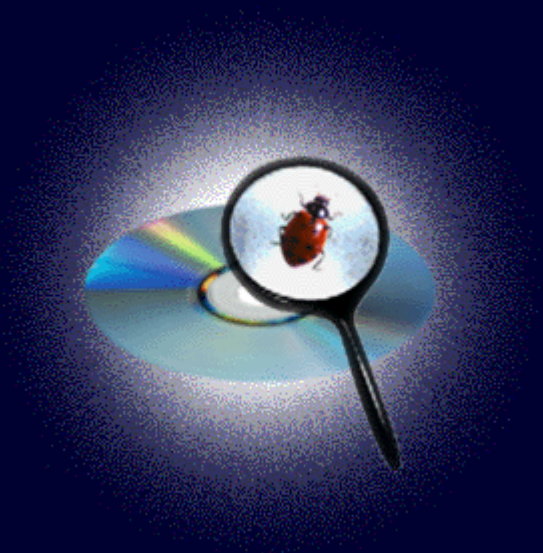
ソフトウェアテストの動向



@IT ソフトウェアテスト・ミーティング

2005/5/31 (火)

電気通信大学 電気通信学部 システム工学科

西　康晴

© NISHI, Yasuharu

自己紹介

- **身分**
  - ソフトウェア工学の研究者
    - » 電気通信大学 電気通信学部 システム工学科
    - » ちょっと「生臭い」研究／ソフトウェアテストやプロセス改善など
  - 先日までソフトウェアのよろず品質コンサルタント
- **専門分野**
  - ソフトウェアテスト／プロジェクトマネジメント／ＱＡ／ソフトウェア品質学／ＴＱＭ全般／教育
- **共訳書**
  - 実践ソフトウェア・エンジニアリング／日科技連出版
  - 基本から学ぶソフトウェアテスト／日経ＢＰ
  - ソフトウェアテスト293の鉄則／日経ＢＰ
  - 基本から学ぶテストプロセス管理／日経ＢＰ
- **もろもろ**
  - TEF: テスト技術者交流会
  - SESSAME: 組込みソフトウェア管理者技術者育成研究会
  - 情報処理学会 アクレディテーション委員会 ソフトウェアエンジニアリング分科会
  - 経済産業省 組込みソフトウェア開発力強化推進委員会

SESSAME

© NISHI, Yasuharu

TEF: Testing Engineer's Forum

- **テスト技術者交流会**
  - 1998年9月に活動開始
    - » 現在900名強の登録
    - » MLベースの議論と、たまの会合
  - http://www.swtest.jp/forum.html
  - お金は無いけど熱意はあるテスト技術者を
    無償で応援する集まり
  - "JaSST：ソフトウェアテストシンポジウム"も開催している
    - » 実行委員は手弁当／参加費は実費＋α
    - » 毎年4Qに開催／今年はのべ約900名の参加者
    - » 2005/7/15（金）に大阪でも開催
  - 「基本から学ぶソフトウェアテスト」や
    「ソフトウェアテスト293の鉄則」の翻訳も手がける
    - » ほぼMLとWebをインフラとした珍しいオンライン翻訳チーム

© NISHI, Yasuharu

SESSAME: 組込みソフトの育成研究会

- **組込みソフトウェア技術者管理者育成研究会**
  - Society for Embedded Software Skill Acquisition for Managers and Engineers
  - 2000年12月に活動開始
    - » 200名強の会員／MLベースの議論と、月イチの会合
  - http://www.sessame.jp/

- **中級の技術者を10万人育てる**
  - ＰＣソフトウェアのような「そこそこ品質」ではダメ
    - » 創造性型産業において米国に劣り、コスト競争型産業でアジアに負ける
    - » ハードウェアとの協調という点で日本に勝機があるはず
  - 育成に必要なすべてを開発する
  - オープンプロダクト／ベストエフォート
    - » 文献ポインタ集、知識体系（用語集）、初級者向けテキスト、スキル標準など
    - » 7つのワークグループ：組込みMOT・演習・MISRA-C・ETSS・子供・高信頼性
  - セミナーだけでなく、講師用セミナーも実施

© NISHI, Yasuharu

講演の流れ

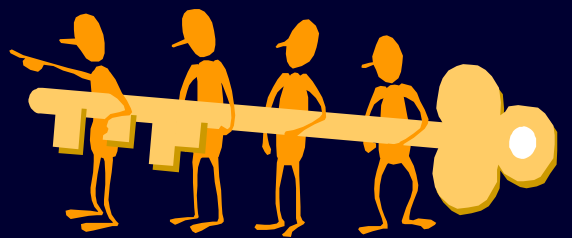
- 最近のソフトウェアテストの動向
- テスト今昔物語
- テストツールのよい使い方
- テストの基本の「キ」
- テストプロセス
- テスト改善
- ブラックボックスによる組み合わせテスト
- リスクベースドテスト

