件の組み合わせ(因果関係)、結果を除外する条件組み合わせ(not)、すべて真でないと結果が発生しない複数の条件(and)、およびどれか1つでも真だとすると特定の結果が発生する複数の条件(or)を示す。これらの関係は、原因結果グラフを使用することで、説話的な記述よりも確認しやすくなる。

制限／注意事項
原因結果グラフ法は、他のテスト設計技法に比べて、習得するのにより多くの時間と労力を必要とする。ツールによるサポートも必要とする。原因結果グラフは特別な表記方法を使用するため、グラフの作成者および参照者は、この表記方法を理解する必要がある。

カバレッジ
最少のカバレッジを達成するためには、原因から結果につながるそれぞれの線をテストしなければならない。原因結果グラフでは、データおよびロジックフローに制約を定義する手段を利用できる。

検出できる欠陥の種類
原因結果グラフを使用すると、デシジョンテーブルを使用して見つかるものと同じような組み合わせの欠陥を見つけることができる。さらに、グラフを作成することにより、テストベースで必要な詳細度合いを定義できるので、テストベースが詳細化され、テストベースの品質が向上し、テスト担当者は欠落している要件を識別できる。

3.2.5 状態遷移テスト

状態遷移テストは、ソフトウェアが、有効または無効な遷移を介して、定義された状態の開始および終了を実行できるかどうかをテストするために使用する。イベントの発生により、ソフトウェアはある状態から別の状態に遷移し、アクションを実行する。テストとして選択したい遷移パスを促す条件(ガード条件または遷移ガードとも呼ばれる)を考慮してイベントを特定する。たとえば、ログインイベントの場合、有効なユーザ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。

状態遷移は、状態間のすべての有効な遷移を図で示す状態遷移図、または有効および無効の両方の可能性のあるすべての遷移を示す状態遷移表を使用して追跡する。

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(たとえば、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、状態遷移表または状態遷移図を定義する作業の最も困難な部分である。ソフトウェアにユーザインターフェースが含まれる場合、一般的に、ユーザ向けに表示されるさまざまな画面を使用して、状態を定義する。組込みソフトウェアの場合、状態はハードウェアで発生する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。これは 0 スイッチとも呼ばれる。すべての遷移をそのままテストすることにより、いくつかの状態遷移の欠陥を見つけることができるが、トランザクションのシーケンスをテストすることにより、より多くの欠陥を見つけることができる。2 つの連続した遷移のシーケンスを 1 スイッチ、3 つの連続した遷移のシーケンスを 2 スイッチ、以降、同様の方式で呼ぶ(これらのスイッチを、N-1 スイッチと呼ぶこともある。ここで N は、遷移するトランザクションの数を表す。たとえば、単一トランザクション(0 スイッチ)は、1-1 スイッチである)[Bath08]。

カバレッジ
他の種類のテスト技法と同じく、テストカバレッジには程度の階層がある。許容できる最低限のカバレッジ度合いは、すべての状態に遷移し、すべての遷移を通ることである。100%のトランザクションカバレッジ(100%の 0

スイッチカバレッジまたは 100%の論理ブランチカバレッジとも呼ばれる)は、システム設計または状態遷移モデル(図または表)に欠陥がない限り、すべての状態に遷移し、すべての遷移を通ったことを保証する。状態と遷移との間の関係によっては、特定の遷移を 1 回実行するために、別の遷移を複数回通ることが必要になる。

用語「N スイッチカバレッジ」は、カバーする遷移の数を表す。たとえば、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テストする。この場合、ループに含まれるすべての状態をテストする必要がある。

これらのいずれの方式でも、より高い度合いのカバレッジにするには、すべての無効な遷移を含める。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する必要がある。

検出できる欠陥の種類
典型的な欠陥としては、以前の状態で発生した処理の結果である現在の状態での不正な処理、不正またはサポートされていない遷移、終了しない状態、および必要にもかかわらず存在しない状態または遷移がある。状態機械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(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。

3.2.6 組み合わせテスト技法

組み合わせテストは、複数の値を持つ複数のパラメータを含むソフトウェアをテストする場合で、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。どのようなパラメータのどのようなオプションでも、他のどのようなパラメータのどのようなオプションと組み合わせられるという意味で、パラメータ間に依存関係はなく、両立させるべきである。クラシフィケーションツリーを使用すると、特定のオプションが両立しない場合に、該当する組み合わせを除外できる。これは、組み合わされたパラメータが相互に影響しないとみなされるものではない。つまり、許容できる範囲で相互に影響する可能性がある。

組み合わせテストは、事前に定義した度合いのカバレッジを達成するために、これらの組み合わせの適切なサブセットを識別する手段を提供する。テストアナリストは、1 つ、2 つ、3 つ、それ以上など、組み合わせに含めるアイテムの数を選択できる[Copeland03]。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存在する(サンプルについては、www.pairwise.org for samples を参照)。これらのツールでは、パラメータおよびそれらの値をリストするか(ペアワイズテストおよび直交表テスト)、視覚的な形式(クラシフィケーションツリー)で表す必要がある[Grochtmann94]。ペアワイズテストは、パラメータをペアとして組み合わせてテストする場合に適用する方法である。直交表は、すでに定義されている数学的に裏付けられた表である。テストアナリストはこの表を使用して、表内の変数をテスト対象となるアイテムに置き換えることで、テスト時のカバレッジ度合いを達成する組み合わせのセットを生成できる[Koomen06]。クラシフィケーションツリーを使用すると、テストアナリストは、テスト対象の組み合わせの規模(たとえば、2 つの値の組み合わせ、3 つの値の組み合わせなど)を定義できる。

適用
パラメータ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。たとえば複数の入力欄のある画面などの、一部のテストケースは、取りうる値が複数存在するパラメータが複数存在する。この場合、パラメータ値の組み合わせで、テストケースのための入力データを作成する。さらに、多数の次元でシステムを構成できる場合、組み合わせの数が非常に大きくなる可能性がある。いずれの状況でも、組み合わせテストを使用して、テスト可能な規模の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では、組み合わせテストを適用して組み合わせの数を減少させる前に、まず、各パラメータに同値クラスまたは、別の抽出の仕組みを適用して、各パラメータに対する値の数を削減する。

これらの技法は通常、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。たとえば、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、その欠陥はこの種類のテスト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。

パラメータとそれぞれの値を識別するのが困難な場合がある。特定カバレッジの度合いを満たす組み合わせの最小セットを手作業で見つけることは困難であり、一般的には、ツールを使用する。一部のツールは、最終的な組み合わせを選択するときに、いくつかの組み合わせ（または下位の組み合わせ）を強制的に追加または削除できる。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、パラメータを重要として扱うかどうかを選択できる。

カバレッジ
この技法は、いくつかのカバレッジの度合いをサポートする。最も低いレベルのカバレッジは、1 ワイズカバレッジまたはシングルカバレッジと呼ぶ。このカバレッジの度合いでは、すべてのパラメータにおけるそれぞれの値を、選択した組み合わせの 1 つ以上に含める必要がある。次のレベルのカバレッジは、2 ワイズカバレッジまたはペアワイズカバレッジと呼ぶ。このカバレッジの度合いでは、任意の 2 つのパラメータについて、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。この考え方は、n ワイズカバレッジに汎用化できる。n ワイズカバレッジでは、n 個のパラメータの任意のセットについて、値のすべての組み合わせを、選択した組み合わせに含める必要がある。n の値が大きいほど、100%のカバレッジを達成するために必要な組み合わせの数が大きくなる。これらの技法の最小度合いのカバレッジは、ツールが生成するすべての組み合わせごとに 1 つのテストケースを持つことである。

検出できる欠陥の種類
この種類のテストで見つかる最も一般的な欠陥のタイプは、いくつかのパラメータ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、システムの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。ユースケースは、アクターと、何らかの目標を達成するシステムとの間の相互作用の観点で定義する。アクターとしては、ユーザまたは外部システムを挙げることができる。

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。統合の状況によっては統合テストにも、またコンポーネントの振る舞いに応じてコンポーネントテストでも使用できる。ユースケースはシステムの実際の使い方を表すので、多くの場合、性能テストの基礎となる。システムに現実的な負荷をかけるために、ユースケースが示すシナリオを仮想ユーザに割り当てることがある。

制限／注意事項
有効なユースケースは、現実的なユーザトランザクションを反映する必要があり、この情報は、ユーザまたはユーザの代表者から入手する。ユースケースが実際のユーザの活動を正確に反映しない場合、その価値は低下する。テストカバレッジを徹底するには、さまざまな代替パス（フロー）を正確に定義する必要がある。ユースケースは、ガイドラインとして扱い、テスト対象の完全な定義として扱ってはならない。これは、ユースケースが要件全体の明確な定義を提供しない場合があるためである。ユースケースの記述からフローチャートなど他のモデルを作成することも、テストの正確性を向上させ、ユースケース自体を検証するために役立つことがある。

カバレッジ
ユースケースの最小のカバレッジは、主要（正常）パス用の 1 つのテストケースと、代替パスまたはフローごとに 1 つのテストケースを含む。代替パスは、例外パスおよび失敗するパスを含む。また、主要パスの拡張として示

