品質を創造する
VERISERVE

ソフトウェアテスト基礎講座

当研修の特徴

VERISERVE

① ソフトウェアは規模も複雑さも増加傾向にあります。

そのため、対象ソフトウェアのテストを、どのような切り口で、どの順番で、何を確認するのかを決定する「テスト設計」が重要になります。
そこで、当研修は、**テスト設計に焦点**を当てます。
また、テストの実務に必要となるテストに対する考え方や、開発全体における実施タイミングなども含め、講義と演習を通じて学習します。

② 本研修はJSTQB®（http://jstqb.jp/）のFoundation Level（FL）のシラバス（学習範囲を示すテキスト）や、用語集に準拠しています。

本研修後にJSTQB® FLのテスト技術者資格認定試験に挑戦するのもよいでしょう。

なお、本研修は試験範囲をすべて網羅しているわけではありません。

研修で皆さんに伝えたいこと

VERISERVE

① テスト技法を意図的に使い分けます。

基本的なテスト技法については、その特徴、長所、短所、効果的な使用局面、使用上の注意事項などを知り、実務で使いこなすことが重要です。
（知っている、言われてみれば使っている、では足りません。）

② テストと開発の連携・協調が重要であり、分離しないようにします。

プロジェクトのチームメンバーとして、積極的に開発者（設計／実装メンバー）と協力しつつ、よい製品、よいサービスを顧客に提供する活力源となるように行動することが重要です。
（テスト技法が使えるだけでは足りません。）

当研修の目次（JSTQB® FLと同じ）

VERISERVE

1. テストの基礎
   1.1. テストとは何か？
   1.2. テストの必要性
   1.3. テストの7原則
   1.4. テストプロセス
   1.5. テストの心理学

2. ソフトウェア開発ライフサイクル全体を通してのテスト
   2.1. ソフトウェア開発ライフサイクルモデル
   2.2. テストレベル
   2.3. テストタイプ
   2.4. メンテナンス(保守)テスト

3. 静的テスト
   3.1. 静的テストの基本
   3.2. レビュープロセス

4. テスト技法
   4.1. テスト技法のカテゴリ
   4.2. ブラックボックステスト技法
   4.3. ホワイトボックステスト技法
   4.4. 経験ベースのテスト技法

5. テストマネジメント
   5.1. テスト組織
   5.2. テストの計画と見積り
   5.3. テストのモニタリングとコントロール
   5.4. 構成管理
   5.5. リスクとテスト
   5.6. 欠陥マネジメント

6. テスト支援ツール
   6.1. テストツールの考慮事項
   6.2. ツールの効果的な使い方

1. テストの基礎

1.1 テストとは何か？

テストとは？ ①

- 「テストでプログラム中の欠陥の存在は示せても、欠陥が存在しないということは示しえない」
  by E.W. Dijkstra ← テストの7原則の“原則1”と同じ
- 「欠陥を全部見つけるのは無理だと心得よ」
  by Cem Kaner
- 「テストとは、エラーを見つけるつもりでプログラムを実行する過程である」
  by G.J. Myers ← 心理的側面を強調している
- 「ソフトウェアテスティングは、プログラムが、日常生まれる無限の実行ドメインから適切に選択された(selected)テストケースの有限(finite)集合のうえで行われる、期待される(expected)振る舞いを提示するための動的(dynamic)検証から構成される」
  by SWEBOK V3.0 のテストの定義

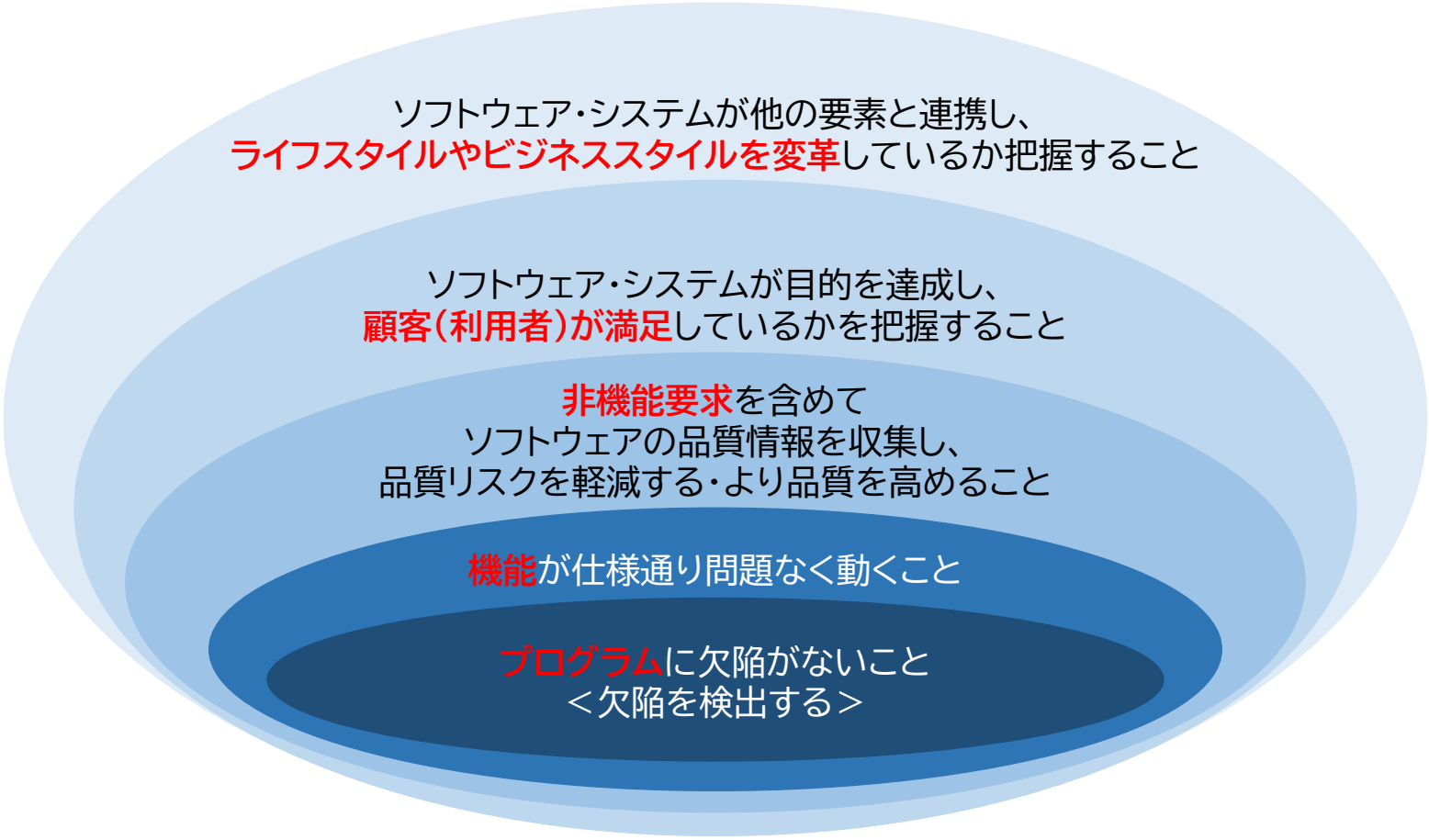
テストとは？ ②

- 「品質保証を支援することにある。すなわち、品質に関連した情報を収集し、プログラマにその情報をフィードバックすることで、過去に発生した不良の再発を防止し、ソフトウェアの品質をよい方向へと導くことである」
  by Boris Beizer

出典：実践的プログラムテスト入門

テスト目的の拡大

機能充足 → 目的達成 → 価値提供
品質～Quality～QOL(Quality of Life)へ



テストに共通する目的

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

出典：Foundation Level シラバス

テストの目的

- テストの目的は、テスト対象のコンポーネントまたはシステム、テストレベル、ソフトウェア開発ライフサイクルモデルといった要因により異なります

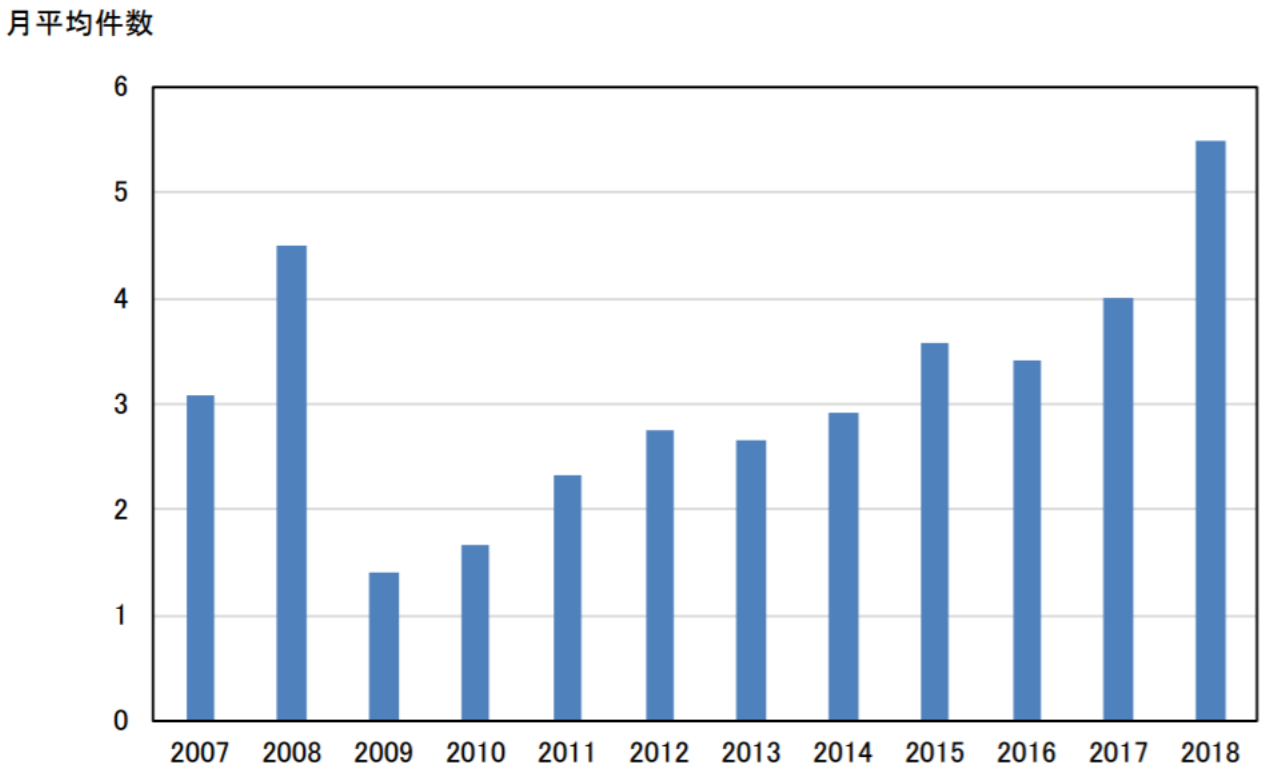
出典:Foundation Level シラバス

1.2 テストの必要性

ソフトウェア開発を取り巻く状況

- ソフトウェアが正しく動かないと社会的な問題になります
  - ソフトウェアは私たちの生活に欠かせないものになっています
  - ソフトウェアの規模や複雑さは増加傾向にあります
  - ソフトウェアが正しく動くことを保証するのが困難になっています
- 動かないソフトウェアは様々な問題を引き起こします
  - 経済的な損失
  - 時間の浪費
  - 信用の失墜
  - 傷害や死亡事故

過去に報道された情報システム障害件数の推移



発生件数	37	54	17	20	28	33	32	35	43	41	48	66

図１　情報システムの障害発生件数の推移

出典:情報システムの障害状況 2018年後半データ

報道された情報システムの事例（抜粋）

No.	システム名	発生日時（上段） 回復日時（下段）				影響	現象と原因	直接原因	情報源
		年	月	日	時				
1864	りそな銀行	2018	12	20	12時00分	インターネットバンキングやATM、店舗窓口など、すべてのシステムで他行口座に振り込めない状態になっていた	正午ごろに発生した障害は2時間程度で復旧した。原因は振り込みシステムの設定の不具合	設定ミス	りそな銀行お知らせ（2018.12.20） 日本経済新聞電子版（2018.12.20） 朝日新聞デジタル（2018.12.20） 読売オンライン（2018.12.20）
		2018	12	20	14時25分				
1865	国土交通省 車検証システム	2018	12	25	朝	全国に93ある運輸支局・自動車検査登録事務所で車検証を発行できなくなった	同日朝から情報処理システムに障害が発生し、同日12時45分に復旧した。各地で車検証の発行を再開している	不明	国土交通省報道・広報（2018.12.25） 読売オンライン（2018.12.25） 朝日新聞デジタル（2018.12.25）
		2017	12	25	12時45分				
1866	筑波大病院 ネットワーク	2018	12	27	8時50分	27日正午から夕方まで電子カルテが使えなくなった。患者約1千人の会計処理もできなくなり、82人の患者の画像検査ができなかった。救急の受け入れも一時断った	院内ネットワークシステムに27日朝から障害が発生し、同日9時ごろから電子カルテに検査画像が表示されないなどの不調が出始め、同日正午から夕方まで電子カルテが使えなくなった。検査画像が表示されない障害は、同日21時過ぎに回復した	不明	茨木新聞（2018.12.28） 朝日新聞（2018.12.29） 読売新聞（2018.12.29）
		2018	12	27	21時00分				

出典：情報システムの障害状況 2018年後半データ

テストの必要性

- 2002年6月に、米商務省の国立標準技術研究所（NIST）が次のレポートを出しました
  [参照] The Economic Impact of Inadequate Infrastructure for Software Testing

- 「ソフトウェアの欠陥が、アメリカに年間約7～8兆円の損害を与えている」
  [参照] Software Errors Cost U.S. Economy $59.5 Billion Annually
  NIST Assesses Technical Needs of Industry to Improve Software-Testing

  （参考）
  - 2002年のアメリカの人口約2.9億人、名目GDP：11兆ドル（1,300兆～1,400兆円）　1ドル＝117～135円
  - 2007年には3倍の20兆円の損害を与えている
  - 約3分の1は、最適なテストができていれば問題回避できたと言われている

出典：The Economic Impact of Inadequate Infrastructure for Software Testing　および、『欠陥ソフトウェアの経済学』

テストの貢献

要件定義

- テスト担当者が要件レビューやユーザーストーリーの洗練作業に関与することにより、これらの作業成果物の欠陥を検出できる
- 要件に対する欠陥の検出と除去を行うことにより、テストができない機能、または正しくない機能が開発されるリスクを低減できる

システム設計

- テスト担当者がシステム設計者と密接に連携して作業することにより、両者が設計とその設計をどうテストするかに対する理解を深めることができる
- 結果として、基本的な設計の欠陥が混入するリスクを低減でき、テストケースを早い段階で識別できる

プログラミング

- テスト担当者が開発担当者と密接に連携して作業することにより、両者がコードとそのコードをどうテストするかに対する理解を深めることができる
- 結果として、コードとテストケースに欠陥が混入するリスクを低減できる

テスト

- テスト担当者はリリース前にソフトウェアを検証および妥当性確認することにより、テストしなければ見逃してしまうかもしれない故障を検出し、故障の原因となる欠陥を除去するプロセス(すなわちデバッグ)を支援できる
- 結果として、ソフトウェアがステークホルダーのニーズに合致し、要件が満たされる可能性が高まる

出典:Foundation Level シラバス

品質の定義とその変遷 ①

- 「品質とは、要件に対する適合である。精密に測定可能で誤りは不可避ではない」（Zero Defect）
  by P.B. Crosby
  ← 一般の工業プロダクトよりの定義となっている

- 「品質は誰かにとっての価値である」
  by G.M.Weinberg
  ← ここで価値という言葉は、「人々はその要求が満たされるなら、喜んで対価を支払う」ということを意味している

- 「『製品の品質』は欧米の考えである狭義の『質』である。
  一方『広義の質』は仕事の質、サービスの質、情報の質、工程の質、部門の質、人の質、システムの質、会社の質等、これら全てを含め捉える。
  品質管理は「広義の質」について管理することを基本姿勢とする」
  by 石川 馨（日本的品質管理の父と呼ばれている）

出典：SQuBOK Guide V2

品質の定義とその変遷 ②

- 「品質は、ユーザーにとっての価値である」
  by 保田勝通
  ← 近代的な品質管理における品質の定義：品質は、ユーザーの満足度である
  出典：ソフトウェア品質保証の考え方と実際

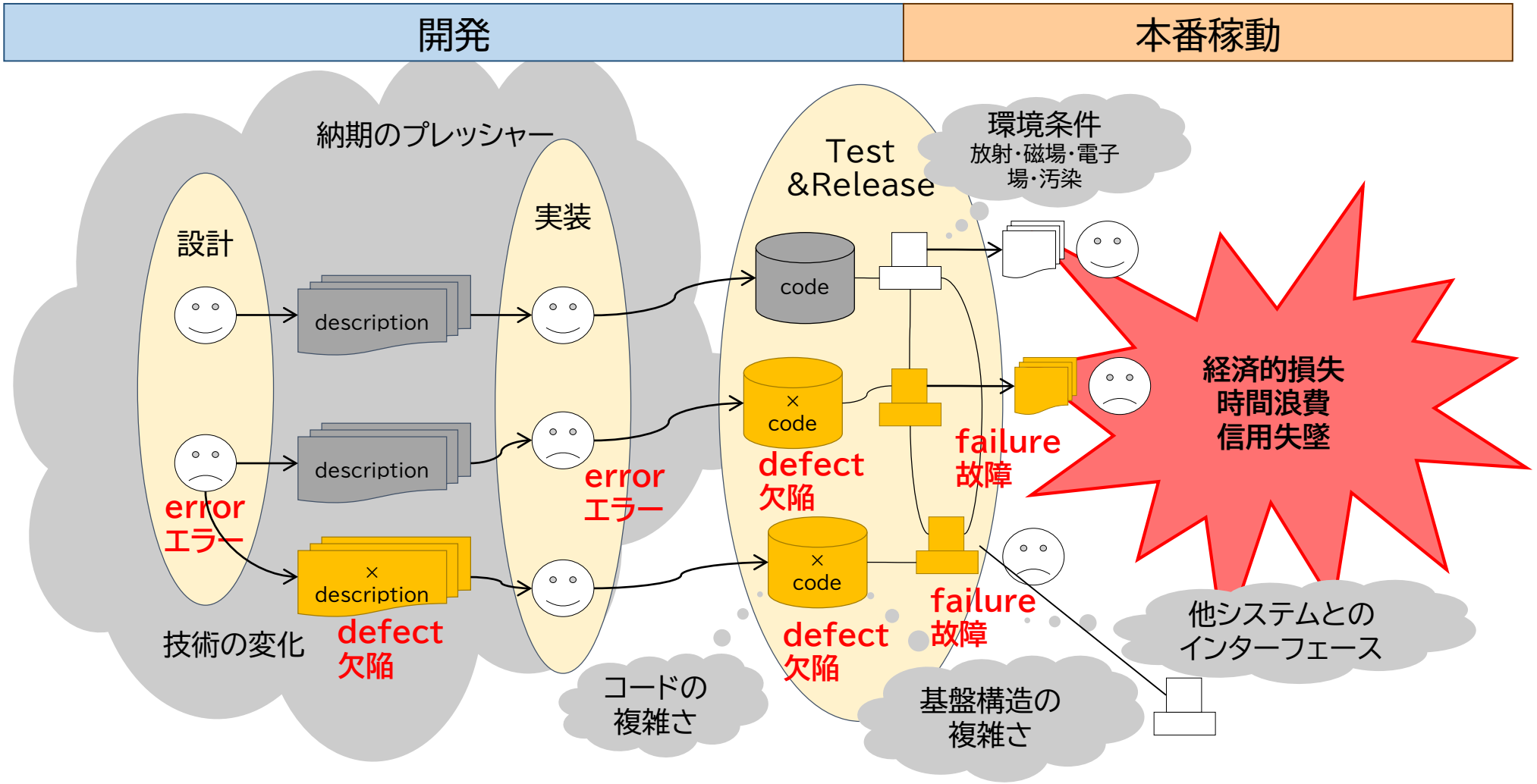
- 「品物又はサービスが、使用目的を満たしているかどうかを決定するための評価の対象となる固有の性質・性能の全体」- JIS Z 8101:1981
  ← 1999年の改訂の際、この定義は削除された

エラー、欠陥、故障 ①

エラー (error)	• 間違った結果を生み出す人間の行為
欠陥 (defect)	• 成果物に存在する、要件または仕様を満たさない不備または欠点
故障 (failure)	• コンポーネントやシステムが定義された範囲内で要求する機能を実行しないこと

出典:ISTQB Glossary

エラー、欠陥、故障 ②



エラー(人の誤り)・欠陥(仕様書・コード)・故障(顕在化した欠陥) ← テストで見つける

エラー、欠陥、故障への対応

エラーへの対応	• 欠陥を作ってしまった原因を究明し、原因に手を打つことで同種の欠陥を未然防止する
	  • 納期のプレッシャー、誤ったコミュニケーション
	  • 人間のもつ誤りを犯しやすい性質
	  • 経験不足または技術不足（不慣れな技術を含む）
	  • コード・設計・解決すべき根本的な問題・使用する技術の複雑さ
	  • システム間のインターフェースに関する誤解

欠陥への対応	• 欠陥レポート、デバッグ、リグレッションテスト、再リリースを行う • 故障の原因は直るが、欠陥の作りこんだ理由には迫っていないため、次のプロダクトで再発したり似た現象が発生する
故障への対応	• ユーザーの今起きている困りごとをなくす • 若化（リジュビネーション：リブートなど）や回避策の提示も有効な対応である • ただし、故障の原因は直っていないため、別のユーザーで同じ故障が起こる

出典：ISTQB Glossary

「バグ」は曖昧な用語です

- 組織によって、人によって、「バグ」は様々な意味を持ちます
  - 機能仕様、目標値とした品質特性、顧客要求を満たしていない
  - 関連する基準、規制、規則、計画、手順に適合していない
  - (ドキュメントに)記載されるべきことが記載されていない
  - 無駄なこと、不必要なことが記載されている
  - 読んでも分からない、不明な内容である
  - 内容が曖昧で分からないこと、何とでも受け取れる
  - 複数の異なる意味に受け取れる可能性のある内容である
  - 要素間で見ると矛盾している、不整合である
- 何を「バグ」とするかは組織で定義している場合もあります
- 使う場合には、確認してから対応します

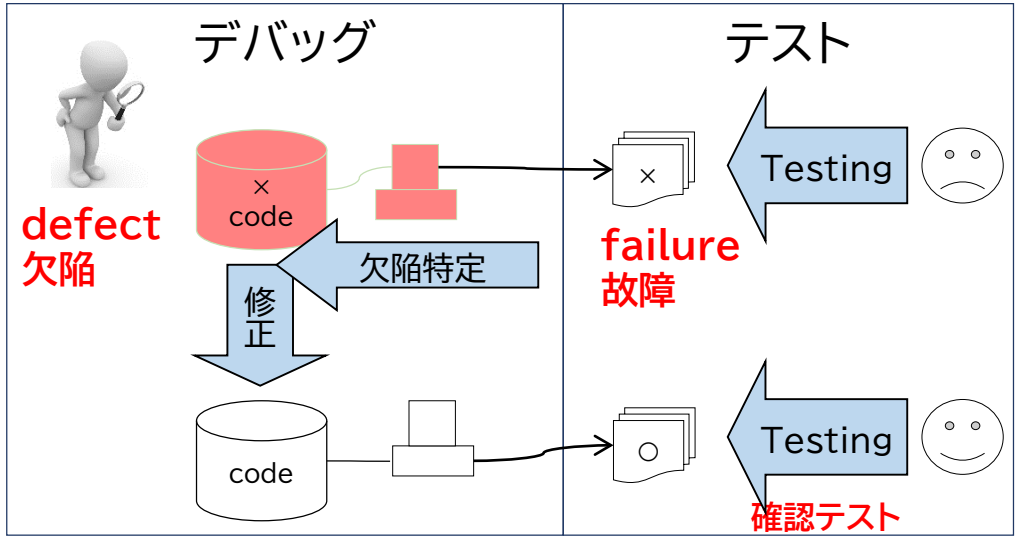
デバッグとテスト

デバッグ(debugging)

ソフトウェアの故障の原因を見つけて、分析して、取り除くプロセス

テスト(testing)

全てのライフサイクルを通じて実施する静的、動的なプロセスにおいて、成果物が特定の要件を満足するかを判定し、目的に合致することを実証し、欠陥を見つけるため、ソフトウェアプロダクトや関連成果物に対し、計画、準備、評価をすること



出典:ISTQB Glossary

1.3 テストの7原則

テストの7原則 ①

原則1　テストは欠陥があることは示せるが、欠陥がないことは示せない

テストにより、欠陥があることは示せるが、欠陥がないことは証明できない

テストにより、ソフトウェアに残る未検出欠陥の数を減らせるが、欠陥が見つからないとしても、正しさの証明とはならない

原則2　全数テストは不可能

すべてをテストすること（入力と事前条件の全組み合わせ）は、ごく単純なソフトウェア以外では非現実的である

全数テストの代わりに、リスク分析、テスト技法、および優先度によりテストにかける労力を集中すべきである

原則3　早期テスト（シフトレフト）で時間とコストを節約

早い段階で欠陥を見つけるために、静的テスト活動と動的テスト活動の両方をソフトウェア開発ライフサイクルのなるべく早い時期に開始すべきである

ソフトウェア開発ライフサイクルの早い時期にテストを行うことにより、コストを低減または削減できる

原則4　欠陥の偏在

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

出典：Foundation Level シラバス

テストの7原則 ②

原則5　殺虫剤のパラドックスにご用心

同じテストを何度も繰り返すと、最終的にはそのテストでは新しい欠陥を見つけられなくなる

ただし、自動化されたリグレッションテストの場合は、同じテストを繰り返すことでリグレッションが低減しているという有益な結果を示すことができる

原則6　テストは状況次第

状況が異なれば、テストの方法も変わる

- 例えば、アジャイルプロジェクトとシーケンシャルライフサイクルプロジェクトでは、テストの実行方法が異なる

原則7　「バグゼロ」の落とし穴

テスト担当者は可能なテストすべてを実行でき、可能性のある欠陥すべてを検出できると期待する組織があるが、原則2と原則1により、これは不可能である

また、大量の欠陥を検出して修正するだけでシステムを正しく構築できると期待することも誤った思い込みである

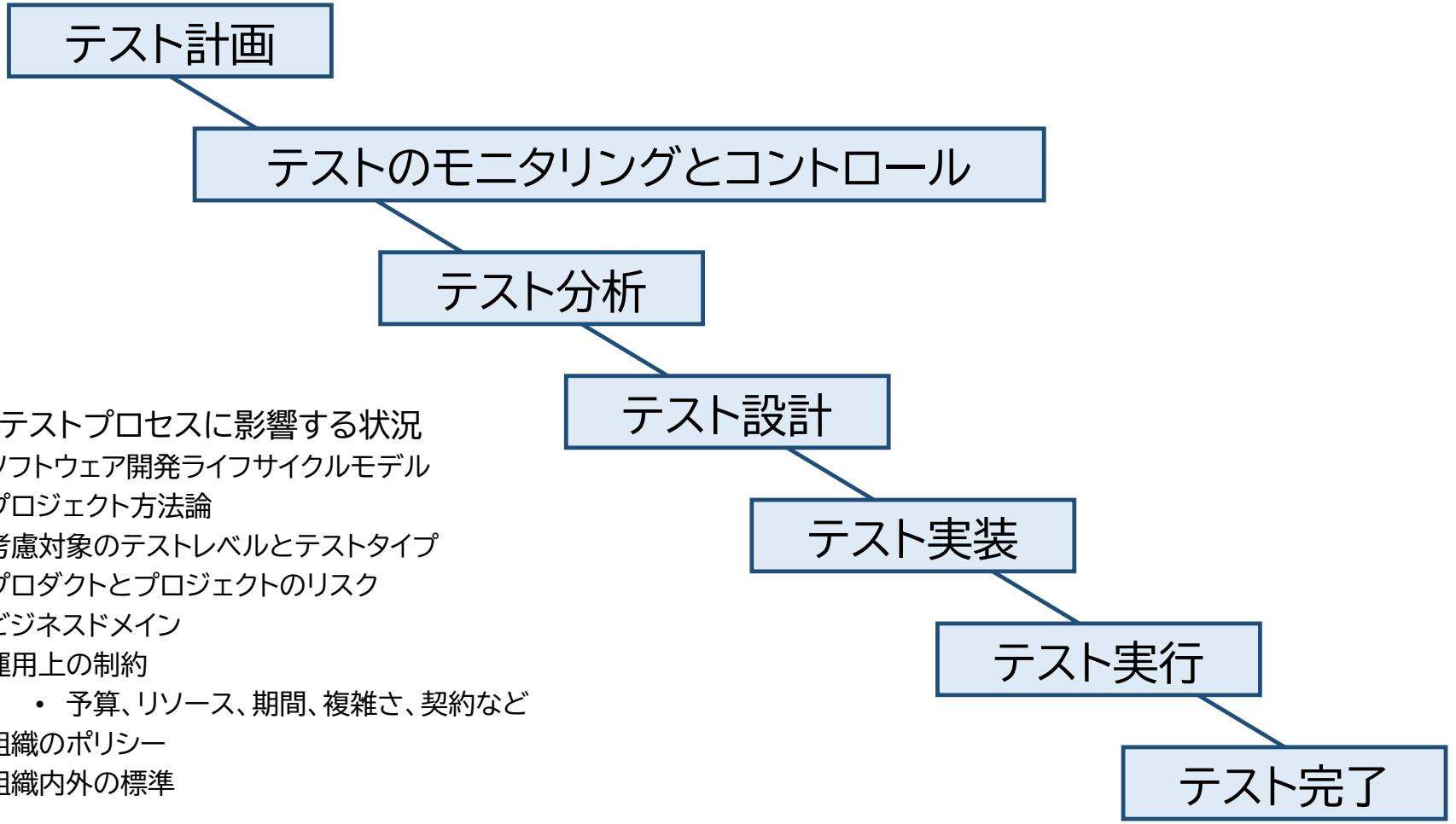
例えば、指定された要件すべてを徹底的にテストし、検出した欠陥すべてを修正しても、使いにくいシステム、ユーザーのニーズや期待を満たさないシステム、またはその他の競合システムに比べて劣るシステムが構築されることがある