© NISHI, Yasuharu

最近のソフトウェアテスト の動向

- **品質事故の報道が増えている**
  - 組込み系が特に多い
    - » 携帯電話やデジタル家電
  - エンタープライズ系も決して少なくない
    - » 不具合による対応コストや手戻りコストが利益を蝕んでいる
    - » 成功するプロジェクトはわずか26.7% (日経コンピュータ)
  - 信頼性を重視されるソフトウェアも品質事故を起こしている
    - » 金融機関、新幹線、自動車・・・
- **ソフトウェアのテストや信頼性のイベントが増えている**
  - JaSST / JaSST in Osaka
  - WOCS (クリティカルソフトウェアワークショップ)
  - ソフトウェア信頼性ワークショップ
  - ソフトウェア品質シンポジウム
  - メディア各社のイベント
  - 有償セミナー: セミナー企業

  など

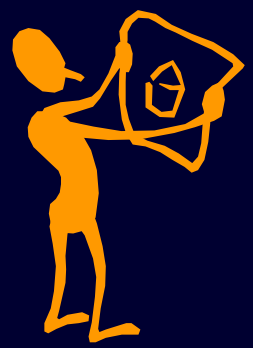


© NISHI, Yasuharu

最近のソフトウェアテストの動向

- **雑誌や Web の記事が増えている**
  - IDG: JavaWorld
  - 日経BP: IT pro、システム構築、ソフトウェア、コンピュータ
  - CQ出版: Design Wave
  - 技術評論社: Web+DB Press、Software Design
    - » 組込みプレス？テストプレス？
  - @IT: ソフトウェアテスト・エンジニアの本音、Rational Edge
  - ITmedia: Webアプリケーション開発における理論的・計画的なテストの実現
- **良質な参考書が増えた**
  - 知識ゼロから学ぶソフトウェアテスト
  - 体系的ソフトウェアテスト入門
  - 基本から学ぶテストプロセス管理
  - ソフトウェアテスト12の必勝プロセス

  など



© NISHI, Yasuharu

最近のソフトウェアテストの動向

- いろいろなコミュニティでテストの重要性を理解してきている
  - 開発系
  - プロセス改善系
  - 保守系
  - スキル標準系
- 構想中の計画もある
  - 某センター
  - 資格試験
- 海外も盛んである
  - アメリカではSTAR、ヨーロッパではICSTESTが盛況
  - 中国やインドでソフトウェア品質に関するカンファレンスが開催されている
  - 各国で資格試験が始まっている
  - アメリカではテスト専門のジャーナルが発刊
  - オープンソース向けのテストプロジェクトもある

© NISHI, Yasuharu

講演の流れ

- 最近のソフトウェアテストの動向
- テスト今昔物語
- テストツールのよい使い方
- テストの基本の「キ」
- テストプロセス
- テスト改善
- ブラックボックスによる組み合わせテスト
- リスクベースドテスト

© NISHI, Yasuharu

テスト今昔物語：テストの位置づけ

- **今は昔：**
  - テストはバグを見つけていればよい
  - バグを見つけるので精一杯だ
  - 納期まで一生懸命テストして、出来る限りのバグを潰せばよい
  - テストをしっかりやるとコストがかかり工数が伸びる
- **となむ語り伝えたるとや、しかし最近は：**
  - テストでバグを見つけるのは当然である。ユーザビリティ、セキュリティ、運用容易性、保守容易性などの品質特性を含め、顧客の満足度をちゃんと評価しなければならない
  - テストでは、バグ検出型と動作保証型の2種類のバランスが重要である
  - テストは、出荷時の「リスク」を測定する技術である。
  - テストをしっかりやると上流が改善され手戻りが減るので、結局は開発全体のコストが減り工数が少なくなる

© NISHI, Yasuharu

テスト今昔物語：いつまでテストするか

- **今は昔：**
  - テストは無限なのでやりきれない
  - とにかく納期まで徹夜して、出荷した後はあきらめる
  - 信頼度成長曲線で出荷を判定したいが、当たらない
- **となむ語り伝えたるとや、しかし最近は：**
  - 期限内にどこまでテストできるか、どこはテストしないか、
    いかに効率を上げるか、QCD のバランスをどうとるかを考える
  - リスクベースドテストを適用して、テスト項目の優先順位を
    体系的・包括的に定める
  - 信頼度成長曲線の合致度はプロセスの成熟度に大きく依存する。
    教科書や学者、他社事例を鵜呑みにせず、自分たちなりの
    使い方をしっかり検討したうえで適用する

