テスト技術者資格制度
Foundation Level シラバス　日本語版

Version 2011.J02

International Software Testing Qualifications Board

(翻訳　Japan Software Testing Qualifications Board)

改定履歴

ISTQB®

バージョン	日付	摘要
ISTQB2011	2011/3/30	テスト技術者資格認定制度 Foundation Level シラバス メンテナンスリリース –付録 E-リリースノート、シラバス 2011 参照
ISTQB 2010	2010/3/30	テスト技術者資格認定制度 Foundation Level シラバス メンテナンスリリース –付録 E-リリースノート、シラバス 2010 参照
ISTQB 2007	2007/5/1	テスト技術者資格認定制度 Foundation Level シラバス メンテナンスリリース –付録 E-リリースノート、シラバス 2007 参照
ISTQB2005	2005/7/1	テスト技術者資格認定制度 Foundation Level シラバス
ASQFV2.2	2003/7	ASQF シラバス Foundation Level 2.2 版
ISEBV2.0	1999/2/25	ISEB Foundation Level シラバス 2.0

JSTQB®

Version2011.J02	2012/06/01	用語集（日本語版） Version 2.1.J01 に対応 以下用語名の変更 ・インシデント管理ツール → インシデントマネジメントツール ・カバレッジ計測ツール → カバレッジ測定ツール ・実際の結果 → 実行結果 ・証明 → 認定 ・テスト駆動型開発 → テスト駆動開発 ・テストの目的 → テスト目的 ・テスト容易性 → 試験性 ・独立性 → テストの独立性 ・メトリクス → メトリック ・ユーザビリティ → 使用性 ・ユーザビリティテスト → 使用性テスト ・リスクベーステスト → リスクベースドテスト ※ただし、用語集で説明されている意味のものに限り変更
Version2011.J01	2011/03/30	ISTQB VER2011 の日本語翻訳版 和訳の表現を全体的に見直し
Version 2007.J01	2009/11/20	用語集に合わせて修正 和訳の表現を全体的に見直し 2.「LO-2.3.5」を追加 2.3.1.「相互運用性(接続性)テスト」の説明を追加 3.「LO-3.1.4」の説明として「レビュー」を追加

		4.1.背景の「テスト分析」の説明において、「テスト設計」と記載されていた箇所を修正 4.2. ブラックボックス技法に関する説明を変更 4.3. 学習時間を「120 分」→「150 分」に変更 5. LO-5.2.4 の説明を変更 5.1.1.テストの形態として「独立したテスト担当者がいない。開発者が自ら自分のコードをテストする。」を追加 6.3.パイロットプロジェクトの説明としての「コンセプトの証明(proof-of-concept)」の記載を削除 「終了基準の検証」を「終了基準の評価」に全体的に変更 バージョン表記方法の変更
Ver1.1.0	2008/07/01	ISTQB　VER2007 の日本語翻訳版 和訳の表現を全体的に見直し
Ver1.0.1	2006/12/29	用語集に合わせて修正 和訳の表現を全体的に見直し
Ver1.0.0	2006/7/21	用語集に合わせて修正
Ver0.6.3	2006/5/1	JTCB→JSTQB に変更 正式名称を Japan Software Testing Qualifications Board に変更
Ver0.6.2	2006/1/12	4　4.4 の K2 の位置を修正 4　4.4 の最後の行に K3 を追加 4.4 背景の説明の「ブランチ」を削除→デシジョンとブランチが違うように見えるため
Ver0.6.1	2005/12/17	2.1.1 テストレベルの意味について脚注を追加 1.5 レベル 2 以上の独立性があると→独立性が高いと 1.5 独立性のレベルを以下に示す→独立性の度合いを以下に示す
Ver0.6	2005/12/13	ISTQB VER2005 の日本語翻訳版

目次

謝辞

国際ソフトウェアテスト資格認定委員会(ISTQB®)の Working Group Foundation Level(Editon2011)
メンバー：Thomas Müller (chair), Debra Friedenberg.
コアチームは現在のシラバスへの提案に対してレビューチーム (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquier Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal)と各国委員会に感謝をしたい。

国際ソフトウェアテスト資格認定委員会(ISTQB®)の Working Group Foundation Level(Editon2010)
メンバー：Thomas Müller (chair), Rahul Verma, Martin Klonk and Armin Beer.
コアチームは現在のシラバスへの提案に対してレビューチーム (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veendendaal)と各国委員会に感謝をしたい。

国際ソフトウェアテスト資格認定委員会(ISTQB®)の Working Group Foundation Level(Editon2007)
メンバー：Thomas Müller (chair), Dorothy Graham, Debra Friedenberg, and Erik van Veendendaal
レビューチーム (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, and Wonil Kwon) と各国委員会がシラバスへの提案をした。

国際ソフトウェアテスト資格認定委員会(ISTQB®)の Working Group Foundation Level(Editon2005)
メンバー：Thomas Müller(Chair), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson and Erik van Veenendaal
レビューチームと各国委員会がシラバスへの提案をした。

イントロダクション

本シラバスの目的

本シラバスは国際ソフトウェアテスト資格認定 Foundation Level のベースとなる。国際ソフトウェアテスト資格認定委員会(以下に ISTQB ®と呼ぶ)は、各国の委員会(日本では JSTQB®が該当する)が、教育機関の認定を行い、各国の言語での試験問題を導入するために本シラバスを提供する。教育機関は、認定のためにコースウェアを製作して、適切な教育方法を確定するが、本シラバスは受験志願者の試験準備の役にも立つであろう。
シラバスに関する来歴と背景に関する情報は付録 A に添付してある。

ソフトウェアテストにおける Foundation Level 認定テスト担当者

Foundation Level の資格認定はソフトウェアのテスト作業に関与する全ての人を対象としている。これにはテスト担当者、テストアナリスト、テストエンジニア、テストコンサルタント、テストマネージャ、ユーザ受け入れテスト担当者、およびソフトウェア開発者の役割を持つ人々が含まれる。この Foundation Level の資格認定はソフトウェアテストについて基本的な理解を望むプロジェクトマネージャ、品質管理者、ソフトウェア開発マネージャ、ビジネスアナリスト、IT 部門長および経営コンサルタントのような人にも適切である。
Foundation Level 認定資格の保持者はソフトウェアテスト資格での上位レベルに進むことが可能である。

学習の目的／知識の認知レベル

本シラバスでは各セクションで以下の分類にて「学習の目的」を示している。

- K1: 記憶
- K2: 理解
- K3: 適用
- K4: 分析

学習の目的の詳細と例は付録 B に添付する。
各章の「用語」の下にリストアップされている全ての用語は、「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。

試験

Foundation Level の認定資格試験は本シラバスに基づく。試験問題に対する解答は本シラバスの 1 セクションよりも多いものを基にした材料を用いることがあるかもしれない。シラバスの全てのセクションは試験対象である。試験の形式は多肢選択式である。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)独立して実施してもよい。公認の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。

認証審査

ISTQB®で認定された各国の委員会にて、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。トレーニングプロバイダーは認定を行う実行委員会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。
教育機関のためのひととおりのガイダンスを付録 D に示す。

レベルの詳細

本シラバスの詳細レベルは国際的に一貫した教育と試験を可能にする。
このゴールを達成するために本シラバスは以下のようになっている。

- 全般的な教育内容の目的(Foundation Level の意図について説明)
- リファレンスのリスト(教えるべき情報、解説を含んだ情報、必要となるかもしれない追加情報源)
- 各知識領域の学習の目的(学習の成果と考え方について説明)
- 用語のリスト(理解でき、思い出すことができなければならない用語)
- 教えるべき主要なコンセプトの説明(文献や標準を情報源として含む)

本シラバスの内容はソフトウェアテストの全知識領域の説明ではない。詳細レベルは、Foundation Level のトレーニングコースでカバーされることを示している。

本シラバスの構成

本シラバスには 6 つの章がある。各章の一番上の見出しは、章の中でカバーされている学習目的の知識の最も高いレベルを示し、学習時間を指定している。例えば

2.　ソフトウェアライフサイクルを通じてのテスト(K2)　　学習時間:115 分

この見出しは、K1(より高いレベルが示されるとき)と K2(しかし K3 ではない)の学習目的があり、章の内容を教えるためには 115 分かかることを示している。

それぞれの章にはいくつかのセクションがあり、各セクションには学習の目的と学習に必要な時間が記してある。時間を提示していないサブセクションはセクションの時間に含まれている。

1. テストの基礎(K2)　学習時間:155 分

本章の学習の目的

本章で記述する学習内容は以下のとおり。

1.1 テストの必要性(K2)

LO-1.1.1 ソフトウェアの欠陥が人、環境、組織にどのような害を与えるかを、例を挙げて説明する。(K2)

LO-1.1.2 欠陥の根本原因と結果を区別する。(K2)

LO-1.1.3 例を挙げてテストの必要性を説明する。(K2)

LO-1.1.4 なぜテストが品質保証の一部分であるのか説明し、品質を確保する上でテストがどう貢献するかを、例を挙げて説明する。(K2)

LO-1.1.5 エラー、欠陥、フォールト、故障という用語、および、それに関連する用語である、誤りとバグとを例を挙げて説明し比較する。(K2)

1.2 テストとは何か？(K2)

LO-1.2.1 テストに共通する目的を再認識する。(K1)

LO-1.2.2 ソフトウェアライフサイクルの異なったフェーズにおけるテストの目的を説明するための例を提供する。(K2)

LO-1.2.3 デバッグとテストとを区別する。(K2)

1.3 テストの 7 原則(K2)

LO-1.3.1 テストにおける 7 原則を説明する。(K2)

1.4 基本的なテストプロセス(K1)

LO-1.4.1 テストの計画から終了作業に至るまでの５つの基本的なテストの活動や、各テスト作業での主な実施内容を再認識する。(K1)

1.5 テストの心理学(K2)

LO-1.5.1 テストの成否は心理的要素に影響されることを再認識する。(K1)

LO-1.5.2 テスト担当者とソフトウェア開発者の心理の違いを対比させる。(K2)

1.1.　テストの必要性(K2)　学習時間:20 分

用語

バグ、欠陥、エラー、故障、フォールト、誤り、品質、リスク

1.1.1.　ソフトウェアシステムの状況(K1)

ソフトウェアシステムは、ビジネス分野(銀行など)から、一般消費財(自動車など)に至るまで、社会を構成する要素として必須となっている。ソフトウェアが期待通りに動かなかった経験は誰もが持っている。ソフトウェアが正しく動作しないと、経済的な損失、時間の浪費、信用の失墜など、いろいろな問題が発生し、時には傷害や死亡事故になることもある。

1.1.2.　ソフトウェアの欠陥の原因(K2)

人間はエラー(誤り)を犯す。そのエラーがソースコード、ドキュメントの欠陥(フォールト、バグ)となる。ソースコードの欠陥が実行されると、システムは実行すべきことを正しく実行できず(あるいは実行してはならないことを実行してしまい)、故障が起きる。ソフトウェア、システム、ドキュメントの欠陥で故障が発生することもあるが、全ての欠陥が故障となるわけではない。

欠陥が混入するのは、人間が誤りを犯しやすいためである。その誘因として、納期のプレッシャー、コードの複雑さ、基盤構造の複雑さ、技術の変化、他システムとのインターフェースなどがある。

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

1.1.3.　ソフトウェアの開発、保守、運用におけるテストの役割(K2)

システムやドキュメントを厳しくテストし、システムが稼動する前に欠陥を摘出して修正するならば、実行環境で問題が発生するリスクを低減でき、ソフトウェアシステムの品質向上に効果がある。

また、契約や法律上の適格要件や各業界の標準に合致していることを証明するため、ソフトウェアのテストが必要になるケースもある。

1.1.4.　テストと品質(K2)

テストにより、摘出した欠陥を基にして、ソフトウェアの機能/非機能要件や、特性(例えば、信頼性、使用性、効率性、保守性)の観点から、ソフトウェアの品質を計測できる。
非機能のテストの詳細は第 2 章を参照されたい。また、ソフトウェアの特性の詳細は[Software Engineering – Software Product Quality(ISO9126[1])]を参照されたい。

テストを実施して、欠陥がほとんど見つからなかった場合や、欠陥がゼロの場合、対象のソフトウェアの品質に自信が持てる。適切に設計したテストに合格すると、システムが抱えるリスク全般の度合いを下げることができる。テストで欠陥

[1] ISO9126 は、国内標準では JIS X 0129 が対応している。

が見つかった場合、その欠陥を修正すれば、システムの品質は上がる。

以前のプロジェクトから学ぶべきことは多い。他のプロジェクトで見つかった欠陥の根本原因を理解すると、プロセスを改善でき、同じ原因の欠陥の作りこみを防げる。この結果、将来開発するシステムの品質を改善できる。
これは品質保証の側面の一つである。

テストは、(開発標準、教育、欠陥の分析と並ぶ)活動の一つとして品質保証に組み込む必要がある。

1.1.5.　テストの十分性(K2)

どこまでテストをすべきかは、技術や安全性、そしてビジネス上のリスクのレベルや、期間や予算などのプロジェクトの制限により決まる。(リスクは第 5 章で詳細に記述する)

テストは、ステークホルダに十分な情報を提供し、テストしたシステムやソフトウェアの次の開発ステップや顧客への次回リリースについて意思決定をするのに十分な情報を提供するべきである。

1.2.　テストとは何か？(K2)　学習時間:30 分

用語

デバッグ、要件、レビュー、テストケース、テスト、テスト目的

背景

テストとは何か、に対する一般的な認識は、テストを実施すること、すなわち、ソフトウェアの実行であることが多い。ソフトウェアの実行はテストの活動の一部であり、全部ではない。

テストの活動は、テスト実施の前後にも存在する。例えば、計画、コントロール、テスト条件の選択、テストケースの設計と実行、実行結果のチェック、テスト完了基準の検証、テストプロセスやテスト対象システムに関する報告、テストのまとめや終了作業(テストフェーズが完了した後)がある。テストにはドキュメント(ソースコードを含む)レビューや、静的解析を実施することも含む。
動的テストと静的テストは、方法は違っても同じような目的のために使え、テスト対象のシステムだけではなく、開発やテストのプロセス改善のための情報提供もできる。
テストには以下のような目的がある。

- 欠陥を摘出する。
- 対象ソフトウェアの品質レベルが十分であることを確認する。
- 意志決定のための情報を示す。
- 欠陥の作りこみを防ぐ。

ライフサイクルの初期にテスト設計の関わりを考えるプロセスと活動(テスト設計を通してテストベース[2]を検証すること)は、コードの中に欠陥が潜りこまないようにする効果がある。ドキュメント(例えば、要件)のレビュー、問題の認識と解決もコードに欠陥が入るのを防ぐ。
テストの視点が異なると、目的も異なる。例えば、開発でのテスト(コンポーネントテスト、統合テスト、システムテストなど)では、なるべく多くの故障をたたきだし、ソフトウェア中の欠陥を特定して修正することを主目的とする。受け入れテストの場合、システムが期待通りに動作し、要件に合致することの確認が目的となる。また、別のケースでは、ソフトウェアの品質をチェックし(欠陥の修正は目的とはしない)、所定の時期にソフトウェアをリリースすればどんなリスクがあるかという情報をステークホルダ(利害関係者)に提供することもある。保守テストでは、ソフトウェアの変更時に、新たに欠陥が混入していないかチェックするテストも実施することが多い。運用テストでは信頼性や可用性などのシステム特性をチェックするのが主目的である。

デバッグとテストは同じではない。動的テストは欠陥から発生する故障を見つけることである。一方、デバッグは、故障の原因を突き止め、解析し、取り除く一連の開発の活動である。その後、テスト担当者が再テストを実施し、修正により故障が解決したことを確認する。通常、テスト担当者はテストに責任を持ち、開発担当者はデバッグに責任を持つことになる。

テストプロセスとその活動は 1.4 節で述べる。

[2] テストの基となる開発関連資料

1.3.　テストの 7 原則(K2)　学習時間:35 分

用語

全数テスト

原則

これまで 40 年以上にわたり、いろいろなテストの原則が編み出され、あらゆるテストで共通に使える一般的なガイドラインとなってきた。

原則 1:テストは欠陥があることしか示せない

テストにより、欠陥があることはわかるが、欠陥がないことは示せない。テストにより、ソフトウェアに残る未摘出欠陥の数を減らせるが、欠陥が摘出できない状態でも、正しさの証明とはならない。

原則 2:全数テストは不可能

全てをテストすること(入力条件の全組み合わせ)は、ごく単純なソフトウェア以外では非現実的である。全数テストの代わりに、リスクや優先順位によりテストの焦点を絞る。

原則 3:初期テスト

早く欠陥を見つけるために、テストはソフトウェア開発もしくはシステム開発のライフサイクルのなるべく早い時期に開始し、あらかじめ定義した目的に集中すべきである。

原則 4:欠陥の偏在

テストは、モジュールごとの欠陥密度の予測や直近の観察結果に比例してテストの焦点を絞らなければならない。リリース前のテストで見つかる欠陥や運用時の故障の大部分は、ある特定の少数モジュールに集中する。

原則 5:殺虫剤のパラドックス

同じテストを何度も繰り返すと、最終的にはそのテストでは新しい欠陥を見つけられなくなる。この「殺虫剤のパラドックス」を回避するため、テストケースを定期的に見直して、改定する必要がある。ソフトウェアやシステムのいろいろな部分に対しテストを実行して多数の欠陥を摘出するには、これまでと違うテストケースが必要となる。

原則 6:テストは条件次第

条件が異なれば、テストの方法も変わる。例えば、高信頼性が必要な 24 時間稼動するシステムのテストは、e コマースのテストとは異なる。

原則 7:「バグゼロ」の落とし穴

欠陥を見つけて修正しても、構築したシステムが使えなかったり、ユーザの要件や期待を満足したりしなければ役に立たない。

1.4.　基本的なテストプロセス(K1)　　学習時間:35 分

用語

確認テスト、再テスト、終了基準、インシデント[3]、回帰テスト、テストベース、テスト条件、テストカバレッジ、テストデータ、テスト実行、テスト結果記録、テスト計画、テスト手順、テストポリシー、テストスイート、テストサマリレポート、テストウェア