出典：Foundation Level シラバス

1.4 テストプロセス

基本的なテストプロセス

- 主なテスト活動は次の通りです



- 組織のテストプロセスに影響する状況
  - ソフトウェア開発ライフサイクルモデル
  - プロジェクト方法論
  - 考慮対象のテストレベルとテストタイプ
  - プロダクトとプロジェクトのリスク
  - ビジネスドメイン
  - 運用上の制約
    - 予算、リソース、期間、複雑さ、契約など
  - 組織のポリシー
  - 組織内外の標準

出典:Foundation Level シラバス

テスト計画

- テストの使命を明らかにし、目的を定義します
  - 開発のゴールと直結していることが大切です
- テストの目的を達成するために、様々な制約を踏まえた上で、テストのアプローチを定義します
  - 例えば、品質目標と納期を踏まえ、適切なテスト技法とタスクを決め、要員を調達し、テストスケジュールを作成します
- 主な成果物はテスト計画書です
  - マスターテスト計画書やレベルテスト計画書など、複数のテスト計画書が作成されることもあります
  - テスト計画書には、テストベースに関する情報や終了基準が書かれています
  - テスト計画書は、モニタリングとコントロールのフィードバックに応じて更新します

出典:Foundation Level シラバス

テストのモニタリングとコントロール ①

- テストのモニタリング
  - テスト計画書に記載された予定と、テスト活動の実績を継続的に比較する活動です
  - テストケースの消化数や欠陥の発生数などのメトリクスを使用して、計画時の数値と実際の進捗を継続的に比較します
- テストのコントロール
  - テスト活動の実績が予定を逸脱していたり、その兆候が見えたりする場合、必要に応じて活動を軌道修正したり、テスト計画を見直します
- 終了基準の評価
  - 事前に設定しているカバレッジ基準に対して、どの程度網羅しているかテスト結果とログをチェックします
  - テスト結果とログに基づいて、コンポーネントまたはシステムの品質のレベルを評価します
  - 欠陥の対応状況を確認します
  - 追加テストの要否を判断します

出典:Foundation Level シラバス

テストのモニタリングとコントロール ②

- 主な成果物はテスト進捗レポートとテストサマリーレポートです
  - テスト進捗レポートを使って、ステークホルダーにテストの状況を伝えます
  - テスト進捗レポートには、計画からの逸脱や、テストの中止を決定するために必要な情報を含みます
  - すべてのテストレポートには、作成時点で提供可能になったテスト実行結果の要約を含みます

出典:Foundation Level シラバス

テスト分析 ①

- 「何をテストするのか」を決めます
  - テストベースを分析します
  - テスト可能なフィーチャーとフィーチャーのセットを識別します
  - 各フィーチャーのテスト条件を決めます
    - テスト技法を活用すると、重要なテスト条件の欠落を防止し、精度が高く正確なテスト条件を決められます
  - 各フィーチャーに優先度を割り当てます
    - 機能/非機能/構造の特性、他のビジネス/技術的要因、リスクのレベルを考慮します
- テストベースとテストアイテムを分析し、評価することで、様々な種類の欠陥を識別することができます
  - 例えば、曖昧、欠落、不整合、不正確、矛盾、冗長なステートメントのような欠陥です

出典:Foundation Level シラバス

テスト分析 ②

- 主な成果物は優先順位を付けたテスト条件です
  - テスト条件はテストベースとの間に双方向のトレーサビリティを確立することが望ましいです
  - 探索的テストのテスト分析では、テストチャーターが成果物になります
  - テストベースの欠陥を検出できた場合、欠陥レポートも成果物になります
- テスト分析は、なぜ行うのでしょうか？
  - 全数テストは組み合わせや操作順序が膨大で実施不可能だから
  - スケジュール、リソース、環境などの制約（前提条件）を踏まえて、もっとも実施すべきテストに絞るため
  - テスト条件（テストすべきもの・こと）を決定するため
- 分析対象には様々なものがあります
  - RFP、要求仕様書、アーキテクチャ定義書、設計仕様書
  - 標準文書（RFC、ISO/IEC標準）、業界標準、社内標準
  - 会議の議事録、設計メモ、メールなど

出典：Foundation Level シラバス

テスト設計 ①

- 「どのようにテストするか」を決めます
  - テストケースおよびテストケースのセットを設計します
    - 様々なテスト技法を活用します
  - テストケースに優先度を割り当てます
  - 必要なテストデータを識別します
  - インフラストラクチャーを識別し、テスト環境を設計します
  - テストツールを識別します

- 主な成果物はテストケースとテストケースのセットです
  - 具体的な入力データと期待結果の値を記載しない高位レベルテストケースは、テスト設計の主な成果物です
    - 具体的な値を記載しないことで、再利用が可能になります
  - テストケースはテスト条件との間に双方向のトレーサビリティを確立することが望ましいです

出典:Foundation Level シラバス

テスト設計 ②

- テストは、
  - ○　時間とお金の都合を考えて、QCDのバランスの取れた情報を提供します（＝どこで妥協するか）
  - ×　納期までの時間を優先し、残った時間でテストします
- テスト設計をすることで、最も適したテストを探ります
  - テストケースを合理的に少なくするための技法を選びます
    - 同値分割法、組み合わせテスト
  - 多くの欠陥が見つかるようにするための技法を選びます
    - 境界値分析、エラー推測、探索的テスト
  - テスト対象を漏れなくテストするための技法を選びます
    - 制御フローテスト、データフローテストなど
    - デシジョンテーブルテスト、状態遷移テスト、ユースケーステストなど

テスト実装 ①

- テスト実行に必要なものをすべて準備します
  - テストケースの実行順序を決めます
    - 効率よくテストできるようテストケースを組み合わせます
  - テスト手順を作成し、優先順位を割り当てます
  - テスト手順からテストスイートを作成します
  - スケジュール内で実行するテストスイートを調整します
  - テスト環境を構築します
    - テスト環境には、テストハーネス、サービスの仮想化、シミュレーター、およびその他のインフラストラクチャーアイテムが含まれます
  - 必要なものすべてが正しくセットアップされていることを確認します
  - テストデータを準備し、テスト環境に設定されていることを確認します

出典:Foundation Level シラバス

テスト実装 ②

- 主な成果物は次のとおりです
  - 低位レベルテストケース
  - テスト手順書
  - テストスイート
  - テスト実行スケジュール
  - テストベース、テスト条件、テストケース、テスト手順、テストスイートの間で、双方向のトレーサビリティを確立することが望ましいです

出典:Foundation Level シラバス

テスト実行 ①

- テスト実行スケジュールに従ってテストスイートを実行します
  - テストアイテムまたはテスト対象、テストツール、テストウェアのIDとバージョンを記録します
  - 手動、またはテスト自動化ツールを用いて、テストを実行します
  - 実行結果と期待結果を比較します
  - テストの実行結果(合格、不合格、ブロックなど)を記録します
  - 実行結果と期待結果が一致しない場合、テストの実行状況に基づいて欠陥を報告します
  - 原因を調査し、特定します
  - 不正(anomaly)に対応します
  - 不正への対応後、テスト活動を繰り返します
    - 修正したテストケースによるテスト、確認テスト、リグレッションテストの実行
  - テストベース、テスト条件、テストケース、テスト手順、テスト結果の間で双方向のトレーサビリティを検証し更新します

出典:Foundation Level シラバス

テスト実行 ②

- 主な成果物は次のとおりです
  - テストケースまたはテスト手順の結果が書かれた文書
    - テスト実行可能、合格、不合格、ブロック、意図的にスキップなど
  - 欠陥レポート
  - テストアイテム、テスト対象、テストツール、テストウェアに関するドキュメント
  - テストベース、テスト条件、テストケース、テスト手順、テスト結果の間で、双方向のトレーサビリティを確立することが望ましいです

出典:Foundation Level シラバス

テスト完了 ①

- テスト活動で得られたことをまとめます
  - すべての欠陥レポートがクローズしていることを確認します
    - 未解決として残されている欠陥がある場合には、変更要求またはプロダクトバックログアイテムを作成します
  - テストサマリーレポートを作成し、ステークホルダーに提出します
  - 再利用できるように、テスト環境、テストデータ、テストインフラストラクチャー、その他のテストウェアを整理し保管します
    - 必要があれば、テストウェアを他のステークホルダーに引き継ぎます
  - 完了したテスト活動から得られた教訓を分析し、次回に活かせるようにまとめます
    - 必要に応じて、テストプロジェクトで振り返りを行います
  - 収集した情報をテストプロセスの成熟度を改善するために利用します

出典:Foundation Level シラバス

テスト完了 ②

- 主な成果物は次のとおりです
  - テストサマリーレポート
  - 後続するプロジェクトまたはイテレーションを改善するためのアクションアイテム
  - 変更要求またはプロダクトバックログアイテム
  - 最終的なテストウェア

出典:Foundation Level シラバス

1.5 テストの心理学

人の心理とテスト

- テストによって欠陥を見つけることは、開発者に対する非難と解釈されることがあります
  - 開発者がテストで得られた情報を理解し、結果を受け入れることが困難になることもあります
- 欠陥や故障に関する情報は、建設的な方法で伝えるべきです
  - そうすることで、ステークホルダー間の緊張を緩和することができます
- コミュニケーションを適切に行う建設的な方法は次のとおりです
  - 対決姿勢ではなく、協調姿勢を取ります
  - テストのデメリットではなく、メリットを伝えます
  - 開発者への非難ではなく、中立的な立場で事実を伝えます
  - 開発者の否定的な反応に対して反応で返すのではなく、否定的に反応した理由を理解するようにします
  - 自分の言ったことを他人が理解しているか確認し、他人の言ったことを自分が理解しているかを確認します

出典:Foundation Level シラバス

開発担当者とテスト担当者のマインドセット

- 開発者のマインドセット
  - 解決策の設計と構築に大きな関心を持ち、解決策の誤りには関心を持ちません
  - 確証バイアスにより、自分自身が犯した誤りを見つけることは困難です
- テスト担当者のマインドセット
  - 好奇心
  - プロとしての悲観的な考え
  - 批判的な視点
  - 細部まで見逃さない注意力
  - 良好で建設的なコミュニケーションと関係を保つモチベーション
- 両者のマインドセットを組み合わせることで、より高いレベルのプロダクト品質を達成できるようになります

出典:Foundation Level シラバス

2. ソフトウェア開発ライフサイクル全体を通してのテスト

2.1 ソフトウェア開発ライフサイクルモデル

ソフトウェア開発における工程別の費用

- 各工程での費用

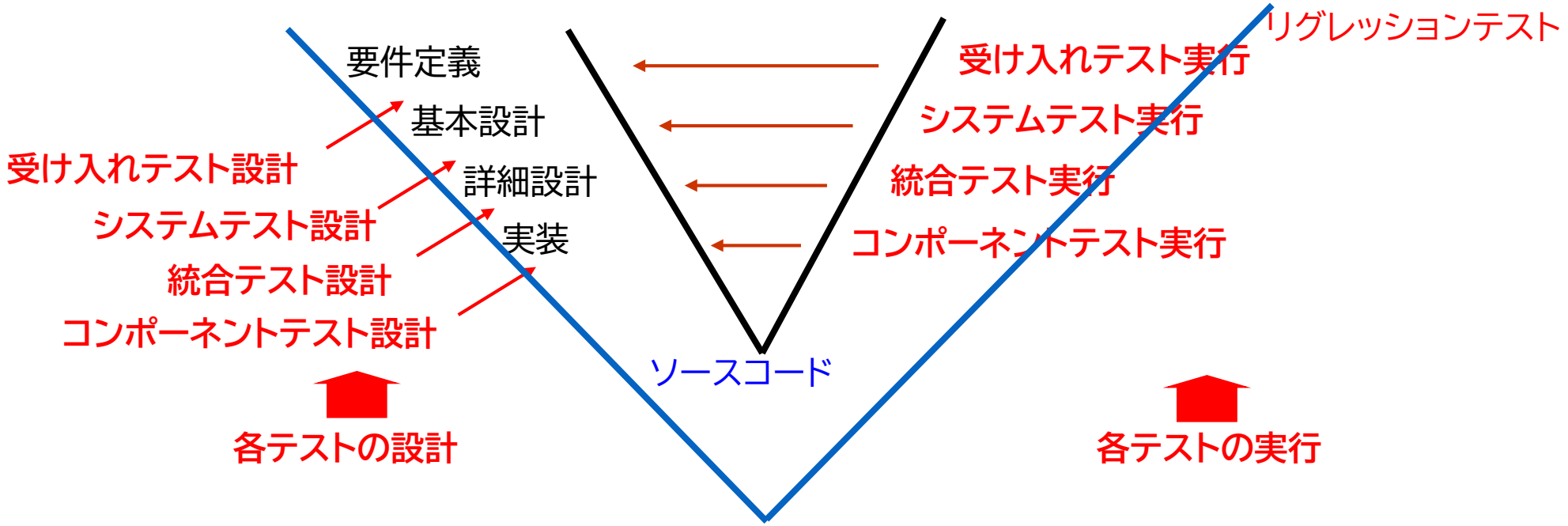
工程	コストの割合	「運用と保守」を除いた割合
要求分析	3%	9%
仕様作成	3%	9%
設計	5%	15%
コーディング	7%	21%
テスト	15%	46%
運用と保守	67%	–

テストが開発工程に占める費用割合は46%にもなります

出典：基本から学ぶソフトウェアテスト

テスト設計の前倒し（早期テストの原則）

- テスト設計を前倒すことで全体がよくなります
  - テストを「テスト設計」「テスト実装」「テスト実行」に分けてシフトレフト
  - テスト設計を、作りこみの段階で実施することによって、それぞれの工程とその成果物の質を見直し、向上させることができます



2.2 テストレベル

コンポーネントテスト ①

定義	• 個別にテスト可能なコンポーネントに焦点をあてるテストのこと • コンポーネント内のコードが正しいロジックかどうかをコードを開発した開発担当者がチェックし、予期しない例外についても確認する • コンポーネントテストのリグレッションテストを自動化して、変更が既存のコンポーネントを破壊していないという信頼を積み重ねていくことが重要である • 単体テスト、ユニットテスト、モジュールテストとも呼ばれる
目的	• リスクの軽減 • コンポーネントの機能的/非機能的振る舞いが設計および仕様通りであることの検証 • コンポーネント品質に対する信頼の積み上げ • コンポーネントに存在する欠陥の検出 • 欠陥がより高いテストレベルまで見逃されることの防止

出典:Foundation Level シラバス

コンポーネントテスト ②

典型的な欠陥と故障

- 正しくない機能(仕様記述とは異なる振る舞い)
- データフローの問題
- 正しくないコードとロジック

出典:Foundation Level シラバス

統合テスト ①

定義	• コンポーネントまたはシステム間の相互処理に焦点をあてるテストのこと

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

出典:Foundation Level シラバス

統合テスト ②

コンポーネント統合 典型的な欠陥と故障	• 不正なデータ、データ不足、不正なデータエンコーディング • インターフェース呼び出しの不正な順序またはタイミング • インターフェースの不整合 • コンポーネント間のコミュニケーション故障 • コンポーネント間のコミュニケーション故障の処理不在、または不適切な処理 • コンポーネント間で渡されるデータの意味、単位、境界の正しくない思い込み
システム統合 典型的な欠陥と故障	• システム間で一貫性のないメッセージ構造 • 不正なデータ、データ不足、不正なデータエンコーディング • インターフェースの不整合 • システム間通信による故障 • システム間通信処理不在、または不適切な処理による故障 • システム間で渡されるデータの意味、単位、境界の正しくない思い込み • 必須のセキュリティ規制への非準拠

出典:Foundation Level シラバス

システムテスト ①

定義	• システムが実行するエンドツーエンドのタスクと、タスクの実行時にシステムが示す非機能的振る舞いといったシステム全体の振る舞いや能力に焦点をあてるテストのこと
目的	• リスクの軽減 • システムの機能的/非機能的振る舞いが設計および仕様通りであることの検証 • システムが完成し、期待通りに動作することの妥当性確認 • システムの全体的な品質に対する信頼の積み上げ • 欠陥の検出 • 欠陥がより高いテストレベルまたは運用環境まで見逃されることの防止

出典:Foundation Level シラバス

システムテスト ②

典型的な欠陥と故障

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

出典：Foundation Level シラバス

受け入れテスト ①

定義	• システム全体の振る舞いや能力に焦点をあてるテストのこと • システムが、ユーザーのニーズ、要件、ビジネスプロセスを満足するかをチェックするための公式なテストのこと
目的	• システムの全体的な品質に対する信頼の積み上げ • システムの機能的/非機能的振る舞いが仕様通りであることの検証 • システムが完成し、期待通りに動作することの妥当性確認

出典:Foundation Level シラバス

受け入れテスト ②

典型的な欠陥と故障

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

出典:Foundation Level シラバス

テストレベル、テストフェーズ

テストレベル (test level)	• 具体的にインスタンス化したテストプロセス
テストフェーズ (test phase)	• テスト活動をプロジェクト中で管理(マネジメント)しやすいフェーズにまとめたセット • 例えば、あるテストレベルの実行活動

出典:ISTQB Glossary

2.3 テストタイプ

テストタイプ

テストタイプ(test type)

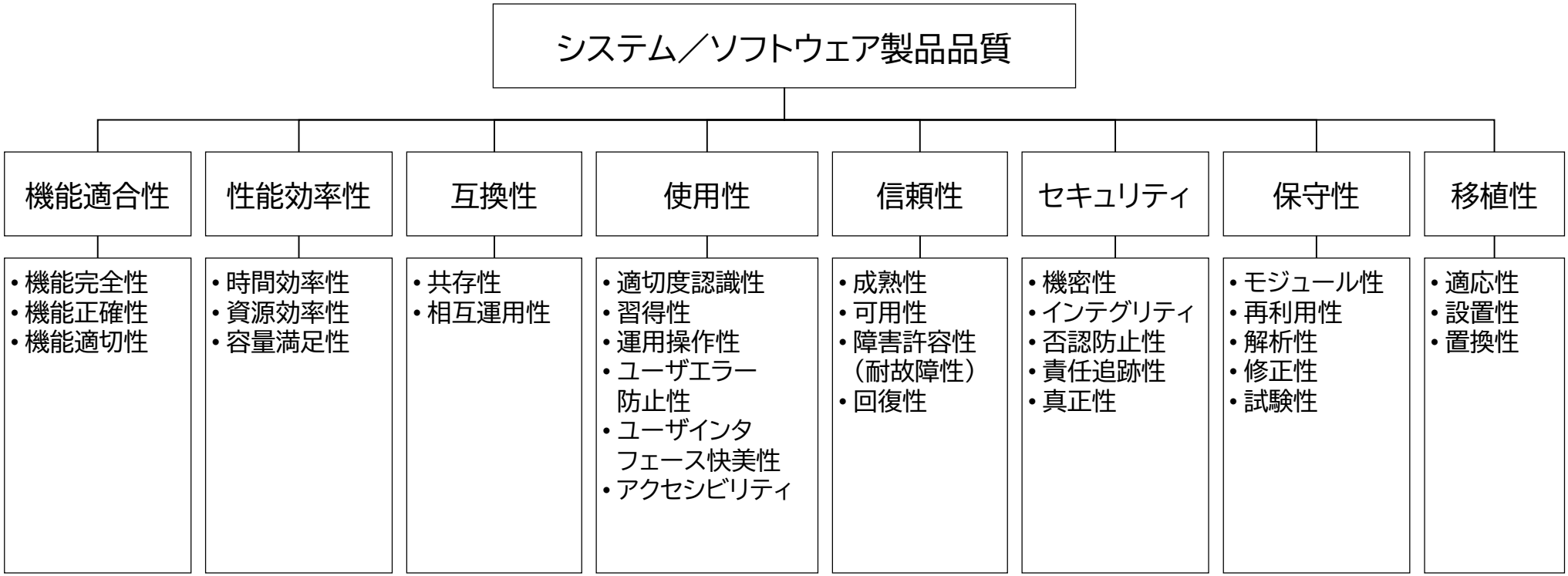
テストタイプは、以下に列挙する特定のテストの目的から見たソフトウェアシステム(あるいはシステムの一部分)の特性をテストするための活動を束ねたものである

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

出典:Foundation Level シラバス

品質特性

システム／ソフトウェア製品の品質モデル



SQuaREには「利用時の品質」と「データ品質」の品質モデルも定義されています

出典:ISO/IEC 25010:2011

機能テスト、非機能テスト

機能テスト	• 機能テストは、システムが「何を」すべきかをテストする • コンポーネントまたはシステムが実行する機能を評価する • ソフトウェアの振る舞いが関心事であり、コンポーネントまたはシステムの機能のためのテスト条件やテストケースをブラックボックス技法で導出できる • 機能カバレッジを用いてテストが十分かを計測できる
非機能テスト	• 非機能テストは、システムが「どのように上手く」振る舞うかをテストする • システムやソフトウェアの使用性、性能効率性、セキュリティといった特性を評価する • 非機能テストではテスト条件やテストケースを抽出するために、ブラックボックス技法を使う場合がある • 非機能カバレッジを用いてテストが十分かを計測できる

出典:Foundation Level シラバス

ホワイトボックステスト、変更部分のテスト

ホワイトボックステスト

- コンポーネントやシステムの内部構造や実装に基づいてテストを導出する
  - 例えば、内部構造とはコード、アーキテクチャー、ワークフロー、システム内のデータフローなど
- 構造カバレッジを用いてテストが十分かを計測できる
  - 例えば、コンポーネント内のテスト済みのコードの割合に基づきカバレッジを計測する(コードカバレッジ)

変更部分のテスト

- 確認テスト
  - 欠陥を修正したら、その欠陥により不合格となった全テストケースを新しいソフトウェアバージョンで再実行するテスト
  - 欠陥が確実に修正されたことを確認する
- リグレッションテスト
  - 意図しない副作用(リグレッション)を検出するテスト
  - 修正および変更が、既存のコンポーネントまたはシステムの振る舞いに意図せず影響を及ぼしていないことを確認する

※欠陥修正時に新たに作りこんだ欠陥を「リグレッション」と呼ぶ

出典:Foundation Level シラバス

テストタイプの具体例 ①

名称	概要
セキュリティテスト	システムのセキュリティポリシーを侵害しようとする行為によって脅威に対するシステムの脆弱性を評価する
資源効率性テスト	システム資源(メモリ、ディスク容量、ネットワーク帯域幅、コネクションなど)の使用状況を、事前に定義したベンチマークに対して評価する
保守性テスト	稼働中のシステムへの変更、または環境の変更が稼働中のシステムに与える影響をテストするために行う
使用性テスト	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
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

出典:Advanced Level テクニカルテストアナリスト シラバス

テストタイプの具体例 ②

名称	概要
信頼性テスト	ソフトウェア成熟性の統計的測定値を定常的に監視し、これをサービスレベルアグリーメント(SLA)として表されることのある期待する信頼性目標と比較する
障害許容性のテスト	想定外の入力値の処理に関するソフトウェアの障害許容性を評価する機能テスト(否定テストとも呼ぶ)に加えて、テスト対象のアプリケーションの外部で発生する障害に対するシステムの許容性を評価するテストが必要である
回復性テスト	ソフトウェアシステムがハードウェアまたはソフトウェアの故障から事前に決定した方法で回復し、正常な運用を再開できることを評価する

出典:Advanced Level テクニカルテストアナリスト シラバス

テストタイプの具体例 ③

名称		概要
性能テスト （時間効率性のテスト）		コンポーネントまたはシステムが、ユーザやシステムの入力に対して指定した時間内に、指定した条件で応答する能力に焦点を当てている
	ロードテスト	現実的な負荷の想定に対するシステムの処理能力に焦点を当てている
	ストレステスト	想定および指定した負荷の限界、および限界を超えた時、あるいはアクセス可能なコンピュータ能力や使用可能な帯域幅などのリソースの可用性が下がった時に、システムまたはコンポーネントが最大負荷を処理する能力に焦点を当てている
	拡張性テスト	現在の要件を超えた将来の効率性要件を、システムが満たす能力に焦点を当てている

出典：Advanced Level テクニカルテストアナリスト シラバス

テストタイプの具体例 ④

名称	概要
移植性テスト	ソフトウェアを目的の環境に、新規にあるいは既存環境から移植できる容易さに関連する
設置性テスト	対象とする環境にソフトウェアをインストールするために使用するソフトウェアと、文書化された手順に対して行う
共存性／互換性テスト	新規のあるいはアップグレードしたソフトウェアを、アプリケーションがすでにインストールされている環境に展開する場合、共存性/互換性テストを実行する必要がある
環境適応性テスト	全ターゲット環境(ハードウェア、ソフトウェア、ミドルウェア、オペレーティングシステムなど)で所定のアプリケーションが正しく機能するかどうかを確認する
置換性テスト	システム内のソフトウェアコンポーネントを他のコンポーネントに置換できる能力に焦点を当てる

出典:Advanced Level テクニカルテストアナリスト シラバス

2.4 メンテナンス（保守）テスト

メンテナンス（保守）テスト

- メンテナンステスト
  - 運用システム自体の変更や、稼動環境の変更が運用システムに与える影響をテストします
  - 変更が正しく適用されていることを評価します
  - システムの変更していない部分での副作用（リグレッション）を確認します
- メンテナンステストの実施範囲
  - 変更のリスクの度合い
    - 例えば、ソフトウェアの変更領域が他のコンポーネントまたはシステムとコミュニケーションする度合い
  - 既存システムの規模
  - 変更の規模

出典：Foundation Level シラバス

メンテナンスの影響度分析

- 影響度分析
  - 変更することにより既存システムがどの程度影響を受けるか分析し、影響を識別します
    - 変更により意図した結果が得られるか
    - 変更が影響するシステムの領域はどこか
    - 変更により予想される副作用があるか
  - 影響度分析は、既存のテストに対する修正が必要な箇所も識別します
- 影響度分析の難しさ
  - 仕様が最新版でない、または存在しない
  - テストケースが文書化されていない、または最新版でない
  - テストとテストベースとの間の双方向のトレーサビリティが維持されていない
  - ツールによる影響度分析のためのサポートが貧弱であるか、まったくない
  - ドメインおよび／またはシステムの知識を持つ担当者がいない
  - 開発時にソフトウェアの保守性が考慮されていない

出典：Foundation Level シラバス

3. 静的テスト

3.1 静的テストの基本

静的テスト

- 作業成果物を人手で調査したり、コードや他の作業成果物をツール主導で評価したりするテストです
  - 静的テスト　＝　レビュー　＋　静的解析
- コードを実行することなく、成果物をテストします

出典:Foundation Level シラバス

静的テストと動的テストの比較

- 静的テストと動的テストのよいところを組み合わせて計画し、早期から、段階的にテストを実行することが大切です

静的テスト	動的テスト

• 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、ウェブページなどあらゆる作業成果物はレビュー可能である
• すべて人力で実施可能
サポートするツールもある
• 故障(failure)自体ではなく、故障の原因である欠陥(defect)を探すという点で動的テストと違いがある
• 機能や処理(例えば要件)の抜けを見つけることができる
  • 動的テストではなかなか見つからない
	• 動作させて、おかしな振る舞いを見つけるサポートツールもある
	• 実行ファイル(ソフトウェアコード)と実行環境にて実施可能
	• 故障を検出する
	  • その後、開発者によるデバッグで欠陥を探す
	• 実機テストでは、実環境での動作・振る舞いが確認できる

出典:Foundation Level シラバス

静的テストのメリット

- 動的テストを実行する前に欠陥を早期に検出できます
  - 欠陥の検出と修正をより効率的に行うことができます
- 動的テストでは容易に検出できない欠陥を識別できます
- 設計時またはコーディング時に欠陥が混入するのを防止できます
  - 要件の不整合、曖昧性、矛盾、欠落、不正確性、冗長性を明らかにします
- 開発の生産性を向上できます
  - 設計の改善、保守性の高いコードなど
- 開発とテストにかかるコストと時間を削減できます
  - ライフサイクルの初期に検出した欠陥の修正は、テストケースを実行して検出した欠陥を修正する場合に比べて、はるかに安価になります
- 全体的な品質コストを削減できます
  - ライフサイクルの終盤または本番リリース後に検出される欠陥数が減少します
- レビューに参加することでメンバー間のコミュニケーションが改善できます

出典:Foundation Level シラバス

静的テストで検出できる欠陥の特徴

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

出典:Foundation Level シラバス

3.2 レビュープロセス

レビュー

- レビューとは静的テストの種類の一つです
- 1人もしくは複数のレビューアが成果物やプロセスを評価して懸念事項を検出し、改善策を提供します

出典:ISTQB Glossary

作業成果物のレビュープロセス ①

計画	• レビューの範囲を定義し、工数と時間を見積る • レビュー特性(レビュータイプ、役割など)を識別する • レビューの参加者を選び、役割を割り当てる • 開始基準と終了基準を定義する • 開始基準が満たされていることをチェックする
レビューの開始	• レビュー資料および関連資料を配布する • レビューの範囲、目的、プロセス、役割、レビュー資料を参加者に説明する • 参加者からのレビューについての質問に答える
個々のレビュー	• レビュー資料のすべてまたは一部をレビューする • 潜在的な欠陥、提案、質問を書き出す

出典:Foundation Level シラバス

作業成果物のレビュープロセス ②

懸念事項の共有と分析	• 識別した潜在的な欠陥を分析する • 分析結果に対して担当者を割り当て、ステータスを付ける • 品質特性を評価し、文書化する • レビューで見つけた欠陥などの結果を終了基準に対して評価し、レビュー判定を行う
修正と報告	• 変更が必要な欠陥について欠陥レポートを作成する • 指摘を受けたレビュー資料を修正する • 欠陥について議論する • 欠陥のステータスを更新する • メトリクスを収集する • 終了基準が満たされていることをチェックする • 終了基準に到達したときに、作業成果物を受け入れる

出典:Foundation Level シラバス

形式的レビューでの役割と責務 ①

作成者	• レビュー対象の作業成果物を作成する • レビュー対象の作業成果物の欠陥を修正する（必要な場合）
マネージャー	• 責任を持ってレビューの計画を行う • レビューの実行を決定する • 担当者、予算、時間を割り当てる • 費用対効果を継続的にモニタリングする • 不適切な結果が発生した状況に対してコントロールをする
ファシリテーター（モデレーター）	• 効果的なレビューミーティングを運営する（開催時） • 様々な意見の調整を行う（必要な場合） • レビューの成功を左右する重要な役割を果たす

