テストの改善、テストによる改善

SPI Japan 2006

2006/10/13（金）

電気通信大学 電気通信学部 システム工学科

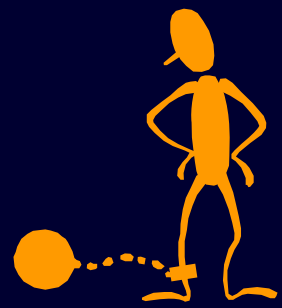
西　康晴

発表の流れ

- プロセス改善についての考察
- テストの改善
- 網羅性の改善
- ピンポイント性の改善
- テストによる改善
- テスト「道」

プロセス改善についての考察

- 私の周りだけですかね？
  - すぐ重量化してしまう
    - »「うちはレベル3で十分ですよ」
    - »「プロセスをキッチリ定義して、ガッチリ教育して、シッカリした社風にしましょう」
  - 「測定したって改善なし」
    - »「何を測ればいいか教えてください」
    - »「いろいろ測ってるんですが、改善しそうにないんですよね」
  - すぐに答えを欲しがる
    - »「とにかくデザインパターンを使えばいいと思ってるんですよ」
    - »「問題分析なんて、このくらいでいいじゃありませんか」
- プロセス改善が硬直化・重量化することが多い
  - プロセス改善を標準化と誤解している場合がある
  - 測定して統計的手法を使えば改善ポイントがすぐに見つかると思っている
  - 問題点の改善は行うが、「根本原因」を分析し未然防止ができない



改善の原則って何だろう

- 改善の本質は、問題点の根本原因やメカニズムを見つけることである
  - 応急処置：現象への対処
  - 再発防止：同じ問題の防止
  - 未然防止：水平展開による類似の問題の防止
  - 改善パラダイムでは、初出の問題の未然防止は非常に難しい
- CMMIにはちゃんと書いてある
  - 注：CMMI初心者の読解力
  - 『原因分析と解決』の目的は、欠陥およびその他の問題の原因を特定すること、および処置をとり将来における発生を予防することである。[PA155]
- しかし例示されている技法は「QC7つ道具」レベル
  - 欠陥およびその他の問題を選択する手法の例を以下に示す：PA155.IG101.SP101.SubP102.N102]
    - » パレート分析／ヒストグラム／プロセス能力分析
  - 根本原因を明らかにする手法の例を以下に示す：[PA155.IG101.SP102.SubP102.N101]
    - » 特性要因図 (魚骨図)／チェックシート

改善そのものをマネジメントし改善する

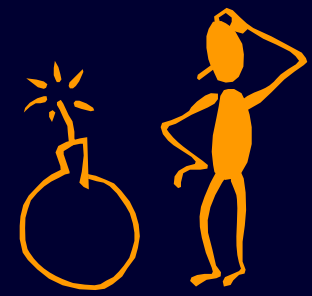
- 改善そのものの質に着目する
  - 泥臭く具体的な「悪さ」に着目して改善しよう
    - » 格好良い標準プロセスに合わせれば良いというものではない
  - 改善に結びつくような「悪さ」の測定をしよう
    - » 結果系の測定だけから得られるものは非常に少ない
  - 改善に結びつくような「悪さ」の原因分析をしよう
    - » 抽象的ななぜなぜ分析はプロセス改善の重量化を導く
- 改善そのものをマネジメントし改善する必要がある
  - 「改善そのもののレベル」を定義する
    - » （誤った）標準化
    - » 気が付いたものを直す
    - » 問題点の現象を測って現象を抑えようとする
    - » 問題点の根本原因やメカニズムを把握し再発防止する
    - » 問題点の根本原因やメカニズムを水平展開し未然防止する
  - 治療に例えると分かりやすい

改善そのもののレベルを向上しよう

- 改善レベル5：予防レベル
  - 問題点の根本原因やメカニズムを水平展開して改善し未然予防するレベル
- 改善レベル4：治療レベル
  - 問題点の根本原因やメカニズムを把握して改善し再発防止を行うレベル
- 改善レベル3：対症療法レベル
  - メトリクスを測定し問題点を把握しているにもかかわらず、問題点の根本原因やメカニズムを理解せずに改善するレベル
- 改善レベル2：民間療法レベル
  - 感覚的・経験的に問題点を把握し改善するレベル
- 改善レベル1：薬漬けレベル
  - 問題点を把握せず標準に闇雲に合わせるだけのレベル
- 改善レベル0：暴飲暴食レベル
  - 全く改善しないレベル

テストプロセス改善モデル

- 欧米ではテストプロセス改善モデルがいくつか発表されている
  - TPI（SOGETI）
  - SW-TMM（イリノイ大学）
  - 参考にする程度でよいだろう
- 基本的思想は開発プロセス改善と同じ
  - テストに関する要素作業を定め達成度をレベル分けする
  - 要素作業に多少の差がある
    - » テスト設計の改善とテスト実施の改善は様相が異なる
  - プロセスモデルに近づけるのではなく、改善の体質を持つプロセスにする
- 開発プロセス改善と乖離させてはいけない
  - テストプロセスのみ改善しても、本質的な問題の解決にはならない
    - » テストをするだけでは品質は向上しない
  - テストプロセスの改善で終わらせずに、
    開発プロセス全体の改善を促進するように進めなくてはならない
    - » テストプロセスの改善モデルに従おうとすると、
      どうしてもテストにしか目がいかなくなってしまう



テストプロセス改善：TPIのKey area

1. テスト戦略
   - A: 単一高位レベルテスト
   - B: 高位レベルテストの組み合わせ
   - C: 高位レベルテストに低位レベルテストor評価の組み合わせ
   - D: 全てのテストレベルと評価レベルの組み合わせ
2. ライフサイクルモデル
   - A: 計画、仕様化、実行
   - B: 計画、準備、仕様化、実行、完了
3. 関与の時点
   - A: テストベースの完成時点
   - B: テストベースの開始時点
   - C: 要求定義の開始時点
   - D: プロジェクトの開始時点
4. 見積もりと計画
   - A: 実証された見積もりと計画
   - B: 統計的に実証された見積もりと計画
5. テスト仕様化技法
   - A: 非公式の技法
   - B: 公式の技法
6. 静的テスト技法
   - A: テストベースのインスペクション
   - B: チェックリスト
7. 尺度
   - A: プロジェクト尺度（成果物）
   - B: プロジェクト尺度（プロセス）
   - C: システム尺度
   - D: 組織尺度（複数システム）
8. テストツール
   - A: 計画ツールと制御ツール
   - B: 実行ツールと分析ツール
   - C: テストプロセスの自動化強化