背景

テストでは、テスト実行が一番目に付くが、効率よく有効にテストを進めるには、テスト計画の立案、テストケースの設計、実行の前準備、結果の評価に時間を費やせるようテストを計画すべきである。

基本的なテストプロセスは以下の活動で構成する。

- 計画とコントロール
- 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

論理的には、上記活動は順列に行われるが、実際には、重複したり並列に行われたりする場合がある。通常、システムやプロジェクトの状況に合わせてこれらの活動を調整する必要がある。

1.4.1.　テスト計画作業とコントロール(K1)

テスト計画作業は、テストの目的を定義し、そのテストの使命や目的に合致するようテストの仕様を決めることである。

テストコントロールは、実際の進捗と計画を比較し、計画からの乖離などの状況をレポートする継続的な活動である。テストコントロールには、プロジェクトの目的や使命に合致させるために取る対策も含む。テストの活動は、プロジェクトを通じてモニタする必要がある。テスト計画作業は、モニタリングとコントロールからのフィードバックを考慮する。

テスト計画およびテストコントロール作業は本シラバスの第 5 章に定義されている。

1.4.2.　テストの分析と設計(K1)

テストの分析と設計により、抽象度の高いテストの目的を具体的なテスト条件やテスト設計に変換する。
テストの分析、および設計では以下を実施する。

- テストベース(例えば、要件、ソフトウェア完全性レベル[4](リスクレベル)、リスク解析レポート、アーキテクチャ、設計、インターフェース仕様)をレビューする。
- テストベースやテスト対象の試験性を評価する。
- テストアイテム、仕様、動作、ソフトウェアの構造などの分析に基づいて、テスト条件を識別し優先順位を付け

[3] 動作結果の事象。事象は欠陥として修正する場合もあるし、示唆や要望として開発者へ伝える場合もある。
[4] ステークホルダによって選択されたソフトウェア、または、ソフトウェアベースのシステム特性(例えば、ソフトウェアの複雑さ、リスクアセスメント、安全レベル、セキュリティー・レベル、期待性能、信頼性、または費用)に適合する度合い。これらのシステム特性は、ステークホルダに対するソフトウェアの重要性を反映させて定義するものである。

る。
- 高度なテストケースを設計し、優先順位を付ける。
- テスト条件やテストケースをサポートする上で必要なテストデータを識別する。
- テスト環境を設計し、必要となるインフラストラクチャやツール類を識別する。
- テストベースとテストケースの間で双方向のトレーサビリティを作成する。

1.4.3. テストの実装と実行(K1)

テスト実装とテスト実行は、必要なテストケースを実行順に組み合わせて、その他のテスト実行に必要な情報を含めてテスト手順、またはスクリプトとして仕様化し、また、テスト環境のセットアップとテストを実行する活動である。

テスト実装と実行では、以下を実施する。
- テストケースを決定し、実装し、優先順位を付ける。(テストデータの識別も含まれる)
- テスト手順を開発し、優先順位を付け、テストデータを作成する。場合によってはテストハーネスを準備し、テストの自動実行スクリプトを書く。
- 効率よくテストを実行するため、テスト手順をベースにしてテストスイートを作る。
- テスト環境を正しくセットアップしたことを確認する。
- テストベースとテストケースの間で双方向のトレーサビリティを確認し更新する。
- 計画した順番に従い、テスト手順を人力、もしくはテストツールで実行する。
- テストの実行結果の記録を取り、テスト対象のソフトウェア、テストツール、テストウェアの ID とバージョンを記録する。
- 実行結果と期待する結果を比較する。
- 両者が一致しない場合、インシデントとして報告し、原因(コード、テストデータやテストドキュメントの欠陥、テスト実行方法の誤りなど)を解明するために、インシデントを分析する。
- 実行結果と期待する結果が一致しないケースごとに、テスト活動を繰り返す。例えば、修正が正しいことを確認するため、前回不一致となったテストケースを再実行(確認テスト)したり、改訂版のテストケースや元のテストケースを実行したり、変更していないプログラム部分に新たな欠陥が入り込んでいないことや、欠陥修正により陰に隠れていた欠陥が現れないことを確認したりする。(回帰テスト)

1.4.4. 終了基準の評価とレポート(K1)

終了基準の評価は、定義した目的に対し、テストの実行が十分かをチェックする活動である。このチェックは、各テストのレベルで実施する必要がある。(2.2 節参照)

終了基準の評価の作業として主なものは以下のとおり。
- テスト結果記録をテスト計画作業で定義した終了基準と比較する。
- 追加テストが必要か、あるいは定義した終了基準を変更するべきかを判断する。
- ステークホルダにテストサマリレポートを書く。

1.4.5. 終了作業(K1)

終了作業では、終了したテストの全活動のデータを集め、プロジェクトから得たこと、テストウェア、実績と数字を集約する。終了作業はプロジェクトの状況が次のようなときに実施する。例えば、システムがリリースされたとき、テストプロジェクトが完了(または打ち切り)したとき、マイルストンに到達したとき、保守版をリリースしたときなどである。
終了作業の主なものは以下のとおり。
- 計画にある成果物がリリースされたかをチェックする。
- インシデントレポートを終了させるか、もしくは未対策の欠陥を変更記録に載せる。
- システムを受け入れるために文書を作成する。
- 次回も使えるように、テストウェア、テスト環境、テストのインフラストラクチャをまとめ、文書に記録する。

- テストウェアを保守部門に引き渡す。
- 次回のリリースやプロジェクトのために教訓とすべきことをまとめる。
- その情報をテストの成熟度を改善するために利用する。

1.5.　テストの心理学(K2)　学習時間:25 分

用語

エラー推測、テストの独立性

背景

テストやレビューのためのマインドセットは、ソフトウェア開発の場合と異なる。適切なマインドセットを持っていれば、開発担当者が自分の作ったコードをテストすることも可能であるが、テストを切り出してテスト担当者に割り当てる方が効率的であり、教育を受けたプロのテスト担当者による、開発から独立した視点でのテストといった付加的な利点が期待できるためである。独立性を確保したテストは、あらゆるレベルのテストで実施可能である。

独立性が高いと、多くの場合、(作者のバイアスを排除できるので)テスト担当者の欠陥や故障を摘出する効果を高くできる。しかし、独立性は熟知に取って代わるものではない。開発者は、自分の作ったコードの中の欠陥を数多く見つけることができる。低レベルから高レベルまでの独立性の度合いを以下に示す。

- ソフトウェアを作成した本人がテストを設計する。(独立性は低い)
- 開発者とは別の人(開発チームの人)がテストを設計する。
- 開発者とは別の部署の人(例えば独立したテストチーム)またはテストの専門家(例えば使用性テストまたは性能テストの専門家)がテストを設計する。
- 開発者とは別の会社の人がテストを設計する。(すなわち、外部のテスト会社へ委託したり、外部団体の認証を受けたりする)

プロジェクトメンバーやプロジェクトの行動は目的次第である。プロジェクトメンバーは管理者やステークホルダが設定した目的、例えば、欠陥の摘出という目的、あるいはソフトウェアがその目的を果たしていることを確認する目的に沿って自分の作業の計画を立てる傾向がある。このためテストの目的を明確に定めることは非常に重要である。

テストで故障を見つけることは、対象ソフトウェアやそれを作成した開発者に対する非難と解釈されることがある。その結果、テストはプロダクトリスクのマネジメントという点では前向きで建設的な作業ながら、破壊的な活動と見ることも多い。システムの故障を見つけるには、好奇心、プロとしての悲観的な考え、批判的な視点、細部まで見逃さない注意力、開発担当者との良好なコミュニケーション、エラーを嗅ぎ出す経験が必要となる。

見つけたエラー、欠陥、故障を建設的に扱うことができれば、テスト担当者と設計者、開発担当者との間の敵対感情を避けられる。これはテストだけでなく、レビュー時に発見される欠陥でも同様である。

テスト担当者やテストリーダは、欠陥、テストの進捗、リスクの情報をやり取りする場合、建設的に作業が進むよう優れたコミュニケーションスキルが必要となる。テストで欠陥を見つけて修正するとリリース後に検出して修正するより時間と金の節約になり、リスクも減る。欠陥の情報はソフトウェアやドキュメントの作成者のスキルを高めることにつながる。

テスト担当者が「悪いニュースを伝える使者」との扱いを受けると、コミュニケーションの問題が起きる。テスト担当者とその他の人とのコミュニケーションや関係を改善する方法には以下のものがある。

- 対決ではなく、協調姿勢で開始する。全員のゴールは、高品質のシステムであること再認識するとよい。
- プロダクトに対する指摘は、中立で、事実を中心にして実施し、欠陥を作りこんだ担当者を非難する方向に行かないようにする。例えば、客観的かつ事実に基づいたインシデントレポートを書いたり、見つけたことをレビューしたりする。
- 他人の気持ちや反応を理解するように努力する。
- 自分の言ったことを他人が理解し、他人の言ったことを自分が理解していることを確認する。

1.6.　行動規範(K2)　学習時間:10 分

ソフトウェアテストに関与することで、重要な機密情報を知る場合がある。特に、このような情報が不適切に使用されないようにするために行動規範が必要になる。ISTQB®では、ACM および IEEE が制定したエンジニアのための行動規範を基に以下の行動規範を示す。

公人 – 認定されたソフトウェアテスト担当者は、常に公人として行動しなければならない。

顧客と雇用主 – 認定されたソフトウェアテスト担当者は、常に公人として行動しつつ、顧客や雇用主に最大限の利益をもたらさなければならない。

プロダクト – 認定されたソフトウェアテスト担当者による成果物(自身がテストしたプロダクトやシステムに関するもの) は、プロフェッショナルとして高いレベルのものでなければならない。

判断 – 認定されたソフトウェアテスト担当者によるプロフェッショナルとしての判断は、誠実なものであり、かつ自身でなしたものでなければならない。

マネジメント – 認定されたソフトウェアテストマネージャおよびリーダは、ソフトウェアテストのマネジメントに対する倫理的なアプローチに同意した上で、これを推進しなければならない。

専門職としての地位 – 認定されたソフトウェアテスト担当者は、公共の利益に寄与することで、専門職としての地位向上に努めなければならない。

同僚 – 認定されたソフトウェアテスト担当者は、同僚に対し公正かつ協力的でなければならず、ソフトウェア開発者と協調しなければならない。

自身 – 認定されたソフトウェアテスト担当者は、生涯その専門性を磨くための学習を続けるとともに、実践の場でも倫理的なアプローチを広めなければならない。

参考文献

1.1.5 Black, 2001, Kaner, 2002

1.2 Beizer, 1990, Black, 2001, Myers, 1979

1.3 Beizer, 1990, Hetzel, 1988, Myers, 1979

1.4 Hetzel, 1988

1.4.5 Black, 2001, Craig, 2002

1.5 Black, 2001, Hetzel, 1988

2. ソフトウェアライフサイクルを通じてのテスト(K2) 学習時間:115 分

本章の学習の目的

本章で記述する学習内容は以下のとおり。

2.1 ソフトウェア開発モデル(K2)

LO-2.1.1 開発ライフサイクルにおける開発作業、テスト活動、成果物の関係を、例として与えられたプロジェクト、プロダクトを利用して説明する。(K2)

LO-2.1.2 プロジェクトやプロダクトの特性を背景としたソフトウェア開発モデルを適用する必要性を認識する。(K1)

LO-2.1.3 どんなライフサイクルモデルにも適用できる良いテストの特性を再認識する。(K1)

2.2 テストレベル(K2)

LO-2.2.1 いろいろなレベルのテストを比較する。例えば、テストの主目的、テストの主な対象物、典型的なテストのターゲット(例えば、機能テスト、構造テスト)、テストに関連する開発成果物、テスト担当者、識別される欠陥や故障の種類。(K2)

2.3 テストタイプ(K2)

LO-2.3.1 4 つのテストタイプ(機能テスト、非機能テスト、構造テスト、変更部分のテスト)の例を示して比較する。(K2)

LO-2.3.2 機能テストと構造テストはどんなテストレベルでも現れることを認識する。(K1)

LO-2.3.3 非機能要件をベースにした非機能的なテストがどのようなものかを説明する。(K2)

LO-2.3.4 ソフトウェア、システムの構造やアーキテクチャの分析をベースにしたテストがどのようなものであるかを説明する。(K2)

LO-2.3.5 確認テストと回帰テストの目的を説明する。(K2)

2.4 保守テスト(K2)

LO-2.4.1 保守テスト(既に開発済みのシステムのテスト)をテストタイプ、テストを開始するきっかけ、テストの量の点で、新規開発アプリケーションのテストと比較する。(K2)

LO-2.4.2 保守テストを実施する指針(変更、別システムへの移行、ハード入れ替えなど)を認識する。(K1)

LO-2.4.3 保守において回帰テスト、および影響度分析の役割を説明する。(K2)

2.1.　ソフトウェア開発モデル(K2)　　学習時間:20 分

用語

市販ソフトウェア(COTS)、イテレーティブ-インクリメンタル開発モデル、妥当性確認、検証、V 字モデル

背景

テストは単独では存在しない。すなわち、テスト活動はソフトウェアの開発作業と密接に関係する。開発のライフサイクルモデルが異なれば、テストに対するアプローチも異なる。

2.1.1.　V 字モデル(シーケンシャル開発モデル)(K2)

V 字モデルにはいろいろな派生モデルがあるが、一般的な V 字モデルでは開発の 4 つのレベルに対応して、4 つのテストレベルがある。本シラバスで扱う 4 つのテストレベル[5]は以下のとおり。

- コンポーネントテスト(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

実際にはV字モデルでの開発やテストレベルは、プロジェクトや開発するソフトウェアの特性により、これより多かったり少なかったり、また、種類が異なる場合がある。例えばコンポーネントテストの後に、コンポーネント統合テストを設けたり、システムテストの後にシステム統合テストを実施したりすることがある。

ソフトウェアの開発で作成する成果物(例えば、ビジネスシナリオ、ユースケース、要求仕様、設計ドキュメント、コード)は、上記レベルでのテストのベースとなる。一般的な成果物に対する参考文献として CMMI(Capability Maturity Model Integration)や、ソフトウェアライフサイクルプロセス(IEEE/IEC 12207)[6]などがある。検証や妥当性確認(および、初期のテスト設計)は、ソフトウェアの成果物を開発しているときに実施する。

2.1.2.　イテレーティブ-インクリメンタル開発モデル(K2)

イテレーティブ-インクリメンタル開発は、システムの要件、設計、構築、テストを何回かの短い開発サイクルで繰り返してソフトウェアを作るモデルである。これに該当する方式として、プロトタイピング、RAD、RUP、アジャイル開発モデルがある。イテレーション(反復)によって開発したシステムは、それぞれのイテレーションにおいて複数のテストレベルでテストされる。開発済みのソフトウェアに追加する場合も、システムの一部を構成するのでテストが必要である。開発を繰り返すたびに回帰テストの重要性は増していく。検証や妥当性確認は、各繰り返しにおいて、実施できる。

2.1.3.　ライフサイクルモデルの中のテスト(K2)

どのようなライフサイクルモデルだとしても、以下のように、良いテストを行うときの特徴がいくつかある。

- 各開発活動に対応してテスト活動がある。
- 各テストレベルには、そのレベル特有の目的がある。

---

[5] テストフェーズ(ステージ)と同義である。

[6] 原文には「'Software life cycle processes' (IEEE/IEC 12207)」とあるが、「ソフトウェアライフサイクルプロセス(ISO/IEC 12207)」である。

- テストレベルのテスト分析や設計は、対応する開発活動を進めている間に実施すべきである。
- 開発ライフサイクルで、ドラフトのドキュメントが入手でき次第、テスト担当者はレビューを実施すべきである。

テストレベルは、プロジェクトやシステムアーキテクチャの特性により、統合したり、再編成したりしてもよい。例えば、市販ソフトウェアプロダクト(COTS)をシステムに組み込む場合、システムレベル(例えば、インフラストラクチャや他システムへの統合、あるいは、システムのデプロイ)で、統合テストを実施したり、受け入れテスト(機能テスト、非機能テストやユーザテスト、運用テスト)を実施したりする。

2.2.　テストレベル(K2)　学習時間:40 分

用語

アルファテスト、ベータテスト、コンポーネントテスト、ドライバ、フィールドテスト、機能要件、統合、統合テスト、非機能要件、ロバストネステスト、スタブ、システムテスト、テスト環境、テストレベル、テスト駆動開発、ユーザ受け入れテスト

背景

各テストレベルで、押さえておく必要があるのは、一般的な目的、テストケースを導き出す場合に参照する開発成果物(すなわち、テストベース)、テスト対象(すなわち、何をテストするか)、テストで見つかる典型的な欠陥や故障、テストツールの必要性やツールによる環境、特定のアプローチと責任割り当てなどがある。

もし、そのようなデータがシステムの一部ならば、システム構成データのテストはテスト計画中に考慮されるべきである。

2.2.1.　コンポーネントテスト(K2)

テストベース:

- コンポーネント要件
- 詳細設計
- ソースコード

典型的なテスト対象:

- コンポーネント
- プログラム
- データ変換／移行プログラム
- データベースモジュール

コンポーネントテスト(ユニットテスト、モジュールテストもしくはプログラムテスト)は、分離してテストが可能なソフトウェア、モジュール、プログラム、オブジェクト、クラスなどの欠陥を摘出し、機能の正しさを検証するテストである。開発のライフサイクルやシステムの特性により、コンポーネントテストは、システムの他の部分と切り離したテストが可能である。この場合、スタブ、ドライバ、シミュレータなどを使う。