出典：Foundation Level シラバス

形式的レビューでの役割と責務 ②

レビューリーダー	• レビューに関して全体的な責任を持つ • 参加者を人選し、レビューを実施する期間と場所を決定する
レビューア	• レビュー対象の作業成果物の欠陥を識別する • 業務や技術など特定分野の専門家、プロジェクトメンバーなどステークホルダーが、それぞれに異なる観点でレビューを行う
書記(記録係)	• 個々のレビュー活動の期間に見つかった潜在的な欠陥を照合する • (開催時に)レビューミーティングで見つかった新しい潜在的な欠陥、未決事項、決定事項を記録する

出典:Foundation Level シラバス

レビュータイプ ①

非形式的レビュー

（バディチェック、ペアリング、ペアレビュー）

- 決まったプロセスはなく、お金や時間をかけずにある程度の効果を出す
- レビューアの質で効果の大小が決まる
- 主な目的
  - 潜在的な欠陥の検出

ウォークスルー

- 作成者が議事進行、机上で処理をたどる
  - シナリオ、ドライラン、シミュレーションの形態を取る場合がある
- レビュー対象物の作成者が進行する
- 主な目的
  - 欠陥の発見
  - ソフトウェアプロダクトの改善
  - 異なる実装方法の検討
  - 標準や仕様への準拠の評価

出典：Foundation Level シラバス

レビュータイプ ②

テクニカルレビュー	• レビューアは、技術の専門家が行い、経験を積んだファシリテーターが主導するのが理想である
	• 新しいアイデアの創出、作業成果物の改善、異なる実装方法の検討を行う
	• 主な目的
	  • 合意の獲得
	  • 潜在的な欠陥の検出

インスペクション	• 作成者ではなく、経験を積んだ進行役がレビューの議事を主導する
	• ルールやチェックリストに基づいたプロセスで進行し、形式に沿ったドキュメントを作成する
	• 開始基準と終了基準が指定され、書記は必須である
	• 主な目的
	  • 潜在的な欠陥の検出
	  • 作業成果物の品質の評価と信頼の積み上げ
	  • 作成者の学習と根本原因分析による将来の類似欠陥の防止

出典:Foundation Level シラバス

レビュー技法 ①

アドホック （決まりなし）	• 準備を必要としない場合によく行う • レビューアは、このレビューに関するタスクのガイダンスをほとんどまたはまったく提供されない状態でレビューする • レビューアは作業成果物を順番に読んで懸念事項を識別するごとに記録に残す • レビューアのスキルに大きく依存し、複数のレビューアから重複した懸念事項が多く報告されてしまう場合がある
チェックリストベース （チェックリスト使用）	• 体系的に行われる • レビューアは、レビュー開始時に配布されるチェックリストを使用して懸念事項を検出する • レビューに使用するチェックリストは、経験から導出した起こりえる欠陥に基づく一連の質問を列挙したものである • 主な利点は、典型的な懸念事項の種類を体系的にカバーできることである

出典：Foundation Level シラバス

レビュー技法 ②

シナリオとドライラン
（シナリオを用いて、ドライラン）

- レビューアに、作業成果物を読むための構造化されたガイドラインを提供する
  - レビューアはシナリオ（作業成果物を読むための構造化されたガイドライン）を使用して、作業成果物の期待する使い方に基づいて、作業成果物に対して「ドライラン」予行演習、空運転）を行う
  - このシナリオは、単純なチェックリスト項目よりも、特定の種類の欠陥を検出するためのよいガイドラインとなる

ロールベース
（役割の観点）

- レビューアは個々のステークホルダーの役割の観点から作業成果物を評価する
- 典型的な役割は次のとおり
  - 特定のエンドユーザーの種類（熟練者、初心者、年配者、子供など）
  - 組織内の特定の役割（ユーザーアドミニストレーター、システムアドミニストレーター、性能テスト担当者など）

出典：Foundation Level シラバス

レビュー技法 ③

パースペクティブベース
（様々な立場からの視点
＋ 成果物の作成）

- レビューアはそれぞれに異なるステークホルダーの観点でレビュー活動を行う
  - レビューアごとに異なる観点を持つことで、より深く掘り下げたレビューが可能になる
  - レビューア間における重複が少なくなる
  - 典型的なステークホルダーの観点
    - エンドユーザー
    - マーケティング担当者
    - 設計者
    - テスト担当者
    - 運用担当者　など
- チェックリストを使用することがある

出典：Foundation Level シラバス

レビューの成功要因

- レビューの主な成功要因
  - 達成する目的、計測可能な終了基準、およびソフトウェア成果物と参加者の種類とレベルに応じてレビュータイプを選択する
  - 適切なレビュー技法を1種以上使用する
  - 大きなドキュメントは小さく分割してレビューする
  - 参加者に十分な準備時間を与える
    - レビューのスケジュールは適切に通知する
  - マネージャーがレビュープロセスを支援する
  - レビューの目的に対して適切な人たちに関与させる
  - テスト担当者は、有効なテストを早期に準備する
  - 見つかった欠陥は客観的な態度で確認、識別、対処をする
  - 自分の言動が退屈感、憤り、敵意だと受け取られないように気を付ける
  - レビューアに十分なトレーニングを提供する

出典:Foundation Level シラバス

4. テスト技法

4.1 テスト技法のカテゴリ

テスト設計の視点

ユーザの要求に合うか、利用時に満足しているか

User-oriented
ユーザ指向

ホワイトボックステスト

仕様通り実装しているか

Spec-oriented
要件指向

バランス
が大切

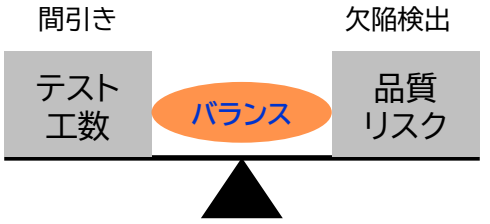
Design-oriented
コード指向

ロジックは正しいか、実行されないコードはないか

ブラックボックステスト

Fault-oriented
フォールト指向

発生しやすい、あるいは、推測される欠陥が検出されるか



※ oriented = view

※ Ostrandの4個のビューより

テスト技法の選択 ①

- テスト技法を選択する場合、様々な要素に依存します
  - システムの種類や複雑さ
  - 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とスキル
  - テスト技法を使用した経験
  - 利用できるツール
  - スケジュールと予算
  - ソフトウェア開発ライフサイクルモデル
  - ソフトウェアの想定される使用方法
  - 想定される欠陥の種類

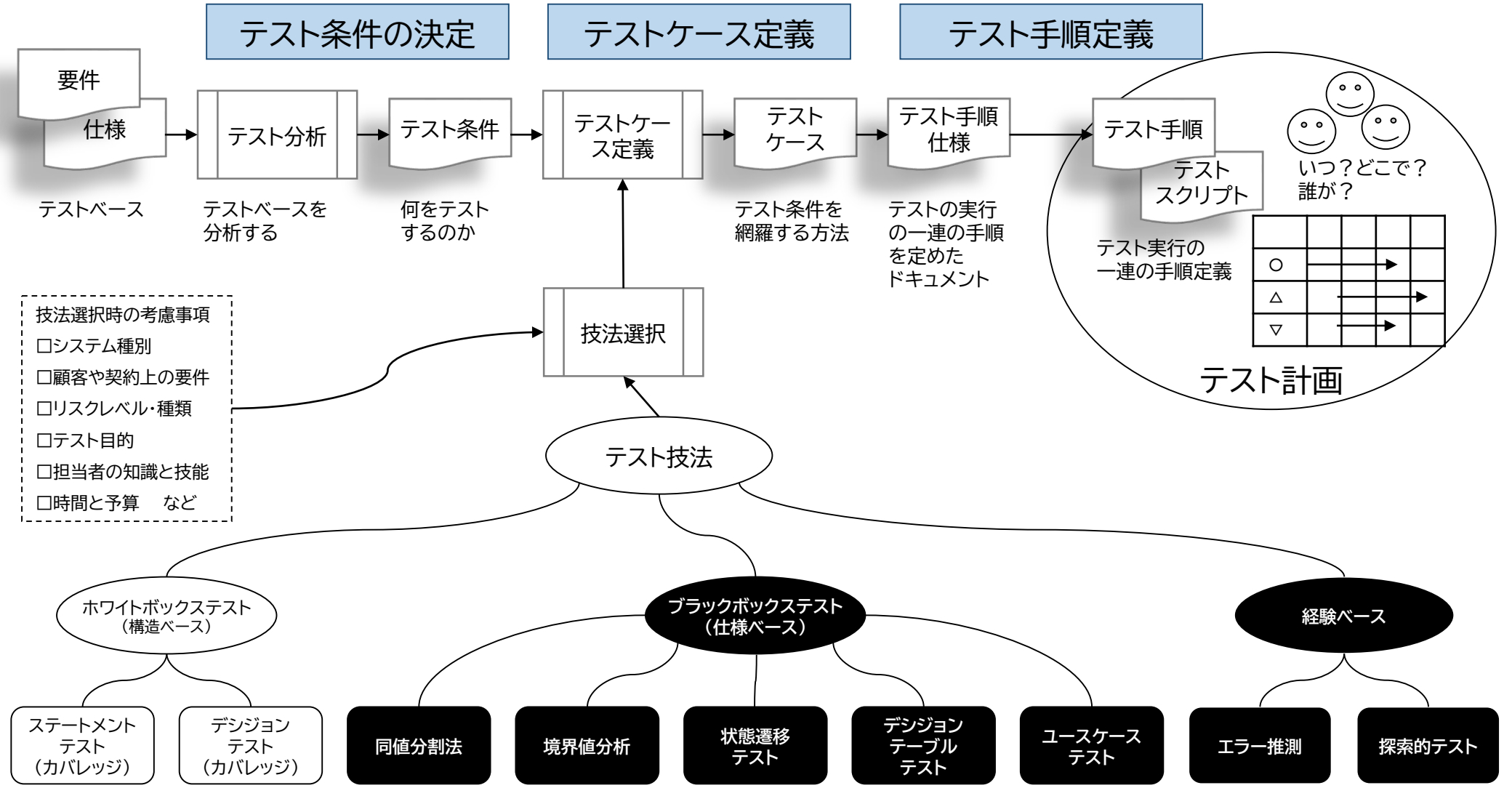
出典:Foundation Level シラバス

テスト技法の選択 ②

- 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よい場合もありますし、すべてのテストレベルで使える技法もあります
- テストケースを作成する際、テスト対象を適切に網羅するために、テスト技法を組み合わせて使います

出典:Foundation Level シラバス

テスト技法のカテゴリと特徴



4.2 ブラックボックステスト技法

演習　テスト設計の必要性

- 三角形の判定仕様
  - このプログラムでは、ユーザが入力ダイアログから3つの整数を入力します
  - この3つの値は、それぞれ三角形の3辺の長さを表すものとします
  - プログラムは、三角形が、不等辺三角形、二等辺三角形、正三角形のうちのどれであるかを決めるメッセージを出力します

- プログラムをテストするのに十分と思われる一連のテスト内容(何を確認するか？)を書きなさい

20～25分

出典:ソフトウェア・テストの技法

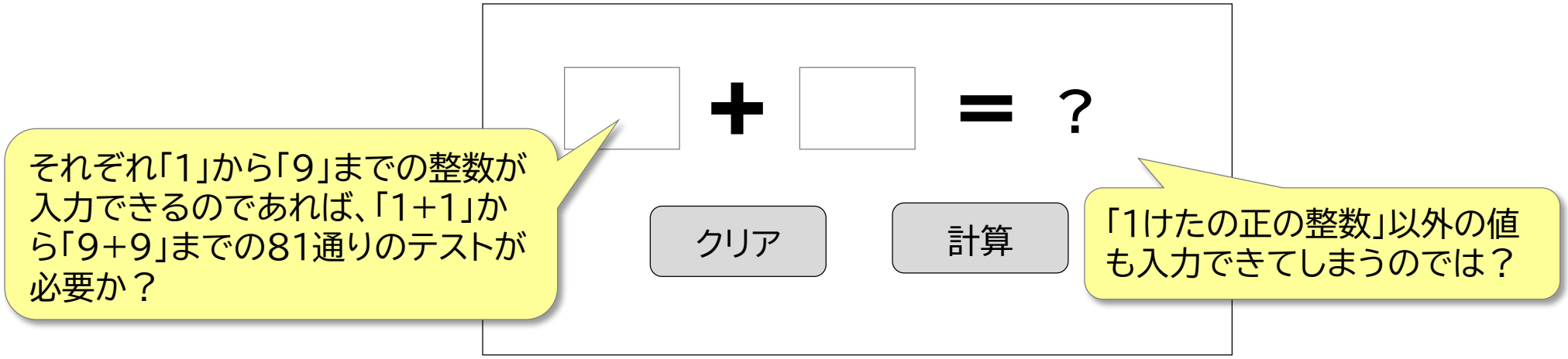
代表的なブラックボックステスト技法

同値分割法	• 同等に処理されると想定したデータすべてを同じパーティションに振り分け、各パーティションから少なくとも1個の値を選んでテストする
境界値分析	• 同値分割法の拡張で、パーティションが数値または順序付け可能な値で構成される場合に、パーティションの最小値と最大値（または最初の値と最後の値）を選んでテストする
デシジョンテーブルテスト	• 条件と結果的に起きる動作を表の行とし、条件の一意な組み合わせとその結果として実行する動作により表される判定の規則を表の列とし、規則ごとにテストする
状態遷移テスト	• 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
ユースケーステスト	• 1個のサブジェクト（コンポーネントまたはシステム）と1個以上のアクターとの相互作用による振る舞いを表すユースケースのシナリオをテストする
組み合わせテスト	• 複数の値を持つ複数のパラメータを含むソフトウェアに対して、組み合わせに含めるアイテムの数を選択して、作成した組み合わせをテストする

同値分割法 ①

- 同等に処理される、または同様の結果をもたらすと想定したデータすべてを、同じパーティションに振り分け、各パーティションから少なくとも1個の値を選んでテストする技法です
  - 同等に処理される、または同様の結果をもたらすと想定したデータを「同値」と呼びます
  - 有効な値の同値パーティションを「有効同値パーティション」と呼びます
  - 無効な値の同値パーティションを「無効同値パーティション」と呼びます

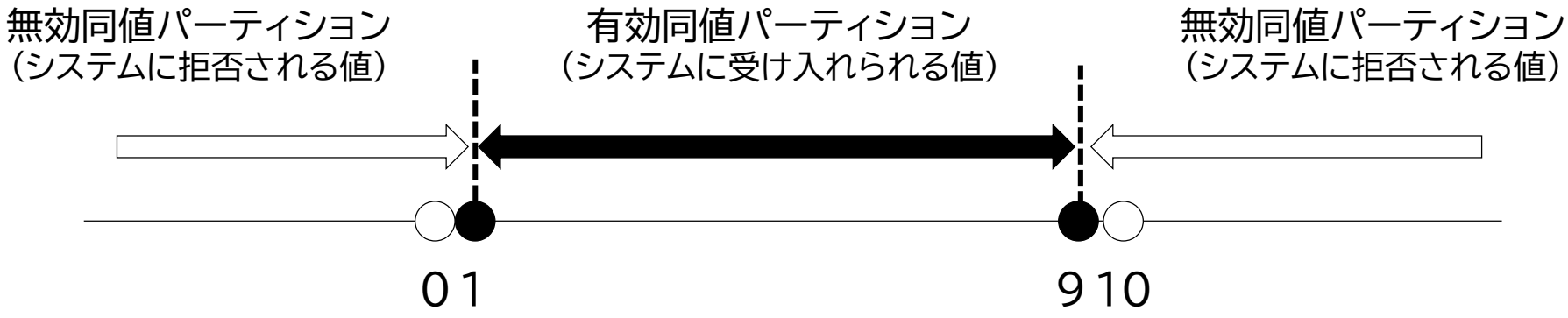
（例）1けたの正の整数2個を足し算するプログラム



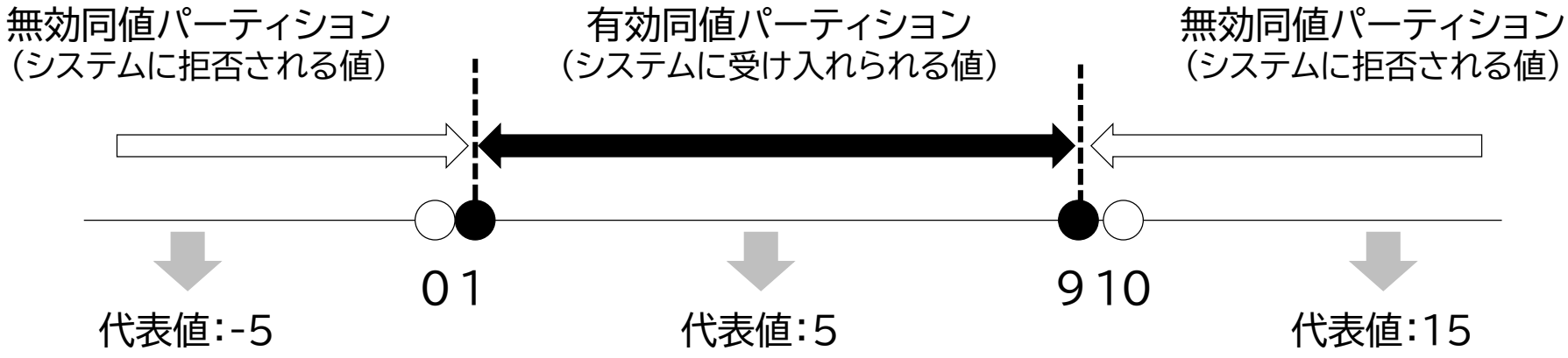
出典：Foundation Level シラバス

同値分割法 ②

- 同等に処理されると想定したデータすべてを、同じパーティションに振り分けます



- 各パーティションから、少なくとも1個の値を選んでテストします



出典:Foundation Level シラバス

同値分割法 ③

- 同値分割法のポイント
  - 同値パーティションを考えることで、無駄なテストを抑え、テスト回数の大幅な削減が可能です
  - 様々な同値パーティションがあります
    - 引数
    - 返り値
    - 内部値
    - 仕様上の範囲
    - 型や条件文の値の範囲（0～255など）
    - 文字コードの範囲
    - ファイルの数や大きさ
    - 時間に関する値
      - イベントの前後、タイミング、タイムアウト、データの入力や転送時間など
    - メモリサイズ

    など

出典：Foundation Level シラバス

同値分割法 ④

- 同値分割法のポイント
  - 出力結果を同値に分割した場合

（例）2個の入力をともに「3」とした場合
3 ＋ 3 ＝ 6

答え（出力）は6

2個の入力をともに「5」とした場合
5 ＋ 5 ＝ 10

答え（出力）は10

表示位置がずれるかも

1桁のときは「2桁めに0」が表示されてしまうかも

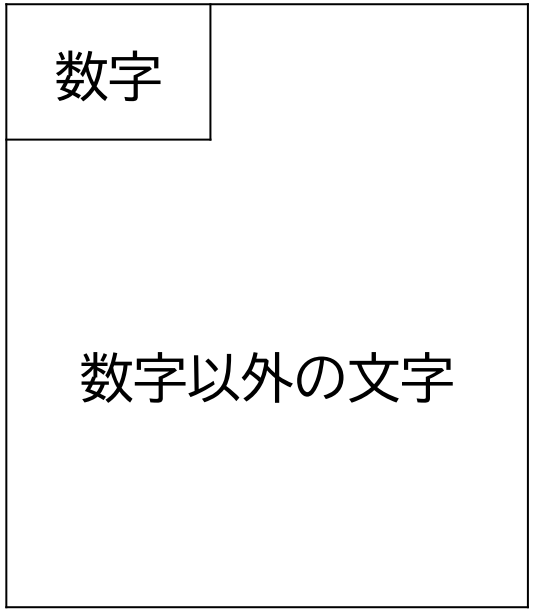
答えが1桁の場合と2桁の場合で別々のパーティションとするなど

出典：Foundation Level シラバス

同値分割法 ⑤

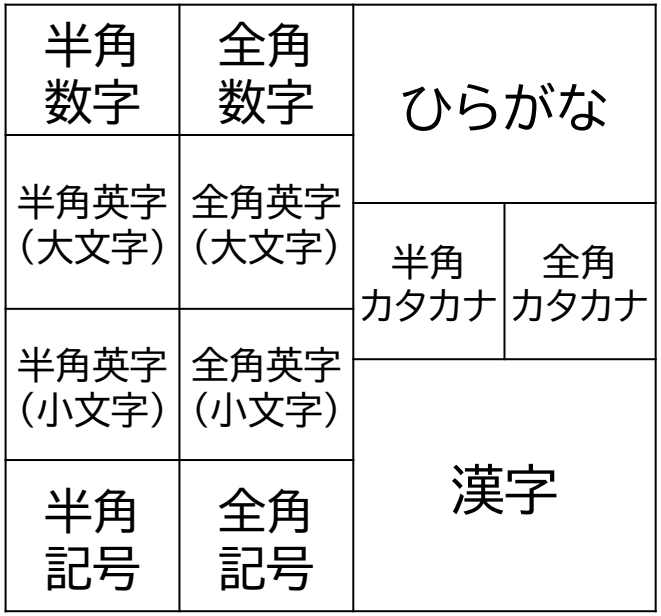
- 同値分割法のポイント
  - ズームイン、ズームアウト
    - パーティションは、必要に応じて細かく分割(ズームイン)する場合もあれば、粗く分割(ズームアウト)する場合もあります

文字種の分割例①



文字種の分割例②

文字種の分割例③

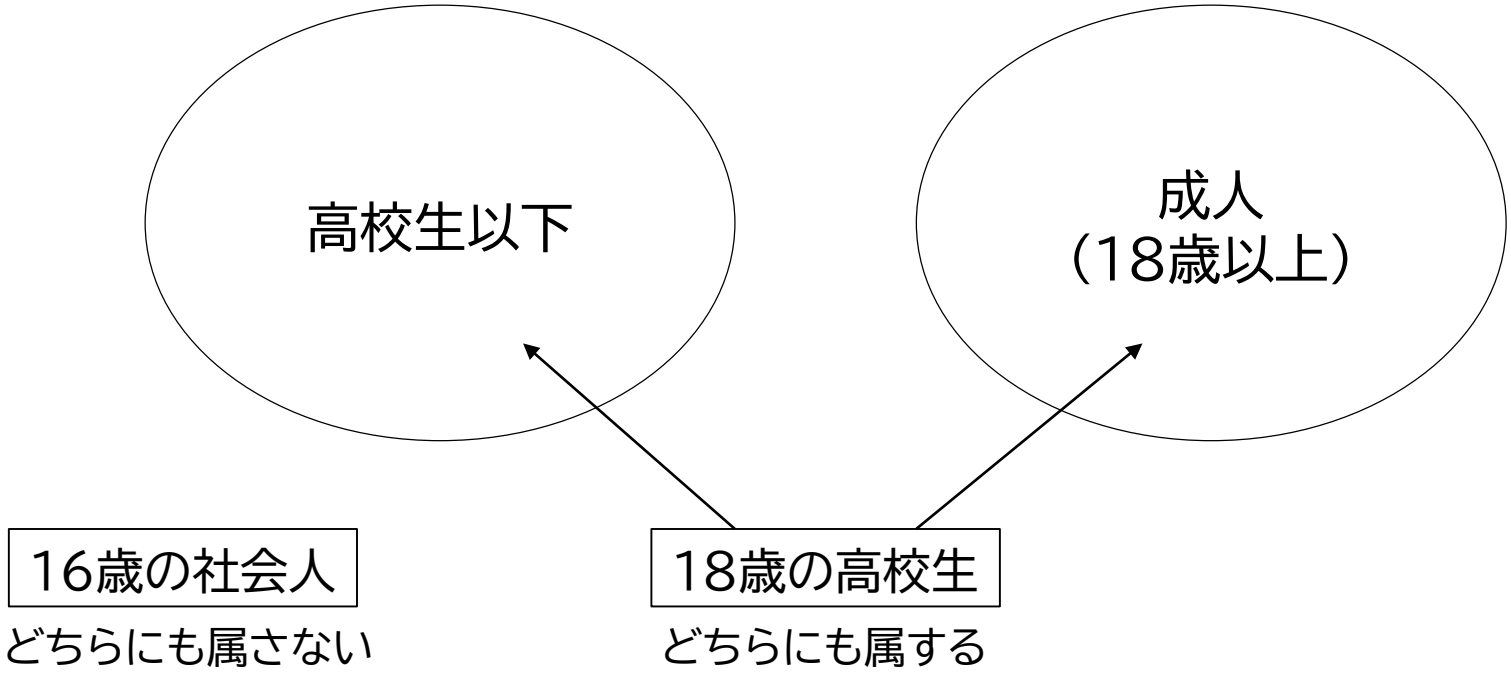


出典:Foundation Level シラバス

同値分割法 ⑥

- 同値分割法のポイント
  - テストに用いる値は、必ず1個の同値パーティションだけに属する必要があります

誤った同値分割の例

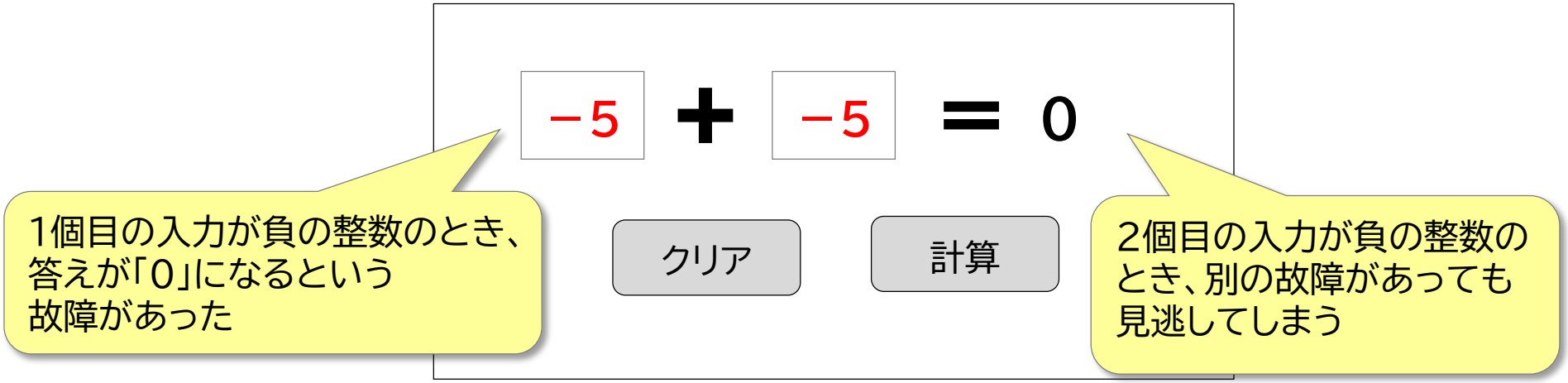


出典：Foundation Level シラバス

同値分割法 ⑦

- 同値分割法のポイント
  - 無効同値パーティションは、他の無効同値パーティションとは組み合わせずに、単独でテストします

（例）1けたの正の整数2個を足し算するプログラム



出典：Foundation Level シラバス

演習　同値分割法

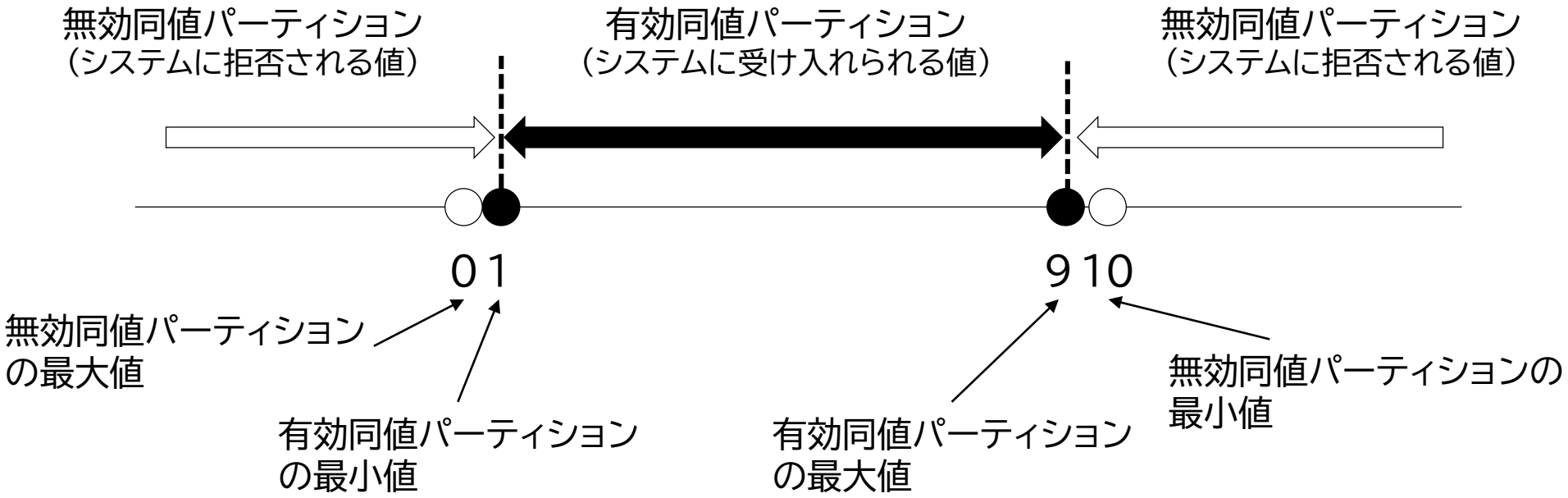
- JR運賃の区分
  - **区分　　説明**
  - おとな　　12歳以上(12歳でも小学生は「こども」です)
  - こども　　6歳～12歳未満(6歳でも小学校入学前は「幼児」です)
  - 幼児　　1歳～6歳未満
  - 乳児　　1歳未満