されることもある。カバレッジの割合を求めるには、テスト済みのパスの数を、主要パスと代替パスの合計で除算する。

検出できる欠陥の種類
欠陥には、定義済みシナリオの誤った処理、代替パス処理の欠落、提示された条件の誤った処理、読みにくいまたは不正なエラーレポートがある。

3.2.8 ユーザストーリーテスト

スクラムなど一部のアジャイル方法論では、単一のイテレーション内で設計、開発、テスト、および実証可能な小さい機能ユニットを示すユーザストーリーを使用して、要件を準備する[Cohn04]。これらのユーザストーリーは、実装対象の機能性の記述、すべての非機能条件、さらには、ユーザストーリーが完了したとみなすための満たすべき受け入れ基準を含む。

適用
ユーザストーリーは主として、アジャイル、および類似の反復性または拡張性のある環境で使用する。また、機能テストと非機能テストの両方で使用する。開発者が開発したコードを次のテストレベルのタスク(たとえば、統合テスト、性能テスト)を実行するチームメンバに渡す前に、ユーザストーリー向けに実装された機能性を実証することを想定している場合には、ユーザストーリーをすべてのテストレベルで使用する。

制限／注意事項
各ストーリーでは機能が少しずつ変化しているので、実現する機能性の一部を実際にテストするために、ドライバおよびスタブを生成する要件が存在することがある。この要件を満たすには、通常、プログラミング能力と、API テストツールなどテストに役立つツールを使用する能力が必要である。ドライバおよびスタブの作成は、通常、開発者が担当するが、テクニカルテストアナリストもこのコードを生成したり、API テストツールを使用したりする。ほとんどのアジャイルプロジェクトがそうであるように、継続的インテグレーションモデルを使用する場合、必要とするドライバとスタブを最小限に抑えることができる。

カバレッジ
ユーザストーリーの最小カバレッジは、指定した各受け入れ基準の達成を検証することである。

検出できる欠陥の種類
欠陥は、通常、指定した機能をソフトウェアが提供できないという機能面での欠陥である。既存機能に関する新しいストーリーにより、機能の統合問題という欠陥が検出される。ストーリーは個別に開発するので、性能、インターフェース、およびエラー処理の問題を検出することもある。テストアナリストは、新しいストーリーをテスト用にリリースした場合は常に、提供している個々の機能のテストと、統合テストの両方を実施することが重要である。

3.2.9 ドメイン分析

ドメインは、定義済みの値のセットである。このセットは、1 つの変数の値の範囲(1 次元ドメイン。たとえば、25 歳以上 65 歳以下の男性)として、または相互作用する変数の値の範囲(多次元ドメイン。たとえば、25 歳以上 65 歳以下かつ体重が 70kg 以上 89kg 以下)として定義する。多次元ドメイン用の各テストケースは、関連する各変数の妥当な値を含む必要がある。

1 次元のドメイン分析では、通常、同値分割法と境界値分析を使用する。分割領域を定義したら、テストアナリストは各分割領域から分割領域内(IN)、分割領域外(OUT)、分割領域の境界上(ON)、および分割領域境界に隣接(OFF)を表す値を選択する。これらの値を決定したら、これらの境界条件を使用して各分割領域をテストする[Black07]。

ドメイン理論に基づく方式を用いると、(1次元ドメインでは)テストケース数は変数の数に比例することになるが、多次元ドメインでこれらの方法を使用して生成されるテストケース数は、関連する変数の数が多くなるに従って指数的に増加する。また、形式的な方式は、同値分割法と境界値分析を使用せず欠陥理論(フォールトモデル)を取り込んだものであり、その小さなテストセットは、大量かつヒューリスティックなテストセットが見逃すような

多次元ドメインの欠陥を見つけられる。多次元ドメインに対処する場合、そのテストモデルをデシジョンテーブル（または「ドメインマトリクス」）として構築してもよい。3 次元を超える多次元ドメインのテストケースを識別するには、多くの場合、コンピュータの支援を必要とする。

適用
デシジョンテーブル、同値分割法、および境界値分析は、重要な領域と故障の発生する可能性の高い領域をカバーするための小さなテストセットを作成する。ドメイン分析は、それらの技法に使用される。多くの変数が潜在的に相互作用しているためにデシジョンテーブルでは扱いづらい場合、ドメイン分析を適用することが多い。ドメイン分析はすべてのテストレベルで使用できるが、統合テストとシステムテストで頻繁に使用する。

制限／注意事項
徹底したドメイン分析を実施するには、さまざまなドメインとドメイン間の潜在的な相互作用を識別するために、ソフトウェアを十分に理解する必要がある。識別されないままのドメインが存在すると、テストの不足が大量に発生する可能性があるが、OFF および OUT の変数が未検出のドメインに存在する可能性があるので、そのドメインを検出できる。ドメイン分析は、テスト領域を定義するために開発者と連携する際に使用する強力な技法である。

カバレッジ
ドメイン分析の最小カバレッジは、各ドメインで IN、OUT、ON、および OFF のそれぞれの値をテストすることである。たとえば、あるドメインの OUT の値が別のドメインの IN の値であるといった場合など、値が重複している場合にテストを重複して実行する必要はない。このため、実際に必要なテストの数は、ドメイン当たり 4 未満となることが多い。

検出できる欠陥の種類
欠陥には、ドメイン内の機能的な問題、境界値の取り扱い、変数の相互作用の問題、およびエラー処理（特にドメインに対する無効な値に関するエラー処理）がある。

3.2.10 技法の組み合わせ

ときには、技法を組み合わせてテストケースを作成することがある。たとえば、デシジョンテーブルで識別される条件が、条件を満たすための複数の方法を発見するために、同値分割法の対象になることがある。この場合、テストケースは条件のすべての組み合わせだけでなく、分割されている条件もカバーする。また、追加テストケースの生成により、同値分割もカバーする。適用する特定の技法を選択する場合、テストアナリストは、技法の適用、制限／注意事項、およびテストの目標を、カバレッジと検出すべき欠陥の観点から考慮する必要がある。状況によっては、「最善」の技法が 1 つではないことがある。技法を組み合わせると多くの場合、最も完全なカバレッジを達成するが、それぞれの技法を正しく適用するための十分な時間とスキルが必要になる。

3.3 欠陥ベースの技法

3.3.1 欠陥ベースの技法の使用

欠陥ベースのテスト設計技法は、探す欠陥のタイプをテスト設計の基本として使用する技法であり、欠陥のタイプについて既知の情報からテストを体系的に導き出す。テストを仕様から導き出す仕様ベースドテストと異なり、欠陥ベースのテストでは、テスト対象のソフトウェアから完全に独立している欠陥分類法（つまり、分類された一覧）からテストを導き出す。分類法は、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を含むことができる。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も、テスト対象を絞るための基礎として使用することがある。テスト担当者はこのテスト技法を使用することにより、欠陥のタイプの種類に対象を絞ったり、特定のタイプの既知および一般的な欠陥の欠陥分類法を通して体系的に作業したりできる。テストアナリストは分類法データを使用して、テストのゴール（つまり、特定の欠陥のタイプを見つけること）を決定できる。また、この情報から欠陥が存在する場合に、その欠陥を出現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。複数の種類のソフトウェアに適用できる標準の分類法が存在する。この、プロダクト固有ではない種類のテストは、業界標準の知識を活用して、特定のテストを導き出すのに役立つ。業界固有の分類法に従うことにより、欠陥の存在に関するメトリクスをプロジェクト全体および組織全体で追跡できる。

制限／注意事項
複数の欠陥分類法が存在し、これらを利用することにより、使用性など特定の種類のテストに焦点を当てることができる。利用可能な欠陥分類法が複数存在する場合、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。たとえば、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、可能性が高い、または頻繁に検出す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想定するカバレッジを定義する必要がある。

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。実際、欠陥ベースの技法のカバレッジ基準は、カバレッジに対する汎用的なルールのみを提供するという点において、仕様ベースの技法に比べて体系的ではない傾向にある。有用なカバレッジの範囲の選定に関する特有の判断は、自由裁量である。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、欠陥を見つけるためのこの技法に基づいた有用なテスト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、一般的に使用する分類法により異なる。ユーザインターフェースの分類法を使用すると、検出する欠陥の大半はユーザインターフェースに関連するが、他の欠陥も特定のテストの副産物として検出できる。

3.3.2 欠陥分類法

欠陥分類法は、欠陥のタイプを分類した一覧である。これらの一覧は、非常に汎用的で高位レベルのガイドラインとして使用できるか、または非常に限定的に使用することができる。たとえば、ユーザインターフェースの欠陥に関する分類法は、機能、エラー処理、グラフィック表示、および性能などの一般的なアイテムを含むことがある。詳細分類は、考えられるすべてのユーザインターフェースオブジェクト(特にグラフィカルユーザインターフェース向け)の一覧を含むことがあり、また、これらのオブジェクトに関する、次のような不適切な処理を示すこともある。

- テキスト欄
  - 有効なデータが受け付けられない
  - 無効なデータが受け付けられる
  - 入力長が検証されない
  - 特殊文字が検出されない
  - ユーザエラーメッセージが有益でない
  - ユーザは誤りのあるデータを修正できない
  - ルールが適用されない
- 日付欄
  - 有効な日付が受け付けられない
  - 無効な日付が拒否されない
  - 日付範囲が検証されない
  - 精度データが正しく処理されない
  - ユーザは誤りのあるデータを修正できない
  - ルールが適用されない(例: 終了日は開始日よりも未来にある必要がある)