テストプロセス改善：TPIのKey area

9. テスト環境
- A: 管理され制御された環境
- B: 最適環境におけるテスト
- C: 要求に即応できる環境

10. オフィス環境
- A: 適切かつタイムリーなオフィス環境

11. コミットメントと意欲
- A: 予算と時間の割り当て
- B: プロジェクト組織に統合されたテスト
- C: テストエンジニアリング

12. テスト役割と訓練
- A: テスト管理者とタスク
- B: （公式の）手法的、技法的、機能的支援、管理
- C: 公式の内部品質保証

13. 方法論の展開
- A: プロジェクトで固有
- B: 組織で共通
- C: 組織で最適化、研究開発

14. コミュニケーション
- A: 内部コミュニケーション
- B: プロジェクト内コミュニケーション（欠陥、変更管理）
- C: テストプロセスの品質についての組織内コミュニケーション

15. 報告
- A: 欠陥
- B: 進捗（テストと成果物のステータス）、アクティビティ（コスト、時間、マイルストーン）、欠陥と優先度
- C: リスクと提言、尺度による実証
- D: ソフトウェアプロセス改善特性への提言

テストプロセス改善：TPIのKey area

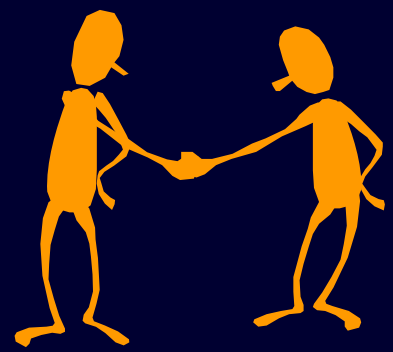
16. 欠陥管理
    - A: 内部欠陥管理
    - B: 柔軟な報告機能による広範な欠陥管理
    - C: プロジェクト欠陥管理
17. テストウェア管理
    - A: 内部テストウェア管理
    - B: テストベースとテスト対象物の外部管理
    - C: 再利用可能なテストウェア
    - D: テストケースに対する追跡性システム要求
18. テストプロセス管理
    - A: 計画と実行
    - B: 計画、実行、監視、調整
    - C: 組織における監視と調整
19. 評価
    - A: 評価技法
    - B: 評価戦略
20. 低位レベルテスト
    - A: 低位レベルテストライフサイクルモデル（計画、仕様化、実行）
    - B: ホワイトボックス技法
    - C: 低位レベルテスト戦略

注意点

- 鵜呑みにしない
- 現実・現場の問題点を見る
- 改善を回すプロセスが重要
- 開発プロセス改善につなげる

テストの改善

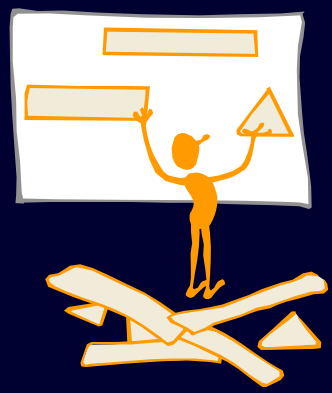
- テストの原則
  - 網羅とピンポイント
    - » 網羅：漏れなくテストすることでバグを全て見つける
    - » ピンポイント：バグがあるところだけをテストすることでバグを全て見つける
- テストの改善
  - 網羅性の改善
    - » 「どうして、このバグが見つからなかったんだろう？」
    - » テスト漏れというテストプロセスの「悪さ」に着目する
  - ピンポイント性の改善
    - » 「どうして、このバグが作り込まれたんだろう？」
    - » バグの作り込みという開発プロセスの「悪さ」に着目する
- テストの改善のゴール
  - 少ない手間で早くたくさん危険なバグを検出する
  - 開発全体の改善を促し、そもそもバグが作り込まれないよう未然予防する