© NISHI, Yasuharu

テスト今昔物語：開発とテスト

- **今は昔：**
  - テストはバグを見つけるのが仕事だから、開発とは関係ない
  - テストの設計や実施や自動化は、ビルドの後に始めればよい
  - 第3者テストチームを開発と明確に分離しなければならない
- **となむ語り伝えたるとや、しかし最近は：**
  - テストではバグを見つけるだけでなく、バグのパターンを把握して
    フィードバックし、上流でバグを減らさねばならない（品質の作り込み）
  - 開発と同期してテストの設計や実施、自動化を進めないと
    非効率になるし、上流で品質が作り込めない
  - 第3者テストチームの独立性は確保しつつ、いかに開発と
    コミュニケーションを取ってフィードバック、フィードフォワードするか
    が重要である

© NISHI, Yasuharu

テスト今昔物語：テスト技術

- **今は昔：**
  - ホワイトボックステストとブラックボックステストなら知っている
  - 組み合わせテストはムリだ
  - 同じようなバグが出てるのに見逃してしまった
  - 時間が無ければレビューや単体テストを省いてよい
  - 受け入れテストって、ただ操作すればよいのだろう
- **となむ語り伝えたるとや、しかし最近は：**
  - ホワイトとブラックは知っていて当然。グレーボックステストで
    いかに組み合わせテストや回帰テストのヒット率を上げるかが重要
  - 組み合わせテストは直交表やAll pair法などで効率化するのは当然。
    禁則などをいかに反映させるか、異なる水準をどう割り付けるか
  - 過去バグの再利用は当然。どうやって過去バグを分析し整理して
    再利用すれば効率やヒット率が上がるかを検討している
  - レビューや単体テストをしっかりやると、
    後半のテストの効率が圧倒的に向上する
  - 受け入れテストでは顧客満足度を構成する要件をしっかり把握し、
    きちんとValidationする。しかし品質向上はValidationだけに頼らない

© NISHI, Yasuharu

テスト今昔物語：テストプロセス

- **今は昔：**
  - どう進めればいいのか分からず、行き当たりばったりになっている
  - ひたすらExcelにテストケースを挙げておりレビューや再利用ができない
  - マネジャーには、テスト項目の消化率を報告している
  - テストは予想外なことばかり起こるので管理できない
- **となむ語り伝えたるとや、しかし最近は：**
  - テストプロセスは（独立して）必要。どうやって改善するかを考えている
  - テストのために分析、設計、実装、実施およびレビューを実施し、開発の粒度とテスト項目の粒度を揃えて再利用を容易にする
  - 様々な視点からのカバレッジを報告するのは当然。達成価値で報告したり、残りリスクで報告するにはどうすればよいかを検討している
  - まず現状をきちんと認識・把握・分析し、発生しそうな問題点を整理しておき、あらかじめ発生しないような軽減策を立て、対策を考えておき、兆候を管理する

© NISHI, Yasuharu

テスト今昔物語：スキルやキャリア

- **今は昔：**
  - テストは初心者にやらせればよい
  - テストは経験がモノを言う
  - 勉強したいけど何も資料がない
- **となむ語り伝えたるとや、しかし最近は：**
  - 開発スキルが無い人間の担当するテストはムダやモレが多いため、結局は損になる。どうやってジョブローテーションをしたり、動機付け・意識付けをしていくか。キャリアパスやスキル標準をどのように作成しようか
  - いろいろな技法を身につけた方が、圧倒的にテストの効率が上がる。直交表、CFD法など最新のテスト技法を身につけたい。品質目標にあわせて技法を組み合わせるべき
  - 教科書、標準、規格などいろいろ情報がある

© NISHI, Yasuharu

テスト今昔物語：テストツール

- **今は昔：**
  - ツールを導入するかどうか悩んでいる
  - ツールは敷居が高い
  - ツールは銀の弾丸である
  - ツールは使えない
- **となむ語り伝えたるとや、しかし最近は：**
  - ツールを導入するのは当然。どのツールを買うか、どれとどれを組み合わせるか、普及はどうするかを検討している
  - フリーのツールや簡単なツールが多く出てきたり、開発環境に統合されるようになってきた
  - ツールにできること、できないこと、難しいことをきちんと切り分け、投資対効果を最大にする
  - ツールを使いこなせるかどうかはプロセスの成熟度によるので、ツールの浸透度とプロセス改善を融合する

© NISHI, Yasuharu

テスト今昔物語：アウトソーシング

- **今は昔：**
  - テストは面倒なのでアウトソーシングすればよい
  - アウトソーシングすれば良いテストをしてくれる
  - アウトソーシング先はテストのプロの企業なので安心だ
- **となむ語り伝えたるとや、しかし最近は：**
  - テストは品質向上のコア技術なので、アウトソーシングや
    コンサルタントを上手に活用して
    自社のテストプロセスをきっちり整備する必要がある
  - アウトソーシング先に良いテストをしてもらうために、
    発注元が品質目標やバランスを明らかにしなくてはならない
  - スキルはアウトソーシング先によって千差万別。組織的な取り組みを
    している企業は非常に少なく、エンジニアごとの当たり外れが大きい