欠陥分類法には、取得可能で形式的な分類法から、さまざまな組織により特定の目的向けに設計されている分類法まで、多くの種類が存在する。また、内部で欠陥分類法を開発して、その組織で一般的に見つかる特定の欠陥を対象にするために使用することもできる。新しい欠陥分類法を作成するか、利用可能な欠陥分類

法をカスタマイズする場合、最初に分類法の目標または目的を定義する必要がある。たとえば、ゴールは、本番システムで検出されたユーザインターフェースの問題を識別することであったり、入力欄の処理に関連する問題を識別することであったりする。

分類法を作成するには、次の手順を実行する。

1. ゴールを作成し、期待する度合いの詳細を定義する
2. 基準として使用する特定の分類法を選択する
3. 値と、組織内または外部での実践で発生した一般的な欠陥を定義する

分類法の詳細度を高めるほど、開発とメンテナンスにかかる時間が多くなるが、テスト結果におけるより高い再現性をもたらす。詳細な分類法は冗長になる可能性があるが、テストチームは、情報またはカバレッジの不足を発生させることなく、テストを分割して実施できる。

適切な分類法を選択したら、テスト条件とテストケースを作成するために使用できる。リスクベースの分類法は、特定のリスク領域に焦点を当ててテストするのに役立つ。分類法は、使用性や性能など、非機能領域に対しても使用できる。多くの分類法の一覧が、IEEE やインターネット上で提供されている。

3.4 経験ベースの技法

経験ベースのテストは、テスト担当者のスキルと直感、および類似のアプリケーションや技術での経験を活用する。これらのテストは欠陥を見つけるのに有効であるが、特定のカバレッジを達成したり、再利用可能なテスト手順を生成したりする場合は、他の技法ほど適していない。システムドキュメントが適切でない場合やテスト時間が厳しく制限されている場合、またはテストチームがテスト対象のシステムに精通している場合には、経験ベースのテストは、より構造化された方法よりも優れた代替策となることがある。経験ベースのテストは、詳細なテストドキュメント、高い再現性、またはテストカバレッジを精緻に評価する能力を必要とするシステムでは、不適切なことがある。

動的またはヒューリスティックな方法では、テスト担当者は通常では経験ベースのテストを使用し、事前に計画した方法よりも、事象に対処する方法でテストを行う。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。

3.4.1 エラー推測

エラー推測を使用する場合、テストアナリストは経験を生かして、コードの設計および開発時に作成され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。たとえば、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、さまざまに異なる値をパスワード欄に入力するようにテストを設計する。

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[Myers79]。

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スコープ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、一般的な誤りやエラーをテストする場合にも効果的に使用できる。チェックリストおよび分類法は、テストをガイドする場合にも役立つことがある。

制限／注意事項
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取り込まれる欠陥の種類に精通している熟練のテスト担当者が使用する場合に最善となる。エラー推測はよく使用されるが、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に劣っていることがある。

カバレッジ
分類法を使用すると、該当するデータフォールトと欠陥の種類によりカバレッジを決定できる。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。この技法の成果は、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、仕様ベースドテストでは検出されないことがある。

3.4.2 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アイテムを、高位レベルで汎用化したリストや、プロダクトに対して検証を行うルール、および基準のセットを使用する。これらのチェックリストは、標準、経験、および他の考慮事項のセットに基づいて構築する。チェックリストベースドテストの例としては、アプリケーションをテストするための基準として採用するユーザインターフェース標準チェックリストがある。

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行うプロジェクトで、最も有効に使用される（たとえば、ユーザ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）。チェックリストは高位レベルであり、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抜ける傾向にある。このため、これらを埋めるために、テスト担当者の知識を使用することが必要になる。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。また、すべてのテストレベルで使用できる。チェックリストは、回帰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
高位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。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、時間と共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
カバレッジはチェックリストと同様であるが、チェックリストが高位レベルであるという特性により、テスト結果は、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には、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もたらされるものがある。チェックリストを使用することにより、テスト時にデータおよび手順の新しい組み合わせが可能になるので、テストを最新の状態に維持できる。

3.4.3 探索的テスト

探索的テストには、テスト担当者がプロダクトとその欠陥の学習、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目標を動的に調整し、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、相互作用的で創造的である。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、ドキュメントが存在しないか、他のテスト技法では適切でない場合に使用することが多い。また、多くの場合、他のテストを補完したり、追加のテストケースを開発するための基準を提供したりするために使用する。

制限／注意事項
探索的テストは、マネジメントおよびスケジュールが困難になることがある。カバレッジがまちまちでまとまりがなく、再現するのが困難である。探索的テストをマネジメントする 1 つの方法として、テストセッションでカバーする領域を指定したチャータと、テストで使える時間を決定するタイムボックスを使用することができる。1回のテストセッション終了時、または複数のセッション終了時に、テストマネージャはテスト結果を収集し、次のセッションのチャータを決定するために報告会を開催する。この報告会は、大きなテストチームまたはプロジェクトの場合調整するのが困難である。

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で、テストケースを作成して行われることもある。これにより、探索的テストに時間を割り当てることが可能になり、計画済みのカバレッジを別のテスト成果を使用して追跡できる。

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。詳細を入念に調査して、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。

カバレッジ
チャータは、タスク、目的、および成果物を特定するために作成されることがある。その後、それらの目的を達成するために、探索的テストセッションを計画する。チャータはまた、テスト作業で重点を置く領域、テストセッションのスコープ内およびスコープ外、計画したテストを完了するために必要なリソースも識別することもできる。形式的な手続き化されたテストの手順を必要とせずに対処できる特定の欠陥のタイプ、および他の潜在的に問題のある領域に重点を置くために、セッションを使用することがある。

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、手続き化された機能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。

3.4.4 最善の技法の適用

欠陥ベースおよび経験ベースの技法では、欠陥検出を向上させるために、テスト対象に関する欠陥および他のテスト経験についての知識を適用する必要がある。これらの知識の範囲は、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、事前計画したセッション、さらには手続き化されたセッションにまで及ぶ。これらは常に有効であり、さらに次の状況では特別な価値を提供する。

- 利用できる仕様書がない
- テスト対象のシステムに関するドキュメントが乏しい
- テストを詳細に設計および作成するための十分な時間がない
- テスト担当者は、ドメイン、技術、またはその両方に十分な経験を持つ

- 手続き化されたテストを基にさまざまテストを行うことが、テストカバレッジを最大化するためのゴールである
- 操作故障を分析する必要がある

欠陥ベースおよび経験ベースの技法は、仕様ベースの技法の体系的な弱点により発生するテストカバレッジの差異を埋めるので、これらを組み合わせて使用することも有効である。仕様ベースの技法がそうであるように、すべての状況に対して 1 つの技法で完全に対処することはできない。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、および影響するその他の要因を考慮して、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必要がある。

4. ソフトウェア品質特性のテスト - 120 分

用語

アクセシビリティテスト、正確性テスト、魅力性、ヒューリスティック評価、相互運用性テスト、習得性、運用性、合目的性テスト、SUMI、理解性、使用性テスト、WAMMI

「ソフトウェア品質特性のテスト」の学習の目的

4.2 ビジネスドメインテストの品質特性

TA-4.2.1 (K2)正確性、合目的性、相互運用性、および標準適合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、例を挙げて説明する。

TA-4.2.2 (K2)正確性、合目的性、および相互運用性の特性に関して、対象とする典型的な欠陥を定義する。

TA-4.2.3 (K2)正確性、合目的性、および相互運用性の特性に関して、これらの特性を、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 (K4)特定のプロジェクトの内容に関して、使用性の要件の実装と、ユーザ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方式を概説する。

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主な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 シラバスで説明する。本章では、ISO 9126 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 25000 [ISO25000]シリーズ(ISO 9126 の改訂版)などの他の標準も使用することがある。ISO の品質特性は、プロダクトの品質特性(属性)に分類され、各品質特性は副特性(副属性)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
機能性	正確性、合目的性、相互運用性、標準適合性	○	
	セキュリティ		○
信頼性	成熟性(頑健性)、障害許容性、回復性、標準適合性		○
使用性	理解性、習得性、運用性、魅力性、標準適合性	○	
効率性	性能(時間効率性)、資源効率性、標準適合性		○
保守性	解析性、変更性、安定性、試験性、標準適合性		○
移植性	環境適応性、設置性、共存性、置換性、標準適合性		○

テストアナリストは、機能性と使用性のソフトウェア品質特性に専念する必要がある。また、アクセシビリティテストも実行する必要がある。副特性として示していないが、アクセシビリティは使用性テストの一部とみなすことが多い。他の品質特性のテストは、通常、テクニカルテストアナリストの担当とみなす。作業のこの割り当ては組織によって異なる可能性があるが、ISTQB シラバスではこの割り当てに従っている。