- 同値パーティションを挙げなさい
- 各同値パーティションの代表値と期待結果(どの区分と判断されるか)を挙げなさい

5～10分

境界値分析 ①

- 同値分割法の拡張で、同値パーティションの最小値および最大値（または最初の値と最後の値）を選んでテストする技法です
  - 同値分割法の拡張です
    - パーティションが、数値または順序付け可能な値で構成されるときに使用できます
  - 境界の値の定義は、要求分析から実装のどの段階でも勘違いしたり、間違えたりしやすい、という経験則に基づいています



境界値分析 ②

On	• 着目している境界値 • 境界上の値で、ドメインの内側にある場合、外側にある場合とがある • 0≦x＜4なら「0」と「4」 • Onポイントが有効値であれば、Offポイントは最も近傍の無効値
Off	• Onポイントに隣接する境界にない値 • 識別可能な最小の量だけドメイン境界から外側にずれた近傍の値 • 0≦x＜4なら「-1」と「3」 • Offポイントが無効値であれば、Onポイントは最も近傍の有効値
In	• すべての境界条件を満たすが境界にはない値 • ドメインの内側にあり、OnでもOffでもない値 • OnやOff以外の、範囲内の任意の値 • 0≦x＜4なら「1」か「2」
Out	• いずれの境界条件も満たさない値 • ドメインの外側にあり、OnでもOffでもない値 • 範囲外の値 • 0≦x＜4なら「-10」や「10」など

境界値分析 ③

- 境界値分析のポイント
  - 境界に対する記述ミスや解釈ミスに起因する欠陥をピンポイントで狙います
    - 例)ソースコードでの不等号の実装ミス(「＞」と「＞＝」など)
    - 例)仕様における境界に関する曖昧な記述
      - システム稼働時間　8:00～20:00
      - システム停止時間　0:00～8:00、20:00～24:00

➡ 8:00、20:00は稼働？停止？

    - 狙う値の例
      - 入力値の下限-1、下限、上限、上限+1
      - 配列の最大長、最大長+1
      - 条件式(等式・不等式)で指定された値、およびその前後の値

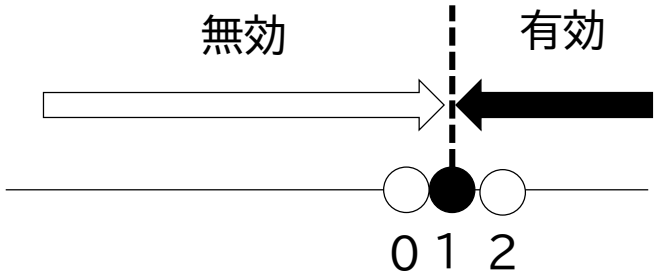
境界値分析 ④

- 境界値分析のポイント
  - 数値の範囲以外にも適用できます
    - 数値属性を持つ非数値変数(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負荷の耐久性
  - 非機能欠陥を発見します
    - 負荷の耐久性
      - システムは10,000人の同時ユーザーをサポートする

境界値分析 ⑤

- 境界値分析のポイント
  - 境界あたり3個の境界値を識別する考え方もあります

境界上および境界の前後



仕様では「1以上」に対して「0以下」と書かれることが多いが、「0以下」ではパーティションが隣接しない

「1以上」に隣接するパーティションは必ず「1未満」として考える

有効同値パーティションの下端の境界
「1以上」→ 1(境界上)、0(前)、2(後)

無効同値パーティションの上端の境界
「1未満」→ 1(境界上)、0(前)、2(後)

→ 有効・無効のどちらから見ても境界値分析した値は 0、1、2

演習　境界値分析

- あるモジュールの仕様
  - short int型(16bit符号付き)の引数aがある
    (short int型変数がとりうる範囲：-32768～+32767)
  - モジュール内には if (a<0) と if (7<a) という2つの条件文がある

- 引数 a には、整数以外が入力されることはないものとみなして、同値パーティションを挙げなさい
- 次に同値パーティションの境界値を挙げなさい
- 最後に、どんな欠陥が見つかるかを考えなさい

10～15分

デシジョンテーブルテスト ①

- デシジョンテーブルテストは、デシジョンテーブルの「規則／ルール」を高位レベルテストケースとして扱う技法です
  - 組み合わせテストのアプローチの1つです
  - 複雑な論理を含んだ仕様に対して、抜け・漏れを防ぐようにテスト設計できます
- デシジョンテーブル
  - ある問題に関する複数の条件についてその成立／不成立の組み合わせを表形式に展開し、それぞれの条件に対応する結果(動作)を示したものです

デシジョンテーブルテスト ②

条件指定部の表記

表記	意味
Y （O/T/1）	• 規則を満足するには、この「Y」に対応する条件が満たされなければならない（条件成立）
N （×/F/0）	• 規則を満足するには、この「N」に対応する条件が満たされてはならない（条件不成立）
値、語句 （コード）	• 条件記述部を補足する形で、この語句、値またはコードによって完成させる • この完成した条件が満たされなければならない • コードを用いる場合、その意味を注記する（条件成立）
－	• 規則を満足するには、「－」に対応する条件は無関係でなければならない（条件無関係）
＃ （N/A）	• 規則を満足するには、「＃」に対応する条件は論理的に成立しない • 「＃」に対応する条件が起こり得ないことを意味する

動作指定部の表記

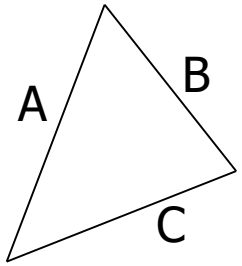
表記	意味
X （eXecute）	• 規則が満足されると、「x」に対応する動作を実行する（動作）
－ （空白）	• 規則が満足されると、「－」に対応する動作は実行されない（無実行）
値、語句 （コード）	• 動作記述部を補足する形で、語句、値またはコードによって完成させる • この完成した動作が実行され、コードを用いる場合、その意味を注記する（動作）

デシジョンテーブルテスト ③

- デシジョンテーブルの具体例
  - プログラムロジックを条件と動作に分けて、マトリクスで表現します
  - Myersの三角形の判定プログラムを例にします

三角形判定表		1	2	3	4
条件	A < B + C	N	Y	Y	Y
	B < A + C	–	N	Y	Y
	C < A + B	–	–	N	Y
動作	三角形ではない	X	X	X	
	種別判定表へ行け				X

種別判定表		1	2	3	4	5
条件	A = B = C	Y	N	N	N	N
	A = B && B ≠ C	–	Y	N	N	N
	A = C && C ≠ B	–	N	Y	N	N
	B = C && C ≠ A	–	N	N	Y	N
動作	正三角形	X				
	二等辺三角形		X	X	X	
	不等辺三角形					X



デシジョンテーブルテスト ④

- デシジョンテーブルの作り方
  - 最初にルール数に合わせて、テーブルを作成します
    - 条件が Y/N の2値である場合、2のn乗がルール数※になります
  - 1行目を列の半分にし、左側をYに、右側をNにします
  - 2行目はさらに列を半分にし、左側をYに右側をNにします
  - 最後の行まで繰り返します

条件数＝3個

問題名称		1	2	3	4	5	6	7	8
条件		Y	Y	Y	Y	N	N	N	N
		Y	Y	N	N	Y	Y	N	N
		Y	N	Y	N	Y	N	Y	N
動作		X							
			X	X	X	X	X	X	X

2^3 ＝ 8個のルール

※
- 条件が3値である場合、3のn乗がルール数になります
- 2値、3値が混在する場合、例えば、2値を持つ条件数が3つ、3値を持つ条件数が1つであるとき、$2^3 \times 3^1 = 24$ から24個のルールになります

デシジョンテーブルテスト ⑤

- メリット
  - 考えられる条件の組み合わせという点において、漏れや抜けがなくなります
  - 入力条件が複雑に絡み合う場合に有効です
  - テスト実行の進捗状況が分かり、他者にも伝えやすい
  - テスト実行を分業可能です
- デメリット
  - 条件そのものが間違っていたり、条件の洗い出しに失敗すると、導出された高位レベルテストケースの意味がなくなってしまう
  - 条件の数が多いと、条件の洗い出しに時間がかかりすぎ、テストを実行する時間がなくなってしまう

デシジョンテーブルテスト ⑥

- デシジョンテーブルの簡単化(圧縮)
  - 条件の組み合わせによっては、デシジョンテーブルが非常に大きくなることがあります
  - そのため、すべての可能な条件の組み合わせから、出力に関連しない条件の組み合わせを削減することがあります
  - この削減したデシジョンテーブルを「簡単化した(圧縮した)デシジョンテーブル」と呼びます
- デシジョンテーブルの簡単化(圧縮)の方法
  - 同じ結果になる共通条件を持つルールをまとめます
    - 結果に影響を及ぼさない条件の組み合わせを排除します
    - YでもNでもよい(無関係)という意味で、「－」に置き換えます
    - 下の条件から順にまとめていきます
- 簡単化(圧縮)の注意事項
  - 条件の処理順が条件欄の記載順と同一であること
  - 不明の場合は単純に簡単化できません

デシジョンテーブルテスト ⑦

- デシジョンテーブルの簡単化(圧縮)の方法

	問題名称	1	2	3	4	5	6	7	8
条件	満30歳以上	Y	Y	Y	Y	N	N	N	N
	年間50万円以上購入	Y	Y	N	N	Y	Y	N	N
	指定地域在住	Y	N	Y	N	Y	N	Y	N
動作	特別会員	X							

当初3･4 → 圧縮後3
当初5･6 → 圧縮後4
当初7･8 → 圧縮後5

	問題名称	1	2	3	4	5
条件	満30歳以上	Y	Y	Y	N	N
	年間50万円以上購入	Y	Y	N	Y	N
	指定地域在住	Y	N	–	–	–
動作	特別会員	X				

当初4･5 → 圧縮後4

	問題名称	1	2	3	4
条件	満30歳以上	Y	Y	Y	N
	年間50万円以上購入	Y	Y	N	–
	指定地域在住	Y	N	–	–
動作	特別会員	X			

テーブル圧縮が正しく行われたかを、
記号「-」の個数を使って検算する
→ 記号「-」は2列を表現する
→ $1+1+2^1+2^2 = 8$個

－:任意
(任意とは、YでもNでも動作は変わらないということ)

演習 デシジョンテーブルテスト ①

- うるう年の判定仕様
  - 西暦年数が4で割り切れる年は、うるう年である
  - ただし、西暦年数が4で割り切れる年のうち、100で割り切れる年の場合は、うるう年ではない(平年である)
  - さらに、2.のケースで平年であっても、400で割り切れる年の場合はうるう年とする

- デシジョンテーブルを作成しなさい
- 作成したデシジョンテーブルの各ルールに対して、入力値、期待結果を列挙しなさい

5~10分

演習　デシジョンテーブルテスト　②

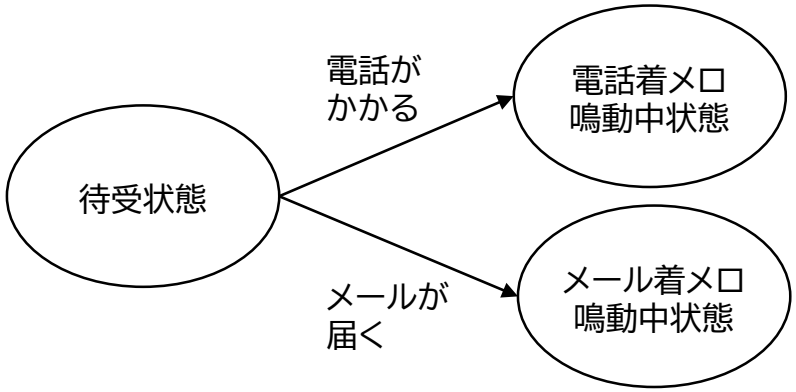
- ある施設の入場料の判定仕様
  - チケット代金が学生料金になる条件は次のとおりです
    - 年齢が6歳以上18歳以下
    - 現住所が埼玉県
    - 学生証の提示あり

- 簡単化した（圧縮した）デシジョンテーブルを作成しなさい
- 次に、同値分割法、境界値分析を考慮し、テストデータ（テストする年齢）を作成しなさい

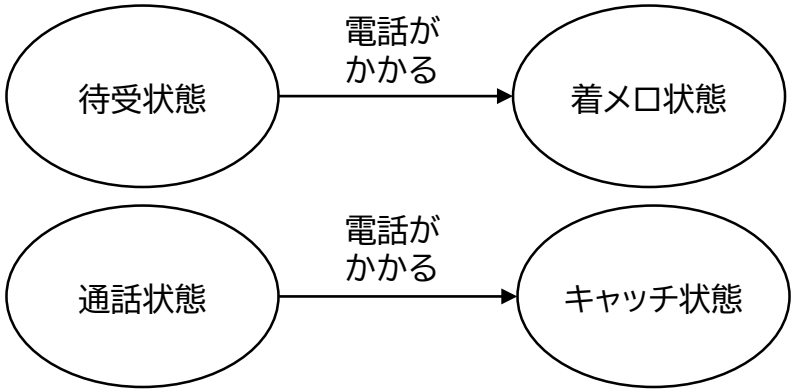
15～20分

状態遷移テスト ①

- 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技法です
  - イベントを起こして、プログラムの状態が正しく変化することをチェックします
  - システム例
    - 入力による画面の遷移を行うソフトウェア
      - GUIアプリケーション、Web、など
    - 同一の入力が内部の状態によって複数の意味を持つようなシステム
      - 携帯電話のボタン、など



同じ状態でもイベントが異なれば、
異なる状態に遷移します



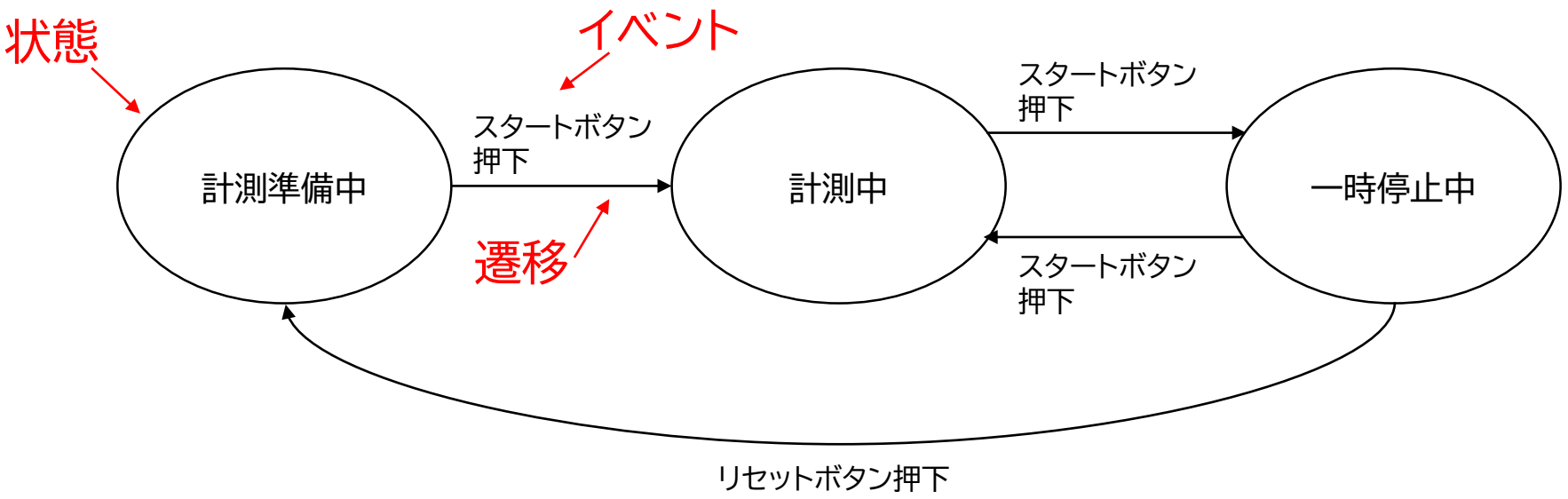
同じイベントでも前状態が異なれば、
異なる状態に遷移します

出典:Foundation Level シラバス

状態遷移テスト ②

- 状態遷移図
  - ソフトウェアの取り得る状態、ソフトウェアが開始および終了する方法、ソフトウェアが状態間で遷移する方法を示します

ストップウォッチの状態遷移図



出典:ソフトウェアテストの練習帳～技法編～

状態遷移テスト ③

- 状態遷移表
  - 状態間の有効な遷移と無効だと思われる遷移のすべてと、イベント、ガード条件、有効な遷移の際のアクションを示します

ストップウォッチの状態遷移表

状態(state) / イベント(event)	計測準備中	計測中	一時停止中
スタートボタン押下	計測中	一時停止中	計測中
リセットボタン押下	N/A	N/A	計測準備中

上の状態で左のイベントが発生したときに遷移する状態

適用不可(無効な遷移)

(注)この例では、アクションは省略しています
また、ガード条件はありません

N/A(Not Applicable):適用不可

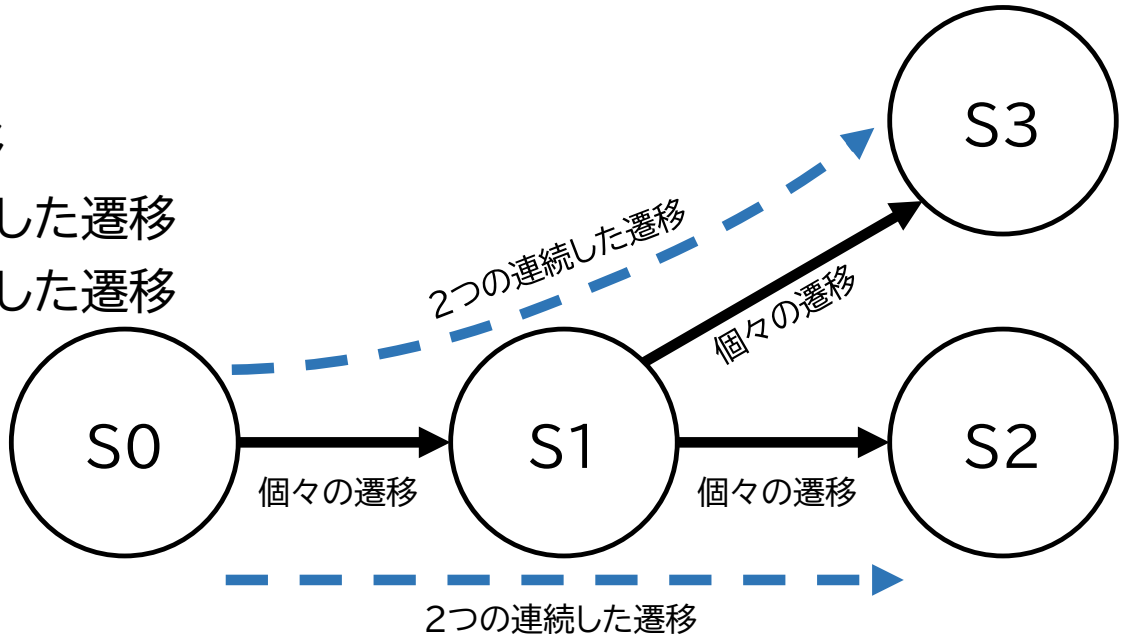
出典:ソフトウェアテストの練習帳～技法編～

状態遷移テスト ④

- 状態遷移テストのテストケース設計
  - 状態の典型的な順序をカバーする
  - すべての状態をテストする
  - すべての遷移をテストする
  - 遷移を特定の順序でテストする
  - 無効な遷移をテストする

状態遷移図には表れないため、
状態遷移表を書いて明らかにする

- N-Switchカバレッジ
  - 0 スイッチ・・・ 個々の遷移
  - 1 スイッチ ・・・ 2つの連続した遷移
  - 2スイッチ ・・・ 3つの連続した遷移



出典:Foundation Level シラバス

状態遷移テスト ⑤

- メリット
  - 状態遷移図と状態遷移表を合わせて使うことで、テストケースの漏れや抜けを防止でき、仕様や設計の誤りを見つけられます
  - 遷移できない欠陥や期待していない状態に遷移する欠陥を発見できます
- デメリット
  - 状態の定義が曖昧だと、期待する効果が得られません
  - 状態や遷移の数が多いと状態遷移図や状態遷移表が複雑になります

演習　状態遷移図と状態遷移表の作成

- 自動販売機の仕様
  - 100円硬貨のみ受け付ける
  - ジュースは100円と150円の2種類のみ（ジュースボタンが2個ある）
  - ジュースボタンを押して購入後に残高があれば、おつりとして商品と同時に返却される
  - コイン返却ボタンがある
  - その他の仕様(ジュースの在庫、温度、おつりの有無など)は考慮しない

- 投入金額(0円、100円、200円など)を状態として、状態遷移図と状態遷移表を作成しなさい

10～15分

ユースケーステスト ①

- 1個のサブジェクト(コンポーネントまたはシステム)と1個以上のアクターとの相互作用による振る舞いを表すユースケースのシナリオをテストする技法です
- ユースケースのシナリオを実行するテストになります
  - ユースケースがユーザーの活動を正確に反映していない場合は、よいテストになりません
  - 網羅的なテストを行うためには、ユースケースの代替系列を正確に定義する必要があります
- 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します
  - 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ます

出典:Foundation Level シラバス

ユースケーステスト ②

- ユースケースは、ユーザの視点から以下を記述したものです
  - どのようなアクターが
  - どのような場面において
  - どのような目的を達成するために
  - どのような活動を行うのか
- ユースケース図
  - ユースケース図は、利用者から見たシステムの「使い方の例（ユースケース）」を示したものです
- ユースケース記述
  - ユースケース図の中の1つのユースケースについて、ユーザやシステムなどのアクターが、ある特定の目的を達成するためにシステムをどう使うかを示すシナリオを記述したものです

ユースケーステスト ③

事前条件	• ユースケースの基本系列が正常に動作するための条件
終了条件 （成功時保証）	• 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
基本系列 （メインフロー）	• 通常、ユースケースにはメインストリームとなる（もっとも頻繁に実行する）シナリオがある
代替系列 例外系列	• 別のシナリオへ向かうブランチもある

ユースケーステスト ④

- ユースケースの例
  - レンタルビデオ店でDVDを借りる際の、店員のシナリオ
    - ユースケース名　：　（レンタルビデオ店で）DVDを貸し出す
    - アクター　：　店員
    - 事前条件　：　客が会員証を持っていること
    - 終了条件　：　DVDを会員が借り出したものとして記録する
    - 基本系列
      1. 店員：会員証を入力する
      2. システム：会員証の有効性を確認する
      3. システム：客への貸し出し状況を確認する
      4. 店員：貸し出すDVDを入力する
      5. システム：貸し出し可能性をチェックし、料金を示す
      6. 店員：料金を会員に提示し、入金する
      7. システム：入金を確認し、レシートを出す
    - 代替系列、例外系列
      2A　会員証の有効期限が切れていた場合、貸出できない
      3A　返却期限を過ぎて貸出中の商品がある場合、新規商品を貸出できない

ユースケーステスト ⑤

- テストケースの作り方
  - 基本系列の1個のテストケースと、代替系列・例外系列ごとに1個のテストケースを作成します
  - 前頁のユースケース記述の場合
    - テストケース1（基本系列） 1→2→3→4→5→6→7
    - テストケース2（代替系列） 1→2→2A
    - テストケース3（代替系列） 1→2→3→3A

出典：Foundation Level シラバス

ユースケーステスト ⑥

- メリット
  - 以下のような欠陥を見つけることができます
    - 定義済みシナリオの誤った処理
    - 代替系列処理の欠落
    - 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
  - ユースケース記述からフローチャートなどを作成すると、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ります
  - ユースケースが示すシナリオを使って性能テストを行うことがあります

ユースケーステスト ⑦

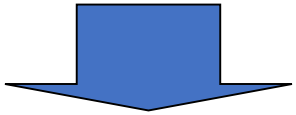
- デメリット
  - ユースケース(記述)は、要件全体の明確な定義を提供しない場合があるため、テスト対象の完全な定義として扱ってはなりません
    - テスト分析・設計時に、ユーザーの立場に立って、改めてユースケースを考えます
      - ユーザーがどのように製品を使うか？
      - 何を目的としてソフトウェアを使用するか？
      - ユーザーの盲点を引き出す
    - 2種類のユーザーの使い方を考えます
      - 一般的なユーザー と ろくでもないユーザー

組み合わせテスト ①

- 複数の値を持つ複数のパラメータを含むソフトウェアに対して、組み合わせに含めるアイテムの数を選択して、作成した組み合わせをテストする技法です
  - 4つの入力フィールドに対して、そこに入力する内容が各5種類だとすると、全組み合わせは、

    5×5×5×5＝625通り　　となります
  - これが、10個の入力フィールドになると

    5^{10}…9,765,625通り　　となります
  - フィールドがもう1つ増えると…更に5倍になります



全組み合わせは指数関数的に増加していきます

組み合わせテスト ②

STEP 1　組み合わせ条件を見つけます

- 様々な環境におけるソフトウェアの動作を確認したい

因子（変数）➡

水準（値）➡

OS
Windows 10
iOS

ブラウザ
Firefox
Chrome

バージョン
2.0
3.0

※テスト対象ソフトウェアのバージョン

組み合わせテスト ③

STEP 2 組み合わせに含める因子の数を選択し、組み合わせ表を作成します

- 以下は、すべての因子・水準(この場合は3個)を組み合わせています

OS
Windows
iOS

ブラウザ
Firefox
Chrome

バージョン
2.0
3.0

全組み合わせ →

	OS	ブラウザ	バージョン
1	Windows	Firefox	2.0
2	Windows	FireFox	3.0
3	Windows	Chrome	2.0
4	Windows	Chrome	3.0
5	iOS	Firefox	2.0
6	iOS	FireFox	3.0
7	iOS	Chrome	2.0
8	iOS	Chrome	3.0

組み合わせテスト ④

STEP 3 組み合わせテスト技法を用いて、組み合わせ表を作成します

- 組み合わせテスト技法にはペアワイズテストと直交表を用いたテストがあります

OS
Windows
iOS

ブラウザ
Firefox
Chrome

バージョン
2.0
3.0

	OS	ブラウザ	バージョン
1	Windows	Firefox	2.0
2	Windows	Chrome	3.0
3	iOS	FireFox	3.0
4	iOS	Chrome	2.0

※2因子間の組み合わせを網羅しています
OSとブラウザ、ブラウザとバージョン、バージョンとOSの
それぞれの組み合わせはすべてテストしています

組み合わせテスト ⑤

STEP 4　それぞれの組み合わせに対して、テストを実行します

- テスト実行後、テスト結果を分析します

	OS	ブラウザ	バージョン	結果
1	Windows	Firefox	2.0	合格
2	Windows	Chrome	3.0	合格
3	iOS	FireFox	3.0	不合格
4	iOS	Chrome	2.0	合格

- iOS×Firefox
- iOS×3.0
- Firefox×3.0

どれかに欠陥がある可能性があります
もしくは、3因子間の組み合わせに欠陥があります

組み合わせテスト ⑥

- 組み合わせテストのポイント
  - 因子(変数)の取りうる水準(値)の数が多いと、因子の数が少なくても、組み合わせは多くなります
    - 変数Aが100個の値、変数Bが100個の値、変数Cが2個の値の場合、ペアワイズテストによる組み合わせ表の行数は、最低でも100 × 100 = 10,000行になります
  - 値の数が多い変数では、組み合わせテストを適用する前に、各変数に同値分割法を適用して、値の数を削減することを検討します
  - あり得ない組み合わせ(制約や禁則と呼ぶ)がある場合は、それを除いて組み合わせを作る必要があります
  - 重要な組み合わせがある場合は、それを必ず含めるように組み合わせを作ります
    - ペアを考慮するだけで本当に十分なのか検討が必要です
  - 2個の因子を組み合わせる技法を適用した場合、3因子以上の組み合わせはテストされないリスクがあります

組み合わせテスト ⑦

- 組み合わせテストのポイント
  - 「ほとんどの欠陥は、単一の変数の値に基づくか、2個の変数の値の組み合わせに基づく」という経験的に知られた仮説に基づいています

バグに関係する要因数の割合

要因数	組込み機器（医療用）	ブラウザ（Netscape）	Webサーバ（Apache）	データベース
1	66	29	42	68
2	31	47	28	25
	97%	76%	70%	93%
3	2	19	19	5
4	1	2	7	2
5		2		
6		1	4	

出典: "Software Fault Interactions and Implications for Software Testing," R. D. Kuhn et al, IEEE Transactions on Software Engineering, 30(6), 2004