© NISHI, Yasuharu

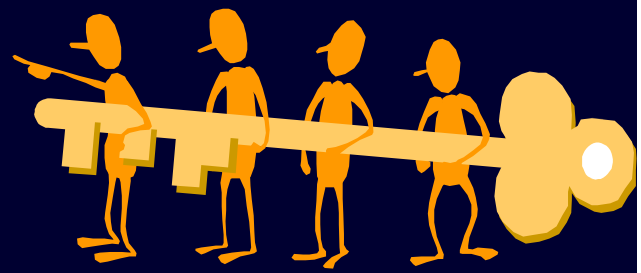
講演の流れ

- 最近のソフトウェアテストの動向
- テスト今昔物語
- テストツールのよい使い方
- テストの基本の「キ」
- テストプロセス
- テスト改善
- ブラックボックスによる組み合わせテスト
- リスクベースドテスト

© NISHI, Yasuharu

各フェーズで用いるテストツールの例

- **コードインスペクション／単体・結合テストフェーズ**
  - コードインスペクションツール
  - 単体テスト支援ツール
- **機能テスト／回帰テストフェーズ**
  - GUI自動操作ツール
  - カバレッジ測定ツール
  - 実行時エラー検出ツール
  - 性能測定ツール
  - バグ管理ツール
- **システムテストフェーズ**
  - 負荷テストツール



© NISHI, Yasuharu

テストツールのよい使い方

- 「弘法は筆を選ぶ」
  - 使い慣れたツールを持っておく
  - 始めは使いにくく感じるもの
  - できればツールエキスパートを社内に育てるとよい
- ツールが自動で何でもやってくれると思ったら大間違い
  - できること、できないこと、難しいこと、に分ける
  - 調達時のスクリーニングに使うという手もある
- テストプロセスが無いとポテンシャルを引き出せない
  - ツールを導入することでプロセス改善のきっかけを掴むという手もある
- 開発部隊にも協力してもらう
  - スクリプト作成、保守、ケース効率化
- テストが始まってからツールを選ぶと効果が薄いことがある
  - 開発の最初でツールを選定しておき、ツールを活用できるところを増やす
  - 余裕のある上流にスクリプト化やメンテナンスを進めておく

© NISHI, Yasuharu

テストツールのよい使い方

- **複数のツールを組み合わせて効果を倍増させる**
  - 負荷テストとメモリ監視、自動操作とカバレッジ測定など
- **テストツール用のスクリプトはソフトウェアと同じようにメンテナンスしないと効果が出ない**
  - この世の中、行き当たりばったりで効果が出るものは少ない
- **まずはフリーのツールで効果を体感する**
  - いろいろなフリーのツールが増えてきた
  - 使い勝手などは商用の方がよい場合が多い
- **高いツールは「親切」なところを選ぶ**
  - サポートがしっかりしていたり、自社の事情に合ったアドバイスが重要
- **風通しのよいベンダを選ぶ**
  - コミュニティやユーザ同士でどんどん情報交換することを促進するベンダを選ぶ

© NISHI, Yasuharu

講演の流れ

- 最近のソフトウェアテストの動向
- テスト今昔物語
- テストツールのよい使い方
- テストの基本の「キ」
- テストプロセス
- テスト改善
- ブラックボックスによる組み合わせテスト
- リスクベースドテスト