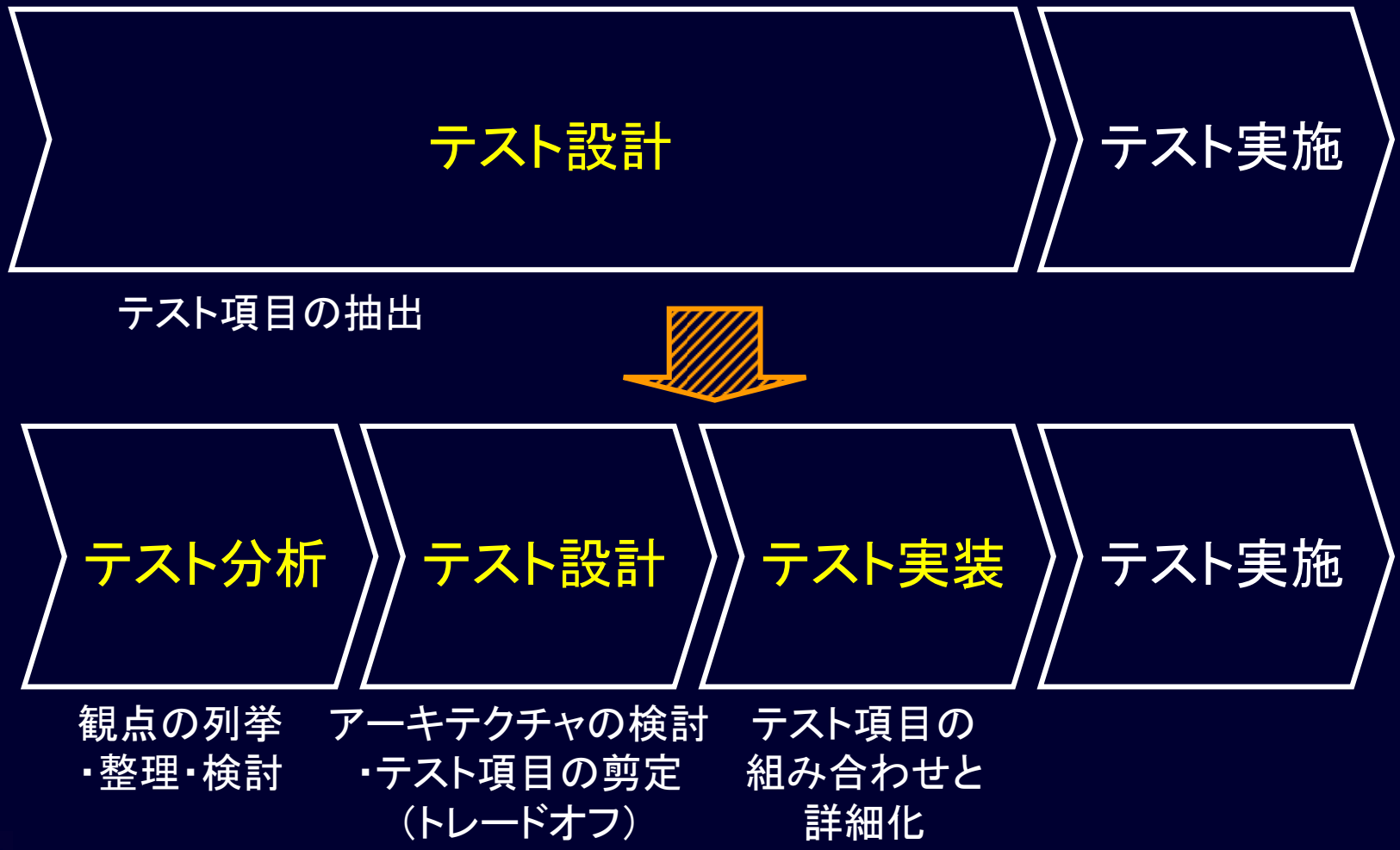
開発とテストが協調することで
品質の高い製品が早く安く出荷できるようになる

網羅性の改善～テスト漏れに着目

- テスト設計プロセスの確立
  - テスト観点の分析
  - 段階的詳細化による設計
  - 同値クラス／カバレッジ／組み合わせの設計
- ディレイ分析によるテスト漏れの原因把握
  - 見逃した（市場で、もしくは出荷前に見つかった）バグが、本来はどのレビューやテストのフェーズで見つかるべきだったか、テスト設計の漏れかテスト実施の漏れか露呈バグの見落としか、を分析することで問題点を把握する
- 実施漏れに対する改善策
  - 実施プロセスの泥臭い改善
    - » ハードウェアの生産プロセスの改善に似た側面もある
  - リスクベースドテストによる優先度設定の改善
  - テスト容易性設計によるテスト実施の効率化
- 設計漏れに対する改善策
  - カバレッジ分析



テスト設計プロセスの確立：分析・設計・実装



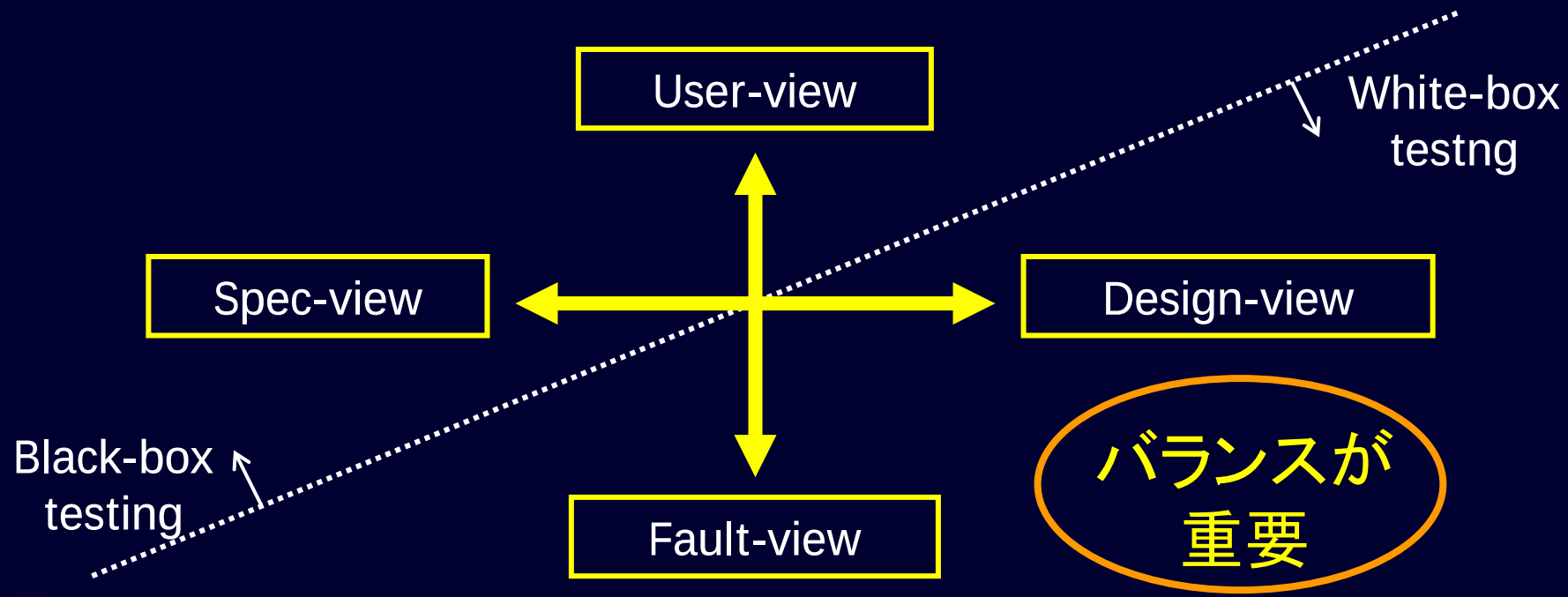
テスト設計プロセスの確立：段階的詳細化

- 分析
  - テスト目標を定める
  - ユーザの要求や要求仕様、システム構造などから
    テストに必要な観点を列挙し整理する
- 設計
  - 整理されたテスト観点を基に、
    テストのアーキテクチャを決める
  - 段階的に詳細化しながらテスト項目を作成する
    - » フローグラフからパスを抽出するなど
    - » 当該製品の具体的なパラメータは当てはめない
- 実装
  - 実施可能なテストケースを作成する
    - » 具体的なパラメータを当てはめる
    - » テスト項目を組み合わせる
    - » 期待結果を導出するなど
- それぞれレビューが必要

段階的詳細化により
適切に粒度を管理でき
再利用性も向上する

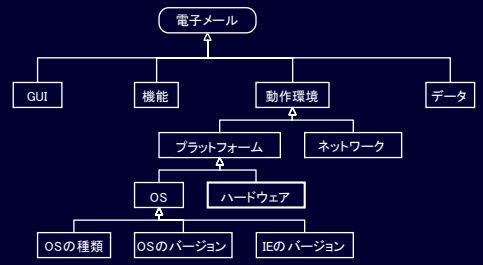
テスト設計プロセスの確立：代表的な4つの観点

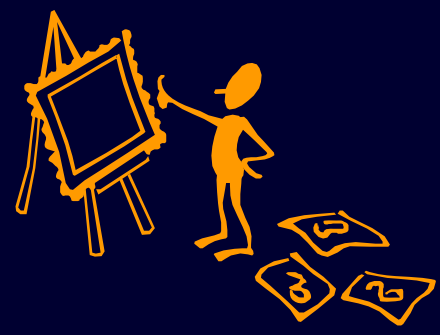
- User-view
  - ユーザが何をするかを考える
- Fault-view
  - 起こしたいバグを考える
- Spec-view
  - 仕様を考える
- Design-view
  - 設計やソースコードを考える