コンポーネントテストには、機能のテスト、特定の非機能的な特性、リソースの動作(例えばメモリリークの検出)やロバストネステスト、構造テスト(例えばデシジョンカバレッジ)も含む。テストケースは、コンポーネントの仕様、ソフトウェアの設計やデータモデル等の成果物をベースに作る。

一般に、コンポーネントテストは、ユニットテストのフレームワークやデバッグツールを備えた開発環境にてコードを対象に行う。現実的には、コンポーネントテストはコードを作成した開発者を巻き込んで実施することが多い。摘出した欠陥は、正式に管理されることなく、見つけたらすぐに修正するのが一般的である。

コンポーネントテストに対する一つのアプローチは、コーディング前にテストケースを準備して実行を自動化することである。これはテストファーストとか、テスト駆動開発と呼ぶ。このアプローチは、何度も繰り返し、テストケースを開発するサイクルをベースとし、部分的なコードを作っては統合し、コンポーネントテストに合格するまで修正と反復(イテレーション)を繰り返すものである。

2.2.2. 統合テスト(K2)

テストベース:

- ソフトウェア設計もしくはシステム設計
- アーキテクチャ
- ワークフロー
- ユースケース

典型的なテスト対象:

- サブシステムのデータベース実装
- インフラストラクチャ
- インターフェース
- システム構成・構成データ

統合テストは、コンポーネント間のインターフェース、OS、ファイルシステム、ハードウェアなどのシステムの別の部分との相互処理や、システム間のインターフェースをテストする。
統合テストにもいくつかのレベルがあり、いろいろな大きさのテスト対象物に対して以下のように実施する。

- コンポーネント統合テストは、ソフトウェアコンポーネント間の相互処理をテストし、コンポーネントテストの後に実施する。
- システム統合テストは、他システム間やハードウェアとソフトウェア間の相互処理をテストする。システムテストの後に実施することが多い。システムテストの後にシステム統合テストを実施するのは、片方の開発組織はインターフェースの片方しか制御できない。これはリスクとして考慮されるかもしれない。ビジネスプロセスは、一連のシステムの中に含まれるワークフローとして実装する。別のプラットフォームで動作するかどうかも重要な問題となる。

統合の範囲が大きくなるほど、欠陥を特定のコンポーネントないしシステムに絞り込むことが難しくなる。これによりリスクが増え、トラブルシューティングの時間が増大する。

体系的に統合する戦略は、システムアーキテクチャ(例えばトップダウンやボトムアップなど)、機能的タスク、トランザクション処理シーケンス、その他システムの形態やコンポーネントがベースとなる。欠陥の切り分けを容易にし、早期に摘出するために、統合は一度に実施するのではなく(ビッグバン統合ではなく)、少しずつ統合するのが普通である。

特定の非機能特性(例えば性能)のテストは機能テストと同様に統合テストに含むことがある。

統合の各段階で、テスト担当者は、統合だけに集中する。例えばモジュールＡとモジュールＢを統合する場合、テスト担当者は、モジュール間のインターフェースのみに興味がある。単一モジュールの機能については、コンポーネントテストで終了している。これには機能的および構造的アプローチが使える。

テスト担当者が統合計画の構造や影響を理解しているのが理想である。コンポーネントやシステム構築前に統合テストを計画する場合、テスト効率が最大になる順序でコンポーネントを構築できる。

2.2.3. システムテスト(K2)

テストベース:

- システム要求仕様もしくはソフトウェア要求仕様
- ユースケース
- 機能仕様
- リスク分析レポート

典型的なテスト対象:

- システム、ユーザとオペレーションマニュアル
- システム構成・構成データ

システムテストは、システムやソフトウェアプロダクトの振る舞いに関わりがある。テストのスコープは、マスターテスト計画、テストレベルごとのレベルテスト計画で明確に定められるべきである。

システムテストは、テストで見つけられない環境関連の故障が残存するリスクを最小限にするため、最終的なターゲットや運用もしくは使用する環境と可能な限り同じテスト環境で行わなければならない。

システムテストには、リスク、または ビジネスプロセス、ユースケース、その他システム動作を高レベルで記述したテキストもしくはモデル、オペレーティングシステムとの相互作用、システムリソースなどの要求仕様に基づいたテストがある。

システムテストは、システムの機能要件と非機能要件およびデータの品質特性を検証する必要がある。テスト担当者は不完全であったり、ドキュメント化していなかったりする要件をも扱わねばならない。
システムテストの対象となる機能要件を検証する最適の技法(ブラックボックス)を使用する。例えば、ビジネスルールを組み合わせるためにデシジョンテーブルを使う。構造的技法(ホワイトボックス)は、例えばメニューの構造やウェブページのナビゲーションをテストするために実施する(詳細は第 4 章参照)。

独立したテストチームがシステムテストを請け負うことが多い。

2.2.4. 受け入れテスト(K2)

テストベース:

- ユーザ要件
- システム要件
- ユースケース
- ビジネスプロセス
- リスク分析レポート

典型的なテスト対象:

- 統合が完了しているシステム上のビジネスプロセス
- 操作と保守プロセス
- ユーザ(の利用)手順
- 書式
- レポート
- 構成データ

受け入れテストは、システムを使う顧客やユーザが実施することが多い。また、ステークホルダが参加することもある。

受け入れテストのゴールは、システム全体、システムの一部、非機能的な特性が正しいことを確認することにある。受け入れテストでは、欠陥を摘出することは主目的ではない。受け入れテストは、システムが稼動できるかをテストするものであり、最終テストである必要はない。例えば受け入れテストの後で、大規模システム統合テストを実施することがある。

受け入れテストは、ライフサイクルの様々な場面で行われる。例えば、以下のようなものがある。

- 市販ソフトウェアプロダクト(COTS)は、インストールや統合時点で受け入れテストを実施する。
- コンポーネントの使用性の受け入れテストは、コンポーネントテストで実施する。
- 新機能の拡張分は、システムテスト前に実施する。

受け入れテストは、通常、以下の形がある。

ユーザ受け入れテスト

システムがビジネスで使えるかをユーザがテストするもの。

運用受け入れテスト

システム管理者による受け入れテスト。これには以下の種類がある。

- バックアップ／リストアのテスト
- 災害復旧テスト
- ユーザマネジメント
- 保守
- データ負荷と移行
- セキュリティの脆弱性の定期的チェック

契約による受け入れテスト、および規定による受け入れテスト

契約による受け入れテストは、カスタムメイドのプロダクトを開発する場合、契約書に記述した判定条件に従って検証する。契約の当事者が契約に同意するとき、受け入れ条件は定義される。
規定による受け入れテストは、法律や安全基準などに合致しているかを検証する。

アルファテスト、ベータテスト(あるいはフィールドテスト)

市販ソフトウェアを開発する場合、実際に市場へリリースする前に将来のユーザや現在の顧客からフィードバックを受けたいと考えることが多い。アルファテストは、開発組織内で実施するテストであるが、開発チームが実施するものではない。ベータテスト、あるいはフィールドテストは、顧客もしくは潜在顧客によるテストである。

顧客のサイトにシステムをセットする前後にテストが必要なシステムでは、工場受け入れテスト、サイト受け入れテスト、という言葉を使うことがある。

2.3.　テストタイプ(K2)　　学習時間:40 分

用語

ブラックボックステスト、コードカバレッジ、機能テスト、相互運用性テスト、ロードテスト、保守性テスト、性能テスト、移植性テスト、回帰テスト、信頼性テスト、セキュリティテスト、ストレステスト、構造テスト、テストスイート、使用性テスト、ホワイトボックステスト

背景

テスト活動群は、ある特定の目的、あるいは、テスト対象をベースにして、ソフトウェアシステム、(あるいは、その一部)の検証をすることに向けることができる。
テストタイプは、次に示すように特定のテスト目的に焦点をあてたものである。

- ソフトウェアの機能
- 信頼性、使用性等の非機能的な特性
- ソフトウェアやシステムの構造やアーキテクチャ
- 関連する変更、すなわち欠陥が正しく修正されていること(確認テスト)や、意図しない変更がないことの確認(回帰テスト)

ソフトウェアのモデルを作って構造テスト(例えば、制御フローモデルやメニュー構造モデル)や非機能テスト(例えば、パフォーマンスモデル、ユーザビリティモデル、セキュリティ脅威モデル)、機能テスト(例えば、プロセスフローモデル、状態遷移モデル、自然言語で記述した仕様)で使うことがある。

2.3.1.　機能のテスト(機能テスト)(K2)

システム、サブシステム、コンポーネントの機能は、要求仕様書、ユースケース、機能仕様書で記述してあったり、全くドキュメント化していない場合もあったりする。機能とはシステムが「何をするのか」である。
機能テストは、機能やフィーチャ(ドキュメント中に記述してあること、あるいはテスト担当者が理解していること)と、それらの特定のシステムとの相互運用性に基づいたものであり、全てのテストレベル(例えば、コンポーネント仕様をベースにしたコンポーネントのテスト)で実施する必要がある。
ソフトウェアやシステムの機能からテスト条件とテストケースを抽出する場合に、仕様ベースの技法を使える(詳細は第4章参照)。機能テストは、ソフトウェアの外部動作を検証する(ブラックボックステスト)。
機能テストの一種であるセキュリティテストでは、ウイルスなど、外部からの悪意ある脅威を検知する機能(例えば、ファイアウォール)をチェックする。
他にも機能テストとして相互運用性(接続性)テストがあるが、これはソフトウェア製品が特定の一つ以上のコンポーネントやシステムと相互作用する能力を評価する。

2.3.2.　機能以外の特性のテスト(非機能テスト)(K2)

非機能テストには、例えば、性能テスト、ロードテスト、ストレステスト、使用性テスト、相互運用性テスト、保守性テスト、信頼性テスト、移植性テストなどがある。このテストは、システムが「何をするのか」のテストではなく、「どのように動作するのか」のテストである。
非機能テストは、全てのテストレベルで実施できる。非機能テストは、システムやソフトウェアの特性を計測するために

必要なテストであり、性能テストにおける応答時間のように、様々なものさしで計測できる。このテストを実施する場合、[Software Engineering – Software Product Quality(ISO9126)]で定義しているソフトウェアの品質モデルを参照するとよい。
非機能テストは、ソフトウェアの外部動作を確認する。そして、多くの場合、その実現のためにブラックボックステスト設計技法を使用する。

2.3.3.　ソフトウェアの構造／アーキテクチャのテスト(構造テスト)(K2)

構造テスト(ホワイトボックステスト)は、全てのテストレベルで実施できる。構造をどの程度網羅したかで、テスト実施の完全性を評価するため、構造テストの技法は、仕様ベースの技法を適用後に実施するのがよい。

カバレッジとは、テストスイートが実施したテストでソフトウェアの構造の何パーセントをカバーしたかを示すものである。カバレッジが 100%でない場合、カバレッジを上げるために未実行部分のテストを設計する。カバレッジ関連の技法は第 4 章で述べる。
全てのテストレベル、特にコンポーネントテストとコンポーネント統合テストではツールを使い、ステートメントカバレッジやデシジョンカバレッジのようにコードのカバレッジを計測できる。構造テストは、呼び出しの上下関係のようにシステムのアーキテクチャをベースにしている。
構造テストのアプローチは、システム、システム統合、受け入れテストレベル(例えば、ビジネスモデルやメニュー構造)でも適用できる。

2.3.4.　変更部分のテスト:再テスト、および回帰テスト(K2)

欠陥を見つけて修正した場合、元の欠陥が正しく修正されたことを確認するための再テストが必要となる。これが、確認テストである。デバッグ(欠陥の修正)は開発上の活動であり、テストの活動ではない。

回帰テストは、変更をした場合、変更によって作りこんだ欠陥や新しい別の欠陥を見つけるため、既にテスト済みのプログラムを何度もテストすることである。この種類の欠陥は、テスト対象のソフトウェアに存在することもあるし、関連するソフトウェア、あるいは全く別のソフトウェアに潜むこともある。回帰テストは、ソフトウェアかソフトウェア動作環境が変わったときに実施する。回帰テストの範囲は、正常に動いていたソフトウェアから欠陥が見つからないリスクを基に決める。
確認テストや回帰テストのために、テストは繰り返し実施できるようにしておくべきである。

回帰テストは、全てのテストレベルで実行可能であり、機能テスト、非機能テスト、構造テストなどを含む。回帰テストのテストスイートは何度も実行し、少しずつ内容が変わるのが普通である。従って、回帰テストは、自動化による効果が非常に大きい。

2.4.　保守テスト(K2)　学習時間:15 分

用語

影響度分析、保守テスト

背景

ソフトウェアがリリースされると、サービスは数年から数十年続く。稼動期間中にシステムや構成データ、または動作環境は、修正、変更、拡張されることが多い。事前のリリース計画は保守テストの成功に欠かせない。計画されたリリースと修正パッチは区別する必要がある。保守テストは、現在稼動しているシステムに実施する。保守テストは、システムやソフトウェアの変更作業、別システムや別ソフトウェアへの移行作業、システムやソフトウェアの回収作業が発生した場合に実施する。

変更作業には、計画に従った拡張(リリース計画)、修正や緊急変更、また、計画的な OS の変更やデータベースのアップグレード、計画的なCOTSソフトウェアのアップグレード、あるいは新しく見つかった OS のセキュリティホールの修正パッチなど、動作環境の変更も含む。

移行作業(例えば、別のプラットフォームへの移行)の保守テストには、変更したソフトウェアだけでなく、新しい環境の動作テストも必要である。維持されているシステムに他のアプリケーションからのデータを移行する際には、移行テスト(コンバージョンテスト)もまた必要である。

システムの入れ替えに関する保守テストで長期間のデータ保有を要求されている場合には、データの移行作業や保管もテストの対象になる。

保守テストでは、変更部分のテストに加え、未修正部分に対する回帰テストも実施する必要がある。保守テストの実施範囲は、変更のリスク、現実のテストの規模、変更の大きさに依存する。変更の程度により、保守テストは全てのテストタイプ、全てのテストレベルに適用できる。
変更により既存システムがどの程度影響を受けるかチェックすることを影響度分析と呼ぶ。この分析により、どの程度回帰テストをしなければならないかを予測できる。影響度分析は、回帰テストスイートを決めるために使われることがある。

仕様が最新版でない場合や存在しない場合、また、ドメイン知識を持つテスト担当者がいない場合には、保守テストは難しくなる。

参考文献

2.1.3 CMMI, Craig, 2002, Hetzel, 1988, IEEE 12207
2.2 Hetzel, 1988
2.2.4 Copeland, 2004, Myers, 1979
2.3.1 Beizer, 1990, Black, 2001, Copeland, 2004
2.3.2 Black, 2001, ISO 9126

2.3.3 Beizer, 1990, Copeland, 2004, Hetzel, 1988

2.3.4 Hetzel, 1988, IEEE Std 829-1998

2.4 Black, 2001, Craig, 2002, Hetzel, 1988, IEEE Std 829-1998

3. 静的技法(K2)　学習時間:60 分

本章の学習の目的

本章で記述する学習内容は以下のとおり。

3.1 静的技法とテストプロセス(K2)

LO-3.1.1 いろいろな静的技法でテストできる成果物を知る。(K1)

LO-3.1.2 ソフトウェア成果物の品質を確認する手段として、静的技法の重要性と価値を述べる。(K2)

LO-3.1.3 ソフトウェアライフサイクルにおける(技法の)目的、検出する欠陥の種類、役割を考慮して静的技法と動的技法の違いを説明する。(K2)

3.2 レビュープロセス(K2)

LO-3.2.1 一般の公式レビューの活動、役割、責任を再認識する。(K1)

LO-3.2.2 いろいろな種類のレビュー、例えば、非公式レビュー、テクニカルレビュー、ウォークスルー、インスペクション等の違いを説明する。(K2)

LO-3.2.3 レビューを成功させるために、必要な要素を説明する。(K2)

3.3 ツールによる静的解析(K2)

LO-3.3.1静的解析で検出できる代表的な欠陥やエラーを認識し、レビューや動的テストで検出できるものと比較する。(K1)

LO-3.3.2 例を用いて静的解析の主な利点を述べる。(K2)

LO-3.3.3 静的解析ツールで検出できるコードや設計の代表的な欠陥を挙げる。(K1)

3.1.　静的技法とテストプロセス(K2)　学習時間:15 分

用語

動的テスト、静的テスト

背景

静的テスト技法では、動的テスト技法(ソフトウェアの実行が必要)と異なり、コードやプロジェクトで作成した開発ドキュメントの人力による確認(レビュー)と自動解析(静的解析)を、コードを実行せずに行う。

レビューは、ソフトウェア成果物(コードも含む)をテストする一つの手段であり、動的テスト実行前に大きな効果がある。ライフサイクルの初期のレビューで摘出した欠陥(例えば要件の欠陥)の修正は、コードを実行して検出した欠陥を修正する場合にくらべ、はるかに安価である。

レビューは、全て人力で実施可能だが、サポートするツールもある。人力による活動の主なものは、成果物の内容をチェックして、コメントすることである。要求仕様、設計仕様、コード、テスト計画、テスト仕様、テストケース、テストスクリプト、ユーザガイド、ウェブページ等あらゆるソフトウェア成果物はレビュー可能である。

レビューの効果には、欠陥を早期に摘出・修正できる、開発生産性が向上する、開発期間を短縮できる、テストのコストや時間を減らせる、システムのライフサイクルを通じたコストを削減できる、欠陥の数が減る、コミュニケーションが良くなるなどがある。レビューでは、機能や処理(例えば要件)の抜けを見つけることができる。これは、動的テストではなかなか見つからない。

レビュー、静的解析、動的テストは、欠陥の検出という同じ目的を持っている。この 3 つの技法は、お互いに補い合っている。すなわち、異なる技法は異なる欠陥を効率よく効果的に摘出する。静的テスト技法は、動的テストと比較した場合、故障自体ではなく、故障の原因(欠陥)を探すという違いがある。