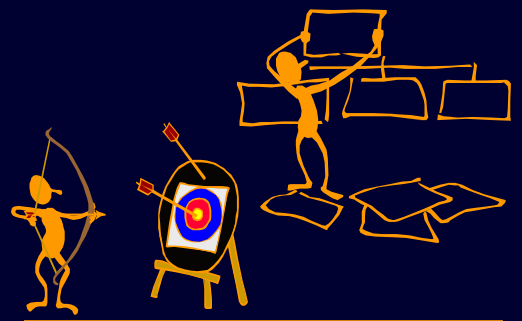
© NISHI, Yasuharu

基本的なテストの方針

- 「テストしなければバグは見つからない」
  - 漏れなくテストを行う
  - 漏れなくテストを行うためには、
    思いつきでテストをあげてはならない

- 「大事なところをテストする」
  - 最もバグを検出しやすいテストを行う
  - バグが起きてはいけない順にテストを行う

網羅



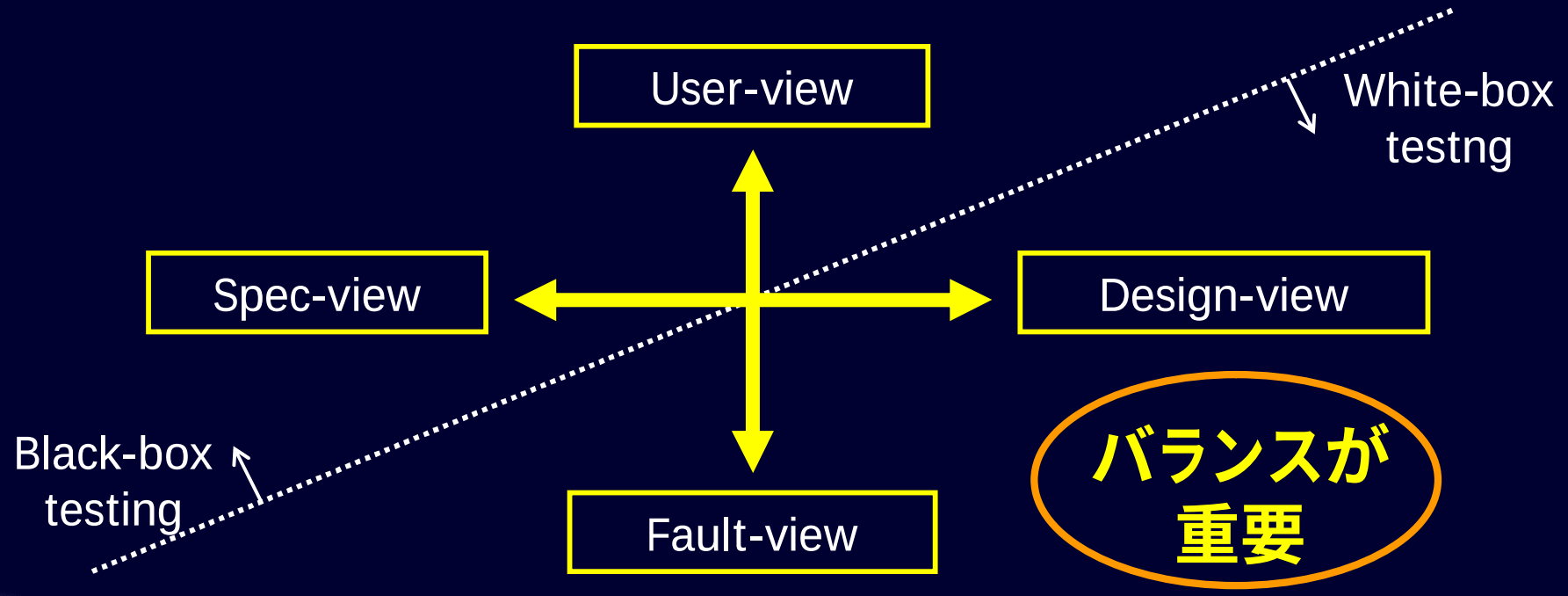
ピンポイント

少ない手間で早くたくさん危険なバグを検出する

© NISHI, Yasuharu

テストの狙い：4つの視点

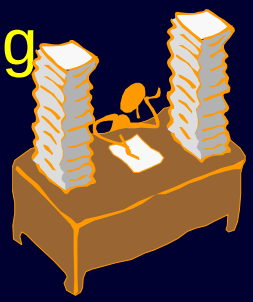
- User-view
  - ユーザが何をするかを考える
- Fault-view
  - 起こしたいバグを考える
- Spec-view
  - 仕様を考える
- Design-view
  - 設計やソースコードを考える



© NISHI, Yasuharu

テスト設計の手法

- User-oriented testing
  - 統計的操作テスト法
  - ユースケース網羅法
- Spec-oriented testing
  - 機能網羅法
  - 同値分割法
  - 境界値分析法
  - GUIパステスト法
  - 原因結果グラフ
  - CFD法
  - トランザクションフローテスト法
  - 直交配列法：HAYST法
- Design-oriented testing
  - 制御パステスト法
  - 状態遷移テスト法
  - データフローパステスト法
- Fault-oriented testing
  - リソースパステスト法



基本的な手法は
最低限マスターしておく
必要がある

© NISHI, Yasuharu

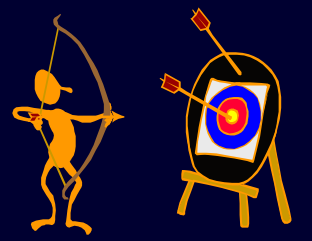
カテゴリ型システムテスト

- システムテストとは
  - 機能の動作の確認後のテスト
  - 意地悪テスト／総合試験
  - 品質特性を確認するテスト
- ストレス系のシステムテスト
  - ボリュームテスト
    - » 大きなデータ、たくさんのデータを与えるテスト
  - ストレージテスト
    - » ディスクやメモリなどを残り少ない状態で使うテスト
  - 高頻度テスト
    - » 短時間にたくさん処理させるテスト
  - ロングランテスト
    - » 長時間実行させるテスト
- 環境系のシステムテスト
  - 構成テスト
    - » 他のものから悪影響が与えられないか、のテスト
  - 両立性テスト
    - » 他のものに悪影響を与えないか、のテスト
  - 互換性テスト
    - » データなどの交換をさせるテスト
- 評価系のシステムテスト
  - 障害対応性テスト
    - » 電源を引っこ抜くなどの障害を起こしてみるテスト
  - セキュリティテスト
    - » 機密保護などのセキュリティ機能の穴を突くテスト
  - ユーザビリティテスト
    - » 操作性や視認性などを評価するためのテスト