組み合わせテスト（ペアワイズテスト）

- 入力パラメータの各ペアを、設定可能な個々の組み合わせの全てで実行するためのテストケースを設計する技法です

OS
Windows
iOS

ブラウザ
Firefox
Chrome

バージョン
2.0
3.0

→

OSとブラウザの全組み合わせ

	OS	ブラウザ
1	Windows	Firefox
2	Windows	Chrome
3	iOS	FireFox
4	iOS	Chrome

→

OSとブラウザの組み合わせ（グループ）とバージョンの全組み合わせ

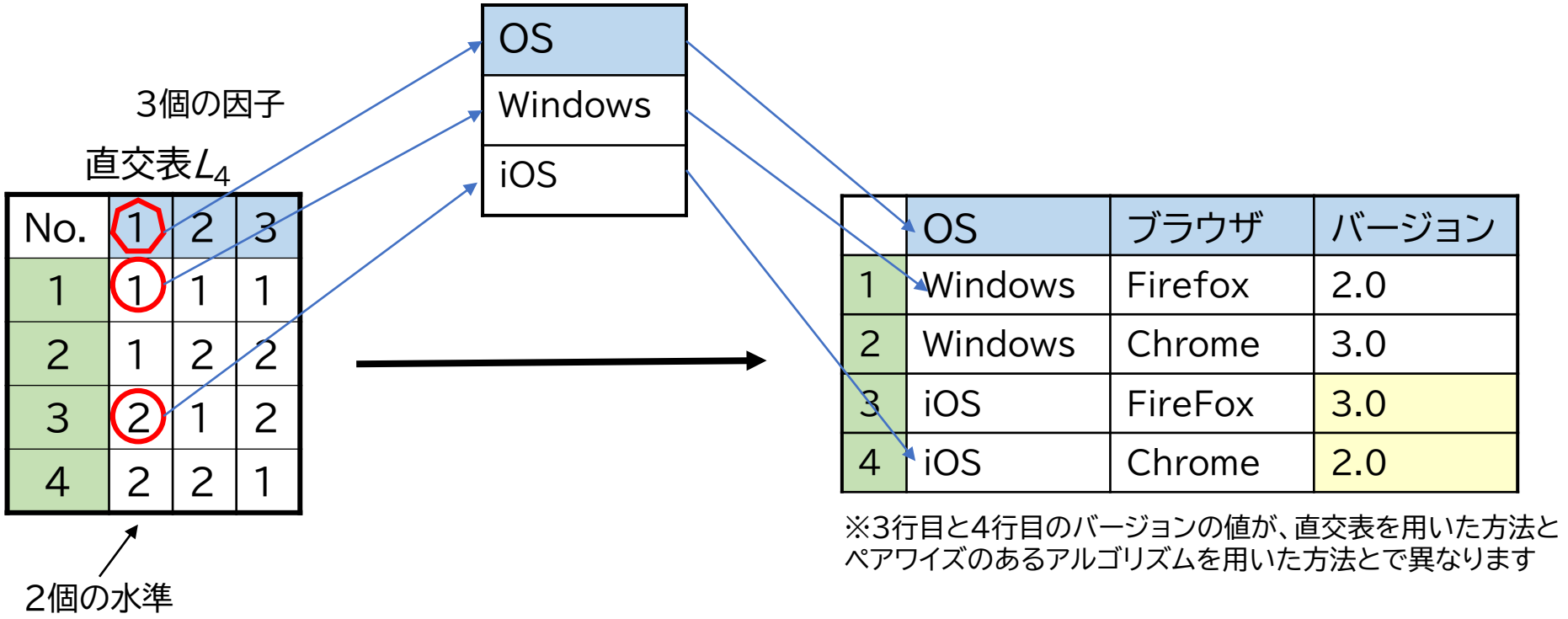
	OS	ブラウザ	バージョン
1	Windows	Firefox	2.0
2	Windows	Chrome	3.0
3	iOS	FireFox	2.0
4	iOS	Chrome	3.0

※この例では、OSとバージョンの全組み合わせはありますが、ブラウザとバージョンの全組み合わせはありません

出典：ISTQB Glossary

組み合わせテスト（直交表テスト）

- 直交表を使った変数のオールペア組み合わせテストの体系的な方法です
  - 変数を全て組み合わせたときの数を、オールペア組み合わせでテストできるまでに減らします



No.	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

	OS	ブラウザ	バージョン
1	Windows	Firefox	2.0
2	Windows	Chrome	3.0
3	iOS	FireFox	3.0
4	iOS	Chrome	2.0

※3行目と4行目のバージョンの値が、直交表を用いた方法とペアワイズのあるアルゴリズムを用いた方法とで異なります

出典：ISTQB Glossary

【参考】直交表

- 特殊な数学的性質を使って構築した2次元の配列であり、配列の中から選択した二個の列から、その配列の中の各値に対して全てのペアの組み合わせを提供します

直交表L_4(2水準の因子3つ)

No.	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

直交表L_8(2水準の因子7つ)

No.	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

直交表L_9(3水準の因子4つ)

No.	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

出典:ISTQB Glossary

演習　直交表を用いたテストケース作成

- 携帯型ミュージックプレーヤーの因子と水準

因子	水準
イコライザ	OFF
	ON
省電力モード	OFF
	ON
消音	OFF
	ON

割り付け

L_4直交表	A	B	C
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

因子（Factor）

水準（Level）

- 直交表を利用して、携帯型ミュージックプレーヤーの因子と水準に対する組み合わせ表を作成しなさい
- 作成した組み合わせ表が、以下を満たすことを確認しなさい
  - 任意の2因子間で水準の総当たりを実現していること
  - 3因子間では総当たりになっていないこと

10～15分

非機能要件と非機能要件テスト

- 非機能要件
  - 機能性に関係しない要件で、信頼性、効率性、使用性、保守性、移植性などの品質特性に関係する要件のことです
    - 拡張性、再利用性も非機能要件に含まれます
    - 使いやすさ（ユーザビリティ、UX、ユニバーサルデザイン、感性）も非機能要件に含まれます
- 非機能要件テスト
  - どのように動作するのかを確認するテストです
  - 非機能要件をテストする場合、品質特性は互いに関連していることを理解する必要があります
    - 例えば、セキュリティのレベルを上げると、性能が落ちることがあります
  - 「非機能要件に対して十分なテストを実現できる包括的なテスト設計の手法は存在しない」と言う人もいます
    - 非機能要件のモデル化や機能のリストアップが困難なため
  - テスト担当者は非常に多くの創造性が要求されます

シナリオテストの設計

- 本番を意識してゴール(目的・ビジネス)が達成できることを確認します
  - 様々なデータ、環境、エラーをテストします （代替シナリオ）
  - 運用時に何が行っているかを知ることが大切です
- テスト結果の分析が必須です
- シナリオテストを利用し、性能テスト、負荷テストを設計することもあります
  - シナリオテストは業務フローを正しく遂行できることを確認するテストです
  - 性能テストは性能要件を満たすかどうかを確認するテストです
  - 負荷テストは予想される負荷がかかっても正常に動作するかを確認するテストです
- シナリオの網羅性を確保するために重要管理項目の組み合わせを網羅することもあります
  - 例えば、お客様区分として、ブロンズ、シルバー、ゴールドを網羅するなど

性能テスト ①

- 性能要件を満たすかを確認するテストです
  - テストレベルによって着目すべき性能要件が異なります
    - コンポーネントテストの場合、コンポーネントの性能要件が対象となります
- 早め早めにテストを実行します
- レイヤーを分けて実施できる場合は、レイヤーごとに測定を実施します
  - Webアプリケーションの場合では
    - Webサーバー層で測定
    - アプリケーションサーバー層で測定
    - データベースサーバー層で測定

性能テスト ②

- テストケース作成（できる限り具体的に記述します）
  - 性能要求を基に、測定すべき内容を記載します
    - （サービスの）スループット（量） ： xx [トランザクション/秒]
    - （クライアントの）レスポンスタイム（速度） ： xx[m秒]
    - （力の源泉となる）リソース使用量 ： CPU xx [%]、MEMxx [GB]
    - 例えば、「50ユーザー同時に使用時に3秒で応答」が要件の時に、一人ずつアクセスを増やしながら上記を測定します
  - 性能測定を行う環境を明確にします
    - 特に「高負荷」や「境界値」
  - 記録および、高負荷の実現のため、ツールの使用を検討します
  - 期待される結果（時間・資源）が要件に書かれないことも多いです
    - 書かれていない場合、常識的な性能値、競合製品の性能値を使うと言われています
  - 性能限界を知ります
    - 仕様の範囲を超えた条件下でも、データロスなどの致命的なことが起こらないことを確認します

負荷テスト

- テストケース作成（できる限り具体的に記述します）
  - 要求される負荷を基に、測定すべき内容を記載します
    - CPU負荷
    - I/O負荷
    - ストレージ負荷
    - ネットワーク負荷
    - 同時アクセス負荷

      など
  - 性能テストと同時にできないかを検討します
  - 折れ線グラフ化し変化を把握します
  - 期待される結果をテスト前に作成・合意します（動作、対応負荷）
    - スペック外でも……データの喪失などの財産を損なうこと、健康を損なうこと、環境を損なうこと、セーブティ＆セキュリティ問題を起こすなどしてはダメ

リグレッションテスト

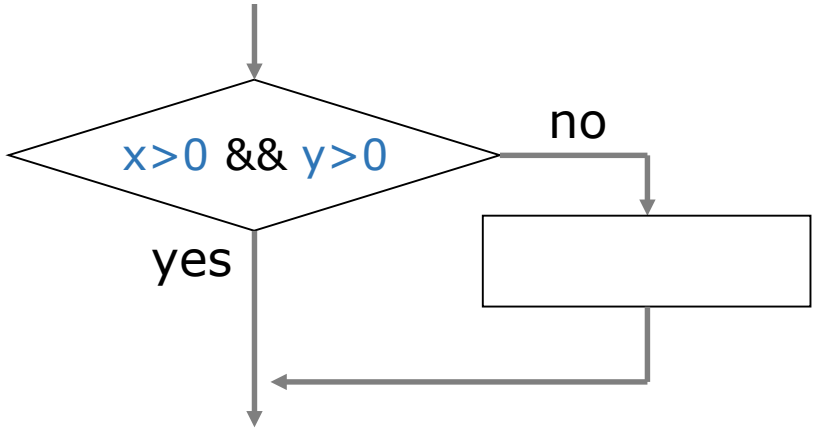
- 欠陥修正(デバッグ)後に行うテストの流れ
  - 発見した故障が起こらなくなったことを確認します(確認テスト)
  - 欠陥修正に伴う影響範囲を確認します(影響度分析)
    - 影響度分析により、修正による副作用や影響を受ける領域を識別します
  - 意図しない副作用の検出を目的とした確認を行います(リグレッションテスト)
  - なお、リグレッションテストの自動化の検討を行います
    - リグレッションテストスイートは何度も実行し、通常は少しずつ拡充をしていくため、自動化による効果が非常に大きいです
- リグレッションテストのポイント
  - リグレッションテストの実行条件をテスト計画で明確にしておきます
    - タイミング　：　欠陥修正ごと行うか、テストレベル終了時にまとめるか
毎日1回行うこともある(デイリービルド)
    - 実行回数　：　開発者の意見と、利用者の視点の両面で決める
    - 合格条件　：　欠陥修正ごとの場合、どこまでテストするかを決めておく

4.2 ホワイトボックステスト技法

制御フローテスト ①

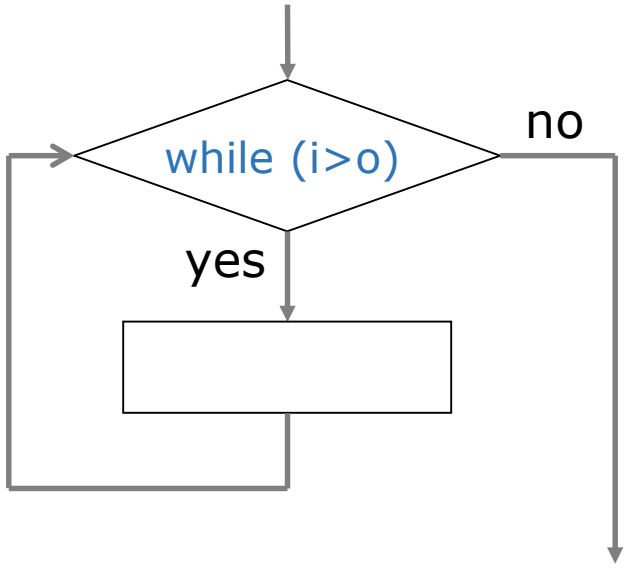
- 関数やメソッドのロジックを網羅します
  - カバレッジ(網羅率, 被覆率)の考え方
    - 被覆とは部分集合の族で覆うときの部分集合の族のこと
  - カバレッジ(Coverage) ＝ テストした項目／テストすべき項目 (×100)
    - C0(命令網羅、ステートメントテスト)
      - 命令文(ノード)を少なくとも1回実行する
    - C1(分岐網羅、デシジョンテスト)
      - 分岐(エッジ)を少なくとも1回実行する

右のフローチャートの場合、
C0なら、1回
C1なら、2回
少なくともテストを行う



制御フローテスト ②

- 制御フローテストのポイント
  - 少なくとも1回はコードを実行したことを保証します
    - 出荷後に、テストで実行しなかったコードが実行される危険性がなくなります
  - 実行できないコード(デッドコード)が見つかります
  - テスト実行の進捗状況が分かり、他者にも伝えやすい
  - 一方で、以下には弱い
    - ループでの誤り
    - 例外処理
      - コード実行が困難な場合が多い
    - 仕様そのものの誤り
      - ある機能がごっそり実装されていない　など
  - ループに対する指針
    - 0回、1回、2回、複数回のテストを実行すること



制御フローテスト ③

- McCabeのサイクロマチック数(複雑度)
  - グラフ理論に基づいて導出された数です
  - プログラムの複雑度を測るメトリクスの一つです
  - C(複雑度) ＝ e－n＋2p

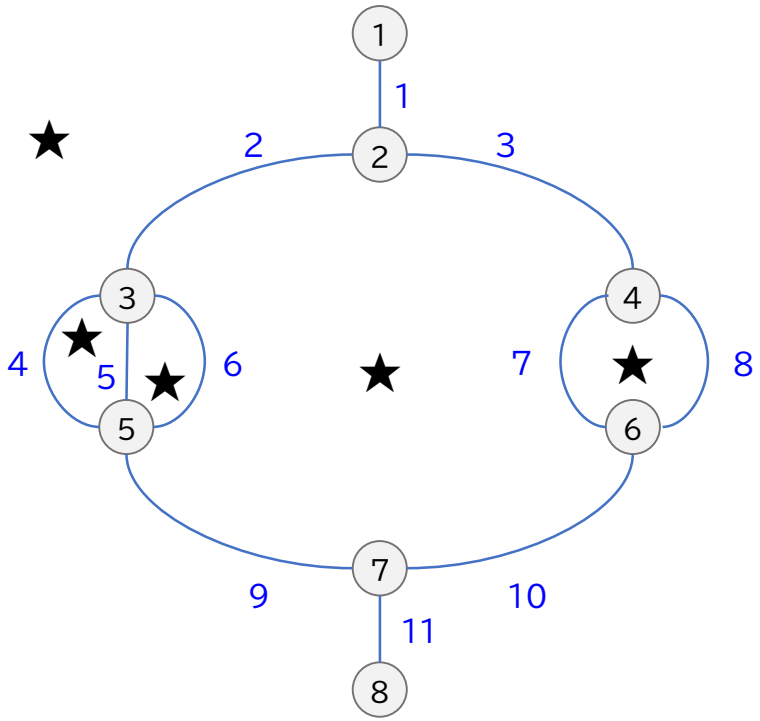
    e: グラフ内のエッジの数(含まれるルート数)

    n: グラフ内のノードの数(分岐点の数)

    p: グラフのつながっていない部分の数
  - 複雑度が大きいプログラム(モジュール)は欠陥が潜みやすい
  - C(複雑度)は、C1を満足するパスの数に(ほぼ)等しい
    - コードの構造を対象としたコンポーネントテストのテストケース数の妥当性の目安となります
    - 複雑度よりも極端に少なければ、十分なテストケース数とは言えない可能性があります

制御フローテスト ④

- サイクロマチック数(複雑度)の計算例



C(複雑度) = e−n+2p

C = e − n + 2 × 1
= 11 − 8 + 2
= 5(★:領域数と同じ)

＜複雑度の解釈例＞
複雑度10 以下であればよい構造
複雑度30 を越える場合、構造に疑問
複雑度50 を越える場合、テストが不可能
複雑度75 を越える場合、いかなる変更も誤修正を生む原因を作る

＜対策例＞
複雑度が15以上の関数は、コードレビューを必須とする

演習　制御フローテスト

- 次のC言語で書かれたソースコードのロジックをすべて通す(C1カバレッジを満たす)テストデータの組み合わせを考えなさい
- ソースコードの一部を切り出したものなので特に意味のあるコードではありません

```
void foo(int x, int y){
	if ( x != 0) {
		y = y / x ;
		if ( y > 0 ) {
			y = y -1 ;
		}
	}
}
```

5～10分

データフローテスト ①

- 制御フローグラフを使用して、データに起こり得る不合理なことを探ります
- プログラムのデータについてその取り扱いを網羅します
  - データや変数の使用の仕方に矛盾がないかを調べるためのテスト
    - 存在しないものを使用したり消滅させたりする欠陥を検出します
  - 定義、使用、消滅などが、順番通りかを調べるテストデータの組み合わせを与えます
    - 定義(defined)
      - 変数の値が代入される場所(代入文の左辺)
    - 使用・参照(used)
      - 変数の値が参照される場所(代入文の右辺や条件式)
    - 消滅(killed)、未定義(undefined)
      - 変数が解放される、または内容が不確定である場合
      - 全定義使用法(All-DU)
        - すべての変数について、定義(defined)から使用(use)までのパスを網羅する
      - 全使用法(All-uses)
        - 変数を使用するすべてのパスを網羅する

データフローテスト ②

後のアクション / 前の状態・アクション	定義(d)	使用(u)	消滅(k)
存在しない	○	×	×
定義(d)	△ 実害なし？	○	△ 保守で多い コード消滅
使用(u)	△	○ 問題なし	○ 問題なし
消滅(k)	○	× 欠陥-危険 メモリ解放後	△／× 実害有無は言語で 異なるが欠陥

○:問題なし、△:要注意、×:欠陥

4.3 経験ベースのテスト技法

経験ベースのテスト ①

エラー推測

- 起こりえるエラー、欠陥、故障のリストを作り、それらの故障やそれらを引き起こす欠陥を検出するテストケースを設計する
- エラー、欠陥、故障のリストは、テスト担当者の経験、欠陥や故障のデータ、ソフトウェアが不合格となる理由に関する一般的な知識に基づいて作成できる
  - アプリケーションの過去の動作状況
  - 開発担当者が犯しやすい誤りの種類
  - 他のアプリケーションで発生した故障

探索的テスト（Exploratory Testing）

- テスト設計とテスト実行を同時並行で行い、学習しながらテストを進める
  - テスト実行時に動的に設計、実行、ログ記録、および評価をする
  - テスト結果を使用しテスト対象の理解を深め、さらにテストを行わなければならない領域のテストケースを作成する
- セッションベースドテスト
  - あらかじめ決められた時間枠内で探索的テストを行う
  - テスト目的を含むテストチャーターに従ってテスト実行をする

出典：Foundation Level シラバス

経験ベースのテスト ②

チェックリスト
ベースドテスト

- チェックリストにあるテスト条件をカバーするように、テストケースを設計、実装、および実行する
- テスト担当者は、新しいチェックリストの作成、もしくは既存のチェックリストの拡張をテスト分析の一環として行う

出典:Foundation Level シラバス

5. テストマネジメント

5.1 テスト組織

組織の独立性

- 独立とは,「技術面、管理面、財務面で独立」することです（ISO/IEC/IEEE 24765）
  - 技術面：　異なる技術（視点）、認知バイアスを使います（異なる種類の故障を見つける）
  - 管理面：　出荷権限をテストに与えることでより厳格な品質保証活動が期待できます
  - 財務面：　テスト組織でのリソース（予算）の確保が楽になります

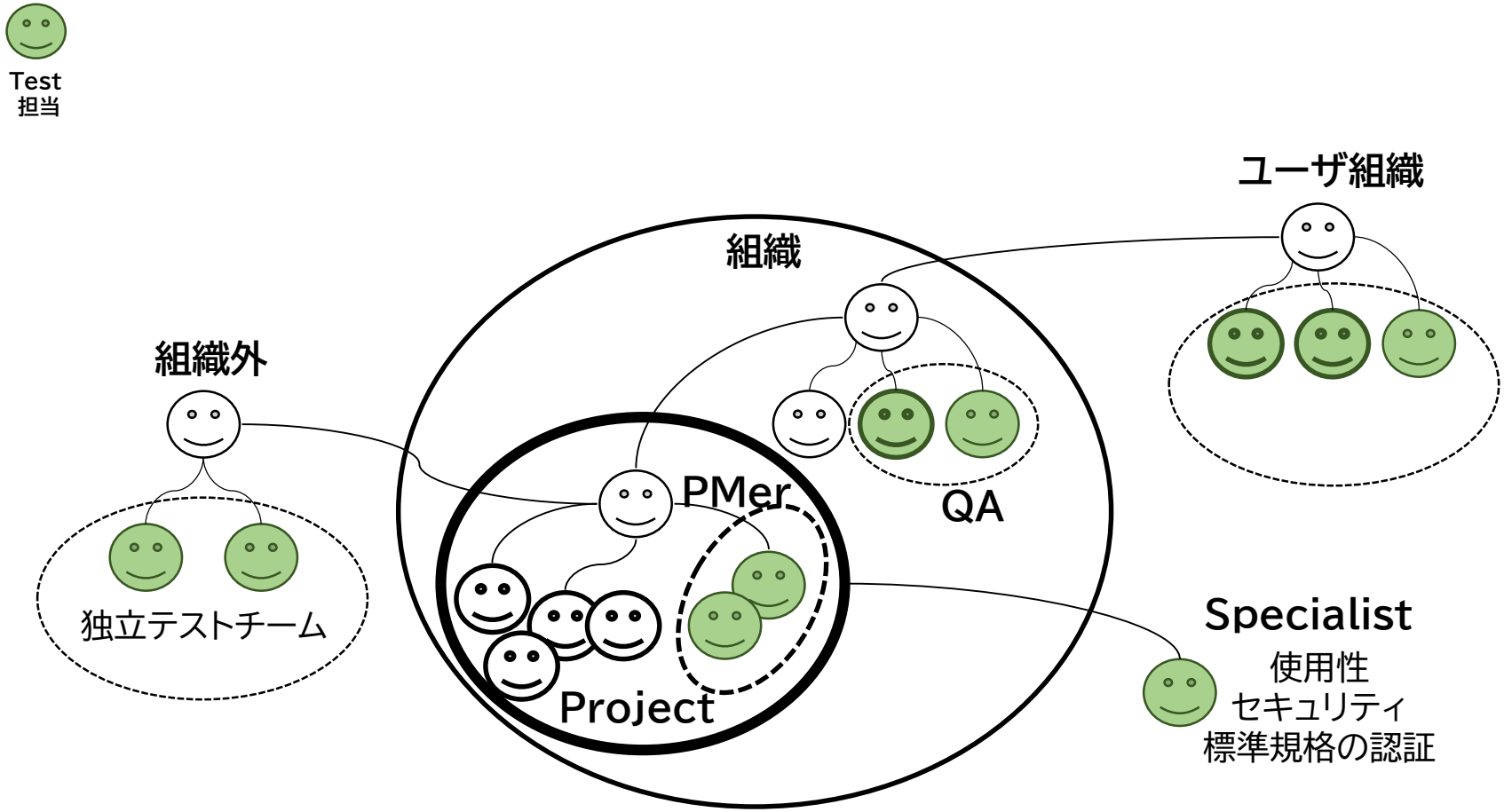
出典：Foundation Level シラバス

テストの独立性の度合い

- テストの独立性の度合い
  - 独立したテスト担当者不在
    - 開発担当者が自分のコードをテストするのみです
  - 開発チーム、またはプロジェクトチーム内に所属する、独立した開発担当者、またはテスト担当者
    - 開発担当者が同僚の成果物をテストすることもあります
  - 組織内にある独立したテストチームまたはグループで、PMや上位管理者の直属組織
  - 顧客またはユーザーコミュニティから派遣された独立したテスト担当者、または、使用性、セキュリティ、標準適合など、特定のテストタイプを専門に行うテスト担当者
  - 組織外の独立したテスト担当者
    - オンサイトまたはオフサイトで作業する

出典:Foundation Level シラバス

テスト組織



独立した組織によるテスト

メリット	• 開発者とは異なる種類の故障を検出する可能性が高い • 独立したテスト担当者は、開発担当者とは異なる背景、技術的視点、バイアスを持つため • ステークホルダーが行った仮定について、検証、説明の要求、または反証を行うことができる • 先入観がなく客観視できる • 設計中に検証できる

デメリット	• 開発チームとの協力関係の欠落、断絶、不和、対立の可能性がある
	• 開発者の品質に対する責任感が薄れることがある
	  • 開発担当者が手を抜く可能性
	• 独立したテスト担当者は、ボトルネックとして見られたり、リリース遅延で責任を問われたりすることがある
	• 重要な情報が伝わらないことがある
	  • 例えば、テスト対象の情報など

出典:Foundation Level シラバス

「テスト対策」の実施

- ソフトウェア開発者は、テストを軽視している？
  - 「テストは新人にやらせておけばいい」
  - 「ちゃんと作ったから、テストは必要ないよ」
  - 「テスト屋はあら探しばかりしてむかつく」
  - 「だったら、お前がプログラミングしてみろ！」
- 「テスト工程が開発におけるボトルネック」である理由は？
  - コストがかかるのは、欠陥を「作りこむ→検出→修正」が原因です
  - テストで欠陥が見つかると、デバッグ作業が必要となります
  - テスト工程は開発の大部分を占めますが、デバッグ作業が多いです
- 「テスト対策」とは？
  - テスト容易性（Testability（後述））を考えます
  - テスト設計の前倒しを行います
    - （開発の）自工程完結・フロントローディング・品質の作りこみを、（テストが）サポートします

テスト容易性（Testability）とは

実行円滑性 (Operability)	• ソフトウェアがうまく動けば動くほど、テストはどんどん効率的になる
観測容易性 (Observability)	• 見えるものしかテストできない
制御容易性 (Controllability)	• ソフトウェアをうまく制御できればできるほど、テストをより自動化し最適化できる
分解可能性 (Decomposability)	• テストの対象範囲を制限することにより、速やかに問題を切り分け、手際よく再テストを行える
単純性 (Simplicity)	• テストする項目が少なければ少ないほど、テストを速やかに行える
安定性 (Stability)	• 変更が少なければ少ないほどテストへの障害が少なくなる
理解容易性 (Understandability)	• より多くの情報があれば、それだけ手際よくテストができる

出典：実践ソフトウェアエンジニアリング

品質の作りこみ

- テスト容易性を高めることで、レビューやテストの効率が向上します
- その結果として開発全体の生産性向上につながる
- テストの準備量、実行数を削減するために
  - すべての同値クラスを明確化 ／ 同値の連続性を保証
  - 最上限、最下限を明確化 ／ すべての条件で動作、結果を明確化
  - 要素間依存を極力減らした設計 など

5.2 テストの計画と見積り

テストポリシー、戦略、アプローチ

テストポリシー	• 組織にとってのテストに関わる原理原則、アプローチ、主要な目的を記述する高位レベルのドキュメント
テスト戦略	• 組織内で実行する1つ以上のプロジェクトをテストするための全般的な要件をドキュメントにしたもの • 通常、プロダクトまたは組織のレベルでの、テストプロセスに関する汎用的な考え方を提供する • テストポリシーに沿ってテストをどのように行うかを詳しく説明している
テストアプローチ	• 特定のプロジェクトのためのテスト戦略を実現化したもの • プロジェクトの複雑さ、ゴール、開発対象プロダクトの種類、プロダクトリスクを考慮する • 開始基準、終了基準の定義、適用するテスト技法、実行するテストレベル、テストタイプの選択を含む

出典：Foundation Level シラバス、ISTQB Glossary

テスト戦略 ①

分析的 テスト戦略	• 要件を分析したり、リスクを分析したりなど、分析をベースにしたテストの考え方 • リスクのレベルに基づいてテストを設計し、優先度付けを行うリスクベースドテストが代表的である
モデルベースドの テスト戦略	• ビジネスプロセス、機能、内部構造、信頼性など、モデルをベースにしたテストの考え方 • ビジネスプロセスモデル、状態遷移モデル、信頼度成長モデルなどのモデルを使う
系統的テスト戦略	• 事前に用意してあるテスト条件またはテストケースを使用する考え方 • テストケースだけでなく、故障を分類したリスト、品質特性のリスト、Webページなどのルックアンドフィール標準も含む

プロセス準拠
テスト戦略
(標準準拠)
	• 外部のルールや標準を使用してテスト分析、設計、実装を行う考え方
	  • 業界特有のルール、標準、プロセスに従う
	  • 社内のルール、標準、プロセスに従う

出典:Foundation Level シラバス

テスト戦略 ②

指導ベースドのテスト戦略 （コンサルテーションベース）	• 外部の専門家からの助言、ガイダンス、指示に基づいてテストを考える • ビジネスドメインの専門家、技術の専門家など

リグレッション回避テスト戦略	• 既存では実現されていた能力のリグレッションを避けることを目的としてテストを考える
	  • 既存のテストウェア（テストケースやテストデータ）の再利用
	  • 高度に自動化されたリグレッションテストの使用
	  • 標準テストスイートの再利用

対処的テスト戦略	• 先に実行したテストの結果により得られた知識に応じて、テスト設計、実装、実行を行う考え方 • 他の戦略とは異なり、事前にテストを計画せず、テストケースも作成しない • 探索的テストが代表的である

出典：Foundation Level シラバス

テスト工数見積りと影響要因

プロダクトの特性	• プロダクトに関連するリスク • テストベースの品質 • プロダクトの規模 • プロダクトドメインの複雑度 • 品質特性の要件（例えば、セキュリティ、信頼性） • テストドキュメントの詳細度に関する要求レベル • 法規制への適合性の要件
開発プロセスの特性	• 組織の安定度合いと成熟度合い • 使用している開発モデル • テストアプローチ • 使用するツール • テストプロセス • 納期のプレッシャー
人の特性	• 参加メンバーのスキルや経験、特にドメイン知識のような類似プロジェクトやプロダクトのスキルや経験 • チームのまとまりとリーダーシップ
テスト結果	• 検出した欠陥の数と重要度 • 必要な再作業の量

出典：Foundation Level シラバス

テスト見積り技術

メトリクスを活用する

- 以前の類似したプロジェクトのメトリクスや、典型的な値を基にしてテスト工数を見積る
- アジャイル開発の場合
  - バーンダウンチャート
- シーケンシャル開発の場合
  - 欠陥除去モデル

専門家の知識を基にする

- テストのタスクの所有者の経験、または専門家による見積りを基にしてテスト工数を見積る
- アジャイル開発の場合
  - プランニングポーカー
- シーケンシャル開発の場合
  - ワイドバンドデルファイ見積り技法

出典:Foundation Level シラバス

テスト計画

1. 文書情報
    - 1.1 概要
    - 1.2 文書ID
    - 1.3 発行組織
    - 1.4 承認者
    - 1.5 変更履歴

2. はじめに
    - 2.1 適用範囲
    - 2.2 参考文献
    - 2.3 用語集

3. テストの背景
    - 3.1 プロジェクト
    - 3.2 テストアイテム
    - 3.3 テスト対象／テスト範囲
    - 3.4 前提と制約
    - 3.5 利害関係者

4. テストでのコミュニケーション

5. リスク管理表
    - 5.1 プロダクトリスク
    - 5.2 プロジェクトリスク

6. テスト戦略
    - 6.1 テストサブプロセス
    - 6.2 テスト成果物
    - 6.3 テスト技法
    - 6.4 テスト完了基準
    - 6.5 収集するメトリクス
    - 6.6 テストデータ要件
    - 6.7 テスト環境要件
    - 6.8 確認テストとリグレッションテスト
    - 6.9 テスト中止基準とテスト再開基準
    - 6.10 組織レベルのテスト戦略からの逸脱