動的テストよりレビューの方が簡単に見つけられる欠陥には、規格からの逸脱、要件の欠陥、設計の欠陥、保守の不十分性、インターフェース仕様の不正などが含まれる。

3.2.　レビュープロセス(K2)　学習時間:25 分

用語

開始基準、公式レビュー、非公式レビュー、インスペクション、メトリック、モデレータ、ピアレビュー、レビューア、書記、テクニカルレビュー、ウォークスルー

背景

レビューには様々なタイプがあり、きわめて非公式なもの(レビューア向けに記述された説明書がないもの)から、きわめて公式なもの(チームの参加、文書化されたレビュー結果、レビューを実施するための文書化された手続き)に至るまで多岐にわたる。レビュープロセスの形式は、開発プロセスの成熟度、法律や規則から来る必要性、監査上の必要性などで決まる。

レビューの実施方法は、レビュー目的として合意したこと(例えば、欠陥を見つける、内容を理解する、テスト担当と新しいチームメンバーを教育する、協議の上で判断を下す)により異なる。

3.2.1.　公式レビューの活動(K1)

一般的な公式レビューには以下の活動がある。

1. 計画:
   - レビュー基準を定義する
   - 人選する
   - 役割を割り当てる
   - より公式なレビュータイプ(例えばインスペクション)の場合、開始条件と終了条件を定義する
   - ドキュメントのどの部分をレビュー対象にするか選定する
   - 開始基準をチェックする(より公式なレビュータイプの場合)
2. キックオフ:
   - ドキュメントを配付する
   - 参加者に目的、プロセス、ドキュメントの説明をする
3. 個々の準備:
   - レビューミーティングの準備としてドキュメントをレビューする
   - 可能性のある欠陥、質問、コメントを書き出す
4. 実施する/評価する/結果を記録する (レビューミーティング):
   - 議論を行い、結果や議事内容を文書として記録する(より公式なレビュータイプの場合)
   - 欠陥について述べ、欠陥の扱いに対する意見を示し、欠陥を書き出す
   - 物理的なミーティングの最中に、実施、評価、記録を行う。またグループの電子的なコミュニケーションの追跡を行う。
5. 再作業:
   - 検出した欠陥の修正 (通常、作成者が実施する)
   - 更新された欠陥のステータスを記録する(公式なレビューの場合)
6. フォローアップ:
   - 欠陥を処理したかチェックする

- メトリクスを収集する
- 終了基準のチェック（より公式なレビュータイプの場合）

3.2.2. 役割と責任(K1)

一般的なレビューでは、以下の役割の人がいる。

- マネージャ:レビューを実施する人。実施のスケジュールを立て、レビューの目的が適切かどうかを判断する。
- モデレータ:ドキュメントのレビューを取りまとめる人。取りまとめには、レビューの計画、レビューの運営、ミーティング後のフォローアップも含む。必要なら、様々な意見の仲裁を行う。レビューの成功はこの人によることが多い。
- 作成者:レビュー対象のドキュメントに責任を持つ人。
- レビューア:ある特定の技術やバックグラウンドを持つ各個人(チェッカーとかインスペクタと呼ぶこともある)。必要な準備の後、レビュー対象物で、気がついたこと(例えば欠陥)を示したり述べたりする。レビューアは、いろいろな分野や、レビュープロセスの役割を代表する人として選ばれ、あらゆるレビューに参加する。
- 書記(記録係):レビューで取り上げた全ての課題、問題点、未解決事項を記録する人。

ソフトウェア成果物や関係する作業成果物を別の視点から見たり、チェックリストを使ったりするとレビューの効率が上がり、効果も大きくなる。例えば、ユーザ、保守担当者、テスト担当者、オペレータのような様々な視点から見たチェックリストや、要件の代表的な問題点を挙げたチェックリストが、以前に見つけられていない課題を明確にすることに役立つ場合がある。

3.2.3. レビューの種類(K2)

一つのソフトウェア成果物、または関連する作業成果物を何度もレビューすることがある。複数回のレビューを実施する場合、レビューの種類により、実施する順番が異なる。例えば、非公式レビューは、テクニカルレビューの前に実施し、要求仕様のインスペクションは、顧客と実施するウォークスルーの前に実施する。代表的なレビューの特性、オプション、目的を以下に示す。

非公式レビュー

- 決まったプロセスはない。
- ペアプログラミングや、設計やコードを技術指導的にレビューする形式を取ることがある。
- レビュー結果をドキュメント化することがある。
- レビューアにより、効果が様々である。
- 主な目的は、金や時間をかけずにある程度の効果を出すことである。

ウォークスルー

- 作成者が進行を主導する。
- シナリオを基に、同僚のグループが机上で処理をたどる形式を取ることがある。
- 制約を付けないセッション。
  - レビューアの事前準備ミーティングを行うことがある。
  - 指摘の一覧を含む、レビューレポートを準備することがある。
- (作成者以外が)記載者になることもある。
- 運営により、きわめて非公式のものから、高度に公式なものまである。
- 主な目的は、ソフトウェアの内容を学習、理解し、欠陥を見つけることである。

テクニカルレビュー

- 欠陥摘出には、同僚や技術のエキスパートが参加し(管理者が参加する場合もある)、そのプロセスはあらかじ

め定義し、ドキュメント化してある。
- 管理者の参加しないピアレビューとして進めることもある。
- 経験を積んだモデレータ(作成者ではなく)が進行を主導するのが理想である。
- レビューアが事前ミーティングを準備する。
- チェックリストを作ることがある。
- 指摘一覧、ソフトウェア成果物が要求事項を満たしているかどうかの判定、そして適宜、指摘事項に関連する推奨事項を含むレビューレポートを準備する。
- 運営により、きわめて非公式のものから、高度に公式なものまである。
- 主な目的は、ディスカッションすること、判断を下すこと、他の方法を評価すること、欠陥を見つけること、技術的な問題を解決すること、仕様や計画、規定、標準に準拠していることをチェックすることである。

インスペクション
- 経験を積んだモデレータ(作成者ではなく)が進行を主導する。
- 同僚によるチェックが一般的である。
- 各人の役割が決まっている。
- メトリクスの収集が含まれる。
- 規則やチェックリストを使い、形式に基づいたプロセスで進める。
- ソフトウェア成果物の受け入れに対する開始、終了基準を決める。
- インスペクション前の準備が必要である。
- インスペクションのレポートや指摘一覧を書く。
- 形式の決まったフォローアッププロセスがある。
  - プロセス改善が含まれる場合がある。
- ドキュメントを読みあげる人を加えたりすることもある。
- 主な目的は欠陥の検出である。

ウォークスルー、テクニカルレビュー、インスペクションは、仲間内(すなわち、同じ組織レベルにおける同僚間)で実行することができる。このタイプのレビューは「ピアレビュー」と呼ばれている。

3.2.4.　レビューを成功させるために(K2)

レビューの成功要因は、以下のようなものがある。
- レビュー開始前に目的を明確にする。
- レビューの目的にふさわしい人に参加してもらう。
- テスト担当者は、レビューに寄与する価値あるレビューアであり、また、製品について学ぶことで、より早いテスト準備を可能にする。
- 欠陥を見つけることは目的に合致しており、歓迎すべきことである。
- 人の問題や心理的な要素も扱う(作成者にとってそれもプラスの経験にする)。
- レビューは信頼の下で実施される。その結果は参加者の評価に使用しない。
- 目的を達成するために、対象のソフトウェアやレビューアのタイプやレベルに最適のレビュー技術を適用する。
- 欠陥を効率よく見つけられるのであれば、チェックリストを使ったり、各人の役割を決めたりする。
- インスペクションのように、公式性が高いレビューの場合、レビュー実施の技術教育を実施する。
- きちんとレビューのプロセスを進めるには、管理者の支援が必要(例えば、プロジェクトのスケジュールにレビュー時間をあらかじめ計画するなど)である。
- レビューの方法を学習したり、プロセスを改善したりすることも目的である。

3.3.　ツールによる静的解析(K2)　学習時間:20 分

用語

コンパイラ、複雑度、制御フロー、データフロー、静的解析

背景

静的解析の目的は、ソースコードやソフトウェアのモデルの欠陥を見つけることである。動的テストでは、対象ソフトウェアを実行するが、静的解析では、実行せずにツールで分析する。静的解析では、動的テストで摘出が難しい欠陥を見つけることができる。レビューと同様に、静的解析は故障よりも欠陥を検出する。静的解析ツールは、プログラムコード(例えば、制御フローやデータフロー)だけでなく、HTML、XML 等も解析する。

静的解析の効果は以下のとおり。

- テスト実施前の早い段階に欠陥を摘出する。
- 複雑度の計測などのメトリクスを測定することにより、欠陥が潜む可能性のあるコードや設計を早期に警告する。
- 動的テストでは簡単に見つからない欠陥を摘出する。
- リンクのように、ソフトウェアモデル間の依存性、非依存性を見つける。
- コードや設計の保守性を改善する。
- 開発中に学習することで欠陥を予防する。

静的解析ツールで摘出できる代表的な欠陥は以下のとおり。

- 値を定義していない変数の参照
- モジュールやコンポーネント間のインターフェース不整合
- 定義はしてあるが使わない、または不適切に宣言されている変数
- 到達できないコード(デッドコード)
- 欠落した、誤ったロジック(潜在的な無限ループ)
- 過度に複雑な構造
- コーディング規約の違反
- セキュリティの脆弱性
- コードやソフトウェアモデルの構文違反

静的解析ツールは、コンポーネントテストや統合テストの実施前や実施中に、あるいは、構成管理ツールにコードをチェックインする際に、開発担当者が使う(あらかじめ定義したルールやコーディング規約に違反しているかをチェックする)。また、ソフトウェアモデルの設計の過程で設計者が使う。静的解析ツールを使うと、大量の警告メッセージが出るので、ツールを有効に活用するには、警告メッセージの取り扱いにも注意する必要がある。コンパイラの中には、メトリクスの計算など、静的解析を支援するものもある。

参考文献

3.2 IEEE 1028

3.2.2 Gilb, 1993, van Veenendaal, 2004

3.2.4 Gilb, 1993, IEEE 1028

3.3 Van Veenendaal, 2004

4. テスト設計技法(K4)　学習時間 285 分

本章の学習の目的

本章で記述する学習内容は以下のとおり。

4.1 テスト開発プロセス(K3)

LO-4.1.1 テスト設計仕様、テストケース仕様、テスト手順仕様の違いを明確化する。(K2)

LO-4.1.2 テスト条件、テストケース、テスト手順の用語を比較する。(K2)

LO-4.1.3 テストケースの質を、要件と期待結果との間の明確なトレーサビリティの観点からチェックする。(K2)

LO-4.1.4 テストケースを構造化されたテスト手順仕様に変換し、テスト担当者に必要な知識レベルで詳細化する。(K3)

4.2 テスト設計技法のカテゴリ(K2)

LO-4.2.1 仕様ベース(ブラックボックス)と構造ベース(ホワイトボックス)のテスト設計技法がどちらも有効である理由を認識し、それぞれの一般的な技法をリストアップする。(K1)

LO-4.2.2 仕様ベースのテスト、構造ベースのテスト、経験ベースのテストの特徴と共通点、および違いを説明する。(K2)

4.3 仕様ベース/ブラックボックスのテスト技法 (K3)

LO-4.3.1 同値分割法、境界値分析、デシジョンテーブル、状態遷移図/表を用いて、ソフトウェアモデルからテストケースを作成する。(K3)

LO-4.3.2 上記の各テスト技法の目的を理解し、どんなレベルのどんなテストで使用でき、カバレッジをどのように計測するかを説明する。(K2)

LO-4.3.3 ユースケーステストの概念や利点を説明する。(K2)

4.4 構造ベース/ホワイトボックスのテスト技法 (K4)

LO-4.4.1 コードカバレッジの概念、価値を説明する。(K2)

LO-4.4.2 ステートメントカバレッジとデシジョンカバレッジの概念を説明し、この概念は、コンポーネントテスト以外のテストレベル(例えば、システムテストレベルでのビジネスプロセス)でも使える理由を説明する。(K2)

LO-4.4.3 ステートメントテスト、デシジョンテストのテスト設計技法を使い、制御フローのテストケースを作成する。(K3)

LO-4.4.4 定義された終了基準を考慮し、完全性のために、ステートメントカバレッジとデシジョンカバレッジを評価する。(K4)

4.5 経験ベースのテスト技法 (K2)

LO-4.5.1 よくある欠陥を摘出するために、直感、経験、知識をベースにしてテストケースを作る理由を考える。(K1)

LO-4.5.2 経験ベースのテスト技法を仕様ベースの技法と比較する。(K2)

4.6 テスト技法の選択 (K2)

LO-4.6.1 テストベース、それぞれのモデル、およびソフトウェア特性の目的にあった特定の背景に応じたテスト設計

技法を分類する。(K2)

4.1.　テスト開発プロセス（K3）　学習時間：15 分

用語

テスト設計、テストケース仕様、テスト実行スケジュール、テスト手順仕様、テストスクリプト、トレーサビリティ

背景

テスト開発プロセスは、ドキュメントをほとんど作らない非常に非形式的なものから、きわめて形式的なもの(詳細は本章で記述)まで、いろいろな方法で実施する。形式性のレベルは、テストの状況、例えば、テストや開発プロセスの成熟度、時間的な余裕、安全性や法規制への要求、人員に依存する。

テスト分析の期間中、何をテストするか決定するため、すなわち、テスト条件を決めるために、テストのベースとなるドキュメントを分析する。テスト条件は、一つ、あるいは、複数のテストケースで検証できる項目やイベントとして定義する。(例えば、機能、トランザクション、品質特性、構造的要素など)

テスト条件から仕様や要件へのトレーサビリティを確立できると、要件を変更したとき、テストによる要件のカバレッジと効果的な影響度分析の両方が可能となる。テスト分析では、テストへの具体的なアプローチの詳細は、いろいろな要素で決まるが、中でも、対策が必要なリスクに大きく影響を受ける。(リスク分析の詳細は、第 5 章を参照)

テストケースやテストデータは、テスト設計にて考え出し、仕様化する。テストケースは、あるテスト目的、あるいはテスト条件を網羅するために定義したものであり、構成する要素として、入力データ群、実行のための事前条件、期待結果、実行後の条件(事後条件)がある。[Standard for Software Test Documentation (IEEE Std 829-1998)]では、テスト設計仕様(テスト条件を含む)、テストケース仕様の内容を記述している。

期待結果は、テストケース仕様の一部として作成する必要があり、そこには、出力、データや状態の変化、その他、テストにより発生する結果を記す。期待する結果を定義していないと、間違っているのにもっともらしい結果が出た場合に、正しいと誤認する可能性がある。期待する結果は、テストの実行前に定義することが望ましい。

テストケースは、テスト実装にてテスト手順仕様(IEEE Std 829-1998)として開発し、実装し、優先順位付けし、実行する順番に並べかえる。テスト手順は、テストを実行する場合の一連の手順を定義したものである。ツールを使ってテストを実行する場合、実行手順は、テストスクリプト(自動実行手順)として定義する。

いろいろなテスト手順や、自動実行のテストスクリプトをまとめて、一つのテスト実行スケジュールとする。テスト実行スケジュールでは、様々なテスト手順や自動実行のテストスクリプトを実施する順番を決める。テスト実行スケジュールは、回帰テスト、優先順位、技術的、論理的な依存性も考慮して決める。

4.2.　テスト設計技法のカテゴリ（K2）　学習時間：15 分

用語

ブラックボックステスト設計技法、経験ベースのテスト設計技法、テスト設計技法、ホワイトボックステスト設計技法

背景

テスト設計技法の目的は、テスト条件とテストケース、テストデータを決定することである。

テスト技法の古典的な分類法として、ブラックボックスとホワイトボックスがある。ブラックボックステスト設計技法（仕様ベースの技法と呼ぶこともある）は、テストベースドキュメントの分析に基づき、テスト条件やテストケース、テストデータを導き出し、選択する方法である。これには、機能テストと非機能テストの両方が含まれる。ブラックボックステストは、定義によれば、テストするコンポーネントやシステムの内部構造についての情報を使用しない。一方、ホワイトボックステスト設計技法（構造技法、構造ベースの技法と呼ぶこともある）は、コンポーネントやシステムの内部構造の分析に基づく。ブラックボックステストとホワイトボックステストは、開発者、テスト担当者、ユーザの経験を活用して何をテストすべきか見つけるために、経験ベースのテストと組み合わせて利用することもある。

テスト技法の中には、どちらかのカテゴリに納まるものもあるが、どちらの要素も備えた技法もある。

本シラバスでは、ブラックボックス技法として仕様ベースのテスト設計技法を、そしてホワイトボックス技法として構造ベースのテスト設計技法について述べている。更に、経験ベースのテスト設計技法をカバーしている。

仕様ベースのテスト設計技法に共通する特徴は以下のものを含む。

- 対象となる問題、ソフトウェア、コンポーネントを定義する場合、形式的、非形式的に関わらず、モデルを使用する。
- モデルから、体系的にテストケースを導き出す。

構造ベースのテスト設計技法に共通する特徴は以下のものを含む。

- ソフトウェアの構成に関する情報（すなわち、コードや詳細化された設計情報）を基に、テストケースを導き出す。
- テストケースを基に、ソフトウェアの構造に関するカバレッジを計測し、カバレッジを上げるためのテストケースを体系的に導き出せる。

経験ベースのテスト設計技法に共通する特徴は以下のものを含む。

- 担当者の知識や経験をベースにテストケースを導き出す。
- テスト担当者、開発担当者、ユーザ、その他のステークホルダのソフトウェアに関する知識、使用法、使用環境は、一つの情報源である。
- 類似した欠陥や、存在する箇所に関する知識は、もう一つの情報源である。

4.3.　仕様ベース/ブラックボックスのテスト技法（K3）　学習時間：150 分

用語

境界値分析、デシジョンテーブルテスト、同値分割法、状態遷移テスト、ユースケーステスト

4.3.1.　同値分割法（K3)