© NISHI, Yasuharu

回帰テスト(リグレッションテスト)

- **変更・修正・保守の際のデグレードを防止するテスト**
  - デグレードを検出して直すのは非常に難しい
    - » 変更個所に新たに作り込んでしまうバグ
    - » 変更個所以外に思わぬ影響を及ぼしてしまったバグ
    - » 抑えられていた"タガ"が外れてしまうバグ
  - 変更などのたびに、今まで行ったテストを全て行うのが原則
    - » 設計工程から機能間の関連情報をテスト工程に伝え、上手に間引きする
    - » 変更の影響をあらかじめ十分検討できるプロセスとアーキテクチャが必要
  - 回帰テストで見逃した不具合を綿密に分析し、設計力を向上する
- **回帰テスト設計のガイドライン**
  1. 重要な機能をテストする
  2. 操作フローで関連する機能をテストする
  3. 似たような機能名の機能をテストする
  4. 構造的に関連する機能をテストする
     - » モジュール呼び出し
     - » データ共有
     - » 外部エンティティ共有
     - » タイミング共有



回帰テストを
合理的に減らせるよう
設計を改善する

© NISHI, Yasuharu

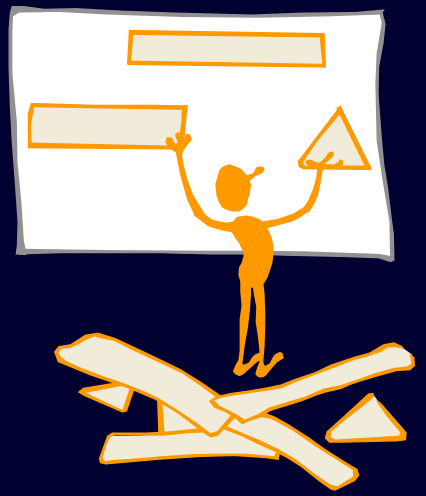
講演の流れ

- 最近のソフトウェアテストの動向
- テスト今昔物語
- テストツールのよい使い方
- テストの基本の「キ」
- テストプロセス
- テスト改善
- ブラックボックスによる組み合わせテスト
- リスクベースドテスト

© NISHI, Yasuharu

テスト設計プロセス：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からテスト仕様を作成する
    - » 機能一覧を作成したり、テストリスクを定めるなど
- **設計**
  - テストの方針（戦略）を決める
  - 段階的に具体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
- **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**