7. テスト活動と見積もり

8. スタッフ
    - 8.1 役割、活動、責任
    - 8.2 雇用ニーズ
    - 8.3 教育ニーズ

9. スケジュール

出典:ISO/IEC/IEEE 29119-3 を参照し翻訳

5.3 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

テストモニタリング	• テスト計画書で定義したテストモニタリングのメトリクスを使用して、テスト計画書の内容と実際の進捗を継続的に比較します
テストコントロール	• テスト計画書の目的に合致させるために対策を講じます • テスト計画書を継続的に更新します

出典:Foundation Level シラバス

一般的なメトリクス

メトリクス	• 定義された測定方法と測定量

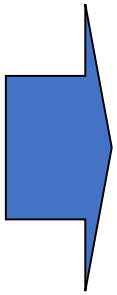
プロダクト
メトリクス
	• プロダクトの属性を測定する
	  • （例）ソフトウェアの品質を測定するメトリクス
	  規模を測定するメトリクス

プロセスメトリクス	• プロセスの属性を測定する
	  • （例）プロセスの実施に要した時間・工数
	  • 生産性のメトリクス（投入時間または投入工数当たりの開発規模）

テストで使用するメトリクス

代表的なテストメトリクス

- 計画したテストケースの準備が完了した割合
- 計画したテスト環境の準備が完了した割合
- テストケースの実行
  - 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
  - 欠陥密度
  - 検出および修正した欠陥数
  - 故障率
  - 確認テスト結果
- テスト設計カバレッジ、テスト実行カバレッジ
  - 要件
  - ユーザーストーリー
  - 受け入れ基準
  - リスク
  - コード
- タスクの完了、リソース毎の稼働状況、工数
- テストに費やすコスト



テストメトリクスで評価できる項目

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

出典:Foundation Level シラバス

開始基準、終了基準

開始基準

- テスト可能な要件、ユーザーストーリー、および／またはモデルが準備できている
- 前のテストレベルで終了基準を満たしたテストアイテムが準備できている
- テスト環境が準備できている
- 必要なテストツールが準備できている
- テストデータや他の必要なリソースが準備できている

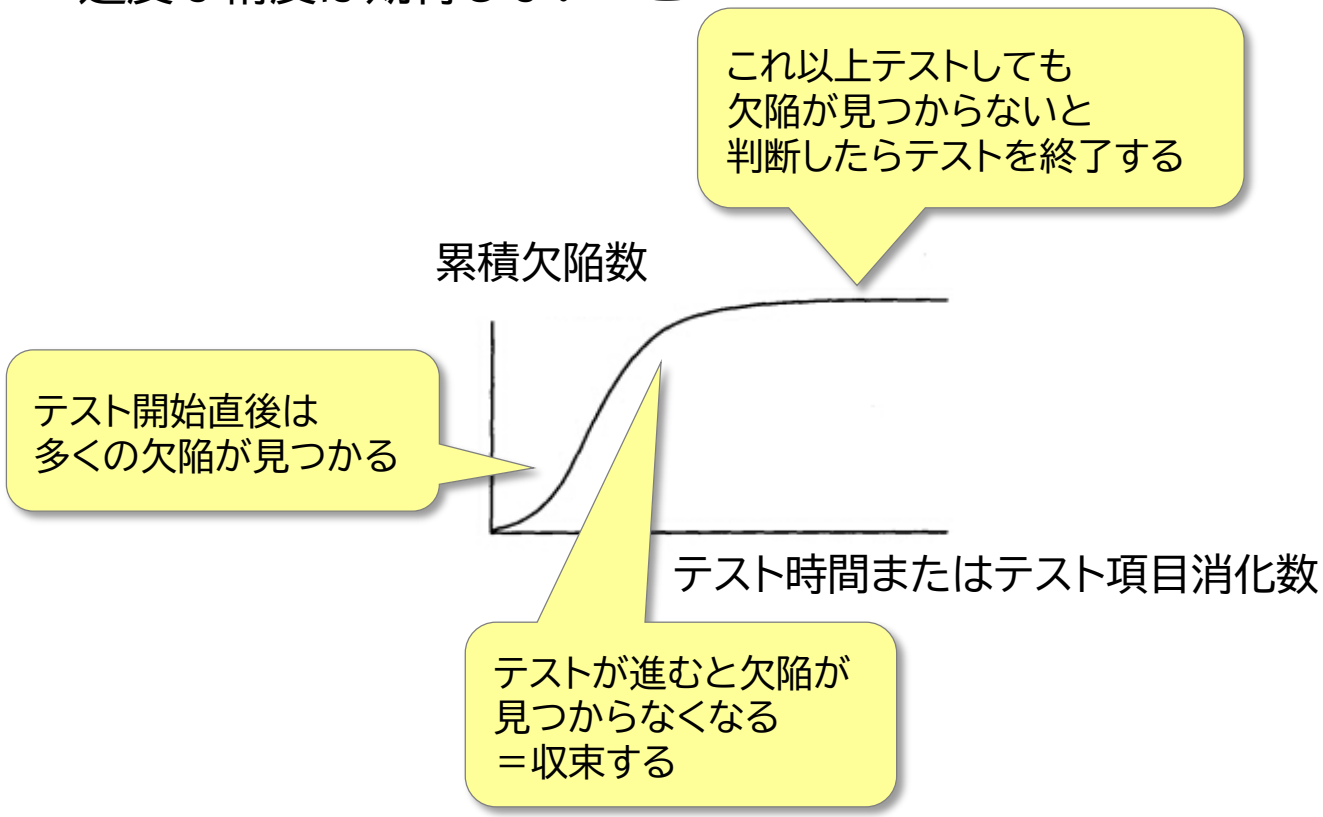
終了基準

- 計画したテスト実行が完了している
- 定義済みのカバレッジを達成している
- 未解決の欠陥の件数は合意された制限内である
- 残存欠陥の想定数が十分に少ない
  - 信頼度成長曲線などを使って推定する
- 信頼性、性能効率性、使用性、セキュリティ、他の関連する品質特性を十分に評価している

出典：Foundation Level シラバス

信頼度成長曲線

- テストにかけた工数と発見した欠陥数から、ソフトウェアに内在する欠陥の総数を推定する方法です
  - 別の視点のテストを追加すると、そこから曲線が新たに伸びることがあります
  - 過度な精度は期待しないこと



テストサマリーレポート

- レポートに含む主な内容
  - 行ったテストの要約
  - テスト期間中に発生したこと
  - 計画からの逸脱
    - テスト活動のスケジュール
    - 期間、工数　など
  - 終了基準に対するテストとプロダクト品質の状況
  - 進捗を妨げた、または引き続き妨げている要因
  - テストメトリクス
    - 欠陥
    - テストケース
    - テストカバレッジ
    - 活動進捗
    - リソース消費のメトリクス　など
  - 残存リスク
  - 再利用可能なテスト作業成果物

出典:Foundation Level シラバス

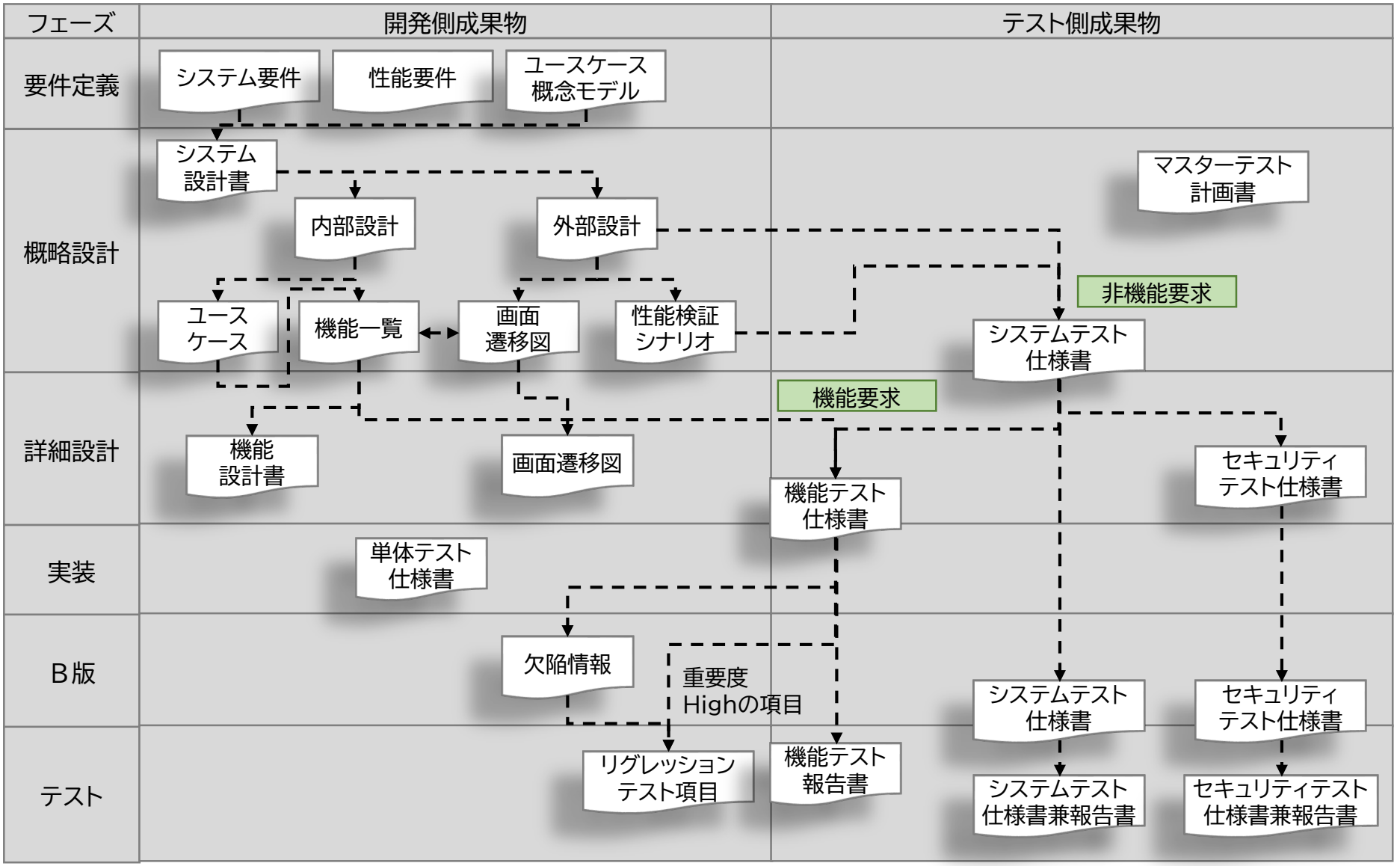
5.4 構成管理

構成管理

- 構成管理とは技術的かつ管理的な指示と監視を適用する規範のことです
  - この規範には次の目的があります
    - 構成アイテムの特性を機能的、物理的に識別・文書化すること
    - 特性に対する変更をコントロールすること
    - 処理の変更と実装の状況を記録し、報告すること
    - 特定の要求への整合を実証すること
- テストウェアの全アイテムを一意に識別して、バージョンコントロールを行い、変更履歴を残し、各アイテム間を関連付けます
  - また、テストアイテムのバージョンとの関連付けを行い、テストプロセスを通してトレーサビリティを維持できます
- 識別したすべてのドキュメントやソフトウェアアイテムは、テストドキュメントに明確に記載してあります

出典:Foundation Level シラバス、ISTQB Glossary

開発成果物とテスト成果物の関係（例）



テスト関連ドキュメント

組織レベルのテスト文書

- テストポリシー（A4で一枚程度）
- 組織における テスト戦略（複数タイプOK）

テスト管理者レベルの文書

- （マスタ）テスト計画（プロジェクト全体）
- （レベル）テスト計画（サブプロセスごと）
- テスト完了報告（サブプロセスごと → プロジェクトごと）

担当者：動的テストドキュメント（テスト仕様書→{データ、環境}→結果・報告）

- テスト設計仕様
- テストケース仕様
- テスト手順仕様
- テストデータ要求
- テストデータ準備報告
- テスト環境要求
- テスト環境準備報告
- テスト実行結果（記録）
- テスト結果（合否判定）
- テスト実行ログ
- テスト進捗報告
- 欠陥報告

5.5 リスクとテスト

リスク

リスク	• 将来、否定的な結果を生む要素
リスクレベル	• 影響度と発生確率に基づいて、定性的または定量的に表現したリスクの重要度
プロダクトリスク	• プロダクトの品質に影響を与えるリスク
プロジェクトリスク	• プロジェクトの成功に影響を与えるリスク • プロジェクトのマネジメントとコントロールに関係するリスク

出典:ISTQB Glossary

プロダクトリスクの例

プロダクトリスクの例

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

出典:Foundation Level シラバス

プロジェクトリスクの例 ①

プロジェクトの懸念事項	• リリース、タスク完了、終了基準の達成が遅延する • 不正確な見積り、優先度の高いプロジェクトへの資金の再割り当て、組織全体での経費節減により資金が不足する • プロジェクト終盤での変更により作業の大規模なやり直しが必要になる
組織の懸念事項	• 人員不足、および人員のスキルやトレーニング不足の場合がある • 人間関係によって、衝突や問題が発生することがある • ビジネス上の優先度の競合によってユーザー、ビジネススタッフ、特定の分野の専門家の都合がつかないことがある

出典:Foundation Level シラバス

プロジェクトリスクの例 ②

政治的な懸念事項	• テスト担当者が自分たちのニーズおよび／またはテスト結果の十分性を上手く伝えられない • 開発担当者やテスト担当者がテストやレビューで見つけた事項を上手くフォローアップできない • 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 • 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　など
供給者側の懸念事項	• 供給者が必要なプロダクト／サービスを提供できない、もしくは撤退する • 契約上の懸念事項がプロジェクトの問題の原因となる

出典：Foundation Level シラバス

プロジェクトリスクの例 ③

技術的な懸念事項

- 要件
  - 要件を十分に定義できない
  - 制約があるために、要件を満たさない
- テスト準備
  - テスト環境が予定した期限までに用意できない
  - データ変換および移行の計画、それらのツールによる支援が遅れる
- 開発プロセスの弱点が、作業成果物間の整合性や品質に影響を与える
- 不適切な欠陥マネジメントおよび類似の問題によって、欠陥や他の技術的負債が累積する

出典:Foundation Level シラバス

リスクベースドテスト

- 対応するリスクのタイプとリスクのレベルに基づき、テストの活動とリソースの利用をマネジメントし、選択し、優先順位付けするテストです
  - 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低減させ、ステークホルダにその状態を通知します
- プロダクトリスク分析の結果を使用して以下を行い、適用するテスト技法を決めます
  - 実行するテストレベルおよびテストタイプを決める
  - 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
  - テスト以外の活動でリスクを減らす方法があるか検討する

機能	故障の影響度	故障発生確率	リスク度	テスト重要度
機能A	2	2	4	低
機能B	3	4	12	中
機能C	5	4	20	高

5.6 欠陥マネジメント

不正と欠陥 ①

不正 (anomaly)	• 要求仕様、設計ドキュメント、ユーザドキュメント、標準など、または知見、経験から逸脱するあらゆる状態
偽陽性結果	• テスト対象には欠陥が存在しないにもかかわらず、欠陥として報告したテスト結果
偽陰性結果	• テスト対象に存在する欠陥を識別できなかったテスト結果

出典:ISTQB Glossary

不正と欠陥 ②

偽陽性、偽陰性とテスト	• テスト担当者は、欠陥として報告する偽陽性の数を最小限にする必要があるが、偽陽性を厳しくとがめれば、今度は萎縮が原因となり偽陰性が増える • テスト中に問題が生じた場合、欠陥レポートを作成する
欠陥 (defect)	• 作業成果物に存在する、要件または仕様を満たさない不備または欠点 • 実行中に欠陥に遭遇した場合、コンポーネントまたはシステムの故障を引き起こす

出典：Foundation Level シラバス、ISTQB Glossary

欠陥レポートの目的

- 開発担当者やその関係者に対し、発生したあらゆる期待に反する事象についての情報を提供します
  - 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します
  - 具体的な影響を識別して、最小の再現テストで問題の切り分けを行い、欠陥の修正ができるようにします
- テストマネージャーに対し、テスト実行中のシステムの品質や、テストへの影響を追跡する手段を提供します
  - 欠陥の報告数が多いと、テスト担当者は多くの時間をテスト実行ではなく報告作業に費やす必要があり、さらに多くの確認テストが必要になります
- 開発プロセスとテストプロセス改善のためのヒントを提供します

出典:Foundation Level シラバス

欠陥レポート

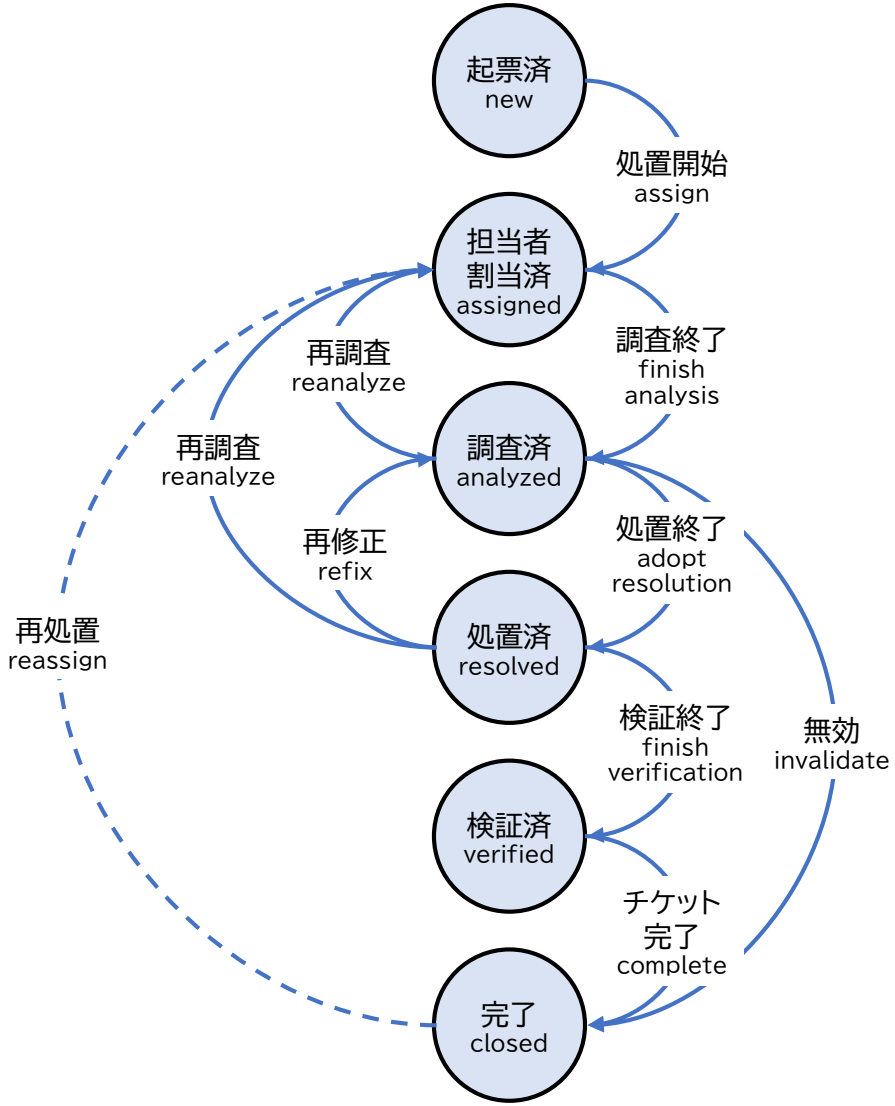
- 欠陥レポートに含める内容は以下のとおり
  - 識別子
  - レポート対象の欠陥の件名と概要
  - 欠陥レポートの作成日付、作成した組織、作成者
  - テストアイテム(テスト対象の構成アイテム)および環境を識別する情報
  - 欠陥を観察した開発ライフサイクルのフェーズ
  - ログ、データベースのダンプ、スクリーンショット、(テスト実行中に検出された場合は)実行状況などの欠陥の再現と解決を可能にする詳細な説明資料
  - 期待結果と実際の結果
  - ステークホルダーに与えるインパクトの範囲や程度(重要度)
  - 修正の緊急度/優先度
  - 欠陥レポートのステータス
    -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

出典:Foundation Level シラバス

欠陥の分析

- 発見された欠陥の分析結果は様々な用途で活用できます
  - エラーを推測します
  - 品質を分析します
  - リスクを分析します
  - リグレッションテストの効果を測定します
  - 今後の工数・テストケースを分析します
  - プロセスを改善します
- 欠陥をデータとして生かしていくためには
  - 欠陥マネジメントツールの構築が必要不可欠です
  - 欠陥の内容、手順を分かりやすく記載します
  - 欠陥の深刻度、レベルを分類し、優先順位付けを行います
    - リグレッションテスト設計時のデータとして利用します

欠陥ライフサイクル



出典:組込みソフトウェア開発における品質向上の勧め[バグ管理手法編]

欠陥レポートのステータス

起票済 new	• バグ管理システムにチケットが投稿された状態 • まだ、チケットに担当者は割り振られていない
担当者割当済 assigned	• 担当者が割り振られた状態 • 原因、対応などの調査活動を開始する
調査済 analyzed	• バグの再現性が確認され、原因が調査され、修正案が検討された状態 • 修正処置を開始する
処置済 resolved	• 修正案に従い、処置が終了した状態 • 修正内容の検証を開始する
検証済 verified	• ピアレビューなどが行われ、修正内容の検証が完了した状態 • チケットの最終確認を開始する
完了 closed	• チケットがクローズされた状態 • これ以上の対応は行われない（再処置となる場合もある）

出典：組込みソフトウェア開発における品質向上の勧め［バグ管理手法編］

状態遷移のイベント内容 ①

処置開始
assign

- バグの内容を確認し、適切な担当者を割り振る
- バグが発見された環境やバージョン情報など、発見時の情報が記録されていることを確認する

→ 担当者割当済 assigned

調査終了
finish analysis

- 担当者がバグ内容を確認し、分析作業を開始する
- 分析結果から処置方法を検討する
- 代替案を含めた処置方法のリスト、検討結果を記録する
- 処置期限や対応予定バージョンなどの情報を入力する

→ 調査済 analyzed

処置終了
adopt resolution

- 担当者が分析結果を確認し、修正作業を実行する
- 修正作業のレビューを行う
- 処置内容やレビュー結果を記録する

→ 処置済 resolved

出典：組込みソフトウェア開発における品質向上の勧め［バグ管理手法編］

状態遷移のイベント内容 ②

検証終了 finish verification	• 処置内容の検証を開始する • レビュー記録や確証情報の参照先などを記録する	→ 検証済 verified
無効 invalidate	• 担当者がバグ内容を確認し、分析した結果、バグではないと判断された場合	→ 完了 closed
チケット完了 complete	• チケットの内容を確認する • レビュー記録や確証情報の参照先などを記録する	→ 完了 closed

出典：組込みソフトウェア開発における品質向上の勧め［バグ管理手法編］

状態遷移のイベント内容 ③

再調査
reanalyze

- 分析結果、処置内容を確認して、再調査が必要だと考えた場合
- 再調査理由を明記する

→ 担当者割当済 assigned

再修正
refix

- 処置内容を確認して、再修正が必要だと考えた場合
- 再修正理由を明記する

→ 調査済 analyzed

再処置
reassign

- リグレッション(バグの再発)が発見された場合など、このバグを再度取り扱う状態になったとき

→ 担当者割当済 assigned

出典:組込みソフトウェア開発における品質向上の勧め[バグ管理手法編]

6. テスト支援ツール

6.1 テストツールの考慮事項

テストツールの目的

- テストツールの目的は様々です
  - テスト実行(リグレッションテストを含む)の効率を改善するため
    - 手動で行うと大量のリソースを必要とする反復作業を自動化します
  - テスト活動の効率を改善するため
    - テストプロセス全体を通して、手動で行うテスト活動を支援します
  - テスト活動の品質を改善するため
    - テストの一貫性や欠陥の再現性を高めます
  - 手動では実行できない活動を自動化することで、実行できるようにするため
    - 例えば、大規模な性能テストなど
  - テストの信頼度を向上させるため
    - 例えば、大量のデータの比較を自動化で実行します
    - 動作のシミュレーションをする　など
- 開発者向けのツールとテスト担当者向けのツールが存在します

テストツールの分類 ①

テストとテストウェアのマネジメントの支援ツール	• テストマネジメントツールとアプリケーションライフサイクルマネジメントツール(ALM) • 要件マネジメントツール(テスト対象へのトレーサビリティなど) • 欠陥マネジメントツール • 構成管理ツール • 継続的インテグレーションツール
静的テストの支援ツール	• レビューを支援するツール • 静的解析ツール
テスト設計とテスト実装の支援ツール	• テスト設計ツール • モデルベースドテストツール • テストデータ準備ツール • 受け入れテスト駆動開発(ATDD)ツールや振る舞い駆動開発(BDD)ツール • テスト駆動開発(TDD)ツール

出典:Foundation Level シラバス

テストツールの分類 ②

テスト実行と結果記録の支援ツール	• テスト実行ツール
	  • 例えば、リグレッションテストの実行
	• カバレッジツール
	  • 例えば、要件カバレッジ、コードカバレッジ
	• テストハーネス
	• ユニットテストフレームワークツール

性能計測と動的解析の支援ツール	• 性能テストツール • モニタリングツール • 動的解析ツール

特定のテストに対する支援ツール	• 以下の目的に使用される支援ツールも存在する
	  • データ品質の評価
	  • データのコンバージョンとマイグレーション
	  • 使用性テスト、アクセシビリティテスト
	  • ローカライゼーションテスト
	  • セキュリティテスト
	  • 移植性テスト

出典:Foundation Level シラバス

ツールの利点

- 反復作業が減ります
  - リグレッションテストの実行
  - 環境の準備／復旧タスク
  - 同じテストデータの再入力　など
- 一貫性や再実行性が増加します
  - 整合性のあるテストデータ
  - 同じ頻度と順序でのテスト実行
  - 要件からの一貫したテストケースの抽出　など
- 客観的な評価が可能になります
  - カバレッジの測定　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ります
  - テスト進捗
  - 欠陥率や性能計測結果の集計
  - グラフの作成　など

出典:Foundation Level シラバス

ツールのリスク ①

- ツールの効果を過大に期待します
  - 例えば、ツールの機能性や使いやすさなど
- ツールを初めて導入する場合に必要な時間、コスト、工数を過小評価します
- 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します
- ツールによって生成されたテスト資産を保守するために必要な工数を過小評価します
- ツールに過剰に依存します
  - 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ります
- ツールが保守されなくなるリスクがあります
  - ビジネスを廃業したり、販売から撤退したり、別のベンダーに買収されたり

出典:Foundation Level シラバス

ツールのリスク ②

- 既に導入している、またはこれから導入を検討している重要なツール間での関係性と相互運用性の問題を軽視します
- ツールのサポート、アップグレード、欠陥修正に対するベンダーの対応が悪い
- オープンソースプロジェクトが停止します
- 新しいプラットフォームや新規技術をサポートできないことがあります
- ツールに対する当事者意識が明確でないことがあります
  - 例えば、助言や手助け、および更新など

出典:Foundation Level シラバス

6.2 ツールの効果的な使い方

ツールを選択する際の基本原則

- ツールが導入目的を満たすかどうかを判断します
- 組織の成熟度、長所と短所を評価します
- テストプロセスを改善する機会として捉えます
- 明確な要件と、客観的な基準に基づいてツールを評価します
  - すでに使用している他のツールとの互換性なども評価します
- ツールベンダーを評価します
  - ライセンスモデル、トレーニング、サポートメニューなど
- ツール教育や訓練に対する組織内の必要事項を識別します
- ツールを使用するためのコーチングおよびメンタリングに関する組織内での要件を識別します
- 具体的なビジネスケースに基づいて、費用対効果を見積ります

出典:Foundation Level シラバス

パイロットプロジェクト

- ツールに関する知識を深め、強みと弱みを理解します
- 現状のプロセスや実践しているやり方にツールをどのように適用するかを評価し、何を変更する必要があるかを特定します
- ツールやテスト資産の標準的な使用方法、管理方法、格納方法、メンテナンス方法を決めます
- 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ます
- ツールによって収集およびレポートをさせたいメトリクスを理解し、メトリクスを確実に記録しレポートするようにツールを設定します

出典:Foundation Level シラバス

ツール導入の成功要因

- ツール未使用の部署にツールを順々に展開します
- ツールが適用できるよう、プロセスを調整、改善します
- ツールのユーザーに対し、トレーニング、コーチング、メンタリングを行います
- 利用ガイドを定めます
- ツールを実際に使用する中で得られる情報の集約方法を実装します
- ツールの利用状況や効果をモニタリングします
- ツールのユーザーサポートを提供します
- すべてのユーザーから、得られた教訓を集めます

出典:Foundation Level シラバス

まとめ

当研修の目次（JSTQB® FLと同じ）

VERISERVE

1. テストの基礎
   1.1. テストとは何か？
   1.2. テストの必要性
   1.3. テストの7原則
   1.4. テストプロセス
   1.5. テストの心理学

2. ソフトウェア開発ライフサイクル全体を通してのテスト
   2.1. ソフトウェア開発ライフサイクルモデル
   2.2. テストレベル
   2.3. テストタイプ
   2.4. メンテナンス(保守)テスト

3. 静的テスト
   3.1. 静的テストの基本
   3.2. レビュープロセス

4. テスト技法
   4.1. テスト技法のカテゴリ
   4.2. ブラックボックステスト技法
   4.3. ホワイトボックステスト技法
   4.4. 経験ベースのテスト技法

5. テストマネジメント
   5.1. テスト組織
   5.2. テストの計画と見積り
   5.3. テストのモニタリングとコントロール
   5.4. 構成管理
   5.5. リスクとテスト
   5.6. 欠陥マネジメント