同値分割法は、ソフトウェアやシステムへの入力を同じ処理をするグループに分割し、グループ内の入力を同等に扱えるようにする技法である。同値分割したグループ(あるいは、同値分割したクラス)は、有効データ、すなわち受け入れられるデータだけでなく、無効データ、すなわち拒否されるデータにもある。同値分割したグループは、出力、内部変数、時間に依存する値(例えば、イベントの前と後)にも存在し、インターフェースパラメータにもある。(例えば、テスト済みの統合されたコンポーネント)テストでは、同値分割した全ての有効な領域、無効な領域をカバーするように設計する。同値分割法は、あらゆるレベルのテストで適用できる。

同値分割法は、入出力のカバレッジ目標を達成する場合に使用できる。また、人間による入力、インターフェース経由でのシステムへの入力、統合テストでのインターフェースパラメータに適用できる。

4.3.2.　境界値分析(K3)

同値分割したグループの境界上の動作は、グループ内部での(境界ではない)動作よりも正しくないことが多く、境界には多くの欠陥が潜んでいる可能性が大きい。ある領域の最大値、最小値は境界値である。同値分割した場合の有効な領域側の境界値は、「有効な境界値」となり、無効な領域側の境界値は「無効な境界値」となる。有効および無効な境界値の両方をカバーするようにテストを設計する。テストケースを設計するときは、両領域から値を選ぶ。
境界値分析は、あらゆるテストレベルで適用できる。比較的簡単に適用でき、欠陥摘出能力も高い。詳細化された仕様は、重要な境界値を決定するために役に立つ。

この技法は、同値分割法の拡張、または他のブラックボックステスト設計技法と見なされることが多い。人間による画面入力や、例えば、タイミング(タイムアウト、トランザクション速度に対する要求など)やテーブルの範囲(テーブルサイズは 256×256 であるなど)にも使用できる。

4.3.3.　デシジョンテーブルテスト(K3)

デシジョンテーブルは、論理的な条件を含むシステム要件を把握する場合に有効である。また、システムの内部設計をドキュメント化する場合に有効である。デシジョンテーブルは、構築するシステムの複雑なビジネスルールを記録するために使用することがある。デシジョンテーブルを作り出す際には、仕様を分析し、システムの条件や動作を決める。入力条件や動作は、大部分の場合、真か偽(ブール値)で表現する。デシジョンテーブルは、トリガーとなる条件(全ての入力条件の真・偽の組み合わせになることが多い)、条件の各組み合わせに対する処理で構成される。デシジョンテーブルの各列は、特定の条件の組み合わせを定義するビジネスルールと、ルールに対応した行動から構成される。デシジョンテーブルテストにおけるカバレッジは一般的にはそのテーブルにおいて一つの列が最低一回は実行されたこと、すなわち、トリガーとなる組み合わせの全てを網羅することを意味する。
デシジョンテーブルテストの利点は、テストでなければありえない条件の組み合わせを実施できることである。デシジョンテーブルテストは、ソフトウェアの動作が、いくつかの論理的な判定に依存する場合、全ての状況に適用可能であ

る。

4.3.4. 状態遷移テスト(K3)

現在の条件や前の条件、システムの状態により、システムは全く異なる動作をする。この場合システムの動作は状態遷移図で表現できる。これにより、テスト担当者は、ソフトウェアを現在の状態、状態間の遷移、状態を変える(遷移する)入力やイベント、状態遷移の結果起きる処理と見ることができる。テスト対象のシステムやオブジェクトの状態は、別々であり、区別ができ、ある有限個の状態となる。
状態遷移表は状態と入力の関係を示し、不正な可能性のある遷移を明確にすることができる。
状態遷移を用いると、状態の代表的な遷移を網羅したり、全ての状態を網羅したり、全ての遷移を実行したり、特定の遷移を実行させたり、不正に遷移しないことを確認したりする、テストを設計できる。

状態遷移テストは、一般に組込み系ソフトウェアやテクニカルオートメーション分野でよく使う。また、状態遷移テストの技法は特定の状態を持つビジネスプロセスをモデル化したり、画面による対話処理フロー(例えば、インターネットアプリケーションやビジネスシナリオ)をテストしたりする場合にも有効である。

4.3.5. ユースケーステスト(K2)

ユースケースをベースにしたテストもある。ユースケースはアクター(ユーザおよびシステム)とシステムとの相互作用を記述したものであり、ユーザや顧客にとって価値のある結果を説明している。ユースケースは抽象的なレベル(ビジネスユースケース、テクノロジーフリー、ビジネスプロセスレベル)で、または、システムレベル(システム機能レベルのシステムユースケース)で説明される。各ユースケースには事前条件があり、ユースケースが正常に動作するには、その条件を満足させる必要がある。また、各ユースケースは事後条件になると終わる。事後条件は、ユースケースが完了した後の出力結果であり、システムの最終状態である。通常、ユースケースにはメインストリームとなる(すなわち最も頻繁に実行する)シナリオがある。また、代替シナリオもある。

ユースケースは、実際には、どのような使われ方が多いかをベースにして、システムの「プロセスフロー」を記述したものである。従って、ユースケースを基に設計するテストケースは、現実の世界でシステムが稼動している状態で、プロセスフローの欠陥を摘出する場合に効果がある。ユースケースは、顧客やユーザが参画する受け入れテストを設計する場合に有効である。他のコンポーネントとのやり取りやインターフェースが原因で発生する統合時の欠陥を検出するのに効果がある。この欠陥は、個別のコンポーネントテストでは見つからない。ユースケースからのテストケース設計は、他の仕様ベースのテスト技法と組み合わせることができる。

4.4.　構造ベース/ホワイトボックスのテスト技法（K4）学習時間：60 分

用語

コードカバレッジ、デシジョンカバレッジ、ステートメントカバレッジ、構造テスト、構造ベースのテスト

背景

構造ベース/ホワイトボックステストは、ソフトウェアやシステムの特定の構造に着目したテストである。特定の構造の例を以下に示す。

- コンポーネントレベル：この場合の構造はソフトウェアコンポーネントである。すなわちステートメント、判定(デシジョン)、分岐(ブランチ)、その他のパス。
- 統合レベル：この場合の構造は、コールツリー(モジュールが他のモジュールをコールするダイアグラム)である。
- システムレベル：この場合の構造は、メニューの構造、ビジネスプロセス、ウェブページの構造である。

本節では、コードの構造をベースとしたコードカバレッジのためのテスト設計技法のうち、ステートメントと判定、分岐をベースにした 3 つの技法について述べる。デシジョンテストの場合、取りうるデシジョンを視覚化するため、制御フローダイアグラムを使う場合がある。

4.4.1.　ステートメントテストとカバレッジ(K4)

コンポーネントテストでは、テストスイートが実行可能ステートメントの何パーセントを実行したか評価する場合に、ステートメントカバレッジを使う。ステートメントテスト技法では、特定のステートメント群を実行するよう(通常は、ステートメントカバレッジ率を上げるため)テストケースを設計する。

ステートメントカバレッジは、テストケースにより網羅(テスト設計、または実行)したステートメント数を全ての実行可能なステートメント数で割った数である。

4.4.2.　デシジョンテストとカバレッジ(K4)

ブランチテスト(ブランチカバレッジ)とも呼ばれるデシジョンカバレッジは、テストスイートが実行した判定の結果(例えば、IF 文の True と False)の実施パーセントを評価する。デシジョンテスト技法では、特定の判定を実行するようにテストケースを導き出す。ブランチとはもともとコードの中のデシジョンポイントからきており、コードの中の別の場所に制御を移すことを示す。

デシジョンカバレッジは、テストケースにより網羅(テスト設計、または実行)した判定数を全ての実行可能な判定数で割った数である。

デシジョンテストは、デシジョンポイントを通り、特定の制御フローに続くため、制御フローテストの一形式である。デシジョンカバレッジは、ステートメントカバレッジよりも厳しい評価基準である。100％のデシジョンカバレッジは、100％のステートメントカバレッジとなるが、逆は成立しない。

4.4.3. その他の構造ベース技法(K1)

デシジョンカバレッジより評価基準の厳しい構造カバレッジがある。例えば、条件カバレッジや複合条件カバレッジである。

カバレッジの概念は、他のテストレベルでも利用できる。例えば、統合レベルでは、テストスイートでモジュール、コンポーネント、クラスを実行したときのパーセンテージがそれぞれのカバレッジとして表現できる。

コードの構造テストではツールを使うと効果が上がる。

4.5.　経験ベースのテスト技法（K2）　学習時間：30 分

用語

探索的テスト、(フォールト)攻撃

背景

経験ベースのテストでは、テスト担当者のスキルと直感の他、同様のアプリケーションまたはテクノロジーにおける経験からテストを抽出する。これらの技法は、体系的な技法を補強したいとき、公式な技法では容易に把握できない特別なテストを識別するのに役に立つ。特に公式なアプローチの後に適用すると有効であるが、経験ベースの技法は、テスト担当者の経験に大きく依存するため、効率には非常に大きなバラツキがある。

エラー推測は、よく使われる経験ベースの技法である。一般的にテスト担当者は、経験に基づいて欠陥を予測している。エラー推測の構造的なアプローチの一つに、発生しうる欠陥をリストアップし、その欠陥を攻撃するためのテストケースを設計するやり方がある。こうした体系的アプローチをフォールト攻撃という。この欠陥や故障のリストは、経験、入手可能な欠陥や故障のデータや、ソフトウェアの故障に対する一般知識から作成する。

探索的テストは、テスト目的が含まれたテストチャータを基にしたもので、テスト設計、テスト実行、テスト記録や学習を並行して同じ時間枠内で実行する。このアプローチは、仕様がほとんどなかったり、不十分であったり、スケジュール的な余裕がない場合や、他の形式的なテスト技法を補完する場合に効果が大きい。探索的テストは、テストプロセスのチェックや、きわめて重大な欠陥を見つけ出すのに役に立つ。

4.6.　テスト技法の選択（K2）　学習時間：15 分

用語

なし

背景

どのテスト技法を選ぶかは、システムの種類、規則や標準、顧客や契約上の要件、リスクのレベルや種類、テスト目的、入手可能なドキュメント、テスト担当者の知識、時間と予算、開発ライフサイクル、ユースケースモデル、様々な種類の欠陥を摘出した過去の経験など多くの要素に依存する。
状況やテストレベルによっては、あるテスト技法を適用するのが良い場合もあるし、全てのテストレベルで使える技法もある。

通常、テスト担当者はテストケースを作成する際に、テスト対象を適切に網羅するために、テスト技法(プロセス、規則、およびデータ駆動テスト技法を含む)を組み合わせて使う。

参考文献

4.1 Craig, 2002, Hetzel, 1988, IEEE Std 829-1998
4.2 Beizer, 1990, Copeland, 2004
4.3.1 Copeland, 2004, Myers, 1979
4.3.2 Copeland, 2004, Myers, 1979
4.3.3 Beizer, 1990, Copeland, 2004
4.3.4 Beizer, 1990, Copeland, 2004
4.3.5 Copeland, 2004
4.4.3 Beizer, 1990, Copeland, 2004
4.5 Kaner, 2002
4.6 Beizer, 1990, Copeland, 2004

5. テストのマネジメント(K3)　学習時間:170 分

本章の学習の目的

本章で記述する学習内容は以下のとおり。

5.1 テスト組織(K2)

LO-5.1.1 独立した組織によるテストの重要性を認識する。(K1)

LO-5.1.2 開発部門と同じ組織内に独立した組織によるテストチームがある場合の利点、欠点を説明する。(K2)

LO-5.1.3 テストチームを組織する場合、いろいろなメンバーを入れることを認識する。(K1)

LO-5.1.4 テストリーダとテスト担当者の一般的な作業内容を認識する。(K1)

5.2 テスト計画作業と見積り(K3)

LO-5.2.1 テスト計画作業には、いくつかのレベルや目的があることを認識する。(K1)

LO-5.2.2 [Standard for Software Test Documentation(IEEE Std 829-1998)]に準拠したテスト計画、テスト設計仕様、テスト手順のドキュメント内容をまとめる。(K2)

LO-5.2.3 分析的、モデルベース、方法論的、プロセス・標準準拠、動的・ヒューリスティック、コンサルテーションベース、回帰的など、概念的に異なるテストアプローチの違いを明確にする。(K2)

LO-5.2.4 システムのためのテスト計画と単なるテスト実行スケジュールとの違いを明確化する。(K2)

LO-5.2.5 優先順位や、テスト間の技術的、論理的依存性を考慮し、テストケースの実行スケジュールを立てる。(K3)

LO-5.2.6 テスト計画作業で考慮すべきテスト準備活動やテスト実行活動をリストアップする。(K1)

LO-5.2.7 テスト関連作業に影響を与える要素を再確認する。(K1)

LO-5.2.8 メトリクスをベースにしたアプローチと、経験をベースにしたアプローチという概念的に異なる見積りアプローチの違いを明確にする。(K2)

LO-5.2.9 特定のテストレベルやテストケースのグループにおける開始基準および終了基準が適切かを見分け、その正当性を説明する。(例えば、統合テスト、受け入れテストなどのテストレベルや使用性テストのためのテストケースのグループなど)(K2)

5.3 テスト進捗のモニタリングとコントロール(K2)

LO-5.3.1 テストの準備や実行をモニタリングするメトリクスを再認識する。(K1)

LO-5.3.2 テストのレポートやテストのコントロール(例えば、摘出、修正した欠陥の数、合格・不合格のテストケース数)でのメトリクスを目的と活用に関係させて説明し、比較する。(K2)

LO-5.3.3 [Standard for Software Test Documentation(IEEE Std 829-1998)]に準拠したテストサマリレポートの目的や内容をまとめる。(K2)

5.4 構成管理(K2)

LO-5.4.1 構成管理がテストをどのように支援するかまとめる。(K2)

5.5 リスクとテスト(K2)

LO-5.5.1 ある問題により、ステークホルダのプロジェクト目的を遂行できなくなる可能性があるという観点でリスクを考

える。(K2)

LO-5.5.2 リスクのレベルは、(発生する)確率と、(実際に発生した場合の被害の)インパクトにより決まることを再認識する。(K1)

LO-5.5.3 プロジェクトリスクとプロダクトリスクの違いを認識する。(K1)

LO-5.5.4 代表的なプロジェクトリスクとプロダクトリスクを認識する。(K1)

LO-5.5.5 テスト計画作業で、リスク分析やリスクマネジメントをどのように使うのか例を挙げて説明する。(K2)

5.6 インシデント管理(K3)

LO-5.6.1[Standard for Software Test Documentation(IEEE Std 829-1998)]のインシデントレポートの内容を認識する。(K1)

LO-5.6.2 テスト期間中に見つけた故障のインシデントレポートを書く。(K3)

5.1.　テスト組織(K2)　学習時間:30 分

用語

テスト担当者、テストリーダ、テストマネージャ

5.1.1.　テスト組織と独立性(K2)

テストやレビューで欠陥を摘出する場合、独立したテスト担当者を使うと効率が上がる。独立した組織によるテストとして以下の形態を含む。

- 独立したテスト担当者がいない。開発者が自ら自分のコードをテストする。
- 同じ開発プロジェクト内の独立したテスト担当者
- 同じ組織内にある独立したテストチームまたはグループで、プロジェクト管理者や上位管理者の直属組織
- 顧客、ユーザコミュニティから派遣された独立したテスト担当者
- 使用性、セキュリティ、標準準拠(ソフトウェアが標準や規程に準拠していることを認定する)など、ある特定のテストタイプを行う独立したテストのスペシャリスト
- 組織外の独立したテストチーム

一般に、複雑なソフトウェアや安全性が最優先のソフトウェアを開発するプロジェクトでは、複数レベルのテストを実施し、そのレベルのいくつか、あるいは、全てを独立したテストチームが担当するのがよい。(特に下位のテストレベルでは)開発スタッフもテストに参画するが、客観性が欠如しているために、その有効性には限界がある。独立したテストチームは、テストプロセスを要求したり、定義したりする権利があるが、プロセスに関連する事柄を決める場合は、上位管理者からの明確な要請を基に実施すべきである。

独立したテストの利点は以下のとおり。

- 先入観がないため、異なる欠陥を見つけられる。
- システムの仕様策定中や実装中に想定通りかを検証できる。

独立したテストの欠点は以下のとおり。

- 開発チームから隔絶されている。(完全に独立の場合)
- 開発担当者の品質に対する責任感が薄れる。
- 独立したテストチームは、ボトルネックと見られたり、リリース遅延のために非難されたりしがちである。

テストは、ある特定のテスト作業を担当する人が実施してもよいし、それ以外の作業の担当者、例えば、プロジェクト管理者、品質管理者、開発担当者、対象ビジネスドメインのエキスパート、システム内部のオペレータが実施してもよい。

5.1.2.　テストリーダとテスト担当者の作業(K1)

本シラバスでは、テストリーダとテスト担当者という 2 つの職務を解説している。この 2 つの職務でどんな活動や作業をするかは、プロジェクトやソフトウェアの背景、各職務を担当する人や組織により決まる。
テストリーダは、テストマネージャまたはテストコーディネータと呼ぶこともある。テストリーダの職務は、プロジェクト管理者、開発管理者、品質保証管理者、テストグループの管理者などが行う。大きなプロジェクトでは、テストリーダとテストマネージャという 2 つ職務が存在する場合がある。テストリーダは一般に、テストを計画し、テスト活動をモニタリング、コントロールし、第 1.4 節で記述した作業を実施する。