標準適合性の副特性を、品質特性ごとに示している。特定のセーフティクリティカルな環境または規制を受ける環境では、場合によっては、各品質特性は特定の標準または規制に適合する必要がある(たとえば、チップとのデータの送受信を可能にするために特定の通信プロトコルを使用するなど、機能性が特定の標準に適合する必要があることを機能性の標準適合性が示す場合がある)。それらの標準は業界により大きく異なるので、ここでは詳細に説明しない。標準適合性要件の影響を受ける環境でテストアナリストが作業する場合、それらの要件を理解し、テストとテストドキュメントの両方が標準適合性要件を確実に満たすようにする必要がある。

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、典型的なリストを認識して、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、ライフサイクルでのタイミング、必要なツール、ソフトウェアとドキュメントの可用性、および技術的な専門知識に、特に注意を払う必要がある。各特性とそれぞれの固有のテストニーズに対処する戦略を計画することにより、テスト担当者は、適切な計画、準備、およびテスト実行期間のスケジュールへの組み込みを行うことができる。使用性テストなど、このテストの一部では、特別な人的リソース、広範な計画、専用のラボ(調査施設)、特定のツール、特定のテストスキル、および、ほとんどの場には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。

品質特性および副特性のテストを全体的なテストスケジュールに統合し、適切なリソースをテスト作業に割り当てる必要がある。次の節で説明するように、品質特性のテストはそれぞれに固有のニーズを持ち、特定の問題を対象にし、ソフトウェア開発ライフサイクルの異なるタイミングで行う可能性がある。

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必要がある。たとえば、性能テストに合格しないプロダクトは、ユーザが効率的に使用するには時間がかかりすぎる場合、使用性テストにも合格しない可能性が高い。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、環境が変化した場合に、より基本的な問題が検出されなく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。

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、主に機能テストに重点を置く。機能テストは、プロダクトが「何をするのか」に重点を置く。機能テストのテストベースは、一般的に、要件または仕様のドキュメント、固有のドメイン知識または暗黙のニーズである。機能テストは、実施するテストレベルにより異なり、ソフトウェア開発ライフサイクルの影響を受けることもある。たとえば、統合テスト時に実施する機能テストでは、単一の定義済み機能を実装するインターフェースモジュールの機能性をテストする。システムテストレベルでは、機能テストは全体としてのアプリケーションの機能性をテストする。システムオブシステムズの場合、機能テストは主に、統合したシステム全体でのエンドツーエンドのテストに重点を置く。アジャイル環境では、通常、特定のイテレーションまたはスプリントで利用可能になる機能に限定して、機能テストを実施する。ただし、イテレーションに対する回帰テストは、リリースしたすべての機能性をカバーすることがある。

機能テスト時には、非常に広範なテスト技法を使用する（第 3 章を参照）。機能テストは、専任のテスト担当者、ドメインエキスパート、または開発者（通常、コンポーネントレベル）が実施することがある。

この節で説明する機能テストに加えて、テストアナリストの担当範囲の一部であり、（プロダクトが「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みなすことができる 2 つの品質特性もある。これらの 2 つの非機能特性は、使用性とアクセシビリティである。

この節では、次の品質特性について説明する。

- 機能的な品質副特性
  - 正確性
  - 合目的性
  - 相互運用性
- 非機能的な品質特性
  - 使用性
  - アクセシビリティ

4.2.1 正確性テスト

機能の正確性に関しては、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テストする。また、計算の正確性もテストすることがある。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使用する。また、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。正確性テストは、ライフサイクルのすべての段階で実施でき、データや状態の誤った処理を対象にする。

4.2.2 合目的性テスト

合目的性テストでは、機能セットがその意図および指定されたタスクに対して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、ユースケースに基づくことができる。合目的性テストは、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、システムがユーザの要求を受け入れられないことを示す。

4.2.3 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、さらには、以降でその情報を使用できる程度をテストする。このテストでは、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して、データ交換が適切に機能することを確認する必要がある。現実的には、比較的少ない数の環境に対してのみ実現可能である。このため、相互運用性テストは、一部の代表的な環境に限定することがある。相互運用性向けにテストを特定するには、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、機能テストから、環境内に存在するさまざま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。

相互運用性は、異なるソフトウェアシステムが相互に作用する方法に関連がある。優れた相互運用性の特性を備えたソフトウェアは、大きな変更を必要とせずに、多くの他のシステムに統合できる。これらの変更を行うために必要な変更箇所の数と工数は、相互運用性の測定値として使用できる。

ソフトウェアの相互運用性をテストするには、たとえば、次の設計上の特徴に重点を置くことが必要な場合がある。

- XML など業界のコミュニケーション標準の使用
- システムが相互作用するためのコミュニケーションニーズを自動的に検出し、その内容に従って調整する能力

相互運用性テストは、市販ソフトウェア（COTS）や市販ツールを開発する組織、およびシステムオブシステムズを開発する組織にとって特に重要なことが多い。

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テスト時に、システムと環境の相互作用に重点を置いて実行する。システム統合レベルでは、この種類のテストは、開発の完了したシステムが他のシステムと適切に相互作用することを判定するために実施する。システムでは、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さまざまな相互作用を遂行する条件を作成できる必要がある。たとえば、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必要なデータとデータ交換を実行するために必要なトランザクションを作成できる必要がある。すべての相互作用を、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に注意する必要がある。これらの相互作用の多くは、システムアーキテクチャ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検証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箇所を特定し、すべてをテストできるようにその準備をする必要がある。デシジョンテーブル、状態遷移図、ユースケース、および組み合わせ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誤ったデータ交換がある。

4.2.4 使用性テスト

ユーザがシステムを使いにくいと感じる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」という用語が、IT エキスパートから子ども、障害を持つ人に至るまで、広い範囲のさまざまな人を表すことを認識する必要がある。

一部の国家機関（たとえば、英王立盲人擁護協会：British Royal National Institute for the Blind）は、障害、視覚障害、弱視、運動障害、聴覚障害、および認知障害を持つユーザが Web ページにアクセスできるようにすることを推奨している。アプリケーションおよび Web サイトをこれらのユーザが使用できることをチェックすることにより、他の人すべてにとっても使用性が向上する可能性がある。アクセシビリティについては、以降でさらに詳細に説明する。

使用性テストは、ユーザ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、指定された目標を達成できるソフトウェア製品の能力

- 効率性- 明示的な条件の下で、使用する資源の量に対比して適切な性能を提供するソフトウェア製品の能力
- 満足性- 指定された利用の状況で、利用者を満足させるソフトウェア製品の能力

次に示す属性を測定することがある。
- 理解性- ソフトウェアが特定の作業に特定の利用条件で適用できるかどうか、およびどのように利用できるかを利用者が理解できるソフトウェア製品の能力
- 習得性- ソフトウェアの適用を利用者が習得できるソフトウェア製品の能力
- 運用性-利用者がソフトウェアの運用および運用管理を行うことができるソフトウェア製品の能力
- 魅力性-利用者にとって魅力的であるためのソフトウェア製品の能力

使用性テストは、通常、次の 2 つの段階で実施する。
- 発達的使用性テスト - 使用性設計の欠陥を識別して、設計をガイドする(または「形成」していく)ために、設計およびプロトタイプの段階で繰り返し実施するテスト
- 総括的使用性テスト - 開発が完了したコンポーネントまたはシステムの使用性を測定し、問題を識別するために、実装後に実施するテスト

使用性テストの担当者は、次の領域の専門性または知識を含むスキルを備えている必要がある。
- 社会学
- 心理学
- 国家標準(アクセシビリティ標準を含む)への適合
- 人間工学

4.2.4.1 使用性テストの実施

実際の実装の妥当性確認は、使用予定のシステムに可能な限り近い状況の下で行う必要がある。これには、実際のシステムが実際の人に与える影響を開発スタッフが観察できるように、ビデオカメラを備えたユーザビリティラボ、擬似オフィス、レビューパネル、ユーザなどを用意する必要がある。形式的な使用性テストでは、多くの場合、ユーザが従う手続きのセットまたは指示のセットのいずれかを提供して、大量の「ユーザ」を準備する必要がある。ユーザは、実際のユーザまたはユーザ代表を必要とする場合もある。他の自由形式のテストでは、ユーザはソフトウェアを自由に試すことができ、観察者は、ユーザが自分のタスクを達成する方法を見つけるのが困難かまたは容易かを判断できる。

多くの使用性テストは、テストアナリストが他のテスト(たとえば、機能性のシステムテスト)の一部として実行することがある。ライフサイクルのすべての段階で、使用性の欠陥の検出およびレポートに関して一貫した方式を達成するには、使用性ガイドラインが役立つことがある。使用性ガイドラインを使用しないと、何が「受け入れられない」使用性なのかを判断するのが困難になる可能性がある。たとえば、アプリケーションにログインするのにマウスを 10 回クリックする必要があるのは、ユーザにとって理にかなっているか？特定のガイドラインを使用しないと、テストアナリストは、開発者がソフトウェアは「設計どおり」に機能しているとして欠陥レポートを「済み」にしようとした場合に反論できなくなる可能性がある。検証可能な使用性の仕様を要件に定義し、すべての類似プロジェクトに適用する使用性ガイドラインのセットを用意することは非常に重要である。これらのガイドラインには、ガイダンスのアクセシビリティ、プロンプトの明確性、活動を完了するのに必要なクリック数、エラーメッセージ、処理中表示(システムは処理を行っており、さらなる入力を受け付けられないことを示す何らかの種類の表示)、画面レイアウトのガイドライン、ユーザエクスペリエンスに影響を与える色、音、および他の要因の使用、などのアイテムを含む必要がある。