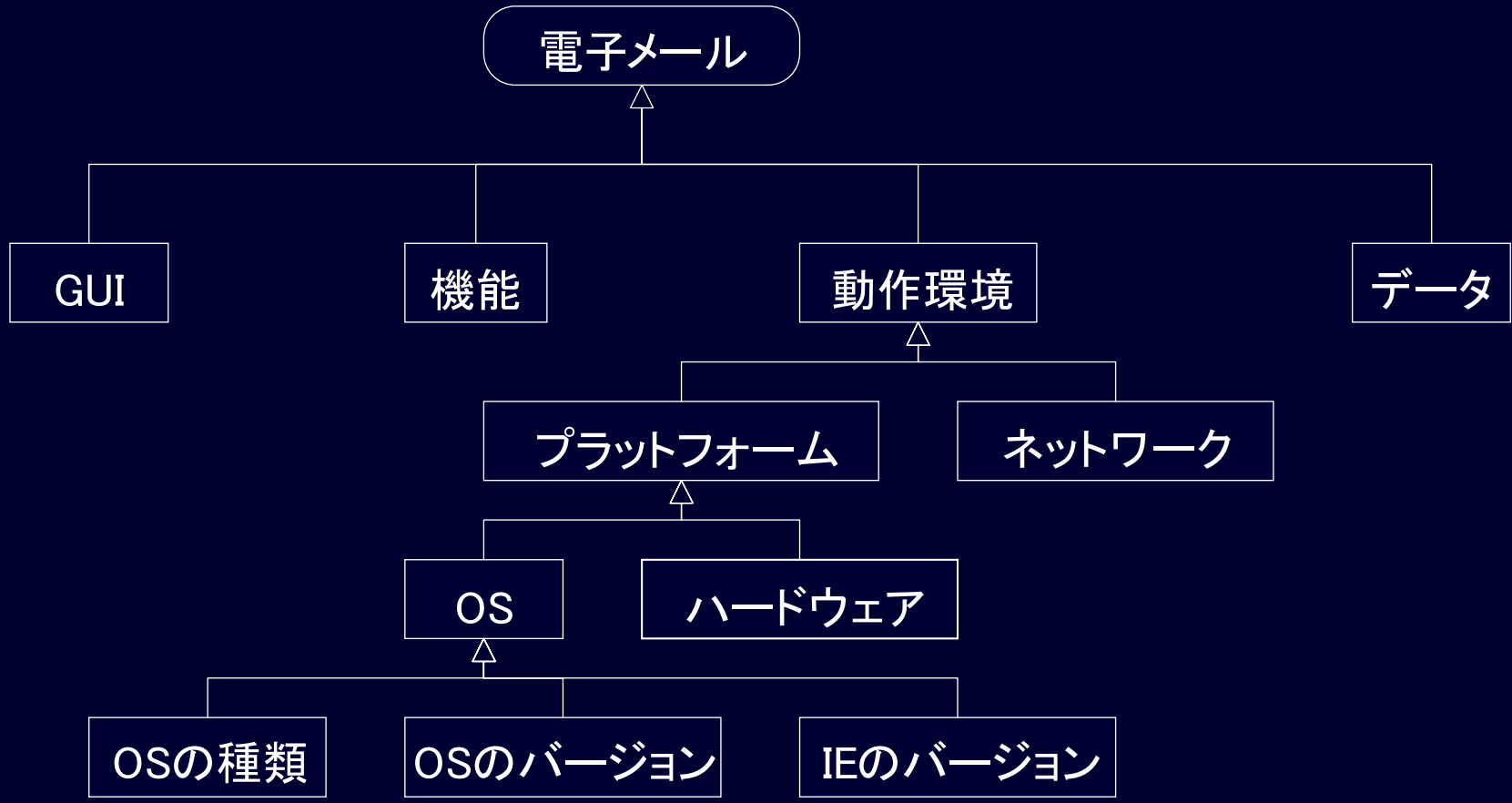
テスト設計プロセスの確立：テスト観点と分析

- テストの観点などをモデリングするための記法：NGT
  - 名称：NGT（Notation for Generic Testing）
    - » 一般的なテストのための記法
    - » テスト分析モデルとテスト設計モデルを記述する
    - » 個々のテスト技法で扱うものは記述しない
  - テスト分析モデル
    - » ビュー、フォーカス・クラス、関連による、テスト対象のモデリング
  - テスト設計モデル
    - » フォーカス・クラス、ズームイン/ズームアウト、
      テスト項目数の概算によるトレードオフ
      確定/暫定フォーカス・クラスによるリスクの明示
  - テスト設計の主な考慮事項
    - » テスト観点と同値クラス
    - » カバレッジ基準と境界値・閾値
    - » 組み合わせ



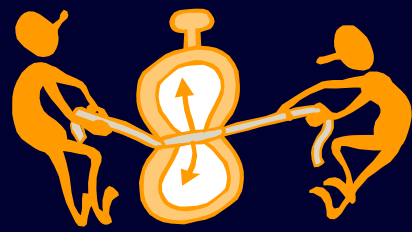


NGTによるテスト観点の分析の例



リスクベースドテストによる優先度設定の改善

- テスト項目ごとに推測された「(出荷後の品質)リスク」を基にしてテストの優先度を決める
  - リスク指数の高いテスト項目を優先してテストする
  - 残りリスク指数が許容範囲内になったらテストを止める
    - » リスクを金額に換算できるとEVMに統合できる
  - テスト進捗・修正進捗をリスク指数で報告する
  - テストできなかった項目のリスクを明らかにし、別の手段を講じて潰す



- 「(出荷後の品質)リスク」とは？
  - リスクファクターの組み合わせによって評価する
    - » 動いた場合の価値／動かない割合／動かない場合のシステムに対するダメージ／動かない場合のプロジェクトに対するダメージ／不具合が見つかる割合
- リスク指数算出の考え方
  - リスクファクターを掛け合わせて考える
    - » リスク評価＝実現確率×損害(Rick Craig流)
    - » リスク優先度指数＝重要度×修正優先度×発生確率(Rex Black流)
  - 単純にかけ算で考えると失敗する
    - » たまにしか発生しないけど致命的な不具合は、リスク指数が低く計算されてしまう場合がある

テスト容易性設計によるテスト実施の効率化

- Testability(テスト容易性)を考慮して
  ソフトウェアの設計やレビューを行う
  - Testability: テスト容易性(DFT: Design For Test)
  - James BachのTestability要素
    - » 操作性: 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
    - » 観測性: 見えるものしかテストできない
    - » 制御性: ソフトウェアをうまくコントロールすれば、
      テストを自動化し最適化できる
    - » 分解性: テストの適用範囲を制限することにより、
      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: 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: 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: 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
  - テスト容易性が高いとテスト実施が効率化されるので、
    同じ工数でより多くの領域をテストできるようになり、テスト漏れが減る