6. テスト支援ツール
   6.1. テストツールの考慮事項
   6.2. ツールの効果的な使い方

研修で皆さんに伝えたいこと

VERISERVE

① テスト技法を意図的に使い分けます。

基本的なテスト技法については、その特徴、長所、短所、効果的な使用局面、使用上の注意事項などを知り、実務で使いこなすことが重要です。
（知っている、言われてみれば使っている、では足りません。）

② テストと開発の連携・協調が重要であり、分離しないようにします。

プロジェクトのチームメンバーとして、積極的に開発者（設計／実装メンバー）と協力しつつ、よい製品、よいサービスを顧客に提供する活力源となるように行動することが重要です。
（テスト技法が使えるだけでは足りません。）

参考文献

参考文献 ①

出典名	タイトル	著者名	出版社	出版年／バージョン
Foundation Level シラバス	ISTQBテスト技術者資格制度 Foundation Level シラバス 日本語版	JSTQB	JSTQB	Version 2018.J02
ISTQB Glossary	ISTQB Glossary (https://glossary.istqb.org/)	ISTQB/JSTQB	ISTQB/JSTQB	Version 3.2
ISO/IEC/IEEE 29119-3	ISO/IEC/IEEE 29119-3:2013	ISO/IEC/IEEE	ISO/IEC/IEEE	2013
組込みソフトウェア開発における品質向上の勧め［バグ管理手法編］	組込みソフトウェア開発における品質向上の勧め［バグ管理手法編］	独立行政法人 情報処理推進機構	独立行政法人 情報処理推進機構	2013
ソフトウェア品質評価ガイドブック	ソフトウェア品質評価ガイドブック	東 基衛（著）	日本規格協会	1995
実践ソフトウェアエンジニアリング	実践ソフトウェアエンジニアリング	ロジャーS.プレスマン（著） 西 康晴（翻訳）， 榊原 彰（翻訳） 内藤 裕史（翻訳）	日科技連出版社	2005
現場の仕事がバリバリ進むソフトウェアテスト手法	現場の仕事がバリバリ進むソフトウェアテスト手法	高橋 寿一（著） 湯本 剛著（著）	技術評論社社	2006
知識ゼロから学ぶソフトウェアテスト	知識ゼロから学ぶソフトウェアテスト	高橋 寿一（著）	翔泳社	2013
基本から学ぶソフトウェアテスト	基本から学ぶソフトウェアテスト	Cem Kaner（著） Hung Quoc Nguyen（著） Jack Falk（著） テスト技術者交流会（翻訳）	日経BP	2001
情報システムの障害状況 2018年後半データ	情報システムの障害状況 2018年後半データ	独立行政法人 情報処理推進機構	独立行政法人 情報処理推進機構	2018
ソフトウェア品質保証の考え方と実際	ソフトウェア品質保証の考え方と実際	保田 勝通（著）	日科技連出版	1995
実践的プログラムテスト入門	実践的プログラムテスト入門	ボーリス バイザー（著） 小野間 彰（翻訳） 石原 成夫（翻訳） 山浦 恒央（翻訳）	日経BP	1997

参考文献 ②

出典名	タイトル	著者名	出版社	出版年／バージョン
ソフトウェア・テストの技法	ソフトウェア・テストの技法 第2版	Glenford J.Myers 著 Tom Badgett 著 Todd M.Thomas 著 Corey Sandler 著 長尾 真 監訳 松尾 正信 訳	近代科学社	2006 第2版
SQuBOK Guide V2	ソフトウェア品質知識体系ガイド(SQuBOK Guide V2)			
ISO/IEC 25010:2011	Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models (JIS X 25010:2013 システム及びソフトウェア製品の品質要求及び評価(SQuaRE)- システム及びソフトウェア品質モデル)			
Advanced Level テクニカルテストアナリスト シラバス	ISTQBテスト技術者資格制度 Advanced Level シラバス 日本語版 テクニカルテストアナリスト	JSTQB	JSTQB	Version2012.J02

テキスト改変箇所について

本ドキュメントは、特定非営利活動法人 ソフトウェアテスト技術振興協会(以下、NPO法人ASTERと表記する)が作成したテキストを複製・改変して作られました。

ベースは「ASTER_Seminar_Text_V3.1_20190901.pptx」です。

NPO法人ASTERの最新テキストは以下に保存されています。

http://aster.or.jp/business/seminar_text.html

次ページより改変箇所を示します。

テキスト全般

VERISERVE

デザイン関連

- スライドのデザインを変更しています
- フォントをモリサワのユニバーサルデザインフォントである「BIZ UDPゴシック」「BIZ UDゴシック」に変更しています
- フォントサイズを変更しています
- 文や図の配置を変更しています

文章

一部「ですます」調に変更しています

ページ番号

表紙の次からページ番号をカウントしています

スライド位置

講義の順番を考慮して、一部のスライドの位置を変更しています

イントロダクションの変更点

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
1	変更	テストエンジニア研修	ソフトウェアテスト研修	開発者向けの研修のため
2	変更	ソフトウェアは規模も複雑さも増加傾向にあるので、	ソフトウェアは規模も複雑さも増加傾向にあります。 そのため、	文の長さを考慮したため
2	変更	テスト設計に焦点を当てるが、	当研修は、テスト設計に焦点を当てます。また、	文の長さを考慮したため
2	削除	その他にも、	−	文のバランスを考慮したため
2	追加	−	なお、本研修は試験範囲をすべて網羅しているわけではありません	試験対策のテキストであるという誤解をなくすため
3	変更	様々なテスト技法のうち、基本的な技法について	基本的なテスト技法については、	文のバランスを考慮したため
4	削除	1.6 行動規範	−	新シラバス対応のため

1 テストの基礎の変更点

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
7	変更	なぜテストをするのか	テストの必要性 1.2 テストの必要性に移動	講義内容に合わせるため
8	追加	－	「欠陥を全部みつけるのは無理だと心得よ」 By Cem Kaner	V2.1テキストの文章を残したため
10	削除	テストとは？(続き)	－	p.23に書いてあるから
12	削除	テストの目的	－	ISTQBによるテストの目的が書かれているため
14	変更	故障のもととなる欠陥を見つけて、解析し、取り除く一連の開発活動のこと	ソフトウェアの故障の原因を見つけて、分析して取り除くプロセス	ISTQB Glossaryを参照したため
14	変更	ソフトウェアに存在する欠陥に起因する故障を見つけること	全てのライフサイクルを通じて実施する静的、動的なプロセスにおいて、成果物が特定の要件を満足するかを判定し、目的に合致することを実証し、欠陥を見つけるため、ソフトウェアプロダクトや関連成果物に対し、計画、準備、評価をすること	ISTQB Glossaryを参照したため
14	削除	通常、テスト担当者はテスト対象の初回のテストと最終的な確認テストに責任を持ち、開発担当者はデバッグとコンポーネントテストに責任を持つことになる	－	開発現場の実態に合わせるため
19,20	削除	動かないソフトウェア③、④	－	講義内容に合わせるため
21	変更	動かないソフトウェアが引き起こす問題	「ソフトウェア開発を取り巻く状況」に移動	講義内容に合わせるため
23	追加	－	← 1999年の改訂の際、この定義は削除された	意味を正確にするため
24	変更	要件レビュー等に関与することにより、	テスト担当者が要件レビューやユーザーストーリーの洗練作業に関与することにより	新シラバス対応のため
24	変更	開発者と密接に連携することにより、テスト担当者と開発者が設計(やコード)に対する理解、およびそれらをどうテストするかの理解を深めることができる。結果として、設計(やコード)の	テスト担当者がシステム設計者と密接に連携して作業することにより、両者が設計とその設計をどうテストするかに対する理解を深めることができる。結果として、基本的な設計の	新シラバス対応のため
24	追加	－	テスト担当者が開発担当者と密接に連携して作業することにより、両者がコードとそのコードをどうテストするかに対する理解を深めることができる。結果として、コードとテストケースに欠陥が混入するリスクを低減できる。	新シラバス対応のため
24	変更	リリース前にソフトウェアを検証および妥当性確認することにより、テストしなければ見逃してしまうかもしれない故障を検出し、デバッグを支援できる。	テスト担当者はリリース前にソフトウェアを検証および妥当性確認することにより、テストしなければ見逃してしまうかもしれない故障を検出し、故障の原因となる欠陥を除去するプロセス(すなわちデバッグ)を支援できる	新シラバス対応のため

1 テストの基礎の変更点

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
25	削除	(ヒューマンエラーのエラーと同じ。)	−	削除しても意味が通じるため
29	変更	早期テストは、シフトレフトとも呼ばれる。	原則3　早期テスト(シフトレフト)で時間とコストを節約	変更しても意味が通じるため
30	削除	テストの労力を集中させるために欠陥の偏在を予測し、テストや運用での実際の観察結果に基づいてリスク分析を行う。(原則2で触れたことと同様)。	−	削除しても意味が通じるため
30	削除	この「殺虫剤のパラドックス」を回避するため、テストとテストデータを定期的に見直して、改定したり新規にテストを作成したりする必要がある(殺虫剤を繰り返し使用すると効果が低減するのと同様に、テストにおいても欠陥を見つける能力は低減する)。	−	削除しても意味が通じるため
31	削除	安全性が重要な産業用制御ソフトウェアのテストは、eコマースモバイルアプリケーションのテストとは異なる。また、	−	削除しても意味が通じるため
33	追加	−	組織のテストプロセスに影響する状況 ソフトウェア開発ライフサイクルモデル プロジェクト方法論 考慮対象のテストレベルとテストタイプ プロダクトとプロジェクトのリスク ビジネスドメイン 運用上の制約 予算、リソース、期間、複雑さ、契約など 組織のポリシー 組織内外の標準	V2.1の記述を残したため
34	変更	使命、目的に合致するようテスト活動を構築する。 → 状況により課せられる制約内でテストの目的を達成するためのアプローチを定義する → 開発ゴールを満たすためのテストプロセスやそのルールを規程する。(適切なテスト技法とタスクを指定し、適切なテストスケジュールを作成するなど。)	テストの目的を達成するために、様々な制約を踏まえた上で、テストのアプローチを定義します 例えば、品質目標と納期を踏まえ、適切なテスト技法とタスクを決め、要員を調達し、テストスケジュールを作成します	講義内容に合わせるため
34	追加	−	主な成果物はテスト計画書です マスターテスト計画書やレベルテスト計画書など、複数のテスト計画書が作成されることもあります テスト計画書には、テストベースに関する情報や終了基準が書かれています	講義内容に合わせるため
34	削除	複雑かつ大規模なものほど最初の計画が しっかりしていないとプロジェクトは破綻する！ これは複雑なソフトウェアやテストでも同じ	−	講義内容に合わせるため

1 テストの基礎の変更点

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
35	変更	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は、テスト計画書で定義したモニタリング用のメトリクスを使用して、テスト計画書の内容と実際の進捗を継続的に比較する活動である。 → テストコントロールには、テスト計画書の継続的な更新活動も含む。 特定のカバレッジ基準に対してテスト結果とログをチェックする。 テスト結果とログに基づいて、コンポーネントまたはシステムの品質のレベルを評価する。 さらなるテストが必要かどうかを判断する。 計画に対するテスト進捗は、テスト進捗レポートを使用してステークホルダーへ伝える。 コントロールとは、テスト計画を使ってテストの諸活動を行い 必要に応じて活動の軌道修正したり計画の見直しを行うこと。	テストのモニタリング テスト計画書に記載された予定と、テスト活動の実績を継続的に比較する活動です テストケースの消化数や欠陥の発生数などのメトリクスを使用して、計画時の数値と実際の進捗を継続的に比較します テストのコントロール テスト活動の実績が予定を逸脱していたり、その兆候が見えたりする場合、必要に応じて活動を軌道修正したり、テスト計画を見直します 終了基準の評価 事前に設定しているカバレッジ基準に対して、どの程度網羅しているかテスト結果とログをチェックします テスト結果とログに基づいて、コンポーネントまたはシステムの品質のレベルを評価します 欠陥の対応状況を確認します 追加テストの要否を判断します 主な成果物はテスト進捗レポートとテストサマリーレポートです テスト進捗レポートを使って、ステークホルダーにテストの状況を伝えます テスト進捗レポートには、計画からの逸脱や、テストの中止を決定するために必要な情報を含みます すべてのテストレポートには、作成時点で提供可能になったテスト実行結果の要約を含みます	講義内容に合わせるため

1 テストの基礎の変更点

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
36	変更	テスト分析 テスト可能なフィーチャーを識別し、テスト条件を決めるためにテストベースを分析する。(テスト分析では計測可能なカバレッジ基準から見た「何をテストするか」を決定する。) テストレベルごとに適切なテストベースを分析する。 テストベースとテストアイテムを評価して、以下のようなさまざまな種類の欠陥を識別する。 ※ 欠陥の例: 曖昧、欠落、不整合、不正確、矛盾、冗長なステートメント テストベースの分析に基づいて、各フィーチャーのテスト条件を決めて優先度を割り当てる。この際には、機能/非機能/構造の特性、他のビジネス/技術的要因、リスクのレベルを考慮する。 テストベースの各要素と関連するテスト条件の間に双方向のトレーサビリティを確立する。 ※ ISTQBでの定義では - テストアイテム: テストを実施する個々の要素。通常、一個のテスト対象に対し、多数のテストアイテムがある。 - テスト条件: コンポーネントやシステムのアイテムやイベントで、テストケースにより検証できるもの。例えば、機能、トランザクション、品質特性、構造要素など。	「何をテストするのか」を決めます テストベースを分析します テスト可能なフィーチャーとフィーチャーのセットを識別します 各フィーチャーのテスト条件を決めます テスト技法を活用すると、重要なテスト条件の欠落を防止し、精度が高く正確なテスト条件を決められます 各フィーチャーに優先度を割り当てます 機能/非機能/構造の特性、他のビジネス/技術的要因、リスクのレベルを考慮します テストベースとテストアイテムを分析し、評価することで、様々な種類の欠陥を識別することができます 例えば、曖昧、欠落、不整合、不正確、矛盾、冗長なステートメントのような欠陥です 主な成果物は優先順位を付けたテスト条件です テスト条件はテストベースとの間に双方向のトレーサビリティを確立することが望ましいです 探索的テストのテスト分析では、テストチャーターが成果物になります テストベースの欠陥を検出できた場合、欠陥レポートも成果物になります テスト分析は、なぜ行うのでしょうか？ 全数テストは組み合わせや操作順序が膨大で実施不可能だから スケジュール、リソース、環境などの制約(前提条件)を踏まえて、もっとも実施すべきテストに絞るため テスト条件(テストすべきもの・こと)を決定するため 分析対象には様々なものがあります RFP、要求仕様書、アーキテクチャ定義書、設計仕様書 標準文書(RFC、ISO/IEC標準)、業界標準、社内標準 会議の議事録、設計メモ、メールなど	講義内容に合わせるため

1 テストの基礎の変更点

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
37	変更	テスト設計 テスト条件を高位レベルテストケース、高位レベルテストケースのセット、およびその他のテストウェアへ落とし込むこと。 (「テスト条件」を「テストケース」に落とし込む際には、さまざまなテスト技法を使用する。) (テスト分析では、「何をテストするか」を決定し、テスト設計では「それをどうテストするか」を決定する。) テストケースおよびテストケースのセットを設計し、優先度を割り当てる。 テスト条件とテストケースを支援するために必要なテストデータを識別する。 テスト環境を設計し、必要なインフラストラクチャーやツールを識別する。 テストベース、テスト条件、テストケース、テスト手順の間で双方向のトレーサビリティを確立する。 テスト設計はテストベースに含まれる以下のようなさまざまな種類の欠陥を識別することもある。 ※ 欠陥の例: 曖昧、欠落、不整合、不正確、矛盾、冗長なステートメント	「どのようにテストするか」を決めます テストケースおよびテストケースのセットを設計します 様々なテスト技法を活用します テストケースに優先度を割り当てます 必要なテストデータを識別します インフラストラクチャーを識別し、テスト環境を設計します テストツールを識別します 主な成果物はテストケースとテストケースのセットです 具体的な入力データと期待結果の値を記載しない高位レベルテストケースは、テスト設計の主な成果物です 具体的な値を記載しないことで、再利用が可能になります テストケースはテスト条件との間に双方向のトレーサビリティを確立することが望ましいです テストは、 〇時間とお金の都合を考えて、QCDの バランスの取れた情報を提供します (=どこで妥協するか) ×納期までの時間を優先し、残った時間でテストします テスト設計をすることで、最も適したテストを探ります テストケースを合理的に少なくするための技法を選びます 同値分割法、組み合わせテスト 多くの欠陥が見つかるようにするための技法を選びます 境界値分析、エラー推測、探索的テスト テスト対象を漏れなくテストするための技法を選びます 制御フローテスト、データフローテストなど デシジョンテーブルテスト、状態遷移テスト、ユースケーステストなど	講義内容に合わせるため

1 テストの基礎の変更点

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
38	変更	テスト実装 テスト実行に必要なテストウェアを作成、および／または完成させること。 (テストケースの順序を決定してテスト手順を作成することも含まれる。) (テスト設計は「それをどうテストするか」の答えであり、テスト実装は「テストの実行に必要なものすべてを準備したか」の答えである。) (テスト設計とテスト実装は一緒に行うことがある。) テスト手順を開発して優先度を割り当てる。場合によっては、自動化のテストスクリプトを作成する。 テスト手順や(存在する場合)テストスクリプトからテストスイートを作成する。 効率的にテスト実行ができるように、テスト実行スケジュール内でテストスイートを調整する。 テスト環境(これには、テストハーネス、サービスの仮想化、シミュレーター、およびその他のインフラストラクチャーアイテムが含まれる)を構築する。また、必要なものすべてが正しくセットアップされていることを確認する。 テストデータを準備し、テスト環境に適切に読み込ませてあることを確認する。 テストベース、テスト条件、テストケース、テスト手順、テストスイートの間での双方向トレーサビリティを検証し更新する。	テスト実行に必要なものをすべて準備します テストケースの実行順序を決めます 効率よくテストできるようテストケースを組み合わせます テスト手順を作成し、優先順位を割り当てます テスト手順からテストスイートを作成します スケジュール内で実行するテストスイートを調整します テスト環境を構築します テスト環境には、テストハーネス、サービスの仮想化、シミュレーター、およびその他のインフラストラクチャーアイテムが含まれます 必要なものすべてが正しくセットアップされていることを確認します テストデータを準備し、テスト環境に設定されていることを確認します 主な成果物は次のとおりです 低位レベルテストケース テスト手順書 テストスイート テスト実行スケジュール テストベース、テスト条件、テストケース、テスト手順、テストスイートの間で、双方向のトレーサビリティを確立することが望ましいです	講義内容に合わせるため
39	変更	テスト実行 テスト実行スケジュールに従ってテストスイートを実行する。 (手動･自動) テストアイテムまたはテスト対象、テストツール、テストウェアのIDとバージョンを記録する。 手動で、またはテスト実行ツールを使用してテストを実行する。 実行結果と期待結果を比較する。 不正(anomaly)を分析して、可能性のある原因を特定する(故障はコード内の欠陥によって発生する可能性があるが、偽陽性が発生することもある。偽陽性とは「欠陥が存在しないにもかかわらず、欠陥として報告したテスト結果」のことをいう。) 故障(failure)を観察し、観察に基づいて欠陥を報告する。 テスト実行の結果(合格、不合格、ブロックなど)を記録する。 不正への対応の結果、または計画したテストの一環として、テスト活動を繰り返す(修正したテストケースによるテスト、確認テスト、および／またはリグレッションテストの実行)。 テストベース、テスト条件、テストケース、テスト手順、テスト結果の間で双方向のトレーサビリティを検証し更新する。	テスト実行スケジュールに従ってテストスイートを実行します テストアイテムまたはテスト対象、テストツール、テストウェアのIDとバージョンを記録します 手動、またはテスト自動化ツールを用いて、テストを実行します 実行結果と期待結果を比較します テストの実行結果(合格、不合格、ブロックなど)を記録します 実行結果と期待結果が一致しない場合、テストの実行状況に基づいて欠陥を報告します 原因を調査し、特定します 不正(anomaly)に対応します 不正への対応後、テスト活動を繰り返します 修正したテストケースによるテスト、確認テスト、リグレッションテストの実行 テストベース、テスト条件、テストケース、テスト手順、テスト結果の間で双方向のトレーサビリティを検証し更新します 主な成果物は次のとおりです テストケースまたはテスト手順の結果が書かれた文書 テスト実行可能、合格、不合格、ブロック、意図的にスキップなど 欠陥レポート テストアイテム、テスト対象、テストツール、テストウェアに関するドキュメント テストベース、テスト条件、テストケース、テスト手順、テスト結果の間で、双方向のトレーサビリティを確立することが望ましいです	講義内容に合わせるため

1 テストの基礎の変更点

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
40	変更	テスト完了 完了したテストの全活動のデータを集め、プロジェクトから得たこと、テストウェア、およびその他の関連する情報すべてをまとめる。(テスト完了活動は、ソフトウェアシステムがリリースされたとき、アジャイルプロジェクトのイテレーションが完了したとき、テストレベルが完了したときなどに実施する。) 終了基準の評価とレポート すべての欠陥レポートがクローズしていることを確認する。テスト実行の終了時に未解決として残されている欠陥については、変更要件またはプロダクトバックログアイテムを作成する。 テストサマリーレポートを作成して、ステークホルダーに提出する。 終了作業 テスト環境、テストデータ、テストインフラストラクチャー、その他のテストウェアを次回も使えるように整理し保管する。また必要に応じて他のステークホルダーに引き継ぐ。 完了したテスト活動から得られた教訓を分析し、次回のイテレーションやリリース、プロジェクトのために必要な変更を決定する。 収集した情報をテストプロセスの成熟度を改善するために利用する。	テスト活動で得られたことをまとめます すべての欠陥レポートがクローズしていることを確認します 未解決として残されている欠陥がある場合には、変更要求またはプロダクトバックログアイテムを作成します テストサマリーレポートを作成し、ステークホルダーに提出します 再利用できるように、テスト環境、テストデータ、テストインフラストラクチャー、その他のテストウェアを整理し保管します 必要があれば、テストウェアを他のステークホルダーに引き継ぎます 完了したテスト活動から得られた教訓を分析し、次回に活かせるようにまとめます 必要に応じて、テストプロジェクトで振り返りを行います 収集した情報をテストプロセスの成熟度を改善するために利用します 主な成果物は次のとおりです テストサマリーレポート 後続するプロジェクトまたはイテレーションを改善するためのアクションアイテム 変更要求またはプロダクトバックログアイテム 最終的なテストウェア	講義内容に合わせるため
41	削除	テスト作業成果物とトレーサビリティ テスト作業成果物 テストプロセスの一環として作成する。 (テスト作業成果物のガイドラインとして、ISO標準(ISO/IEC/IEEE 29119-3)を使用することができる。) テスト計画: テスト計画書 テストのモニタリングとコントロール: テスト進捗レポートとテストサマリーレポート テスト分析: 決定して優先順位を付けたテスト条件 (+検出した欠陥) テスト設計: 高位レベルテストケース (+トレーサビリティ情報+検出した欠陥) テスト実装: テスト手順とそれらの順序付け、テストスイート、テスト実行スケジュール テスト実行: テスト実行のステータスと結果(合否)、欠陥レポート、トレーサビリティ テスト完了; テストサマリーレポート、改善するためのアクションアイテム、完成したテストウェア トレーサビリティを取ることの重要性 変更の影響度分析。テストを監査可能にしてITガバナンス基準を満たす。テストベースの要素のステータスを明らかにする、ビジネスゴールに対するプロダクトの品質、および、プロセス能力およびプロジェクト進捗を評価することができるようになる。テストカバレッジの評価と同様に重要である。	–	テスト作業成果物を各活動の中で解説をしたため

1 テストの基礎の変更点

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
43	変更	人の心理とテスト(事実) 　欠陥や故障を見つけることは、開発担当者に対する非難と解釈されることがある。 　開発担当者には、コードは正しいという思い込み(確証バイアス)がある。 　テストからの情報は悪いニュースを含んでいることが多い。 　悪いニュースをもたらす人は非難される傾向がある。 　テストは、破壊的な活動と見られる場合がある。 　これらの心理的傾向を軽減するために、欠陥や故障に関する情報を建設的な方法で伝える。 テスト担当者と開発担当者との関係(対処法) 　対決ではなく、協調姿勢。 　全員のゴールは、高品質のシステムであることを再認識する。 　テストの利点を強調。 　欠陥情報は、作業成果物の品質向上と開発担当者のスキル向上に役立つ。 　欠陥を検出して修正すると、時間と経費の節約になり、プロダクト品質の全体的なリスクも減る。 　欠陥を作りこんだ担当者を非難しない。 　テスト結果や他の発見事項は、中立な立場で、事実に焦点をあてて伝え、客観的かつ事実に基づいた欠陥レポートを書いたり、発見事項をレビューしたりする。 　他人の気持ちや、他人が情報に対して否定的に反応した理由を理解するように努力する。 　自分の言ったことを他人が理解し、他人の言ったことを自分が理解していることを確認する。	テストによって欠陥を見つけることは、開発者に対する非難と解釈されることがあります 　開発者がテストで得られた情報を理解し、結果を受け入れることが困難になることもあります 欠陥や故障に関する情報は、建設的な方法で伝えるべきです 　そうすることで、ステークホルダー間の緊張を緩和することができます コミュニケーションを適切に行う建設的な方法は次のとおりです 　対決姿勢ではなく、協調姿勢を取ります 　テストのデメリットではなく、メリットを伝えます 　開発者への非難ではなく、中立的な立場で事実を伝えます 　開発者の否定的な反応に対して反応で返すのではなく、否定的に反応した理由を理解するようにします 　自分の言ったことを他人が理解しているか確認し、他人の言ったことを自分が理解しているかを確認します	講義内容に合わせるため
43	追加	－	開発者のマインドセット 　解決策の設計と構築に大きな関心を持ち、解決策の誤りには関心を持ちません 　確証バイアスにより、自分自身が犯した誤りを見つけることは困難です テスト担当者のマインドセット 　好奇心 　プロとしての悲観的な考え 　批判的な視点 　細部まで見逃さない注意力 　良好で建設的なコミュニケーションと関係を保つモチベーション 両者のマインドセットを組み合わせることで、より高いレベルのプロダクト品質を達成できるようになります	講義内容に合わせるため
44～47	削除	1.6 行動規範	－	新シラバスに対応するため

2 ソフトウェア開発ライフサイクル全体を通してのテストの変更点

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
50	削除	円グラフ	−	見にくいため
53	変更	コンポーネントテスト(CT)　(ユニットテスト:UT、モジュールテストとも呼ぶ) モジュール内のコードが正しいロジックかどうかをコードを開発した開発担当者がチェックすることが一般的である。予期しない例外についても確認する。 コンポーネントテストのリグレッションテストを自動化して、変更が既存のコンポーネントを破壊していないという信頼を積み重ねていくことが重要である。	定義 個別にテスト可能なコンポーネントに焦点をあてるテストのこと コンポーネント内のコードが正しいロジックかどうかをコードを開発した開発担当者がチェックし、予期しない例外についても確認する コンポーネントテストのリグレッションテストを自動化して、変更が既存のコンポーネントを破壊していないという信頼を積み重ねていくことが重要である 単体テスト、ユニットテスト、モジュールテストとも呼ばれる 目的 リスクの軽減 コンポーネントの機能的/非機能的振る舞いが設計および仕様通りであることの検証 コンポーネント品質に対する信頼の積み上げ コンポーネントに存在する欠陥の検出 欠陥がより高いテストレベルまで見逃されることの防止 典型的な欠陥と故障 正しくない機能(仕様記述とは異なる振る舞い) データフローの問題 正しくないコードとロジック	講義内容に合わせるため

2 ソフトウェア開発ライフサイクル全体を通してのテストの変更点

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
54	変更	統合テスト(IT: Integration Test) コンポーネントまたはシステム間の相互処理に焦点をあてたテストのこと。 「コンポーネント統合テスト」と「システム統合テスト」は、テスト対象が異なる。	定義 コンポーネントまたはシステム間の相互処理に焦点をあてるテストのこと 目的 リスクの軽減 インターフェースの機能的/非機能的振る舞いが設計および仕様通りであることの検証 インターフェース品質に対する信頼の積み上げ 欠陥の検出 インターフェース自体 コンポーネントに内在 システムに内在 欠陥がより高いテストレベルまで見逃されることの防止 コンポーネント統合　典型的な欠陥と故障 不正なデータ、データ不足、不正なデータエンコーディング インターフェース呼び出しの不正な順序またはタイミング インターフェースの不整合 コンポーネント間のコミュニケーション故障 コンポーネント間のコミュニケーション故障の処理不在、 または不適切な処理 コンポーネント間で渡されるデータの意味、単位、境界の正しくない思い込み システム統合　典型的な欠陥と故障 システム間で一貫性のないメッセージ構造 不正なデータ、データ不足、不正なデータエンコーディング インターフェースの不整合 システム間通信による故障 システム間通信処理不在、または不適切な処理による故障 システム間で渡されるデータの意味、単位、境界の正しくない思い込み 必須のセキュリティ規制への非準拠	講義内容に合わせるため

2 ソフトウェア開発ライフサイクル全体を通してのテストの変更点

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
54	変更	システムテスト(ST: System Test) 　システムの機能的/非機能的振る舞いが設計および仕様通りであることを検証(verification)するテストのこと。 　システムが完成し、期待通りに動作することの妥当性確認を行うテストのこと	定義 　システムが実行するエンドツーエンドのタスクと、タスクの実行時にシステムが示す非機能的振る舞いといったシステム全体の振る舞いや能力に焦点をあてるテストのこと 目的 　リスクの軽減 　システムの機能的/非機能的振る舞いが設計および仕様通りであることの検証 　システムが完成し、期待通りに動作することの妥当性確認 　システムの全体的な品質に対する信頼の積み上げ 　欠陥の検出 　欠陥がより高いテストレベルまたは運用環境まで見逃されることの防止 典型的な欠陥と故障 　正しくない計算 　正しくない、または期待通りではないシステムの機能的または非機能的振る舞い 　システム内の正しくないコントロールフローおよび／またはデータフロー 　エンドツーエンドの機能的タスクを適切かつ完全に実行できない 　本番環境でシステムが適切に動作しない 　システムマニュアルおよびユーザーマニュアルに記載されている通りにシステムが動作しない	講義内容に合わせるため
54	変更	受け入れテスト(UAT: User Acceptance Test) 　システムが、ユーザーのニーズ、要件、ビジネスプロセスを満足するかをチェックするための公式なテストのこと。 　システムが完成し、期待通りに動作することの妥当性確認(validation)を行う。	定義 　システム全体の振る舞いや能力に焦点をあてるテストのこと 　システムが、ユーザーのニーズ、要件、ビジネスプロセスを満足するかをチェックするための公式なテストのこと 目的 　システムの全体的な品質に対する信頼の積み上げ 　システムの機能的/非機能的振る舞いが仕様通りであることの検証 　システムが完成し、期待通りに動作することの妥当性確認 典型的な欠陥と故障 　システムのワークフローがビジネス要件またはユーザー要件を満たさない 　ビジネスルールが正しく実装されていない 　システムが契約要件または規制要件を満たさない 　セキュリティの脆弱性、高負荷時の不適切な性能効率、サポート対象プラットフォームでの不適切な操作などの非機能特性の故障	講義内容に合わせるため
54	追加	–	テストレベル 　具体的にインスタンス化したテストプロセス テストフェーズ 　テスト活動をプロジェクト中で管理(マネジメント)しやすいフェーズにまとめたセット 　例えば、あるテストレベルの実行活動	講義内容に合わせるため