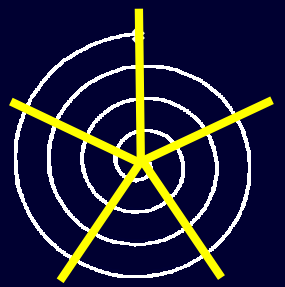


**設計・実装により
粒度を管理し
再利用性を向上する**

© NISHI, Yasuharu

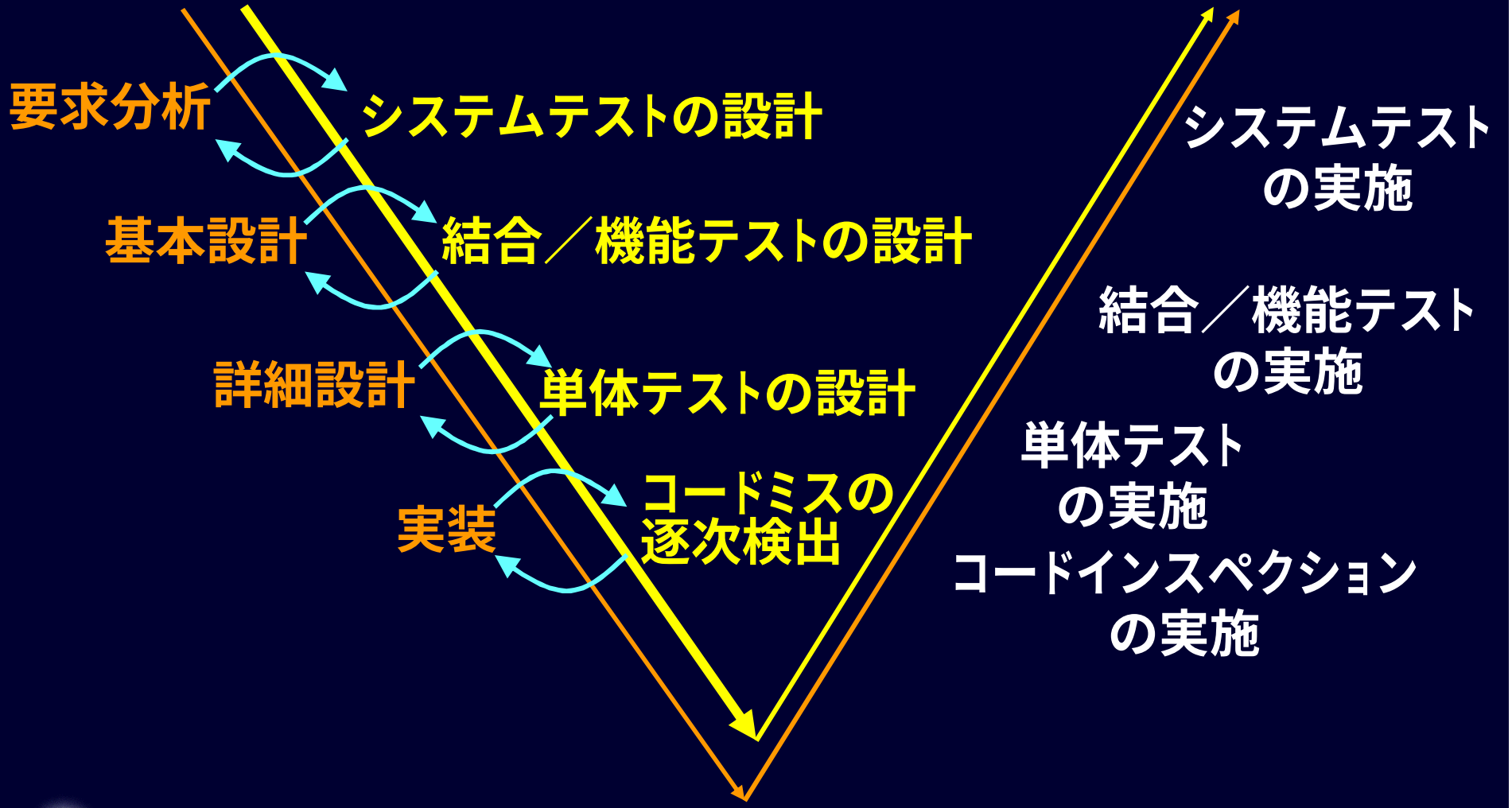
テスト実施プロセス：サイクル型テスト

- **短いテストサイクルで管理する**
  - 1週間前後のテストサイクルを単位とする
    - » 状況に応じたアクションを取るきっかけとする
    - » テストプロジェクトにリズムを作る
    - » 縁起の悪い報告こそ、きちんと行う
  - サイクル毎に問題点を把握し改善する
    - » 全員がプロセスの問題点やバグの状況などを把握し、全員で手を打つ
    - » 始めは計画と合わないが、テスト後半にはきちんと合うようにする
  - サイクルごとに種々のテストのバランスを変化させる
    - » テスト側受け入れテストの内容を高度にする
    - » 保証型テスト（機能テスト、回帰テスト）と
      検出型テスト（ピンポイントテスト、負荷テストなど）のバランスを返る



© NISHI, Yasuharu

ダブル・Vモデル



© NISHI, Yasuharu

テスト容易性設計

- テスト容易性を確保して設計する
  - テストに5割以上の工数がかかっていることを忘れない
    - » 分析:設計:実装:テストが2:2:1:5の場合、設計を5割増しにしてテストを半減できれば、トータルで1.5の工数削減になる
- 組み合わせのボトルネックを分析してテスト工数を抑える
  - テスト工数の爆発やTestabilityの低下につながる設計を防止する
    - » 製品設計時にモジュール性を考慮した方がよいことは皆知っている
    - » でもモジュール性を減らした時にどうなるかを定量的に把握したことはない
    - » 要するに、すっきりキレイに作ればテスト工数が激減し、打ち切り時のリスクも減る
    - » 製品設計で工夫して工数がほんの少し増えても、テストで激減すればトータルで得
- テスト項目数以外のテスト容易性も考慮する
  - » 操作性／観測性／制御性／分解性／単純性／安定性／理解容易性

© NISHI, Yasuharu

Testability (テスト容易性) を考慮して設計を行う

- **Testabilityを考慮した設計を行う／レビューをする**
  - Testability: テスト容易性 (DFT: Design For Test)
    - » 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