網羅性の改善～テスト漏れに着目

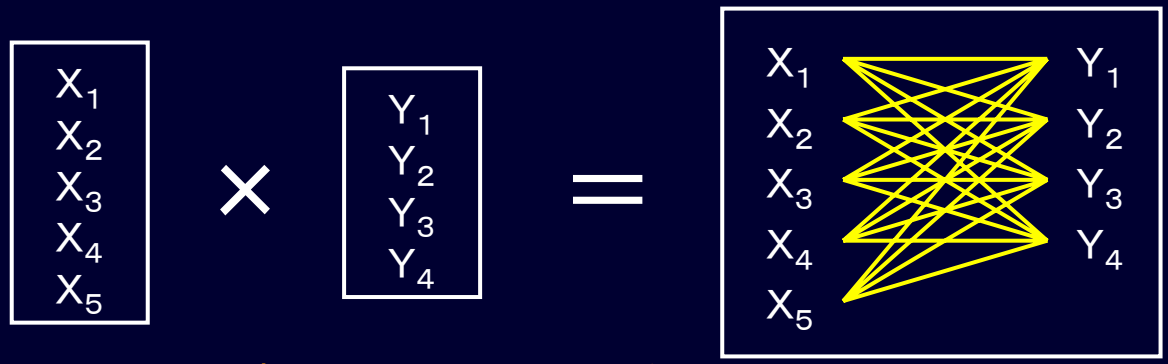
- カバレッジ分析によるテスト漏れの原因把握
  - カバレッジ基準を決めて、カバレッジを測定し、評価する
    - » 各フェーズの成果物のトレーサビリティを全て評価すべきである
    - » 決めたカバレッジは100%達成するようにする
    - » 適用除外をきちんと管理し、分析するプロセスを決める
    - » カバレッジ基準を達成するようにテストする方法と、別の基準で設計・実施したテストを評価する方法がある
  - 見逃したバグが、どのカバレッジを網羅しなかったことによるものか、を分析することで問題点を把握し、改善する
    - » カバレッジは制御パステストだけの概念ではない
    - » 実施カバレッジの問題か、設計カバレッジの問題か
    - » 観点や同値クラスの抜けか
    - » カバレッジ基準の甘さか、カバレッジ率の低さか
    - » 境界値テストのミスか、閾値が存在したか
    - » 組み合わせバグなのでカバレッジを上げても見つからないか
    - » 工数がかかりすぎるための間引きの失敗か
    - » デグレードか

網羅性の改善～テスト漏れに着目

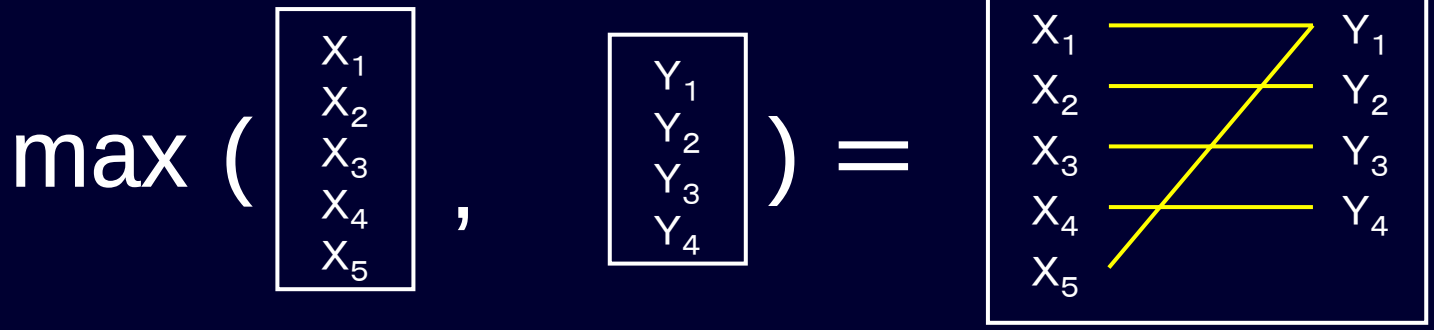
- カバレッジ分析によるテスト漏れの原因把握
  - 同値クラス漏れに対する改善策
    - » 観点のモデリングによるステークホルダー間のコミュニケーションの促進
    - » 要求分析へのフィードバックによる再発防止・未然防止
  - カバレッジ漏れに対する改善策
    - » 境界値テストの徹底
    - » カバレッジ基準の見直し
    - » 同値クラスの確定のためのレビュー
  - 組み合わせ漏れに対する改善策
    - » 直交配列表やAll pairs法などの組み合わせテスト手法の導入
    - » グレーボックステストによる依存関係を分析し、組み合わせを間引く
    - » Wモデルで組み合わせのボトルネックとなる部分の依存関係を低く保つ

グレーボックステストによる依存関係の間引き

- 組み合わせ網羅テストと単一網羅テスト
  - 単機能テスト項目5件の機能Xと、単機能テスト項目4件の機能Yがある
    - » 組み合わせテスト項目数は 5×4 ＝ 20 件