4.2.4.2 使用性テストの詳細

使用性テストで使用する主な技法を次に示す。
- インスペクション、評価、またはレビュー
- プロトタイプを使用した動的な操作
- 実際の実装の検証と妥当性確認
- 調査およびアンケートの実施

インスペクション、評価、またはレビュー
要求仕様および設計を使用性の観点からインスペクションまたはレビューすることは、ユーザの関与を高めて問題を早期に発見できるので、費用対効果が高い方法である。ヒューリスティック評価（使用性に関するユーザ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のに使用できるので、イテレーティブな設計プロセスの一部として使用できる。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。たとえば、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文書による記述よりも、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必要である。

プロトタイプを使用した動的な操作
プロトタイプを開発したら、テストアナリストはプロトタイプで作業し、開発者がユーザのフィードバックを設計に反映してプロトタイプを進化させるのを支援する必要がある。この方法を使用すると、プロトタイプを洗練することができ、ユーザは完成品のルックアンドフィールが期待どおりかどうかを把握できる。

実際の実装の検証と妥当性確認
要件でソフトウェアの使用性を指定している場合（たとえば、特定の目標を達成するためにマウスをクリックする回数）、テストケースを作成して、ソフトウェアがこれらの特性を実装していることを検証する必要がある。

実際の実装の妥当性確認をする場合、機能性のシステムテスト向けに指定するテストを、使用性テストシナリオとして開発することがある。これらのテストシナリオは、機能的な側面ではなく、習得性または操作性など、特定の使用性を測定する。

使用性のテストシナリオは、特にシンタックスおよびセマンティクスをテストするために開発することがある。シンタックスはインターフェースの構造または文法（たとえば、入力欄に入力できるものなど）で、セマンティクスはインターフェースの意味および目的（たとえば、理にかなった分かりやすいユーザ向けのシステムメッセージおよび出力など）を記述する。

ブラックボックス技法（たとえば、3.2 節で記述されている技法）、特に平文または UML（Unified Modeling Language）を使用して定義できるユースケースを、使用性テストで使用することがある。

使用性テストのテストシナリオも、ユーザガイダンス、ガイダンスを提供しフィードバックを受け取るためのテスト前およびテスト後のインタビューを実施する時間配分、セッションを実施するための合意した手続きを含む必要がある。この手続きは、テストの実行方法の説明、タイミング、メモの記録やセッションの結果記録、インタビューやアンケートで使用する方法を含む。

調査およびアンケートの実施
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。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、以前の使用性測定のデータベースに対するベンチマークを実施できる。また、SUMI は使用性に関する明確な測定値を提供し、これらの測定値は、完了基準／受け入れ基準のセットとなる。

4.2.5 アクセシビリティテスト

ソフトウェアの使用に際して特定のニーズまたは制限を持つユーザ向けに、ソフトウェアへのアクセシビリティを考慮する必要がある。これらのユーザには、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・ガイドラインなどの関連する標準、および障害者差別禁止法（英国およびオーストラリア）や第 508 条（米国）などの法律を考慮する必要がある。使用性と同様に、アクセシビリティは設計段階で考慮する必要がある。テストは、多くの場合、統合テストレベルで発生し、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。

5. レビュー - 165 分

用語

なし

「レビュー」の学習の目的

5.1 イントロダクション

TA-5.1.1 (K2)テストアナリストにとって、レビューの準備が重要である理由を説明する。

5.2 レビューでのチェックリストの使用

TA-5.2.1 (K4)シラバスが提供するチェックリストの情報に従って、ユースケースまたはユーザインターフェースを分析し、問題を識別する。

TA-5.2.2 (K4)シラバスが提供するチェックリストの情報に従って、要求仕様またはユーザストーリーを分析し、問題を識別する。

5.1 イントロダクション

レビュープロセスを成功させるには、計画、参画、およびフォローアップが必要である。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、独自の見解を提供する必要がある。テストアナリストは公式なレビュートレーニングを受け、レビュープロセスにおける自分の役割についてよく理解しなければならない。すべてのレビュー参加者は、レビューが適切に実施されるように寄与する必要がある。レビューが正しく実施された場合、全体の品質に最大限寄与する最も費用対効果の高い手段になる可能性がある。

実行するレビューの種類に関係なく、テストアナリストは適切な準備時間を確保できる必要がある。この準備時間は、成果物をレビューする時間、一貫性を検証するために相互参照しているドキュメントをチェックする時間、および成果物に何が欠けているかを判定する時間を含んでいる。適切に準備を行わないと、テストアナリストは、レビューチームの時間を最大限に活用して可能な限り最善のフィードバックを提供する効率的なレビューに参加できず、ドキュメントにすでに記述されている内容の編集のみに活動が制限される可能性がある。優れたレビューは、記述された内容の理解、欠けている内容の判定、および記述されたプロダクトが、すでに開発されたまたは開発中の他のプロダクトと整合性があることの検証を含んでいる。たとえば、テストアナリストは統合レベルテスト計画のレビューで、統合するアイテムについて統合テストレベルに進むにはアイテムはどのような条件を満たす必要があるか、文書化する必要がある依存関係が存在するか、および統合箇所のテストに使用できるデータが存在するかを検討する必要がある。レビューは、レビューする成果物のみを対象とするのではなく、そのアイテムと、システム内の他のアイテムとの相互作用も考慮する必要がある。

レビュー対象のプロダクトの作成者は、非難されていると感じやすい。テストアナリストは、できるだけ最高のプロダクトを作成するために作成者と協力しているという観点で、レビューコメントを提示する必要がある。このアプローチを使用することで、コメントの表現が建設的になり、コメントの対象を作成者ではなく、成果物にすることができる。たとえば、記述が曖昧な場合は、「この要件は曖昧なので、誰も理解できない」と述べるよりも、「この要件が正しく実装されていることを検証するために、何をテストするべきか私には理解できません。私が理解できるように教えてくれませんか？」と述べる方がよい。テストアナリストはレビューにおいて、成果物で提供された情報が、テスト作業をサポートするのに十分であることを確認する。その情報が存在しない、明確でない、または必要な詳細度合いを提供しない場合、それは欠陥である可能性があり、作成者が修正する必要がある。批判的なアプローチではなく肯定的なアプローチを続けることで、コメントが受け入れられやすくなり、会議がより生産的になる。

5.2 レビューでのチェックリストの使用

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。たとえば、「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。たとえば、汎用チェックリストは、一意の ID を持つ、TBD 参照がない、適切な書式である、規格に適合している、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」と「推奨」の適切な使用の検証、記述されている要件の試験性の検証などを含んでいることがある。要件の書式は、使用するチェックリストの種類を示すこともある。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、プログラマ／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を対象にすることもある。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。これらのチェックリストは、以下に示すようなアイテムを含むことがある。

要件、ユースケース、およびユーザストーリーに対して使用するチェックリストは、一般的にコードまたはアーキテクチャに対して使用するチェックリストとは焦点が異なる。これらの要件指向のチェックリストは、次の項目を含むことがある。

- 各要件の試験性
- 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／ユーザストーリーの一意な識別子
- 各要件／ユースケース／ユーザストーリーのバージョン
- ビジネス／マーケティング要件からの各要件のトレーサビリティ
- 要件とユースケースとの間のトレーサビリティ

上記の項目は例として挙げているだけである。要件がテストできない場合、つまりテストアナリストがテスト方法を決定できるように要件が定義されていない場合、その要件には欠陥が存在すると認識することが重要である。たとえば、「ソフトウェアは非常にユーザフレンドリでなければならない」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フレンドリであるかどうか、または非常にユーザフレンドリであるかどう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で規定されている使用性の標準に適合する必要がある」と記述されており、使用性の標準ドキュメントが実際に存在する場合、これは、テスト容易化要件である。この要件はインターフェースの各アイテムに適用されるので、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、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、必要に応じて更新しなければならないので、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。

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。たとえば、「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)の可用性を提供する必要がある」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を構成する設問の例を次に示す。

- メインパス(シナリオ)は明確に定義されているか？
- すべての代替パス(シナリオ)は識別されており、エラー処理は完全に備わっているか？
- ユーザインターフェースメッセージは定義されているか？
- メインパスは 1 つだけ存在するか？複数存在する場合、複数のユースケースが存在するか？
- 各パスはテスト可能か？

アプリケーションのユーザインターフェースについて、使用性をテストする場合の単純なチェックリストを構成する設問の例を次に示す。

- 各フィールドとその機能は定義されているか？
- すべてのエラーメッセージは定義されているか？
- すべてのユーザプロンプトは定義されており、一貫しているか？
- フィールドのタブ順序は定義されているか？
- マウス操作を代替するキーボード操作は存在するか？
- ユーザ向けに定義された「ショートカット」キーの組み合わせは存在するか？(たとえば、カットアンドペースト)
- フィールド間に依存関係は存在するか？(たとえば、特定の日付は別の日付よりも未来である必要がある)
- 画面レイアウトは存在するか？
- 画面レイアウトは指定した要件に一致しているか？
- システムで処理中であることを示す、ユーザ向けの表示はあるか？
- 画面は最小マウスクリック要件を満たしているか？(定義されている場合)
- ユースケース情報に基づいて、ユーザを論理的にナビゲーションできるか？
- 画面は習得性に関するすべての要件を満たしているか？
- ユーザはヘルプを利用できるか？
- ユーザはホバーテキストを利用できるか？
- ユーザはこれを魅力的と思うか？(主観的なアセスメント)
- 色の使用は他のアプリケーションおよび組織の標準と一貫しているか？
- 効果音は適切に使用されており、設定可能か？
- 画面はローカライズ要件を満たしているか？