テストリーダの代表的な作業は以下のとおり。

- テスト戦略とテスト計画をプロジェクト管理者やその他の関係者と調整する。
- プロジェクトのテスト戦略や組織のテストポリシーを策定したりレビューしたりする。
- テストの観点から、プロジェクトの他の活動(例えば、統合計画など)に貢献する。
- プロジェクトの背景を考慮し、テスト目的とリスクを理解してテストを計画する。これにはテストアプローチの選択、テストにかかる時間、工数、コストの見積り、資源の獲得、テストレベルやテストサイクルの定義、インシデント管理の計画を含む。
- テストの仕様化、準備、実装、実行を開始し、テスト結果の測定と終了基準を満たしていることの確認を行う。
- テスト結果やテストの進捗に基づいて計画を修正し(状況報告の中に記載することが多い)、問題を補正するために必要なあらゆる対策を取る。
- トレーサビリティを向上させるため、テストウェアの十分な構成管理をセットアップする。
- テストの進捗を計測したり、テストや対象ソフトウェアの品質を検証するため、適切なメトリクスを導入したりする。
- どんな作業をどの程度、どのように自動化するかを決定する。
- テストを支援するツールを選び、テスト担当者が使えるようにトレーニングを企画する。
- 構築するテスト環境を決定する。
- テスト期間中に収集した情報をベースにテストレポートを書く。

テスト担当者の代表的な作業は以下のとおり。

- テスト計画に対してレビューによって貢献する。
- 試験性の観点で、要件、仕様、モデルを分析し、レビューし、評価する。
- テスト仕様を作成する。
- テスト環境(システム制御やネットワーク管理と協調する場合が多い)をセットアップする。
- テストデータを作成したり、入手したりする。
- 全てのテストレベルのテストを実装し、テストを実行して記録を取り、テスト結果を評価し、期待した結果との差異を文書化する。
- 必要に応じて、テストコントロールツールやマネジメントツール、モニタ用ツールを使う。
- テストを自動化する。(これには、開発担当者や、テストの自動実行のエキスパートの支援が必要)
- 必要ならば、コンポーネントやシステムの性能を計測する。
- 他の人が実施したテストをレビューする。

テストの分析、テスト設計、特殊なテストの実施、テストの自動化を担当する人は、各分野のスペシャリストである。テストレベルや、プロダクトまたはプロジェクトのリスクの状況により、テスト担当者の職務は、ある程度の独立性を確保しながら別の人が担当する場合がある。コンポーネントテストや統合テストレベルでは開発担当者がテストを実施し、受け入れ段階ではビジネスエキスパートやユーザが担当し、運用受け入れテストはオペレータが実施することもある。

5.2.　テスト計画作業と見積り(K3)　学習時間:40 分

用語

テストアプローチ、テスト戦略

5.2.1.　テスト計画作業(K2)

本節では、開発、実装プロジェクトや保守の活動におけるテスト計画作業の目的を記述する。テストの計画は、マスターテスト計画に文書化し、そしてシステムテストや受け入れテストといったテストレベルに応じた個別のテスト計画に分割して文書化する。テスト計画書の概要は、[Standard for Software Test Documentation(IEEE Std 829-1998)]を参照。

テスト計画作業は、組織のテストポリシー、テストの範囲、目的、リスク、制限、緊急度、試験性、使えるリソースなどの影響を受ける。プロジェクトやテストの計画を進めていくと、より多くの情報が利用可能になり、テスト計画の内容がより具体化する。

テスト計画作業は、連続的な活動であり、全てのライフサイクルプロセスや活動で実施する。テスト活動からのフィードバックにより、リスクの変化を認識し計画を調整できるようにする。

5.2.2.　テスト計画策定(K3)

システム全体、またはシステムの一部に対するテスト計画の活動には以下のものがある。

- スコープとリスクを特定し、テストの目的を識別する。
- テストに対する包括的なアプローチを定義する。これには、テストレベル、開始基準、終了基準などの定義も含む。
- ソフトウェアライフサイクル(取得、供給、開発、運用、保守)での活動にテスト活動を統合し、協調させる。
- 何をテストするか、テスト活動でどのような職務を実行するか、いつどのようにテスト活動を進めるか、テスト結果をどのように評価するか、いつテストを終わらせるか(終了基準)を決める。
- テスト分析と計画の活動をスケジューリングする。
- テスト実装と実行、テスト結果の評価にかかる活動をスケジューリングする。
- 定義したいろいろな活動にリソースを割り当てる。
- テストドキュメントの量、詳細度、構成、テンプレートなどを決める。
- テストの準備、実行、欠陥の解決、リスク問題のモニタリングやコントロールをするためのメトリクスを選ぶ。
- テストの準備や実行の再現性を確保するのに十分な情報を得るためにはテスト手順をどの程度詳細化すればよいか、を決める。

5.2.3.　開始基準(K2)

開始基準は、いつそのテストレベルを始めるかまたはいつテストセットの実行準備ができるかといったように、テストの開始を定義することである。

代表的な開始基準は、次のものである。

- テスト環境の可用性と準備
- テスト環境におけるテストツールの準備
- テスト可能なコードの可用性
- テストデータの可用性

5.2.4. 終了基準(K2)

終了基準は、いつそのテストレベルを終了するか、またはいつテストセットがあらかじめ決められたゴールに達するかといったように、テストの終了を定義することである。

代表的な終了基準は、次のものである。

- コード、機能性、リスクのカバレッジのような、徹底度
- 欠陥密度や信頼性の見積値
- コスト
- 残存リスク、例えば、未修正の欠陥や、特定領域のカバレッジ不足など
- 出荷時期などのスケジュール

5.2.5. テスト見積り(K2)

テスト工数見積りのアプローチは次の 2 つがある。

- メトリクスをベースにしたアプローチ：以前のプロジェクトや類似プロジェクトのメトリクスや、代表的な値をベースにテスト工数を見積る。
- 経験をベースにしたアプローチ：作業の実行者やエキスパートによる見積りをベースにテスト工数を見積る。

テスト工数の見積りが終了すると、必要となるリソースを決定でき、スケジュールを立てることができる。
テスト工数に影響を与える要素として、以下のものがある。

- プロダクトの特性：例えば、テストモデル(すなわち、テストベース)として使用する仕様やその他の情報の品質、プロダクトのサイズ、問題のドメインの複雑性、信頼性やセキュリティに対する要件、ドキュメンテーションに対する要件。
- 開発プロセスの特性：組織の安定度、使用するツール、テストプロセス、関係者のスキル、時間のプレッシャーなど。
- テストの出力結果：欠陥の数、修正に要する工数など。

5.2.6. テスト戦略、テストアプローチ(K2)

テストアプローチは、特定のプロジェクトにおいてテスト戦略を実装することである。テストアプローチは、テスト計画とテスト設計の中で定義され、改良される。これには、一般的に(テスト)プロジェクトのゴールとリスクアセスメントに基づく決定を盛り込む。それは、テストプロセスの計画、テスト設計技法の選択とテストタイプの適用、開始・終了基準の定義などを行うための出発点である。

選択されたアプローチはプロジェクトの状況と、リスク、危険性と安全性、使用可能な資源とスキル、テクノロジー、システムの性質(例えば、カスタムメイドなのか COTS なのか)、テスト目的、および法規制などの考慮によって決まる。

代表的なアプローチとして、以下のものがある。

- 分析的アプローチ：例えば、緊急度が最も高い分野を重点的にテストするリスクベースのテストなど。
- モデルベースアプローチ：故障率(例えば、信頼度成長モデル)や、使用法(例えば、運用プロファイル)に関する統計的情報を使う確率的テストなど。
- 方法論的アプローチ：フォールトをベースにしたもの(例えば、エラー推測、フォールト攻撃)。チェックリストをベースにしたもの。品質特性をベースにしたものなど。
- プロセス準拠アプローチや、標準準拠アプローチ：例えば、いろいろなアジャイル方法論や、業界固有の標準など。
- 動的で経験則に基づいたアプローチ：例えば、事前に計画するのでなく、発生した事項に対応する方法や、テストの実行と評価を同時に実行する探索的テストなど。

- コンサルテーションベースのアプローチ：テストチーム外のビジネスドメインのエキスパート、または技術的エキスパートの助言や指導を受けた部分から優先的にテストカバレッジを上げていく。
- 回帰的アプローチ：例えば、既存のテストを再利用したり、回帰テストを高度に自動化したり、標準のテストスイートを使うなど。

例えば、リスクをベースにした動的アプローチのように、違うアプローチを組み合わせることもできる。

5.3.　テスト進捗のモニタリングとコントロール(K2)　　学習時間:20 分

用語

欠陥密度、故障率、テストコントロール、テストモニタリング、テストサマリレポート

5.3.1.　テスト進捗モニタリング(K1)

テストモニタリングの目的は、テストの活動を可視化し、フィードバックをかけることにある。モニタリングする情報は、手動あるいは自動で収集し、例えばカバレッジの様に終了基準を計測するために使う。メトリクスにより、実際の進捗と、予定スケジュール、コストとを比較することもある。代表的なテストメトリクスには以下のものがある。

- テストの準備作業が完了したパーセンテージ(あるいは、作成したテストケースのパーセンテージ)
- テスト環境準備で完了したパーセンテージ
- テストケースの実行(例えば、実行／未実行のテストケース数、合格／不合格のテストケース数)
- 欠陥情報(例えば、欠陥密度、摘出した欠陥と修正した欠陥、故障率、再テストの結果)
- 要件、リスク、コードにおけるテストカバレッジ
- プロダクトの品質に対するテスト担当者の主観的な信頼度
- テスト作業のマイルストン
- テストに要するコスト、例えば、次の欠陥を摘出することや次のテストによる利益とコストの比較

5.3.2.　テストレポート(K2)

テストレポートは、以下の様にテスト作業のサマリ情報を勘案して作成する。

- テスト期間中に発生したこと、例えば、終了基準を満足した日
- 将来の方針を決めたり参考にしたりすることができるような分析情報やメトリクス、例えば、残存欠陥の評価、テストを続けることによるコスト上の効果、発生する可能性の高いリスク、テスト完了ソフトウェアに対する信頼度のレベルなど

テストのサマリレポートの概要は、[Standard for Software Test Documentation(IEEE Std 829-1998)]を参照。

メトリクスは各レベルのテスト期間中や終了時に計測し、以下の項目を評価する。

- 各テストレベルのテスト目的に対する十分性
- 採用したテストアプローチに対する十分性
- テスト目的に対する有効性

5.3.3.　テストコントロール(K2)

テストコントロールとは、収集、報告されたテストの実施結果やレポートをベースに次に進む方向を決めたり、作業を補正したりすることである。補正作業は、あらゆるテスト活動を網羅し、他のソフトウェアライフサイクルの活動や作業に影響を与える。

テストコントロールの作業には、例えば以下のものがある。

- テストモニタリングの情報をベースに意思決定をする。
- あるリスクが発生した場合(例えば、スケジュール遅延)、テストの順位を見直す。
- テスト環境の利用の可不可状況により、テストスケジュールを変更する。
- 修正をビルドに反映する前に開発者自身による再テスト(確認テスト)を実施することを開始基準とする。

5.4.　構成管理(K2)　学習時間:10 分

用語

構成管理、バージョンコントロール

背景

構成管理の目的は、プロジェクトやプロダクトのライフサイクルを通じて、ソフトウェアのプロダクト(コンポーネント、データ、ドキュメント)やシステムの完全性を確立し、維持することである。
テストの場合、構成管理では、以下を確認する。

- テストウェアの全アイテムを識別し、バージョンコントロールをし、変更履歴を取り、各アイテム間、あるいは、開発アイテム(テスト対象)との関連付けがされ、テストプロセスを通してトレーサビリティを維持できる。
- 全てのドキュメントやアイテムを識別し、テストドキュメントに明確に記載してある。

テスト担当者にとって、構成管理は、テストアイテム、テストドキュメント、テストケースやテストハーネスを一意に識別し、再作成する場合の強力なツールとなる。

テスト計画作業では、構成管理の手順や仕組み(ツール)を選択し、文書化し、実行に移す必要がある。

5.5.　リスクとテスト(K2)　学習時間:30 分

用語

プロダクトリスク、プロジェクトリスク、リスク、リスクベースドテスト

背景

リスクとは、イベント、危険、脅威、発生する状況等の見込みと、その望ましくない結果、あるいは可能性のある問題と定義できる。リスクのレベルは、不利なイベントが起きる見込みとそのインパクト(イベントによる結果がもたらす損害)で決まる。

5.5.1.　プロジェクトリスク(K2)

プロジェクトリスクはプロジェクトを遂行する際に影響を与えるリスクである。例えば、以下のようなものがある。

- 組織問題の要素
  - スタッフのスキル、およびトレーニング不足、人員不足
  - 人事の問題
  - 以下のような政治的な問題
    - □ テスト担当者のニーズやテスト結果が上手く伝わらない。
    - □ テストやレビューで見つかった事項がチームで上手くフォローアップできていない。(例えば、開発やテストの実施方式が改善されていない)
  - テストから期待できるものを正しく評価しようとしない。(例えば、テストで見つかった欠陥の情報を真摯に受け止めようとしない)
- 技術的な問題
  - 要件を正しく定義できない
  - 制約があるために、要件を拡張できない
  - テスト環境が予定通りに用意できない
  - データ変換の遅れ、移行計画の遅れ、データ変換／移行ツールの開発・テストの遅れ
  - 設計、コード、構成データ、テストデータ、テストの低い品質
- 供給者側の問題
  - サードパーティ製品の故障
  - 契約上の問題

リスクを分析し、マネジメントし、軽減する場合、テストマネージャは、きちんと確立したプロジェクト管理のルールに従う。[Standard for Software Test Documentation(IEEE Std 829-1998)]のテスト計画の概要には、リスクやリスクの回避策を記述する旨をうたっている。

5.5.2.　プロダクトリスク(K2)

ソフトウェアやシステムで失敗する可能性のある領域(次に起きる事象が意に沿わなかったり、危険を引き起こしたりする可能性)をプロダクトリスクと呼ぶ。これらは、プロダクトの品質に対するリスクであり、以下のものが含まれる。

- 故障の起きやすいソフトウェアの出荷
- ソフトウェアやハードウェアが個人や会社に対し損害を与える可能性
- 貧弱なソフトウェア品質特性(例えば、機能性、セキュリティ、信頼性、使用性、性能など)
- 貧弱なデータの完全性と品質(例えば、データ移行問題、データ変換問題、データ伝送問題、データ標準へ

の違反）
- 予定の機能が動作しないソフトウェア

どの部分をテストするか、更なるテストが必要なのはどの部分かを決める場合にプロダクトリスクを使う。一方、期待しないイベントが発生するリスクを減らしたり、その影響を少なくしたりするためにテストを使う。

プロダクトリスクは、プロジェクトを成功させる上で特別のタイプのリスクである。リスクコントロールとしてテストを扱うと、重大な欠陥の除去の状況やコンティンジェンシープランによる効果の計測より得られる残存リスクのフィードバックを行うことができる。
テストで、リスクをベースにしたアプローチを取ると、プロジェクトの初期段階からプロダクトリスクのレベルを減らす予防的措置を取れる。このアプローチは、プロダクトリスクを一つずつ洗い出すこと、テスト計画作業のガイダンスとして適用すること、テストの仕様を決め、準備し、実行することなどの作業がある。リスクベースのアプローチでは、洗い出したリスクを以下のように使う。
- 適用するテスト技法を決める。
- テストを実行する範囲を決める。
- 重大な欠陥をなるべく早い時期に検出するため、テストの優先順位を決める。
- テスト以外でリスクを減らす方法があるか検討する。（経験の少ない設計者に教育を実施するなど）

リスクベースドテストでは、リスクとそのリスクを扱う場合に必要なテストレベルを決める上で、プロジェクトのステークホルダの総合的な知識や洞察力を必要とする。

プロダクトが故障を起こす可能性を最小にするため、リスクマネジメントでは、以下のように、きちんと決まったアプローチを備えている必要がある。
- どこが上手くいかないか(リスク)を評価する。（定期的に再評価する）
- リスクの重要性を決める。
- 重大なリスクを処理する方法を作る。

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

5.6. インシデント管理(K3)　学習時間:40 分

用語

インシデントログ、インシデント管理、インシデントレポート

背景

テストの目的の一つが欠陥を摘出することであるため、実際のテスト結果と期待する結果に食い違いがあれば、インシデントとして記録を残す必要がある。インシデントは調査すべきであり、それが欠陥だとわかることがある。インシデントと欠陥を処理するための適切な活動を定義する必要がある。インシデントと欠陥は、検出時点から分類、修正、最終確認まで、きちんと追跡しなければならない。全てのインシデントが完了するまでマネジメントするために、組織でインシデント管理のプロセスを定め、分類のための規則を設ける必要がある。

インシデントは、開発中、レビュー、テスト、稼動中に現れる。インシデントは、コードや稼動システムの中、あるいは、要件、開発ドキュメント、テストドキュメント、ヘルプやインストールガイド等のユーザ情報など、あらゆる種類のドキュメントで問題となる。

インシデントレポートは以下のような目的がある。

- 開発担当者やその関係者に対し、必要に応じて問題を特定、抽出、解決できるようフィードバックをする。
- テストリーダに対し、テスト実行中のシステムの品質や、テストの進捗を追跡する手段を提供する。
- テストプロセス改善のためのヒントを提供する。

インシデントレポートに、以下の詳細情報を記入する。

- インシデントログの作成日付、作成した組織、作成者
- 期待した結果と実行結果
- テストアイテム(構成アイテム)および環境の識別
- インシデントが発生したソフトウェアやシステムのライフサイクルプロセス
- インシデントログ、データベースのダンプ、スクリーンショットなどのインシデントの再現および解決を可能にする詳細な記述
- 対応するステークホルダに与えるインパクトの範囲や程度
- システムに与えるインパクト
- 修正の緊急度／優先順位
- インシデントのステータス(例えば、オープン、次バージョンへ延期、重複、修正待ち、再テスト待ち、解決済みなど)
- 結論、アドバイス、承認
- 外部との問題(例えば、インシデントを修正することで他の分野に及ぼす影響)
- 修正履歴(例えば、インシデントを抽出し、修正し、修正が正しいことを確認するときにプロジェクトチームのメンバーが実施した作業など)
- 問題を摘出したテストケース仕様の ID 番号など参照情報