2 ソフトウェア開発ライフサイクル全体を通してのテストの変更点

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
56	変更	特定のテストの目的から見たソフトウェアシステム(あるいはシステムの一部分)の特性をテストするための活動を束ねたもの 機能テスト 非機能テスト ホワイトボックステスト 変更部分のテスト	テストタイプ(test type) テストタイプは、以下に列挙する特定のテストの目的から見たソフトウェアシステム(あるいはシステムの一部分)の特性をテストするための活動を束ねたものである 機能の品質特性 例えば完全、正確および適切であることなどを評価する 非機能の品質特性 例えば信頼性、性能効率性、セキュリティ、互換性、使用性などを評価する コンポーネントまたはシステムの、構造またはアーキテクチャーが正しく完全で仕様通りであることを評価する 欠陥が修正されていることを確認するなどの変更による影響を評価し(確認テスト)、ソフトウェアや環境の変更によって意図しない振る舞いの変化が発生していないかを探す(リグレッションテスト)	講義内容に合わせるため
65	変更	確認テスト 修正した欠陥により不合格となった全テストケースを再実行し、欠陥が確実に修正されたことを確認する。 リグレッションテスト 修正や変更により同一コンポーネント、同一システム内の他のコンポーネント、または他システムのふるまいに意図せず及ぼす副作用(リグレッション)を検出する。	確認テスト 欠陥を修正したら、その欠陥により不合格となった全テストケースを新しいソフトウェアバージョンで再実行するテスト 欠陥が確実に修正されたことを確認する リグレッションテスト 意図しない副作用(リグレッション)を検出するテスト 修正および変更が、既存のコンポーネントまたはシステムの振る舞いに意図せず影響を及ぼしていないことを確認する ※欠陥修正時に新たに作りこんだ欠陥を「リグレッション」と呼ぶ	講義内容に合わせるため
66	削除	リグレッションテスト	–	p.150にマージしたため
68	変更	メンテナンスの一環として変更が発生した場合に以下の目的で行う。 変更が正しく適用されていることを評価する。 システムの変更していない部分での副作用(リグレッション)を確認する。	メンテナンステスト 運用システム自体の変更や、稼動環境の変更が運用システムに与える影響をテストします 変更が正しく適用されていることを評価します システムの変更していない部分での副作用(リグレッション)を確認します メンテナンステストの実施範囲 変更のリスクの度合い 例えば、ソフトウェアの変更領域が他のコンポーネントまたはシステムとコミュニケーションする度合い 既存システムの規模 変更の規模	講義内容に合わせるため

2 ソフトウェア開発ライフサイクル全体を通してのテストの変更点

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
69	変更	メンテナンスリリース向けに行った変更を評価する。 　変更により意図した結果 　変更により予想される副作用 　変更が影響するシステムの領域 　既存のテストに対する修正が必要な箇所 修正が必要な既存のテストを更新した後に、副作用および影響を受ける領域に対してリグレッションテストを行う必要がある。	影響度分析 　変更することにより既存システムがどの程度影響を受けるか分析し、影響を識別します 　変更により意図した結果が得られるか 　変更が影響するシステムの領域はどこか 　変更により予想される副作用があるか 　影響度分析は、既存のテストに対する修正が必要な箇所も識別します 影響度分析の難しさ 　仕様が最新版でない、または存在しない 　テストケースが文書化されていない、または最新版でない 　テストとテストベースとの間の双方向のトレーサビリティが維持されていない 　ツールによる影響度分析のためのサポートが貧弱であるか、まったくない 　ドメインおよび／またはシステムの知識を持つ担当者がいない 　開発時にソフトウェアの保守性が考慮されていない	講義内容に合わせるため

3 静的テスト

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
73	追加	－	作業成果物を人手で調査したり、コードや他の作業成果物をツール主導で評価したりするテストです 静的テスト ＝ レビュー ＋ 静的解析 コードを実行することなく、成果物をテストします。	講義内容に合わせるため
73	変更	静的解析の主な効果 欠陥の検出と修正をより効率的に、動的テストを実行する前に行うことができる。 動的テストでは容易に検出できない欠陥を識別できる。 要件の不整合、曖昧性、矛盾、欠落、不正確性、冗長性を明らかにして、設計時またはコーディング時に欠陥が混入するのを防止できる。 開発やテストの生産性を向上できる（設計の改善、保守性の高いコードなど）。全体的な品質コストを削減できる。	静的テストのメリット 動的テストを実行する前に欠陥を早期に検出できます 欠陥の検出と修正をより効率的に行うことができます 動的テストでは容易に検出できない欠陥を識別できます 設計時またはコーディング時に欠陥が混入するのを防止できます 要件の不整合、曖昧性、矛盾、欠落、不正確性、冗長性を明らかにします 開発の生産性を向上できます 設計の改善、保守性の高いコードなど 開発とテストにかかるコストと時間を削減できます ライフサイクルの初期に検出した欠陥の修正は、テストケースを実行して検出した欠陥を修正する場合に比べて、はるかに安価になります 全体的な品質コストを削減できます ライフサイクルの終盤または本番リリース後に検出される欠陥数が減少します レビューに参加することでメンバー間のコミュニケーションが改善できます	講義内容に合わせるため
75	追加	－	レビューとは静的テストの種類の一つです 1人もしくは複数のレビューアが成果物やプロセスを評価して懸念事項を検出し、改善策を提供します	講義内容に合わせるため
75	変更	計画 レビューの範囲、開始・終了基準の定義。工数・時間の見積り。 レビュー参加者の選定など レビューの開始 作業成果物（レビュー対象）の配布。参加者への作業説明など。 個々のレビュー 作業成果物のレビュー。潜在的な欠陥、提案、質問の記録など。 懸念事項の共有と分析 欠陥についての議論、分析、オーナーの割り当て。品質特性の評価、レビュー判定など 修正と報告 欠陥レポートの作成。欠陥の修正など。	計画 レビューの範囲を定義し、工数と時間を見積る レビュー特性（レビュータイプ、役割など）を識別する レビューの参加者を選び、役割を割り当てる 開始基準と終了基準を定義する 開始基準が満たされていることをチェックする レビューの開始 個々のレビュー 懸念事項の共有と分析 修正と報告	講義内容に合わせるため

原著作物のページ	区分	改定前	改定後	改定理由
75	変更	計画 　レビューの範囲、開始・終了基準の定義。工数・時間の見積り。 　レビュー参加者の選定など レビューの開始 　作業成果物(レビュー対象)の配布。参加者への作業説明など。 個々のレビュー 　作業成果物のレビュー。潜在的な欠陥、提案、質問の記録など。 懸念事項の共有と分析 　欠陥についての議論、分析、オーナーの割り当て。品質特性の評価、レビュー判定など 修正と報告 　欠陥レポートの作成。欠陥の修正など。	計画 　レビューの範囲を定義し、工数と時間を見積る 　レビュー特性(レビュータイプ、役割など)を識別する 　レビューの参加者を選び、役割を割り当てる 　開始基準と終了基準を定義する 　開始基準が満たされていることをチェックする レビューの開始 　レビュー資料および関連資料を配布する 　レビューの範囲、目的、プロセス、役割、レビュー資料を参加者に説明する 　参加者からのレビューについての質問に答える 個々のレビュー 　レビュー資料のすべてまたは一部をレビューする 　潜在的な欠陥、提案、質問を書き出す 懸念事項の共有と分析 　識別した潜在的な欠陥を分析する 　分析結果に対して担当者を割り当て、ステータスを付ける 　品質特性を評価し、文書化する 　レビューで見つけた欠陥などの結果を終了基準に対して評価し、 　レビュー判定を行う 修正と報告 　変更が必要な欠陥について欠陥レポートを作成する 　指摘を受けたレビュー資料を修正する 　欠陥について議論する 　欠陥のステータスを更新する 　メトリクスを収集する 　終了基準が満たされていることをチェックする 　終了基準に到達したときに、作業成果物を受け入れる	講義内容に合わせるため
75	追加	–	形式的レビューでの役割と責務	講義内容に合わせるため

原著作物のページ	区分	改定前	改定後	改定理由
76	変更	非形式的レビュー 「潜在的な欠陥の検出」を主な目的とする、形式的な(文書化した)プロセスに基づかない方法。作成者の同僚(buddy)やその他の人により実施する。 ウォークスルー 「欠陥の発見」を主な目的とし、作業成果物の作成者がミーティングを主導して、さまざまな技法やスタイルに関するアイデアの交換、参加者のトレーニング、合意の形成を行う テクニカルレビュー 「合意の獲得、潜在的な欠陥の検出」を主な目的とし、新しいアイデアの創出、作業成果物の改善、異なる実装方法の検討を行う。レビューアは、技術の専門家が行い、経験を積んだファシリテーターが主導するのが理想である。 インスペクション 「潜在的な欠陥の検出」を主な目的とし、ルールやチェックリストに基づいたプロセスで進行し、形式に沿ったドキュメントを作成する。開始基準と終了基準が指定され、書記は必須である	非形式的レビュー 決まったプロセスはなく、お金や時間をかけずにある程度の効果を出す レビューアの質で効果の大小が決まる 主な目的 潜在的な欠陥の検出 ウォークスルー 作成者が議事進行、机上で処理をたどる シナリオ、ドライラン、シミュレーションの形態を取る場合がある レビュー対象物の作成者が進行する 主な目的 欠陥の発見 ソフトウェアプロダクトの改善 異なる実装方法の検討 標準や仕様への準拠の評価 テクニカルレビュー レビューアは、技術の専門家が行い、経験を積んだファシリテーターが主導するのが理想である 新しいアイデアの創出、作業成果物の改善、異なる実装方法の検討を行う 主な目的 合意の獲得 潜在的な欠陥の検出 インスペクション 作成者ではなく、経験を積んだ進行役がレビューを議事を主導する ルールやチェックリストに基づいたプロセスで進行し、形式に沿ったドキュメントを作成する 開始基準と終了基準が指定され、書記は必須である 主な目的 潜在的な欠陥の検出 作業成果物の品質の評価と信頼の積み上げ 作成者の学習と根本原因分析による将来の類似欠陥の防止	講義内容に合わせるため

原著作物のページ	区分	改定前	改定後	改定理由
77	変更	アドホック レビューアは、このレビューに関するタスクのガイダンスをほとんどまたはまったく提供されない。一般的に、レビューアは作業成果物を順番に読んで懸念事項を識別するごとに記録に残す。 チェックリストベース 体系的に行われる。レビューアはレビューの開始時に配布されるチェックリストを使用して懸念事項を検出する。主な利点は、典型的な懸念事項の種類を体系的にカバーできることである シナリオとドライラン レビューアはシナリオ(作業成果物を読むための構造化されたガイドライン)を使用して、作業成果物の期待する使い方に基づいて、作業成果物に対して「ドライラン」(予行演習、空運転)を行う。 ロールベース レビューアは個々のステークホルダーの役割の観点から作業成果物を評価する。典型的な役割には、特定のエンドユーザーの種類や組織内の特定の役割がある。 パースペクティブ レビューアはそれぞれに異なるステークホルダーの観点でレビュー活動を行う。これにより、検出される問題のレビューア間における重複が少なくなる。さらに、レビュー対象の作業成果物から各レビューアの役割で導出する成果物(例えば、暫定的な受け入れテスト)を作成する。要件や技術的な作業成果物のレビューの場合、最も効果的な技法である。	アドホック 準備を必要としない場合によく行う レビューアは、このレビューに関するタスクのガイダンスをほとんどまたはまったく提供されない状態でレビューする レビューアは作業成果物を順番に読んで懸念事項を識別するごとに記録に残す レビューアのスキルに大きく依存し、複数のレビューアから重複した懸念事項が多く報告されてしまう場合がある チェックリストベース 体系的に行われる レビューアは、レビュー開始時に配布されるチェックリストを使用して懸念事項を検出する レビューに使用するチェックリストは、経験から導出した起こりえる欠陥に基づく一連の質問を列挙したものである 主な利点は、典型的な懸念事項の種類を体系的にカバーできることである シナリオとドライラン レビューアに、作業成果物を読むための構造化されたガイドラインを提供する レビューアはシナリオ(作業成果物を読むための構造化されたガイドライン)を使用して、作業成果物の期待する使い方に基づいて、作業成果物に対して「ドライラン」(予行演習、空運転)を行う このシナリオは、単純なチェックリスト項目よりも、特定の種類の欠陥を検出するためのよいガイドラインとなる ロールベース レビューアは個々のステークホルダーの役割の観点から作業成果物を評価する 典型的な役割は次のとおり 特定のエンドユーザーの種類(熟練者、初心者、年配者、子供など) 組織内の特定の役割(ユーザーアドミニストレーター、システムアドミニストレーター、性能テスト担当者など) パースペクティブ レビューアはそれぞれに異なるステークホルダーの観点でレビュー活動を行う レビューアごとに異なる観点を持つことで、より深く掘り下げたレビューが可能になる レビューア間における重複が少なくなる 典型的なステークホルダーの観点 エンドユーザー マーケティング担当者 設計者 テスト担当者 運用担当者　など チェックリストを使用することがある	講義内容に合わせるため

4 テスト技法

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
81	削除	テスト設計	−	講義内容に合わせるため
82	削除	テストの作り方	−	講義内容に合わせるため
84	変更	テスト担当者はテストケースを作成する際に、テスト技法を組み合わせて使用する。←銀の弾丸があるわけではない。 テスト分析、テスト設計、テスト実装の活動では、形式に従わない(ドキュメントがほとんどない)活動から形式に従った活動まで、テスト技法を使用できる。	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よい場合もありますし、すべてのテストレベルで使える技法もあります テストケースを作成する際、テスト対象を適切に網羅するために、テスト技法を組み合わせて使います	講義内容に合わせるため
89	追加	−	同等に処理される、または同様の結果をもたらすと想定したデータを「同値」と呼びます 有効な値の同値パーティションを「有効同値パーティション」と呼びます 無効な値の同値パーティションを「無効同値パーティション」と呼びます	講義内容に合わせるため
91	削除	無駄なテストをなくし、テスト回数を大幅に削減できる。	−	講義内容に合わせるため
92	追加	−	同値分割法のポイント 同値パーティションを考えることで、無駄なテストを抑え、テスト回数の大幅な削減が可能です 無効同値パーティションをテストケースで使用する場合、他の無効同値パーティションとは組み合わせず単独でテストします 様々な同値パーティションがあります 引数 返り値 内部値 仕様上の範囲 型や条件文の値の範囲(0～255など) 文字コードの範囲 ファイルの数や大きさ 時間に関する値 イベントの前後、タイミング、タイムアウト、データの入力や転送時間など メモリサイズ など	講義内容に合わせるため
97	追加	−	同値分割法の拡張です パーティションが、数値または順序付け可能な値で構成されるときに使用できます 境界の値の定義は、要求分析から実装のどの段階でも勘違いしたり、間違えたりしやすい、という経験則に基づいています	講義内容に合わせるため
97	追加	−	On、Off、In、Out　の説明を追加	講義内容に合わせるため

4 テスト技法

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
102	追加	−	デシジョンテーブルテストは、デシジョンテーブルの「規則／ルール」を高位レベルテストケースとして扱う技法です 組み合わせテストのアプローチの1つです 複雑な論理を含んだ仕様に対して、抜け・漏れを防ぐようにテスト設計できます デシジョンテーブル ある問題に関する複数の条件についてその成立／不成立の組み合わせを表形式に展開し、それぞれの条件に対応する結果(動作)を示したものです	講義内容に合わせるため
102	変更	ロジックを条件と動作に分けて、マトリクスで表現した表をデシジョンテーブルと呼び、デシジョンテーブルをテスト設計に使用するテスト技法をデシジョンテーブルテストと呼ぶ。	プログラムロジックを条件と動作に分けて、マトリクスで表現します	講義内容に合わせるため
102	追加	−	デシジョンテーブルの作り方 　最初にルール数に合わせて、テーブルを作成します 　条件が Y/N の2値である場合、2のn乗がルール数※になります 　1行目を列を半分にし、左側をYに、右側をNにします 　2行目はさらに列を半分にし、左側をYに右側をNにします 　最後の行まで繰り返します	講義内容に合わせるため
103	変更	デシジョンテーブルテストのポイント 考えられる条件の組合せを作るという点において、漏れや抜けがなくなる。 入力条件が複雑に(ANDやORで)絡み合う場合に有効。 テスト実施の進捗状況が分かり、他者にも伝えやすい。 テスト実施を分業できる。 デシジョンテーブルを「DT」と略すことがある。 一方で、以下には弱い。 条件そのものが間違っている。 条件の洗い出しが難しい(抜け漏れしやすい条件がある場合など)。 → 条件の洗い出しに失敗(条件の抜け漏れなど)。 条件の数が多い。 → DTの作成に時間がかる(テスト実装・実行に時間がかかる)	メリット 　考えられる条件の組み合わせという点において、漏れや抜けがなくなります 　入力条件が複雑に絡み合う場合に有効です 　テスト実行の進捗状況が分かり、他者にも伝えやすい 　テスト実行を分業可能です デメリット 　条件そのものが間違っていたり、条件の洗い出しに失敗すると、導出された高位レベルテストケースの意味がなくなってしまう 　条件の数が多いと、条件の洗い出しに時間がかかりすぎ、テストを実行する時間がなくなってしまう	講義内容に合わせるため

4 テスト技法

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
105	変更	※簡単化時の注意事項 条件の処理順が条件欄の記載順と同一であること。 不明の場合は単純に簡単化できない。	デシジョンテーブルの簡単化(圧縮) 　条件の組み合わせによっては、デシジョンテーブルが非常に大きくなることがあります 　そのため、すべての可能な条件の組み合わせから、出力に関連しない条件の組み合わせを削減することがあります 　この削減したデシジョンテーブルを「簡単化した(圧縮した)デシジョンテーブル」と呼びます デシジョンテーブルの簡単化(圧縮)の方法 　同じ結果になる共通条件を持つルールをまとめます 　結果に影響を及ぼさない条件の組み合わせを排除します 　YでもNでもよい(無関係)という意味で、「－」に置き換えます 　下の条件から順にまとめていきます 簡単化(圧縮)の注意事項 　条件の処理順が条件欄の記載順と同一であること 　不明の場合は単純に簡単化できません	講義内容に合わせるため
108	追加	－	イベントを起こして、プログラムの状態が正しく変化することをチェックします システム例 入力による画面の遷移を行うソフトウェア GUIアプリケーション、Web、など 同一の入力が内部の状態によって複数の意味を持つようなシステム 携帯電話のボタン、など	講義内容に合わせるため
111	追加	－	N-Switchカバレッジ 0 スイッチ　・・・　個々の遷移 1 スイッチ　・・・　2つの連続した遷移 2スイッチ　・・・　3つの連続した遷移	講義内容に合わせるため
114	追加	－	ユースケースのシナリオを実行するテストになります ユースケースがユーザーの活動を正確に反映していない場合は、よいテストになりません 網羅的なテストを行うためには、ユースケースの代替系列を正確に定義する必要があります 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します 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ます	講義内容に合わせるため
116	変更	基本フロー、代替フロー、例外フロー	基本系列、代替系列、例外系列	講義内容に合わせるため
126	削除	組合せのそれぞれに対してテストを実行し、テスト結果を分析する。	－	講義内容に合わせるため

4 テスト技法

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
134	削除	品質特性：（機能適合性、）性能効率性、互換性（共存性、相互運用性）、使用性、信頼性、セキュリティ、保守性、移植性（ISO 25010より）、「2.3 テストタイプ」を参照ください。	－	講義内容に合わせるため
134	削除	品質特性はMECEではなく、	－	講義内容に合わせるため
134	削除	本テキストでは、シナリオテスト、性能テスト、負荷テスト、リグレッションテストの設計方法を説明する	－	講義内容に合わせるため
134	追加	－	例えば、セキュリティのレベルを上げると、性能が落ちることがあります	講義内容に合わせるため
136	変更	CT	コンポーネントテスト	講義内容に合わせるため
141	削除	代表的なホワイトボックステスト技法	－	講義内容に合わせるため
142	削除	三項演算子を使って網羅率を上げようとする本末転倒なコードに注意する。 例） age = (age >= 17) ? 17 : age;	－	講義内容に合わせるため
143	削除	※ループ2回のテストはHuangの理論による。例えば、ループ内でデータを初期化し、ループの外でそのデータを参照する場合に、初期値設定の二重化の欠陥が検出されることがある。『ソフトウェアテスト技法』（ボーリス バイザー著）より。	－	講義内容に合わせるため
144	削除	→ fault-prone（欠陥の偏在箇所を予測する技術）に使われる。	－	講義内容に合わせるため
145	削除	C（複雑度） = e－n＋2p e：グラフ内のエッジの数（含まれるルート数） n：グラフ内のノードの数（分岐点の数） p：グラフのつながっていない部分の数	－	講義内容に合わせるため
147	追加	－	プログラムのデータについてその取り扱いを網羅します データや変数の使用の仕方に矛盾がないかを調べるためのテスト 存在しないものを使用したり消滅させたりする欠陥を検出します 定義、使用、消滅などが、順番通りかを調べるテストデータの組み合わせを与えます	講義内容に合わせるため
150	追加	－	テスト設計とテスト実行を同時並行で行い、学習しながらテストを進める セッションベースドテスト あらかじめ決められた時間枠内で探索的テストを行う テスト目的を含むテストチャーターに従ってテスト実行をする テスト担当者は、新しいチェックリストの作成、もしくは既存のチェックリストの拡張をテスト分析の一環として行う	講義内容に合わせるため

5 テストマネジメント

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
153	削除	独立性を確立すると、テスト担当者はより効果的に欠陥を発見できる。ただし、独立性は熟知していることの代わりにはならない。開発担当者は自分のコードにある多くの欠陥を効率的に検出できる。また、協力関係の維持や開発者の品質意識を高める工夫が必要である。	－	講義内容に合わせるため
154	追加	－	メリット 独立したテスト担当者は、開発担当者とは異なる背景、技術的視点、バイアスを持つため ステークホルダーが行った仮定について、検証、説明の要求、または反証を行うことができる 先入観がなく客観視できる デメリット 開発担当者が手を抜く可能性 独立したテスト担当者は、ボトルネックとして見られたり、リリース遅延で責任を問われたりすることがある 例えば、テスト対象の情報など	講義内容に合わせるため
155	削除	「品質を上げるにはコストがかかる。」(レビュー、文書・文書・文書)	－	講義内容に合わせるため
155, 156	変更	試験性	テスト容易性	講義内容に合わせるため
157	削除	ソフトウェア品質を例にすると・・・ モジュール性:凝集度が高い・結合度が低い。 単純性:複雑度が小さい。 一貫性、表現性、簡潔性、伝達性 追跡可能性、階層性　など ソフトウェアシステム独立性 (補完要素)完備性、説明性、製品管理性	－	講義内容に合わせるため
158	削除	試験性に影響を与える内部特性の例	－	講義内容に合わせるため
160	削除	テスト計画までの流れ	－	講義内容に合わせるため
161	追加		通常、プロダクトまたは組織のレベルでの、テストプロセスに関する汎用的な考え方を提供する	講義内容に合わせるため
161	変更	テストプロセスの開始ポイント、適用するテスト設計技法、テスト終了基準、実施するテストタイプを含む。	プロジェクトの複雑さ、ゴール、開発対象プロダクトの種類、プロダクトリスクを考慮する開始基準、終了基準の定義、適用するテスト技法、実行するテストレベル、テストタイプの選択を含む	講義内容に合わせるため

5 テストマネジメント

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
162	変更	参加メンバーのスキルや経験	参加メンバーのスキルや経験、特にドメイン知識のような類似プロジェクトやプロダクトのスキルや経験	講義内容に合わせるため
163	削除	テスト計画書に記載する内容	–	講義内容に合わせるため
164	変更	1.概要 2.識別番号 1.スコープ 2.参考資料 2.テスト項目 3.テスト対象 4.仮定と制約 5.ステークホルダー ■コミュニケーション ■リスク 1.サブプロセス 2.成果物 4.テスト十分条件 9.中断と再開の基準 10.組織のテスト戦略からの逸脱 3.訓練ニーズ	1. 1 概要 1. 2 文書ID 2. 1 適用範囲 2. 2 参考文献 3.2 テストアイテム 3.3 テスト対y層／テスト範囲 3.4 前提と制約 3.5 利害関係者 4. テストでのコミュニケーション 5. リスク管理表 6.1 テストサブプロセス 6.2 テスト成果物 6.4 テスト完了基準 6. 9 テスト中止基準とテスト再開基準 6. 10 組織レベルのテスト戦略からの逸脱 8.3 教育ニーズ	講義内容に合わせるため
167	変更	製品	プロダクト	講義内容に合わせるため
168	変更	テストケース数 テストカバレッジ 工数、コスト	計画したテスト環境の準備が完了した割合 テストケースの実行 故障率 確認テスト結果 テスト設計カバレッジ、テスト実行カバレッジ 要件 ユーザーストーリー 受け入れ基準 リスク コード タスクの完了、リソース毎の稼働状況、工数 テストに費やすコスト	講義内容に合わせるため

5 テストマネジメント

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
169	追加	－	開始基準 テスト可能な要件、ユーザーストーリー、そして／またはモデルが準備できている 前のテストレベルで終了基準を満たしたテストアイテムが準備できている テスト環境が準備できている 必要なテストツールが準備できている テストデータや他の必要なリソースが準備できている	講義内容に合わせるため
173	追加	－	構成管理とは技術的かつ管理的な指示と監視を適用する規範のことです この規範には次の目的があります 構成アイテムの特性を機能的、物理的に識別・文書化すること 特性に対する変更をコントロールすること 処理の変更と実装の状況を記録し、報告すること 特定の要求への整合を実証すること	講義内容に合わせるため
174	追加	－	開発成果物とテスト成果物の関係(例)	講義内容に合わせるため
175	削除	開発成果物とテスト成果物の関係(例)	－	講義内容に合わせるため
176	削除	テストケース仕様(詳細)	－	講義内容に合わせるため
178	削除	通常、影響度と発生可能性として表現する。	－	講義内容に合わせるため
178	変更	テスト対象に直接関係するリスク。	プロダクトの品質に影響を与えるリスク	講義内容に合わせるため
178	追加	－	プロジェクトの成功に影響を与えるリスク	講義内容に合わせるため
180	削除	リスク評価の例	－	講義内容に合わせるため
180	追加	－	プロダクトリスクの例 プロジェクトリスクの例	講義内容に合わせるため
182	削除	レビュー、テスト、分析、コンパイルをする中で検出できるが、それだけにとどまらず、ソフトウェアプロダクトや該当するドキュメントを利用するときに検出できることもある	－	講義内容に合わせるため
182	変更	偽陽性	偽陽性結果 テスト対象には欠陥が存在しないにもかかわらず、欠陥として報告したテスト結果	講義内容に合わせるため
182	変更	偽陰性	偽陰性結果 テスト対象に存在する欠陥を識別できなかったテスト結果	講義内容に合わせるため

5 テストマネジメント

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
185	削除	欠陥マネジメントツールを利用しよう。	−	講義内容に合わせるため
185	追加	−	欠陥ライフサイクル 欠陥レポートのステータス 状態遷移のイベント内容	講義内容に合わせるため

6 テスト支援ツール

VERISERVE

原著作物のページ	区分	改定前	改定後	改定理由
188	追加	−	テストツールの目的	講義内容に合わせるため
190	変更	ツールの利点 反復作業が減る。特に「データ駆動方式」。 一貫性や再実行性が増加する。 客観的な評価が可能になる。 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。	反復作業が減ります リグレッションテストの実行 環境の準備／復旧タスク 同じテストデータの再入力　など 一貫性や再実行性が増加します 整合性のあるテストデータ 同じ頻度と順序でのテスト実行 要件からの一貫したテストケースの抽出　など 客観的な評価が可能になります カバレッジの測定　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ります テスト進捗 欠陥率や性能計測結果の集計 グラフの作成　など	講義内容に合わせるため
190	変更	ツールのリスク 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。 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。 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するなど）。	ツールの効果を過大に期待します 例えば、ツールの機能性や使いやすさなど ツールを初めて導入する場合に必要な時間、コスト、工数を過小評価します 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します ツールによって生成されたテスト資産を保守するために必要な工数を過小評価します ツールに過剰に依存します 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ります ツールが保守されなくなるリスクがあります ビジネスを廃業したり、販売から撤退したり、別のベンダーに買収されたり 既に導入している、またはこれから導入を検討している重要なツール間での関係性と相互運用性の問題を軽視します ツールのサポート、アップグレード、欠陥修正に対するベンダーの対応が悪い オープンソースプロジェクトが停止します 新しいプラットフォームや新規技術をサポートできないことがあります ツールに対する当事者意識が明確でないことがあります 例えば、助言や手助け、および更新など	講義内容に合わせるため

- 本日はお時間をいただき、ありがとうございました。
- 本研修に関するご質問などございましたら、どうぞお気軽にお寄せください。

株式会社ベリサーブ

TEL：03-6629-8540 (代表)

〒101-0061

東京都千代田区神田三崎町3-1-16

神保町北東急ビル 9F

お問い合わせ

https://www.veriserve.co.jp/contact/