- ユーザは何をすべきかを判断できるか？(理解性)(主観的なアセスメント)
- ユーザは何をすべきかを覚えられるか？(習得性)(主観的なアセスメント)

アジャイルプロジェクトでは、一般的に、要件をユーザ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トランザクションであるが、ユーザストーリーはより独立しており、一般的に開発にかかる時間によりそのスコープが決まる。ユーザストーリー向けのチェックリストを構成する設問の例を次に示す。

- 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？
- 受け入れ基準は定義されており、テスト可能か？
- 機能は明確に定義されているか？
- このストーリーと他のストーリーとの間に依存関係は存在するか？
- ストーリーに優先度が割り当てられているか？
- ストーリーは機能の 1 つのアイテムを含んでいるか？

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、言うまでもなく、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。

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、および慣習の考慮)
- 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)
- レビュー対象物(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)

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。

6. 欠陥マネジメント – 120 分

用語
欠陥分類法、フェーズ内阻止、根本原因分析

「欠陥マネジメント」の学習の目的

6.2 欠陥を検出するタイミング
TA-6.2.1　(K2)フェーズ内阻止がどのようにコストを削減するかを説明する。

6.3 欠陥レポートフィールド
TA-6.3.1　(K2)非機能に関する欠陥を記述する場合に、必要となる情報を説明する。

6.4 欠陥の分類
TA-6.4.1　(K4)特定の欠陥の分類情報を識別、収集、および記録する。

6.5 根本原因分析
TA-6.5.1　(K2)根本原因分析の目的を説明する。

6.1 イントロダクション

テストアナリストは、システムの振る舞いをビジネスニーズおよびユーザニーズの観点から評価する。たとえば、特定のメッセージまたは振る舞いに直面した場合に、ユーザが何をすべきか分かるかどうかを評価する。テストアナリストは、期待結果と実際の結果を比較することにより、システムが正しく振る舞っているかどうかを判断する。不正(インシデントとも呼ばれる)は、さらなる調査を必要とする想定外の事象である。不正は、欠陥が引き起こした故障の可能性がある。不正の内容に応じて、欠陥レポートを作成する場合もあれば、作成しない場合もある。欠陥は、解決すべき実際の問題である。

6.2 欠陥を検出するタイミング

欠陥は静的テストを通して検出でき、欠陥の兆候である故障は動的テストを通して検出できる。ソフトウェア開発ライフサイクルの各フェーズは、潜在的な欠陥を検出および除去するための方法を提供する必要がある。たとえば開発フェーズでは、コードレビューおよび設計レビューを行って欠陥を検出する必要がある。動的テストの実行時には、テストケースを使用して故障を検出する。

欠陥を検出し、修正するタイミングが早期であるほど、システムの品質コストは全体として低くなる。たとえば静的テストは、動的テストが可能になる前に欠陥を見つけることができる。これが、静的テストが高品質のソフトウェアを生成するための費用対効果の高いアプローチである理由の 1 つである。

欠陥追跡システムを使用して、欠陥が混入したライフサイクル内のフェーズと、欠陥が見つかったフェーズとをテストアナリストが記録できるようにする必要がある。この 2 つのフェーズが同じである場合、完全なフェーズ内阻止が達成できている。これは、欠陥が混入したが、同じフェーズ内で見つかり、以降のフェーズに「流出」することはなかったことを意味する。たとえば、誤った要件を要件レビューで識別して、このフェーズで修正することがこの例として挙げられる。これは要件レビューを効率的に行えるばかりでなく、その欠陥により組織にとってさらにコストのかかる追加作業が発生するのを防止する。誤った要件が要件レビューから「流出」して、その後、開発者が実装し、テストアナリストがテストし、ユーザがユーザ受け入れテストで検出すると、その要件に費やしたすべての作業時間が無駄になる(ユーザがシステムに対する信頼を無くしたことは言うまでもない)。

フェーズ内阻止は、欠陥のコストを削減する有効な方法である。

6.3 欠陥レポートフィールド

欠陥レポート用に提供するフィールド(パラメータ)は、欠陥レポートが対応可能なレポートとなるように十分な情報を提供することを意図する。対応可能な欠陥レポートとは、次の特性を備えるレポートである。

- 完全 - 必要な情報のすべてがレポートに存在する。
- 簡潔 - 関連性のない情報はレポートに存在しない。
- 正確 - レポートの情報は正しくて、期待結果と実行結果、および再現するための適切な手順を明確に表している。
- 客観的 - レポートは、事実を専門的に記述したものである。

欠陥レポートでは、情報を個々のデータのフィールドに分割して記録する必要がある。フィールドを適切に定義するほど、個々の欠陥をレポートしやすくなり、傾向のレポートや他のサマリレポートを作成しやすくなる。あるフィールドで定義済みのオプションだけを利用できる場合、利用可能な値をドロップダウンリストで提供することにより、欠陥を記録するのにかかる時間を削減できる。オプションの数が決まっており、正しいオプションを選択するのにユーザが長い一覧をスクロールする必要がない場合に、ドロップダウンリストが効果を発揮する。欠陥レポートの種類に応じて必要な情報が異なるので、欠陥マネジメントツールは、欠陥のタイプに応じて正しいフィールドへの入力を促す十分な柔軟性を備えている必要がある。データは個別のフィールドに記録する。データ入力エラーを避け有効なレポートを作成するために、データの妥当性確認をサポートする機能を活用するのが理想的である。

欠陥レポートには、機能テストおよび非機能テストの実行時に検出した故障を記述する。欠陥レポートでは、常に問題を検出した状況を明確に識別するように情報を記録する。記録する情報には、状況を再現するために必要な手順とデータ、および期待結果と実行結果を含める。非機能に関する欠陥のレポートは、環境、他の性能パラメータ(負荷の大きさなど)、手順の順序、期待結果に関するさらに詳細な情報を必要とすることがある。使用性の故障を記述する場合、ソフトウェアの振る舞いに関するユーザの期待を記述する必要がある。たとえば、使用性における標準で、操作を完了するのに必要なマウスクリックは 3 回以下と規定している場合、欠陥レポートには、規定された標準と、実際に必要であったクリック回数を併記する必要がある。標準を利用できず、要件がソフトウェアの非機能の品質側面をカバーしていない場合、テスト担当者は、「一般的な人」を想定したテストを使用して、使用性が受け入れられるかどうかを判断する。受け入れない場合、「一般的な人」の期待値を欠陥レポートに明確に記述する必要がある。非機能要件は要件ドキュメントに記述されていないことがあるので、非機能に関する故障を記述する場合、「期待」と「実際」の振る舞いの違いを文書化することは、テスト担当者にとって、困難になりがちである。

欠陥レポートを記述する一般的なゴールは、問題の解決策を得ることであるが、正確な分類、リスク分析、およびプロセス改善をサポートする欠陥情報も提供する必要がある。

6.4 欠陥の分類

欠陥レポートの分類の度合いは、そのライフサイクルを通してさまざまに変化する。欠陥を適切にレポートするには、欠陥を適切に分類する必要がある。分類は、欠陥のグループ化、テストの有効性の評価、開発ライフサイクルの有効性の評価、および注目すべき傾向を見つけ出すために使用する。

欠陥が識別された場合の一般的な分類情報を次に示す。

- 欠陥を検出した活動
  - 例: レビュー、監査、インスペクション、コーディング、テスト
- 欠陥が混入したプロジェクトフェーズ(判明している場合)
  - 例: 要件、設計、詳細設計、コーディング
- 欠陥を検出したプロジェクトフェーズ
  - 例: 要件、設計、詳細設計、コーディング、コードレビュー、ユニットテスト、統合テスト、システムテスト、受け入れテスト
- 可能性のある原因
  - 例: 要件、設計、インターフェース、コード、データ
- 再現性
  - 例: 1 回のみ、不定、必ず再現
- 現象
  - 例: クラッシュ、ハングアップ、ユーザインターフェースエラー、システムエラー、性能

欠陥を調査した後、さらに細かく分類できることがある。

- 根本原因 - 欠陥をもたらした誤り。
  例: プロセス、コーディングエラー、ユーザエラー、テストエラー、構成の問題、データの問題、サードパーティ製ソフトウェア、外部ソフトウェアの問題、ドキュメントの問題
- 発生元 – 誤りが発生した成果物。
  例: 要件、設計、詳細設計、アーキテクチャ、データベース設計、ユーザドキュメント、テストドキュメント
- 欠陥のタイプ
  例: ロジックの問題、演算の問題、タイミングの問題、データ処理の問題、拡張による問題