インシデントレポートの構成は、[Standard for Software Test Documentation(IEEE Std 829-1998)]を参照。

◆ 参考文献

5.1.1 Black, 2001, Hetzel, 1988

5.1.2 Black, 2001, Hetzel, 1988

5.2.5 Black, 2001, Craig, 2002, IEEE Std 829-1998, Kaner 2002

5.3.3 Black, 2001, Craig, 2002, Hetzel, 1988, IEEE Std 829-1998

5.4 Craig, 2002

5.5.2 Black, 2001, IEEE Std 829-1998

5.6 Black, 2001, IEEE Std 829-1998

6. テスト支援ツール(K2)　学習時間:80 分

本章の学習の目的

本章で記述する学習内容は以下のとおり。

6.1 テストツールの種類(K2)

LO-6.1.1 テストツールをそれらの目的と基本的なテストプロセス、およびソフトウェアライフサイクルでの活動により分類する。(K2)

LO-6.1.3 テストツールに関する用語と、テストへのツールによる支援の目的を説明する。(K2)

6.2 ツールの効果的な使い方:利点とリスク(K2)

LO-6.2.1 テストの自動化やツールによる支援の利点とリスクをまとめる。(K2)

LO-6.2.2 テスト実行ツールと静的解析ツール、テストマネジメントツールに対する固有の考慮事項を認識する。(K1)

6.3 組織へのツールの導入(K1)

LO-6.3.1 組織にツールを導入する際の基本原則を提示する。(K1)

LO-6.3.2 ツール評価のために行うコンセプトの証明(proof-of-concept)のゴールと、ツール導入時のパイロットフェーズのゴールを提示する。(K1)

LO-6.3.3 ツールによる支援を成功させるには単にツールを入手する以外の要因が要求されることを認識する。(K1)

6.1.　テストツールの種類(K2)　学習時間:45 分

用語

構成管理ツール、カバレッジ測定ツール、デバッグツール、動的解析ツール、インシデントマネジメントツール、ロードテストツール、モデリングツール、モニタリングツール、性能テストツール、プローブ効果、要件マネジメントツール、レビューツール、セキュリティツール、静的解析ツール、ストレステストツール、テスト比較ツール、テストデータ準備ツール、テスト設計ツール、テストハーネス、テスト実行ツール、テストマネジメントツール、ユニットテストフレームワークツール

6.1.1.　ツールによるテストへの支援(K2)

テストツールは、一つ以上のテストを支援する活動に用いることが可能であり、以下のようなものが挙げられる。

- テストに直接利用されるツール。例えば、テスト実行ツールやテストデータを生成するツール、テスト結果を比較するツールなどが該当する。
- テストプロセスのマネジメントを支援するツール。例えば、テストやテスト結果、データ、要件、インシデント、欠陥などをマネジメントするために用いられるツールや、テスト実行をモニタしたりレポートしたりするためのツールが該当する。
- 調査や探索のために用いられるツール。例えば、アプリケーションのファイル使用状況を監視するツールが該当する。
- テストを支援するあらゆるツール。この意味では表計算ソフトウェアもテストツールである。

テストを支援するツールは、状況に応じ、以下に示すもののうちの一つ以上を目的とする。

- 反復作業を自動化したり、テスト計画やテスト設計、テストレポート、モニタリングのような手動で行うテストの活動を支援したりすることによって、テストの活動の効率を改善する。
- 手動で行うと大きなリソースを要する活動(例えば、静的テスト)を自動化する。
- 手動では実行できない活動(例えば、クライアントサーバアプリケーションの大規模な性能テスト)を自動化する。
- 例えば、大量のデータの比較を自動化したり、動作をシミュレーションしたりすることによって、テストの信頼性を上げる。

業界において、「テストフレームワーク」という用語は、少なくとも以下の 3 つの意味でしばしば用いられる。

- テストツールに組み込まれている、再利用可能で拡張可能なテストライブラリ(テストハーネスとも呼ばれる)
- テスト自動化のための設計の種類(例えば、データ駆動やキーワード駆動など)
- テストの中で行われるプロセス全体

このシラバスでは、6.1.6 で述べるように、最初の 2 つの意味で「テストフレームワーク」という用語を用いる。

6.1.2.　テストツールの分類(K2)

テストツールにはいろいろなものがあり、テストの様々な部分を支援している。ツールは、目的や市販/フリー/オープンソース/シェアウェア、利用する技術などといった基準で分類できる。本シラバスでは、テストツールが支援するテストの活動によって、ツールを分類している。

ツールの中には、明確に一つの活動しか支援しないものがある。また、複数の活動を支援するものは、代表的な活動で分類する。連携するように設計された複数のツールが、一つのパッケージとして提供されることもある。

テストツールの中には、テストの実際の出力に影響を及ぼすものがある。例えば、ツールによって実行される特別な命令のせいで、実行のタイミングが実際とは異なってしまうかもしれないし、あるいは、測定コードカバレッジの値が異なることがあるかもしれない。このようなツールによる結果を、プローブ効果と呼ぶ。

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

6.1.3. テストマネジメントの支援用ツール(K1)

マネジメント用ツールは、ソフトウェアライフサイクルでの全てのテストの活動に適用可能である。

テストマネジメントツール

これらのツールは、定量的分析やテスト対象のレポートに加えて、テストの実行や欠陥の追跡、要件のマネジメントのためのインターフェースを提供する。テスト対象から要求仕様書への追跡についても支援し、ツールによっては、ツール自身にバージョン管理機能を持つ場合や、外部のバージョン管理ツールとのインターフェースを持つ場合がある。

要件マネジメントツール

これらのツールは、要件や要件の属性(優先順位を含む)を格納し、一意の識別子を提供し、要件から各テストへの追跡を支援する。一貫しない要件や欠けている要件の識別を支援するものもある。

インシデントマネジメントツール(欠陥追跡ツール)

これらのツールは、インシデント(すなわち、欠陥、故障、変更要求、検出した問題、不具合)レポートを格納し、管理する。インシデントのライフサイクルの管理を支援する。また、統計的分析を支援するものもある。

構成管理ツール

厳密にはテストツールではないが、テストウェアや関連するソフトウェアの保管とバージョン管理に必要となる。特に複数のハードウェアやソフトウェア環境(OS のバージョン、コンパイラ、ブラウザなど)で構成する場合は必要である。

6.1.4. 静的テスト支援ツール(K1)

静的テスト支援ツールは、開発プロセスの早い段階で多くの欠陥を見つける方法により、高い費用対効果を提供する。

レビューツール

これらのツールは、レビュープロセス、チェックリスト、レビューガイドラインを支援し、またレビューコメントと欠陥および取り組みについてのレポートを保管したり情報共有したりすることに使用される。大規模なチームや地理的に離れているチームのためにオンラインレビューを支援できるツールもある。

静的解析ツール(D)

これらのツールは、コーディング規約(セキュアコーディングを含む)の遵守、構造や依存関係の分析を支援するもので、開発担当者やテスト担当者が動的テストの前に欠陥を発見するのに役立つ。また、ツールが提供するソースコードメトリクス(例えば、複雑度)は、計画やリスク分析の際に役立つ。

モデリングツール(D)

これらのツールは、ソフトウェアのモデル(例えば、関係データベースの物理データモデル[PDM：Physical Data Model])を検証するために用いられるもので、不整合を列挙したり欠陥を摘出したりすることができる。モデルをベースに、テストケースを生成するツールもある。

6.1.5. テスト仕様の支援ツール(K1)

テスト設計ツール

これらのツールは、要件、グラフィカルユーザインターフェース、設計モデル(状態、データ、オブジェクト)、ソースコードから、テストの入力データや実行可能なテスト、テストオラクルを生成するために用いられる。

テストデータ準備ツール

テストデータ準備ツールは、データベース、ファイル、転送データを操作して、テスト実行中に利用できるセキュリティが確保された架空のテストデータを用意する。

6.1.6. テスト実行と結果記録の支援ツール(K1)

テスト実行ツール

これらのツールは、入力データと期待する出力結果を格納し、スクリプト言語を使用して、テストを自動実行、あるいは半自動実行する。通常、各テストを実行するごとにテスト結果記録を提供する。また、テストを記録することができ、スクリプト記述機能やテストにおけるデータのパラメータ化や他のカスタマイズをするための GUI ベースの設定機能をサポートしていることが多い。

テストハーネス、ユニットテストフレームワークのツール(D)

テストハーネスあるいはユニットテストフレームワークは、スタブやドライバとなる模造のオブジェクトを使用することによってテスト対象を実行する環境をシミュレーションし、コンポーネントやシステムの一部のテスト実行を支援する。

テスト比較ツール

テスト比較ツールは、ファイル、データベース、テスト結果の違いをチェックする。テスト実行ツールは、通常、テスト比較ツールを内蔵しているが、実行後の比較は、別の比較ツールで実施することもある。自動実行する場合、テスト比較ツールはテストオラクルを使うことがある。

カバレッジ測定ツール(D)

測定これらの測定ツールは、特定の種類のコード構造(例えば、ステートメント、ブランチ、判定、モジュールやファンクションの呼び出し)について、実行済みの部分のパーセンテージを測定するもので、侵入的である場合がある。

セキュリティツール

これらのツールは、ソフトウェアのセキュリティ特性を評価するために用いられる。これは、データの機密性、完全性、認証、権限、可用性、否認防止といった、データを保護するためのソフトウェアの能力を評価することを含む。セキュリティツールの多くは、特定の技術やプラットフォーム、目的に焦点をあてている。

6.1.7.　性能・モニタリング支援ツール(K1)

動的解析ツール(D)

動的解析ツールは、ソフトウェアを実行しているときに現れる欠陥、例えば、時間に依存するものや、メモリリークのような欠陥を摘出する。通常コンポーネントテストやコンポーネント統合テスト、ミドルウェアをテストする場合に使う。

性能テスト／ロードテスト／ストレステストツール

性能テストツールは、シミュレーションしたいろいろな使用条件下で、システムの動作をモニタリングしたり、レポートしたりするツールである。使用条件は、同時にシステムを使用するユーザの数や、ユーザの立ち上がり(ramp-up)パターン、トランザクションの頻度や相対的な割合などの観点で指定する。負荷のシミュレーションは、仮想ユーザを作成し、選択したトランザクションのセットを様々なテストマシンへ広がるように実行する。一般的にこれはロードジェネレータという名前で知られている。

モニタリングツール

モニタリングツールは、特定のシステムリソースの使用状況を継続的に分析、検証、レポートし、システムが提供するサービスに問題が発生する可能性がある場合、警告を出す。

6.1.8.　特定のテストに対する支援ツール(K1)

データ品質評価

データコンバージョン(変換)/マイグレーション(移行)のためのプロジェクトやデータウェアハウスのようなアプリケーションにおいては、データがプロジェクトの中心となる。その属性は、データの重要度合いや量によって変わることもある。この状況において、ツールは、データ品質評価のために使われる必要がある。データ品質評価では、処理されたデータが正しく、完全であり、あらかじめ定義済みの状況に応じた標準に変換することが保証できるデータコンバージョン/マイグレーションのルールとなっているかをレビューし検証する。

他には、使用性テストのためのテストツールが存在する。

6.2.　ツールの効果的な使い方:利点とリスク（K2）　学習時間:20 分

用語

データ駆動テスト、キーワード駆動テスト、スクリプト言語

6.2.1.　ツールでテストを支援する利点とリスク（全ツール）(K2)

ツールを購入したり、リースしたりしただけでは、ツールの効果は出ない。各ツール本来の効果を継続的に上げるには、それなりの工数を投下する必要がある。テストでツールを使うことには潜在的な利点もあるし、テストにはツールを使うに適した機会もあるが、リスクもある。

ツールを使うことで期待できる効果は以下のとおり。

- 反復作業が減る。(例えば、回帰テストの実行、同じテストデータの再入力、コーディング規約との比較など)
- 一貫性や再実行性が増加する。(例えば、要件をベースにしたテスト、ツールによる同じ頻度で同じ順序のテスト実行など)
- 客観的な評価が可能になる。(例えば、静的解析、カバレッジなど)
- テストやテストケースの情報へのアクセスが容易になる。(例えば、テスト進捗の統計を取ったり、図的に表現したり、インシデント率や性能計測結果の算出など)

ツールを使うことで発生するリスクは以下のとおり。

- テストツールの効果を過大に期待する。(例えば、ツールの機能性や使いやすさなど)
- テストツールを初めて導入する場合に要する時間、コスト、工数を過小評価する。(例えば、教育や、外部の専門家の支援など)
- 大きな効果を継続的に上げるために必要な時間や工数を過小評価する。(例えば、テストプロセスの変更、ツールの使用法を常に改善するなど)
- ツールが生成するテスト資産を保守するために必要な工数を過小評価する。
- ツールに過剰に依存する。(テスト設計と置き換えたり、手動テストの方が適したケースで自動テストを利用したりするなど)
- ツール内にあるテスト資産のバージョン管理を怠る。
- 要件管理ツール、バージョン管理ツール、インシデントマネジメントツール、欠陥追跡ツール、複数のベンダーからのツールなど、重要なツール間での関係性と相互運用性の問題を無視する。
- ツールベンダーがビジネスを廃業したり、ツールを撤退したり、別のベンダーにツールを売ったりするリスクがある。
- ツールのサポート、アップグレード、欠陥修正に対するベンダーの対応が悪い。
- オープンソース/フリーツールプロジェクトが停止するリスクがある。
- 新しいプラットフォームをサポートできないといった、予想外の出来事が起こる。

6.2.2.　個別ツールの使用上の注意(K1)

テスト実行ツール

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

手動テストの担当者の操作をそのまま記録することでテストの実行手順をキャプチャするのは魅力的である。しかし、このやり方では大量の自動テストスクリプトを活用するよう拡張できない。テストをキャプチャしたときのスクリプトは、各スクリプトの一部であり、特定のデータや動作で線形的に表現したものである。この種のスクリプトは、期待しないイベント

が起きると、動作が不安定になる。
データ駆動によるテストアプローチでは、テスト入力(テストデータ)をスプレッドシートに入れるなどして分類する。そして、入力データを読んだり、異なるテストデータで同じテストスクリプトを実行したりするような、一般的なテストスクリプトを使う。スクリプト言語に慣れていないテスト担当者は、あらかじめ定義済みのスクリプトに対し、テストデータを作ればよい。

データ駆動技術を応用した他の技術もある。スプレッドシートに直接書き込んだ変更できないデータの組み合わせの代わりに、テスト実行時に構成できるパラメータに基づきアルゴリズムを使ってデータを生成しアプリケーションに与えるものがそうである。例えば、ツールはランダムなユーザIDを生成したり、パターンを繰り返すことができるよう乱数を制御するためにシードが用いられるアルゴリズムを使用したりする場合がある。

一方、キーワード駆動のテストアプローチでは、実行させる動作を定義するキーワード(アクションワードとも呼ぶ)と、テストデータをスプレッドシートに設定する。スクリプト言語に慣れていないテスト担当者でも、キーワードを使ってテストを定義できる。キーワード駆動によって、テストをテスト対象のアプリケーション用に調整することができる。

スクリプト言語では、どんなアプローチであっても、技術的な専門知識が必要となる。(テスト担当者が実行する場合でも、テスト自動化のスペシャリストが実行する場合でも)

使用したスクリプト技術に関わらず、各テストで必要な「期待結果」は、後で比較するため、あらかじめ、格納しておく必要がある。

静的解析ツール

ソースコードに静的解析ツールを適用することにより、コーディング規約を遵守させることは可能だが、作成済みのソースコードに適用すると、大量のメッセージを吐き出すこともある。警告メッセージが出たソースコードは、実行形式プログラムに変換できないことはないが、理想を言えば将来、保守が容易になるよう警告の出たソースコードを見直すべきである。解析ツールの導入時は、初期は警告メッセージのいくつかが出力されないようにフィルタを設定するなどして、段階的に導入することが効果的なアプローチである。

テストマネジメントツール

テストマネジメントツールは、組織が必要とするフォーマットで役立つ情報を作成するために、他のツールやスプレッドシートとのインターフェースが必要である。

6.3.　組織へのツールの導入（K1）　学習時間：15 分

用語

なし

背景

組織にツールを導入する際の基本原則は以下のとおり。

- ツールを活用するためのテストプロセス改善の機会や組織の成熟度、長所と短所を評価する。
- (ツールに対する)明確にした要件と、客観的な基準を背景に評価をする。
- 「コンセプトの証明(proof-of-concept)」を行う。つまり、テストツールを評価する中で、テスト対象に効果的に使えるツールなのか、および現在のインフラストラクチャで効果的にツールを使うことができるのかを立証する。もしくは、ツールを効果的に使うために必要となるインフラストラクチャの変更点を識別する。
- ツールベンダー(トレーニングや、サポートメニュー、ビジネス的な要素も含む)もしくは、非商用ツールの場合は、サービスサポート提供者を評価する。
- ツールを使う上で、教育や訓練に対する組織内の必要事項を特定する。
- 現在のテストチームのテスト自動化スキルを考慮して、トレーニングの必要性を評価する。
- 具体的なビジネスケースに基づいて、費用対効果を見積る。

選定したツールを組織に導入する際は、パイロットプロジェクトから始める。

パイロットプロジェクトには次のような目的がある。

- ツールの詳細を学習する。
- 現状のプロセスやプラクティスにツールが適用可能か、変更が必要な場合、どう変更すればよいかチェックする。
- ツールやテストの関連物(例えば、ファイルやテストケースの命名法の決定、ライブラリの作成、テストスイートの部品の定義など)の使用、管理、格納、保守の標準的な方法を決める。
- 期待する効果が妥当なコストで実現可能かどうかを見極める。

組織内でツールの展開を成功させるには、以下が必要になる。

- ツール未使用の部署にツールを順々に展開する。
- ツールが適用できるよう、プロセスを適用、改善する。
- 新規ユーザに対し、トレーニング、指導、助言を提供する。
- 利用ガイドを定める。
- 実際に使用する中で、活用していくための情報を集める方法を実装する。
- ツールの利用状況や効果をモニタリングする。
- ツール利用のために、テストチームへの支援を提供する。
- 全てのチームから、学んだ教訓を集める。