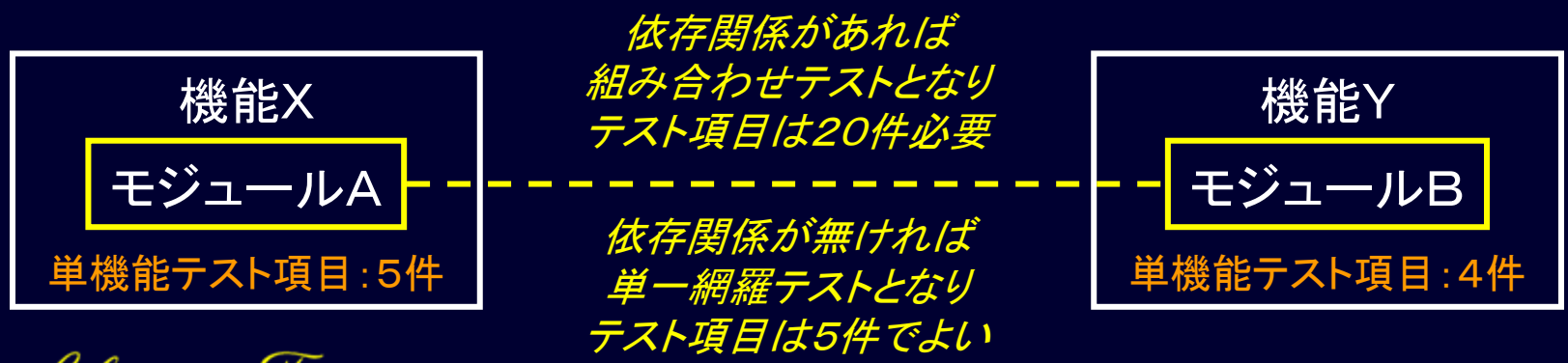


- » 単一網羅テスト項目数は max(5,4) ＝ 5 件

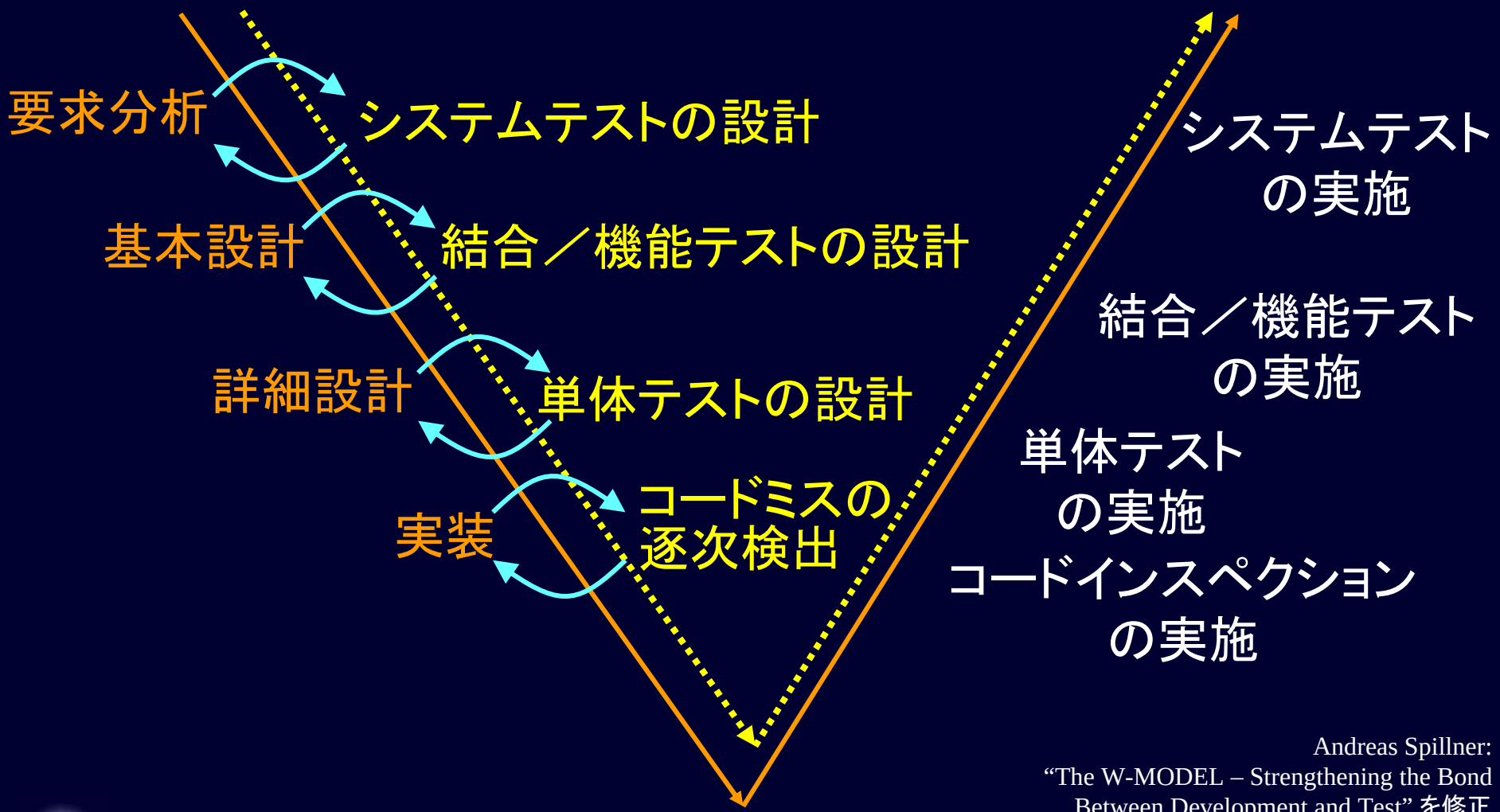


グレーボックステストによる依存関係の間引き

- グレーボックスで依存関係を分析しながらテストを設計すると、機能Xと機能Yの組み合わせテストを構造的にレビューし判断して間引くことができる
  - 組み合わせが必要なのは、機能Xの処理に応じて、機能Yの処理が変わるから
    - » 機能を構成するモジュール間に依存関係があるから
  - 構造的にレビューして、依存関係が無かったり、影響伝播が抑えられていることが保証できれば、単一網羅テストで十分となる
    - » ブラックボックステストに少しホワイトボックス的観点が入るのでグレーボックステスト
    - » 実務的には完全に間引くわけではなく、優先度を落とすことになる



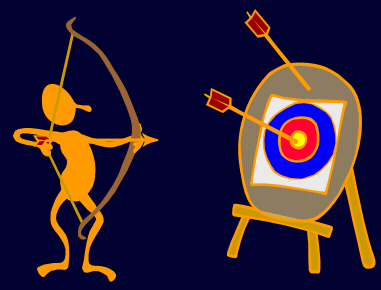
Wモデルによって依存関係を低く保つ



Andreas Spillner:
“The W-MODEL – Strengthening the Bond Between Development and Test” を修正

ピンポイント性の改善～バグの作り込みに着目

- 不具合モード分析
  - 類似バグの分類
  - バグ作り込みの原因把握
  - 不具合モードの抽出
  - 不具合モードに基づいたテスト設計

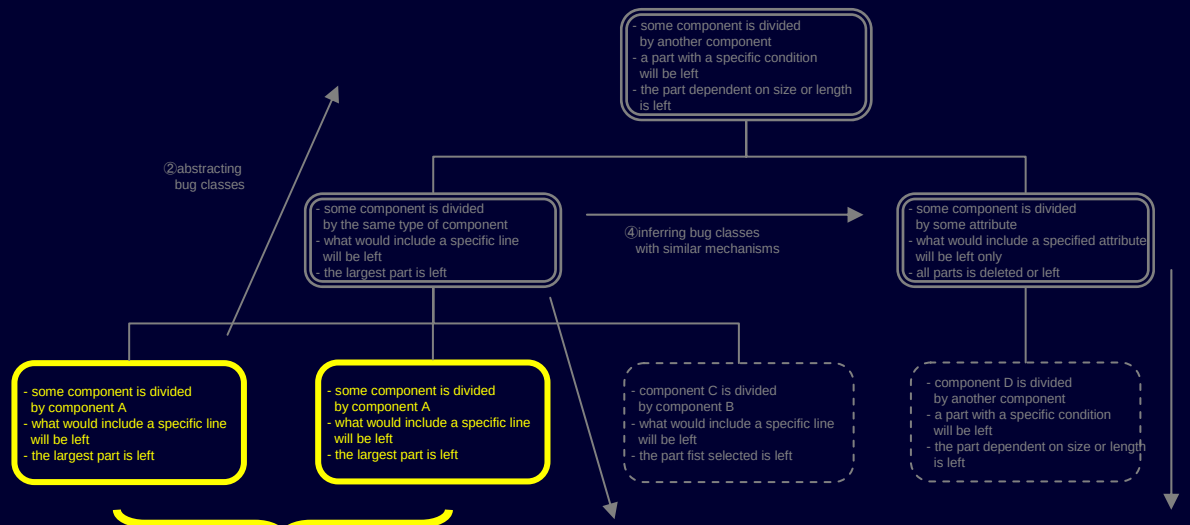


- バグが多いところを狙ってテストするための手法
  - テスト項目が膨大になるため、全てを網羅的にテストすることは不可能
  - そこでバグが偏在しているところを狙い、ピンポイント性を向上する
    - » プロジェクトが異なっても、似たような原因によるバグが多い
    - » 経験の豊富なテスト技術者は偏在しているバグを推測してテストしている
    - » FMEA（故障モードによる影響解析）をソフトウェアテストに応用したものである
  - バグが偏在しているところを不具合モードとして整理しておき、水平展開して類似のバグを狙いテスト精度を上げる
    - » 不具合モードの抽出の観点が精度を大きく左右する
    - » 機能的観点は容易性が高く精度は低い / 構造的観点は容易性が低く精度は高い
    - » 我々は認知的観点（プロダクト・アフォーダンス）に基づいている