欠陥の修正が完了または延期された場合、あるいは欠陥を確認できない場合、次に示すような、さらに多くの分類情報を利用できることがある。

- 解決
  - 例: コード変更、ドキュメント変更、延期、問題なし、重複
- 修正アクション

– 例:要件レビュー、コードレビュー、ユニットテスト、構成の文書化、データの準備、変更なし

これらの分類カテゴリに加えて、多くの場合、重要度と優先度に基づいて欠陥を分類する。また、プロジェクトによっては、使命達成への影響、プロジェクトスケジュールへの影響、プロジェクトコスト、プロジェクトリスク、およびプロジェクト品質への影響に基づいて分類することもある。これらの分類は、いつまでに修正したものを提供するかを合意する際に考慮することがある。

最後の分類は最終的な解決策である。多くの場合、解決策に基づいて欠陥をグループ化する。例としては、解決済み／検証済み、終了／問題なし、延期、対応中／未解決が挙げられる。欠陥の追跡は、欠陥のライフサイクルを通じて行うので、一般的に、プロジェクトを通じてこの分類を使用する。

組織は、多くの場合、分類のオプションをカスタマイズして使用する。前述の分類のオプションは、業界で一般的に使用するオプションのいくつかの例を示している。分類のオプションを有用にするために、一貫して使用することが重要である。分類フィールドが多すぎると、欠陥をオープンしたり対処したりするのに時間が余分にかかるので、欠陥を対処するごとに増加するコストと比較して収集するデータの価値を慎重に考慮する必要がある。ツールを選択する場合、ツールが収集した分類のオプションをカスタマイズできることを重要な要因として挙げることが多い。

6.5 根本原因分析

根本原因分析の目的は、欠陥を引き起こしたものを判定し、欠陥の重要な部分を占める根本原因を除去するためのデータを提供することで、プロセスの変更をサポートするということである。根本原因分析は、一般的に、問題を調査して、その問題を解決する人、またはその問題を解決する必要はないと判断したり修正できないと判断したりする人が実施する。これは通常開発者が担当する。根本原因に仮のオプションを設定するのは、通常、問題を発生させたものを推測するトレーニングを受けたテストアナリストが実施する。修正を確認する場合、テストアナリストは、開発者が入力した根本原因のオプションを検証する。一般的に、根本原因を判定した時点で、欠陥が混入したフェーズも判定または確認する。

典型的な根本原因を次に示す。

- 不明確な要件
- 要件の欠落
- 要件の誤り
- 不正な設計実装
- 不正なインターフェース実装
- コードロジックエラー
- 演算エラー
- ハードウェアエラー
- インターフェースエラー
- 無効なデータ

この根本原因情報をまとめて活用することにより、欠陥を発生させる共通問題を判定できる。たとえば、不明確な要件により大量の欠陥が発生している場合、より多くの工数をかけて効果的な要件レビューを実施することが理にかなっている。同様に、開発グループ間でインターフェース実装が問題になっている場合、共同の設計会議を開催することが必要な場合がある。

プロセス改善に根本原因情報を使用することにより、組織は効果的なプロセスの変更の利点を見ることができ、特定の根本原因に帰属する欠陥のコストを算出できる。これにより、追加のツールおよび機器の購入、およびスケジュールタイミングの変更を必要とするプロセス変更向けの予算を確保できる。根本原因分析の詳細については、ISTQB Expert Level シラバスの「Improving the Test Process」[ISTQB_EL_ITP]を参照されたい。

7. テストツール – 45 分

用語

キーワード駆動テスト、テストデータ準備ツール、テスト設計ツール、テスト実行ツール

「テストツール」の学習の目的

7.2 テストツールおよび自動化

TA-7.2.1 (K2)テストデータ準備ツール、テスト設計ツール、およびテスト実行ツールを使用する利点を説明する。

TA-7.2.2 (K2)キーワード駆動による自動化でのテストアナリストの役割を説明する。

TA-7.2.3 (K2)自動化したテスト実行が失敗した場合のトラブルシューティングの手順を説明する。

7.1 イントロダクション

テストツールは、テスト作業の効率性と正確性を大きく改善するが、適切なツールを適切な方法で実装した場合のみ実現できる。テストツールは、テスト組織を適切に運営することとは別の側面として管理する必要がある。テストツールごとに洗練度および適用性は大きく異なり、ツール市場は常に変化している。ツールは一般的に市販ツールベンダ、またはさまざまなフリーウェアやシェアウェアのツールサイトから入手できる。

7.2 テストツールおよび自動化

テストアナリストは担当する作業の多くで、ツールを有効に使用する必要がある。どのツールをどのタイミングで使用するかを把握することにより、テストアナリストの作業効率が向上し、許容される時間内でより広いテストカバレッジを達成できる。

7.2.1 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を作成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製品（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、テストカバレッジ、システムのコンフィデンス（確信度合い）、またはプロダクトリスク軽減アクションの目標とする度合いを達成するために必要なテストタイプを決定するための情報を、テストアナリストに提供する。たとえば、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する。

7.2.2 テストデータ準備ツール

テストデータ準備ツールには、いくつかの利点がある。一部のテストデータ準備ツールは、要件ドキュメントなどのドキュメントや、さらにはソースコードも分析して、カバレッジのレベルを達成するためにテストで必要なデータを決定できる。他のテストデータ準備ツールには、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」または「匿名化」を実行して個人情報を削除できるものがある。選別したデータは、個人情報のセキュリティリーク、または誤使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に、特に重要となる。さらに、特定の入力パラメータのセットから（たとえば、ランダムテスト用に）テストデータを生成するために使用できるデータ生成ツールもある。これらのツールの一部は、データベース構造を分析して、テストアナリストが必要とする入力を決定する。

7.2.3 テスト自動実行ツール

テスト実行ツールは、ほとんどの場合テストを実行し、テストの結果をチェック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、一般的に次の 1 つまたは複数の項目に該当する。

- コスト削減（工数、時間、またはその両方の観点）のため
- より多くのテストを実行するため
- 同じテスト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、コスト削減と同時にカバレッジを上げるという主要目的と一致する。

7.2.3.1 適用

テスト実行ツールの投資効果は、通常回帰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、自動化ツールを有効に使用できる。この理由として、頻繁なテストケースの利用、結果を迅速に取得することの必要性、およびメンテナンスコストは高くなるが、新しいビルドを継続的インテグレーション環境で評価するための自動化した方法を持つ能力などが挙げられる。

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のレベルで使用する。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レベル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。

7.2.3.2 テスト自動化ツールの基本

テスト実行ツールは、スクリプト言語と呼ばれることが多いプログラミング言語で記述された命令セットを実行することにより動作する。ツールに提供する命令は、入力、入力の順序、入力に対して使用する特別な値、および期待結果を非常に詳細に記述する。スクリプトを詳細に記述することで、特にツールがグラフィカルユーザインターフェース(GUI)と相互作用している場合に、テスト対象のソフトウェア(SUT)の変更に対して影響を受けやすくなる可能性がある。

ほとんどのテスト実行ツールは、実行結果と格納した期待結果を比較できる比較機能を備えている。

7.2.3.3 テスト自動化の実装

テスト実行自動化は、プログラミングと同様に低レベルの詳細な処理から、ライブラリ、マクロ、およびサブプログラムを使用するより高レベルの言語に移行する傾向にある。キーワード駆動やアクションワード駆動などの設計技法は、一連の処理を捕捉し、それらを特定の「キーワード」または「アクションワード」に結びつける。これにより、テストアナリストは、基になるプログラミング言語やより低レベルの機能などを気にせずに人間の言葉でテストケースを記述できる。このモジュール式の記述技法を使用することにより、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を変更する場合に、より容易にメンテナンスできるようになる[Bath08]。自動化スクリプトでのキーワードの使用については、以降でより詳細に説明する。

キーワードまたはアクションワードを容易に作成するために、モデルを使用できる。要件ドキュメントに含まれることの多いビジネスプロセスモデルを調査することにより、テストアナリストは、テストする必要のある主要なビジネスプロセスを決定できる。その後、各プロセスにおいて発生する可能性のある判定箇所を含む、これらのプロセスの手順を決定できる。判定箇所は、テスト自動化がキーワードまたはアクションワードのスプレッドシートから取得して使用できるアクションワードになることができる。ビジネスプロセスモデル化は、ビジネスプロセスを文書化するための方法で、これらの主要なプロセスと判定箇所を判定するために使用できる。このモデル化は、手動でもツールを使用しても実行できる。ツールは、ビジネスルールとプロセス記述に基づく入力に従って動作する。

7.2.3.4 自動化作業の成功度合いの改善

どのテストを自動化するかを決定する場合、候補となるテストケースまたはテストスイートのそれぞれに対して、自動化が利点をもたらすかどうかを評価する必要がある。自動化に失敗したプロジェクトの多くは、自動化により実際に利点を得られるかどうかを確認することなく、容易に実行できる手動によるテストケースを自動化している。特定のテストケース(またはテストスイート)のセットでは、手動、半自動化、および完全自動化のテストを組み合わせて含めることが最適な場合がある。

テスト実行の自動化を行う場合には、次の側面を考慮する必要がある。

達成可能な利点:

- 自動化テストでは、実行時間がより予測可能になる。

- 自動化テストを使用した回帰テストおよび欠陥の確認が、プロジェクトの後半でより短時間に終了し、信頼性も高くなる。
- 自動化ツールの使用により、テスト担当者またはテストチームの状態が改善し、技術的な成長が促進されることがある。
- 自動化は、特にイテレーティブおよびインクリメンタル開発ライフサイクルに役立ち、各ビルドまたはイテレーションに対してより優れた回帰テストの実行を可能にする。
- 特定のテストタイプのカバレッジは、自動化ツールを使用した場合のみ実行できる(例: 大規模なデータ妥当性確認作業)。
- テスト実行自動化は、高速で一貫性のある入力および検証を可能にするので、大量データの入力、変換、および比較のテストにおいて、手動テストよりも費用対効果が高くなる可能性がある。

考えられるリスク:

- 不完全、非効率、または誤りのある手動テストを、「そのまま」自動化してしまう可能性がある。
- テスト対象のソフトウェアの変更により、複数の変更が必要となるなど、テストウェアのメンテナンスが困難となる可能性がある。
- テスト担当者がテスト実行に直接関与する機会が少なくなり、欠陥の検出が少なくなる可能性がある。
- テストチームが、自動化ツールを有効に使用するための十分なスキルを備えていない可能性がある。
- 全体的なテストカバレッジに貢献しない無意味なテストを、存在かつ安定しているという理由で自動化に含めてしまう可能性がある。
- ソフトウェアが安定するにつれて、テストが非生産的になる可能性がある(殺虫剤のパラドックス)。

テスト実行自動化ツールを展開する場合、手動によるテストケースをそのまま自動化せず、自動化の有効性が発揮されるようにテストケースを再定義することが望ましい。再定義には、テストケースの書式化、再使用パターンの考慮、値をそのまま埋め込んで使用する代わりに変数を使用することによる入力の拡張、およびテストツールのすべての利点の活用が挙げられる。テスト実行ツールは、一般的に複数のテストの横断的な実行、テストのグループ化、テストの繰り返し、および実行順の変更を行う機能を備えており、同時に分析およびレポートの機能も併せ持つ。

多くのテスト実行自動化ツールで、効率性と有効性の高いテスト(スクリプト)とテストスイートを作成するために、プログラミングスキルが必要である。一般的に大規模な自動化のテストスイートは、入念に設計しないと更新とマネジメントが非常に困難である。テストツールのすべての利点を活用するにはツールの適切なトレーニング、プログラミング技術、および設計技術が必要である。

テスト計画作業では、テストが動作する仕組みに関する知識の維持、正しい操作の検証、および入力データの有効性とカバレッジのレビューを行うために、自動化のテストケースを定期的に手動で実行する時間を割り当てる必要がある。

7.2.3.5 キーワード駆動自動化

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高位レベルの相互作用(たとえば、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の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)の列は、テストケースを特定するために使用する[Buwalda01]。

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列として記述されている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、モジュールを高度な方式で実装する。これらのモジュール型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自動化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、ドメインエキスパートが定義できる。このため、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)。
- キーワードを使用して記述したテストケースは、テスト対象のソフトウェアの詳細に対する変更が発生した場合に変更する必要性が少ないので、メンテナンスが容易である。
- テストケース仕様は、その実装から独立しているため、キーワードはさまざまなスクリプト言語およびツールを使用して実装できる。

キーワード／アクションワード情報を使用する自動化スクリプト(実際の自動化スクリプトのコード)は、一般的に開発者またはテクニカルテストアナリストが記述し、テストアナリストは、一般的にキーワード／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。キーワード駆動の自動化は、一般的に、システムテストフェーズで実行するが、スクリプトのコード開発は統合フェーズ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、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、一般的に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必要がある。不正を検出したら、テストアナリストは故障の原因を調査して、問題がキーワード、入力データ、自動化スクリプト自体、またはテスト対象のアプリケーションのどれに存在するのかを判断する必要がある。一般的に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(おそらく不正なデータを生成している)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必要がある。

7.2.3.6 自動化が失敗する原因

テスト実行自動化プロジェクトは、目標を達成できないことが多々ある。これらの失敗の原因としては、テストツールを柔軟に使用できないこと、テストチームに十分なプログラミングスキルがないこと、またはテスト実行の自動化で解決できる問題に非現実的な期待を抱いたことが挙げられる。すべてのテスト実行自動化は、すべてのソフトウェア開発プロジェクトと同じく、マネジメント、工数、スキル、およびモニタリングを必要とすることに注意する必要がある。持続可能なアーキテクチャを作成するために時間を十分にかける必要があり、その後に適切な設計、構成管理、優れたコーディングを行う必要がある。自動化テストスクリプトは、欠陥を含んでいる可能性があるので、十分にテストする必要がある。また、性能を向上させるために調整が必要なこともある。ツールの使用性は、対象とする使用者として、開発者だけでなくツールを使用してスクリプトを実行する人も考慮する必要がある。ツールとユーザとの間のインターフェースを設計することが必要な場合がある。このインターフェースは、テスト担当者には論理的にまとめられており、ツールが必要とするアクセシビリティも提供する方法でテストケースにアクセスできることが必要である。

8. 参考文献

8.1 標準

- [ISO25000]: ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE)
  第 1 章と第 4 章
- [ISO9126]: ISO/IEC 9126-1:2001, Software Engineering - Software Product Quality
  第 1 章と第 4 章、JSTQB 訳注）日本では JIS X 0129-1 として発行されている。
- [RTCA DO-178B/ED-12B]: Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE ED12B.1992.
  第 1 章

8.2 ISTQB ドキュメント

- [ISTQB_AL_OVIEW] ISTQB Advanced Level Overview, Version 1.0
  JSTQB 訳注）日本では「テスト技術者資格制度 Advanced Level シラバス日本語版概要 Version 2012」として発行されている。
- [ISTQB_ALTM_SYL] ISTQB Advanced Level Test Manager Syllabus, Version 1.0
  JSTQB 訳注）日本では「テスト技術者資格制度 Advanced Level シラバス日本語版 テストマネージャ Version 2012」として発行されている。
- [ISTQB_ALTTA_SYL] ISTQB Advanced Level Technical Test Analyst Syllabus, Version 1.0
- [ISTQB_FL_SYL] ISTQB Foundation Level Syllabus, Version 2011
  JSTQB 訳注）日本では「テスト技術者資格制度 Foundation Level シラバス日本語版 Version 2011」として発行されている。
- [ISTQB_GLOSSARY] Standard glossary of terms used in Software Testing, Version 2.2, 2012
  JSTQB 訳注）日本では「ソフトウェアテスト標準用語集 Version 2.2」として発行されている。

8.3 書籍

[Bath08]: Graham Bath, Judy McKay, "The Software Test Engineer's Handbook", Rocky Nook, 2008, 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　ソフトウェアのブラックボックステスト」（日経 BP, 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", John Wiley and Sons, 2007, ISBN 978-0-470-12790-2

[Buwalda01]: Hans Buwalda, "Integrated Test Design and Automation", Addison-Wesley Longman, 2001, ISBN 0-201-73725-6

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

[Copeland03]: Lee Copeland, "A Practitioner's Guide to Software Test Design", Artech House, 2003, ISBN 1-58053-791-X

JSTQB 訳注）日本では「はじめて学ぶソフトウェアのテスト技法」(日経 BP, 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。

[Gerrard02]: Paul Gerrard, Neil Thompson, "Risk-based e-business Testing", Artech House, 2002, ISBN 1-580-53314-0

[Gilb93]: Tom Gilb, Graham Dorothy, "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」(共立出版,1999 年)として発行されている。

[Graham07]: Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black "Foundations of Software Testing", Thomson Learning, 2007, ISBN 978-1-84480-355-2

JSTQB 訳注）日本では「ソフトウェアテストの基礎:ISTQB シラバス準拠」(ビー・エヌ・エヌ新社, 2008 年)として発行されている。

[Grochmann94]: M. Grochmann (1994), Test case design using Classification Trees, in: conference proceedings STAR 1994

[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

[Myers79]: Glenford J. Myers, "The Art of Software Testing", John Wiley & Sons, 1979,

ISBN 0-471-46912-2

JSTQB 訳注）日本では「ソフトウェア・テストの技法」(近代科学社,1980 年)として発行されている。

[Splaine01]: Steven Splaine, Stefan P. Jaskiel, "The Web-Testing Handbook", STQE Publishing, 2001, ISBN 0-970-43630-0

[vanVeenendaal12]: Erik van Veenendaal, "Practical risk-based testing – The PRISMA approach", UTN Publishers, The Netherlands, 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", Addison-Wesley, 2009, ISBN 0-321-63641-4

8.4 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
  - バグ分類法: www.testingeducation.org/a/bsct2.pdf
  - Boris Beizer の著作に基づくサンプルバグ分類法: inet.uni2.dk/~vinter/bugtaxst.doc
  - さまざまな分類法の優れた概説: testingeducation.org/a/bugtax.pdf
  - James Bach, "Heuristic Risk-Based Testing"
  - "Exploratory & Risk-Based Testing (2004) www.testingeducation.org"
  - Exploring Exploratory Testing , Cem Kaner and Andy Tikam , 2003
  - Pettichord, Bret, "An Exploratory Testing Workshop Report", www.testingcraft.com/exploratorypettichord
- 第 4 章
  - www.testingstandards.co.uk

9. 索引