◆ 参考文献

6.2.2 Buwalda, 2001, Fewster, 1999

6.3 Fewster, 1999

7. 参考文献

7.1. 標準:Standards

- ISTQB Glossary of terms used in Software Testing Version 2.1
- [CMMI] Chrissis, M.B., Konrad, M. and Shrum, S. (2004) CMMI, Guidelines for Process Integrationand Product Improvement, Addison Wesley: Reading, MA
  - See section 2.1
- [IEEE Std 829-1998] IEEE Std 829™ (1998) IEEE Standard for Software Test Documentation,
  - See sections 2.3, 2.4, 4.1, 5.2, 5.3, 5.5, 5.6
- [IEEE 1028] IEEE Std 1028™ (2008) IEEE Standard for Software Reviews and Audits,
  - See section 3.2
- [IEEE 12207] IEEE 12207/ISO/IEC 12207-2008, Software life cycle processes,
  - See section 2.1
- [ISO 9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality,
  - See section 2.3

7.2. 書籍:Books

- [Beizer, 1990] Beizer, B. (1990) Software Testing Techniques (2nd edition), Van Nostrand Reinhold:Boston
  「ソフトウェアテスト技法」（日経 BP，1994 年）
  - See sections 1.2, 1.3, 2.3, 4.2, 4.3, 4.4, 4.6
- [Black, 2001] Black, R. (2001) Managing the Testing Process (3nd edition), John Wiley & Sons: New York
  「基本から学ぶテストプロセス管理」（日経 BP，2004 年）
  - See sections 1.1, 1.2, 1.4, 1.5, 2.3, 2.4, 5.1, 5.2, 5.3, 5.5, 5.6
- [Buwalda, 2001] Buwalda, H. et al. (2001) Integrated Test Design and Automation, Addison Wesley:Reading, MA
  - See section 6.2
- [Copeland, 2004] Copeland, L. (2004) A Practitioner's Guide to Software Test Design, Artech House:Norwood, MA 「はじめて学ぶソフトウェアのテスト技法」（日経 BP，2005 年）
  - See sections 2.2, 2.3, 4.2, 4.3, 4.4, 4.6
- [Craig, 2002] Craig, Rick D. and Jaskiel, Stefan P. (2002) Systematic Software Testing, Artech House:Norwood, MA 「体系的ソフトウェアテスト入門」（日経 BP，2004 年）
  - See sections 1.4.5, 2.1.3, 2.4, 4.1, 5.2.5, 5.3, 5.4
- [Fewster, 1999] Fewster, M. and Graham, D. (1999) Software Test Automation, Addison Wesley:Reading, MA
  - See sections 6.2, 6.3
- [Gilb, 1993] Gilb, Tom and Graham, Dorothy (1993) Software Inspection, Addison Wesley: Reading,MA
  「ソフトウェアインスペクション」(共立出版,1999 年)
  - See sections 3.2.2, 3.2.4
- [Hetzel, 1988] Hetzel, W. (1988) Complete Guide to Software Testing, QED: Wellesley, MA
  - See sections 1.3, 1.4, 1.5, 2.1, 2.2, 2.3, 2.4, 4.1, 5.1, 5.3
- [Kaner, 2002] Kaner, C., Bach, J. and Pettticord, B. (2002) Lessons Learned in Software Testing, John Wiley & Sons: New York
  「ソフトウェアテスト 293 の鉄則」（日経 BP，2003 年）
  - See sections 1.1, 4.5, 5.2
- [Myers 1979] Myers, Glenford J. (1979) The Art of Software Testing, John Wiley & Sons: New York

「ソフトウェア・テストの技法」(近代科学社,1980 年)
- See sections 1.2, 1.3, 2.2, 4.3

- [van Veenendaal, 2004] van Veenendaal, E. (ed.) (2004) The Testing Practitioner (Chapters 6, 8, 10),UTN Publishers: The Netherlands
  - See sections 3.2, 3.3

付録 A – 本シラバスの背景

このドキュメントができるまでの経緯

このドキュメントは 2004 年から 2005 年の間に国際ソフトウェアテスト資格認定委員会(ISTQB®)のメンバーによって選任された Working Group Foundation Level によって作成された。これは最初に審査委員会によってレビューされ、その後、広く各国のソフトウェアテストのコミュニティから選ばれた代表よりレビューされた。このドキュメントを作成するにあたって用いた規則は付録Cに記載されている。

このドキュメントはISTQB® (www.istqb.org)によって承認されたソフトウェアテストの国際的なFoundation Levelの資格認定のためのシラバスである。
本文が記述された時点(2005 年)の ISTQB®のメンバーにはオーストリア、デンマーク、フィンランド、フランス、ドイツ、インド、イスラエル、日本、韓国、ノルウェー、ポーランド、ポルトガル、スペイン、スウェーデン、スイス、オランダ、イギリスとアメリカが含まれている。

Foundation Level 資格認定の目的

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

国際資格認定の目的(2001 年 11 月、ISTQB®のソレンテュナでの会議にて改案)

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

本資格認定試験の受験要件

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

- システムテストやユーザ受け入れテストなどのテスト担当、もしくはソフトウェア開発担当など、最低 6 ヶ月程度

技術者としての経歴があること。
- ISTQB®に加盟している各国の委員会(日本では JSTQB®が該当する)によって認定された教育コースを受けていること。

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

独立したソフトウェアテストの資格認定は、1998 年にイギリスのブリティッシュコンピュータソサエティ情報システム研究委員会(ISEB)によってソフトウェアテスト委員会が設置されたときからスタートした。(www.bcs.org.uk/iseb) 2002 年には、ドイツのASQFがドイツのテスト技術者資格認定のスキームを支援し始めた。(www.asqf.de) 本シラバスは、ISEBとASQFのシラバスが基となっている。これは再編成され、内容を更新、追加したものとなっている。この際最も重要視したことは、テスト技術者にとってより実用的になるように心がけたことである。
この国際資格がスタートする前に認定されたソフトウェアテスト Foundation Level の資格(例えば、ISEB、ASQF、または ISTQB®が認可した各国の委員会によるもの)は国際資格と同等のものだと見なす。同等と見なされたこれらの基準資格は、期限が切れることはなく、更新の必要もない。資格が与えられた日付が認定書に記されている。

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

付録 B – 学習している知識の目的と認知レベル

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

レベル 1:記憶レベル(K1)

用語または概念を認識し、記憶して、想起することができる。

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

例

「故障」の定義を次のように認識できる。

- 「エンドユーザまたは他の関係者にサービスを引き渡しできないこと。」
- 「コンポーネントまたはシステムに期待する出力、サービス、または最終結果が異なっていること。」

レベル 2:理解レベル(K2)

課題に関連する記述について理由または説明を選択することができ、テスト概念、テスト手順(タスクの順番の説明)に関して要約、比較、分類、類別することができ、テスト概念に関して例を挙げることができる。

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

例

できるだけ早くテストを設計しなければならない理由を説明することができる。

- 欠陥除去をより低コストで行うため。
- 重要な欠陥をより早く見つけるため。

統合テストとシステムテストの類似点と相違点を説明することができる。

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

レベル 3:適用レベル(K3)

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

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

例

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

レベル 4:分析レベル(K4)

手順または技術に関連する情報を分類し、理解しやすくするため各部分に分割し構成することができる。同時に事実と推論を区別することができる。典型的な適用例として、問題を解決するためやタスクを実行するために、ドキュメント、ソフトウェア、プロジェクトの状態を分析し、適した活動を提案できることが該当する。

キーワード:分析(analyze)、体系(organize)、一貫性の発見(find coherence)、統合(integrate)、概要(outline)、解析(parse)、構造(structure)、属性(attribute)、分解(deconstruct)、区別(differentiate)、選別(discriminate)、識別(distinguish)、集中(focus)、選択(select)

例

- プロダクトのリスクを分析し、予防および軽減活動を提案する。
- インシデントレポートのどの部分が事実でどの部分が結果からの推察かを判断する。

リファレンス (学習の目的の認識レベル用)

Anderson, L. W. and Krathwohl, D. R. (eds) (2001). *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, Allyn & Bacon.

付録 C – ISTQB®に適用する規則 Foundation Level シラバス

ここに記載した規則は、本シラバスの開発とレビューに用いたものである。
(各規則の最後に省略記号として“タグ”を示す)

総則

- SG1. 本シラバスはテスト経験が 0 から 6 ヶ月位までの人々でも理解でき、かつ、取り入れることができるものでなければならない。(6-MONTH)
- SG2. 本シラバスは理論的であるよりも、むしろ実践的でなければならない。(PRACTICAL)
- SG3. 本シラバスは対象とする読者にとって、わかりやすく、あいまいさのないものでなければならない。(CLEAR)
- SG4. 本シラバスは異なる国の人々に理解され、かつ、他の言語に容易に翻訳できるものでなければならない。(TRANSLATABLE)
- SG5. 本シラバスはアメリカ英語を使用しなければならない。(AMERICAN-ENGLISH)

最新の内容

- SC1. 本シラバスは最新のテストコンセプトを含み、かつ、一般的に認められた最新のソフトウェアテストのベストプラクティスを反映しなければならない。本シラバスは 3～5 年ごとにレビューされる。(RECENT)
- SC2. 本シラバスは、3～5 年の有効期限を可能にするために、最新のマーケット状況など、時間に関する事項をできる限り少なくしなければならない。(SHELF-LIFE)

学習目的

- LO1. 学習目的は、
  - 認識すべき／記憶すべき項目(認識レベル K1)
  - 概念的に理解すべき項目(K2)
  - 実践できる／使用できる項目(K3)
  - 状況に応じて、ドキュメント、ソフトウェア、プロジェクト状況を解析して利用できる項目(K4)

  に区別されなければならない。(KNOWLEDGE-LEVEL)
- LO2. 内容の記述は学習目的と一貫性がなければならない。(LO-CONSISTENT)
- LO3. 学習目的を説明するために、各主要セクションのサンプルの試験問題はシラバスとともに発行されなければならない。(LO-EXAM)

全体構造

- ST1. シラバスの構造は明確でなければならず、また他のパート、試験問題、および他の関連文書と相互に参照できなければならない。(CORSS-REF)
- ST2. シラバスのセクション間の重複は最小限にとどめなければならない。(OVERLAP)
- ST3. シラバスの各セクションはみな同じ構造でなければならない。(STRUCTURE-CONSISTENT)
- ST4. シラバスはバージョン、発行日、ページ番号が全てのページに記されていなければならない。(VERSION)
- ST5. シラバスはセクションごとに費やすべき時間の合計のガイドラインを盛り込まなければならない。(各トピックの相対的な重要性を示すため)(TIME-SPENT)

参照

- SR1. トレーニングの提供者がトピックについてより多くの情報を得られるように、情報源と参考文献とをシラバスのコンセプトに示す。(REFS)
- SR2. 容易に確認できるはっきりとした情報源がない場合、より多くの詳細をシラバスに提供しなければならない。例えば、定義は用語集に記載されており、専門用語のみシラバスに記載されている、など。(NON-REF DETAIL)

情報源

- 本シラバスに記載されている専門用語は Software Testing で使われている ISTQB®の用語集で定義されている。用語集は ISTQB®で入手できる。
- ソフトウェアのテストに関する推薦文献のリストも、本シラバスとともに発行される。主要な本のリストは参照のセクションの一部をなしている。

付録Ｄ－ 教育機関への注意書き

本シラバスの各章見出しには、時間が割り当てられている。 この目的は、認定コースの各セクションの相対的な割合を示すと共に、各セクションの教育に費やす最小時間を示している。
教育機関は示されるより多くの時間を費やすこともあれば、受験志願者は、記載以上の時間を読書と研究に費やすこともあろう。 コースカリキュラムは本シラバスの構成に従う必要もない。
本シラバスは、7.1で示した標準を参照できるようにしてあり、トレーニング資料の準備には、標準を使用しなければならない。 使用される各標準は本シラバスの最新版で引用されたバージョンでなければならない。 また本シラバスでは参考としていない他の刊行物、テンプレートまたは標準が、教材として使用されるかもしれないが、出題範囲とはならない。

全てのK3とK4に関する学習の目的は、演習としてトレーニング資料に含めることが必要である。

付録 E - リリースノート、シラバス 2010

1. 学習目的(LO)の変更は、明確化を含む。
   a. LO-1.2.2、LO-1.3.1、LO-1.4.1、LO-1.5.1、LO-2.1.1、LO-2.1.3、LO-2.4.2、LO-4.1.3、LO-4.2.1、LO-4.2.2、LO-4.3.1、 LO-4.3.2、LO-4.3.3、LO-4.4.1、LO-4.4.2、LO-4.4.3、LO-4.6.1、LO-5.1.2、LO-5.2.2、LO-5.3.2、 LO-5.3.3、LO-5.5.2、 LO-5.6.1、LO-6.1.1、LO-6.2.2、LO-6.3.2 の各 LO で用語を変更した。(LO の内容とレベルは変動なし)
   b. LO-1.1.5 は、記述しなおし K2 に更新した。理由は、欠陥に関連する用語について比較できることを期待するため。
   c. LO-1.2.3(K2)において、デバッグとテストの 2 つの活動の違いを説明することは新しい LO である。内容は、既にカバーしていた。
   d. LO-3.1.3(K2)において、LO-3.1.4 と内容を結合した。
   e. LO-3.1.4 は削除した。LO-3.1.3 と部分的に冗長だった。
   f. LO-3.2.1 において、内容の整合性を取った。
   g. LO-3.3.2 は、LO-3.1.2 と整合性を取るために K2 に更新した。
   h. LO-4.4.4 は既に K4 レベルで書かれていたため、K3 レベルから K4 レベルに変更する。
   i. LO-6.1.2 は、LO-6.1.3 の一部にして削除した。LO-6.1.2 はこのシラバスには存在しない。
2. 用語集の定義に従って、テストアプローチを一貫して使用した。テスト戦略という用語は、思い出す用語として要求されない。
3. 1.4 節は、テストベースとテストケース間のトレーサビリティの概念を、今回から含む。
4. 2.x 節は、テスト対象とテストベースを、今回から含む。
5. 再テストは、今回から用語集と同様重要な用語とし、確認テストは使わない。
6. データ品質とテストを、シラバス上のいくつかの箇所に追加した。2.2 節、5.5 節、6.1.8 節におけるデータ品質とリスクの箇所である。
7. 5.2.3 節において、開始基準を新しい節として追加した。理由は、終了基準との整合性のためである(->開始基準は、LO-5.2.9 として追加した)。
8. 用語集の定義に従って、テスト戦略とテストアプローチの用語を一貫して使用した。
9. 6.1 節は、ツールの記述が 45 分の学習時間に対して多すぎるので、短くした。
10. IEEE Std 829-2008 がリリースされた。今回のシラバスでは、この新版を考慮していない。5.2 節は、マスターテスト計画の文書を参照している。マスターテスト計画の内容は、「テスト計画」の文書が計画の異なるレベルをカバーするという概念によってカバーされている。すなわち、各テストレベルのテスト計画は、複数のテストレベルをカバーするプロジェクトレベル上のテスト計画と同様に作り出すことができる。後者は、このシラバスと用語集で、マスターテスト計画と名付けている。
11. 行動規範は、Advanced Level から Foundation Level に移行した。

2011 年保守リリースでの対応内容

1. 一般: Working Party は Working Group に変更した
2. post-conditions という記述を postconditions に統一した。（ISTQB Glossary 2.1.に合わせた）
   ※日本語対応は無し
3. ISTQB と言う記述は ISTQB®に統一した。
4. イントロダクションでの知識の認知レベルの説明の一部を取り除いた。Appendix B. で確認可能なためである。
5. 1.6 節　行動規範: LO の定義がないため、知識レベルを取り除いた。
6. 2.2.1, 2.2.2, 2.2.3, 2.2.4, 3.2.3 節:フォーマットの問題を解決した。
7. 2.2.2 節:　「故障」を「欠陥」に修正した。
8. 2.3 節　テストタイプ(K2): テスト目的のリストの段落を修正した。
9. 2.3.4 節: デバッグの説明を更新（ISTQB Glossary 2.1.に合わせた）
10. 2.4 節: 「回帰テストもしっかり」から「しっかり」を取り除く。理由は「しっかり」はサイズやリスク、価値などによって変化するからである。
11. 3.2 節: センテンスを明快にするため、「チームの参加、文書化されたレビュー結果、レビューを実施するための文書化された手続きに含まれるもの」から「含まれるものを」を取り除く。
12. 3.2.1 節: シラバス 2007、AL シラバス 2007 と整合を取り、6 つのメインの活動となるようにした。
13. 4.1 節:「開発した」を「定義した」に置き換えた。なぜなら、テストケースは定義されるものであり、開発されるものではないためである。
14. 4.2 節: ブラックボックスとホワイトボックスが経験ベースと一緒になって使われるであろうことを明快にするためにテキストを変更した。
15. 4.3.5 節のテキストを次のように変更した。「(ユーザおよびシステムを含む)アクター」から「アクター(ユーザおよびシステム)」
16. 4.3.5 節 :　「代替パス」を「代替シナリオ」に差し替えた。
17. 4.4.2 節: 4.4 節の中で、ブランチテストの用語を明快にするための対応。
18. 4.5 節, 5.2.6 節: 「経験ベース」の用語修正（“experienced-based” を “experience-based”に変更。日本語対応は無し。）
19. 6.1 節:　「6.1.1 ツールでテストを支援する意味と目的(K2)」を「ツールによるテストへの支援(K2)」に差し替えた。
20. 7. 参考文献/ 書籍: 2nd edition の代わりに The 3rd edition of [Black,2001] をリストした。
21. 付録 D: ISTQB Accreditation Process (Version 1.26)の要求により、演習問題に対する要求を変更した。
22. 付録 E: 2007 から 2010 への LO の変更を正しく列挙した。