不具合モードを用いたバグの水平展開

①類似の不具合を収集する

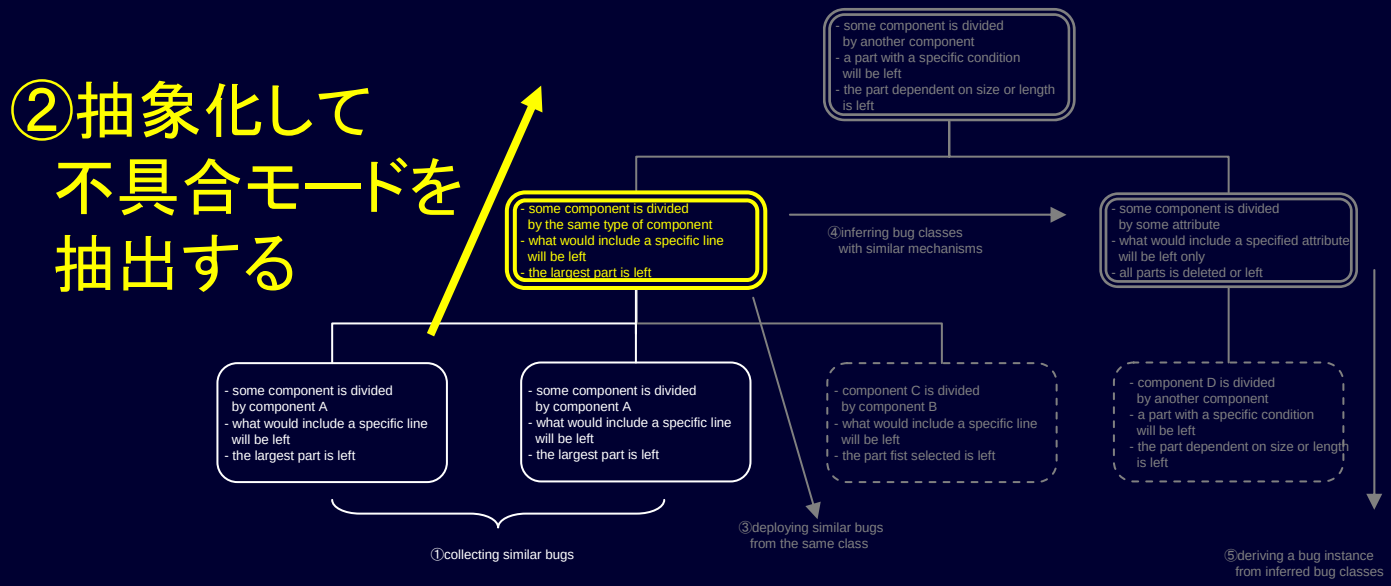
- 不具合の事例をインスタンスとして収集する
- 不具合の記述を書き直す
- 似たような不具合に分類する



不具合モードを用いたバグの水平展開

②抽象化して不具合モードを抽出する

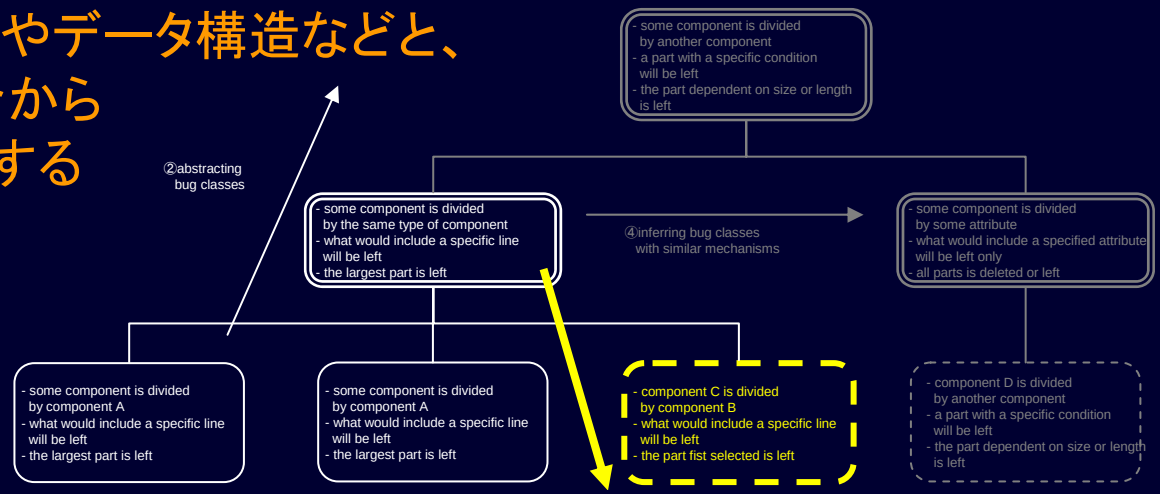
- 似たようなバグを抽象化し不具合モードを抽出する



不具合モードを用いたバグの水平展開

③同じ不具合モードに属する新たな不具合のインスタンスを推測する

- 不具合モードの記述に当てはまるテスト対象の機能やデータ構造を探す
- 当てはまった不具合モードの示すメカニズムや兆候を持つ不具合を推測する
- 当てはまった機能やデータ構造などと、具体化した不具合からテスト項目を設計する



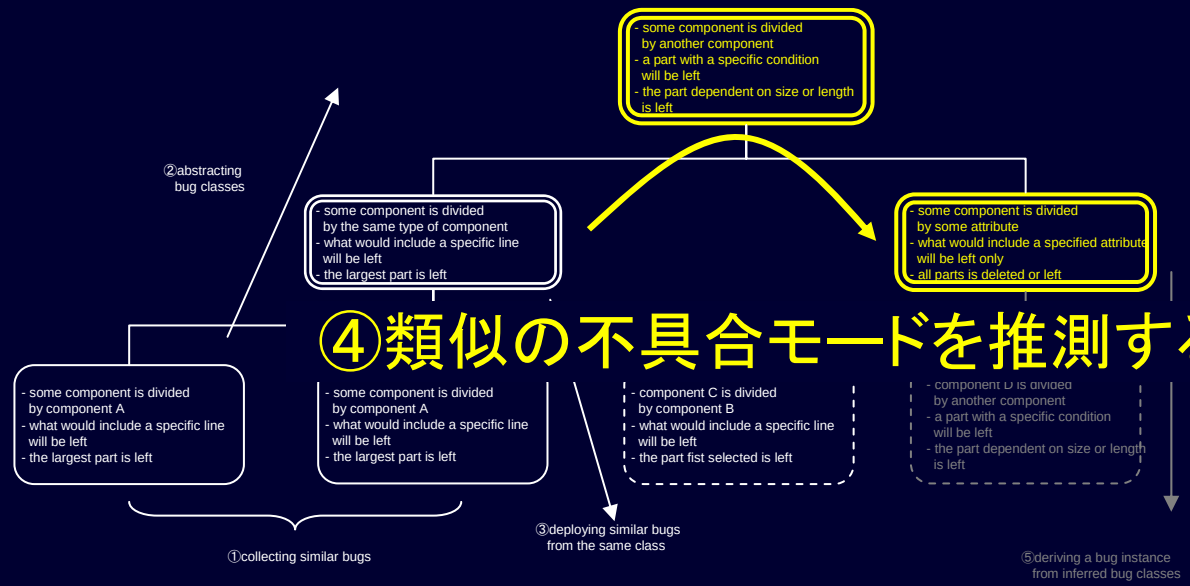
※当てはまる機能やデータ構造などが多ければ、バグを見つけられる可能性は低い

③同じ不具合モードに属する新たな不具合のインスタンスを推測する

不具合モードを用いたバグの水平展開

④類似の不具合モードを推測する

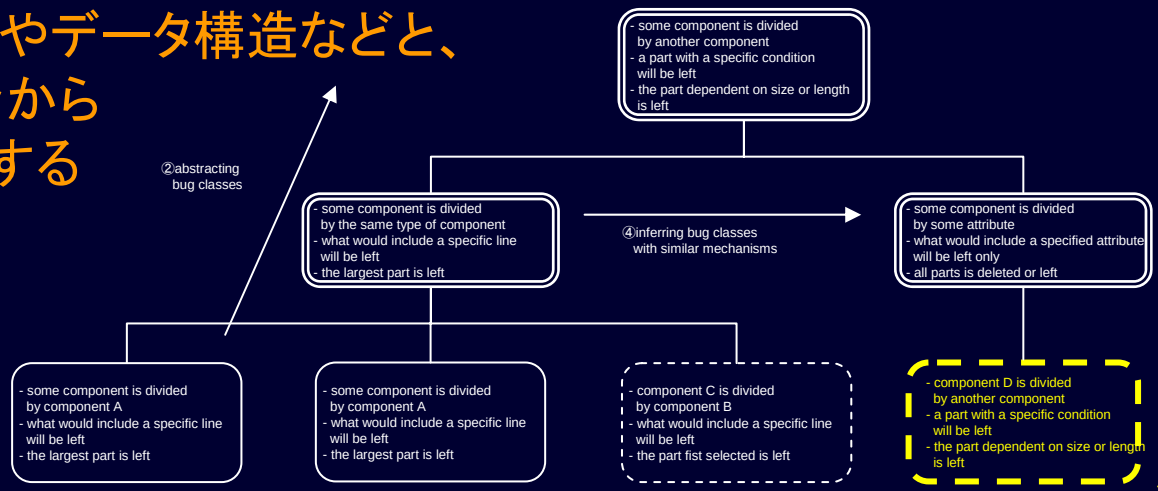
- 上位の不具合モードを抽出し、
  そこから新たに下位の不具合モードを推測する



不具合モードを用いたバグの水平展開

⑤推測された不具合モードから新たな不具合のインスタンスを推測する

- 不具合モードの記述に当てはまるテスト対象の機能やデータ構造を探す
- 当てはまった不具合モードの示すメカニズムや兆候を持つ不具合を推測する
- 当てはまった機能やデータ構造などと、具体化した不具合からテスト項目を設計する



⑤推測された不具合モードから新たな不具合のインスタンスを推測する

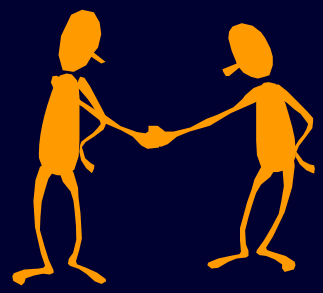
不具合モードによるテスト設計プロセス

- 不具合モード蓄積プロセス
  - 不具合モード抽出対象プロジェクトの不具合列挙
  - 類似不具合分類
  - 不具合モードの抽出
  - 不具合モードツリーの整理
- プロジェクト適用プロセス
  - 不具合モードテストの適用範囲の選定
  - 適用可能な不具合モード群の選定
  - 現プロジェクトで検出された不具合の不具合モードへの当てはめ
  - 当てはめた不具合モードの展開による不具合インスタンスの推測
  - 推測された不具合インスタンスを検出するテストの設計と実施
  - 不具合モードテストの適用範囲と適用可能な不具合モード群の見直し

テストによる改善

- テストの改善を進めると、プロセス全体の改善が必要だと気づく
  - テスト容易性設計
    - » テストが容易になるよう気遣うことで、設計をスッキリさせる
  - 同値クラス漏れの分析・設計へのフィードバック
    - » 分析・設計での同値クラスの考慮不足を改善する
  - 同値クラスの確定のためのレビュー
    - » テスト設計で想定した通りに例外的な処理をしないよう改善する
  - Wモデルの導入
    - » 分析・設計・実装とテスト設計とを同時に進めることにより、レビューを改善するとともに、依存関係を減らして設計をスッキリさせる
  - 不具合モードによるレビュー・開発の改善
    - » バグが作り込まれやすいところをレビューしたり、バグを作り込まないように開発ガイドラインを作る

テストの改善をきっかけにして、
そもそもバグを作り込みにくい
開発プロセスに改善していく



ソフトウェアテストとは

- ソフトウェアテストはどんな技術か
  - 少ない手間で早くたくさん危険なバグを検出する技術
  - 製品出荷後の品質リスクを見積もる技術
  - 開発全体を改善するきっかけとなる技術
- 開発プロセス全体を改善することで
  長期的に品質を改善し続けられるように
  テストを改善していかなくてはならない
  - まずは基本的な手法を体系的に導入し、
    自分たちがどんなテストをしているのかを把握する
  - テスト漏れを少しでも減らすべくテストを分析し改善する
  - テストで見つけたバグの原因を分析し、上流にフィードバックする
  - 組み合わせテスト工数を減らすために、設計をスッキリさせる
  - テストで検出したバグを分析し、バグを作り込んだ原因を明らかにし、
    要件定義や設計、マネジメント、プロセスを改善する

Boris Beizer のテスト「道」

フェーズ0 ― テストはデバッグの一部である

フェーズ1 ― テストの目的は、
ソフトウェアが動くことを示すことである

フェーズ2 ― テストの目的は、ソフトウェアが動かないことを
示すことである

フェーズ3 ― テストの目的は、何かを証明することではなく、
プログラムが動かないことによって発生する
危険性をある許容範囲までに減らすことである

フェーズ4 ― テストは行動ではなく、テストをしないで品質の
高いソフトウェアを作るための精神的訓練である

ご清聴ありがとうございました

電気通信大学　西 康晴
http://blues.se.uec.ac.jp/
nishi@se.uec.ac.jp