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

© NISHI, Yasuharu

講演の流れ

- 最近のソフトウェアテストの動向
- テスト今昔物語
- テストツールのよい使い方
- テストの基本の「キ」
- テストプロセス
- テスト改善
- ブラックボックスによる組み合わせテスト
- リスクベースドテスト

© NISHI, Yasuharu

テスト改善：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

© NISHI, Yasuharu

テスト改善：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

© NISHI, Yasuharu

テスト改善：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

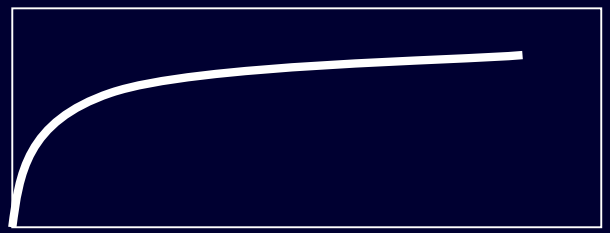
注意点

- 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

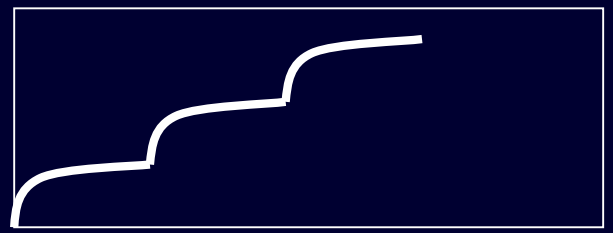
© NISHI, Yasuharu

信頼度成長曲線 (SRGM) によるテスト改善

- **バグの分布と均一なテストを仮定して、残存バグ数を統計的に推定する方法**
  - ゴンペルツ曲線やロジスティック曲線を当てはめる
    - 「ソフトウェア信頼性モデル－基礎と応用」, 山田茂著, 日科技連出版
    - フリーや商用のツールがある
  - 統計的に信頼できるほどプロセスが定まっておらず過去のデータもないのであれば、残存バグ数推定などは難しい
  - そもそも適切にテストが設計できてないので大きな抜けなどがある
  - 中身をきちんと考慮しないと統計値で出荷判断してはいけない



こうなると言われているが



こうなることの方が多い

バグの分布のムラとテストのムラをきちんと考慮しないといけない

© NISHI, Yasuharu

信頼度成長曲線によるテスト改善

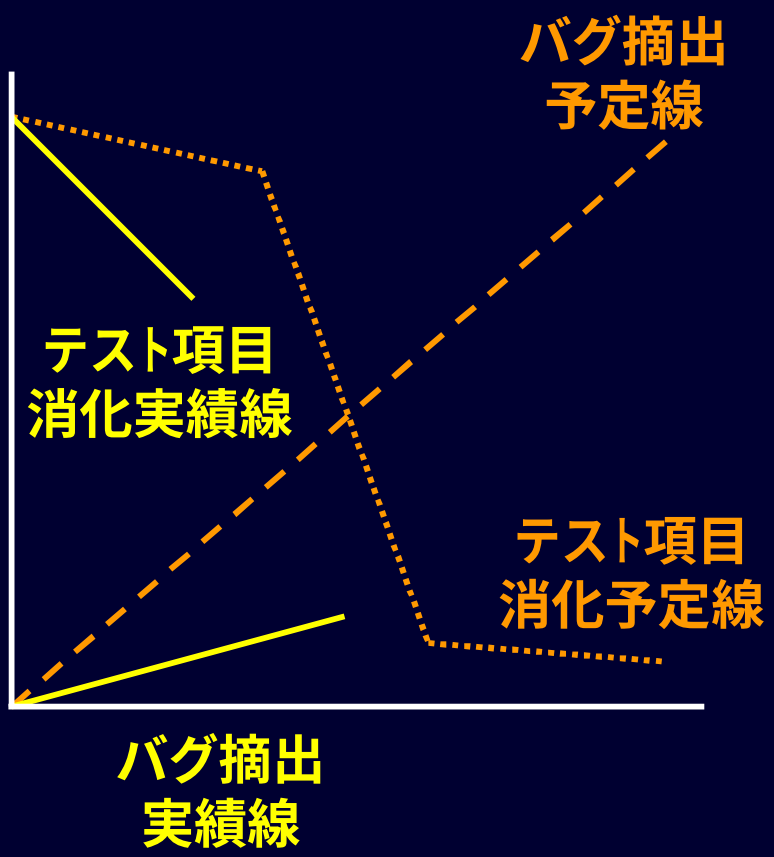
- **最低限何を考えるべきか：統計値が全てではない**
  - どんなテストをどの程度設計したか
    - » 要求カバレッジ、仕様カバレッジ、設計カバレッジ、実装カバレッジ
    - » テスト設計技法の持つ特徴など
  - どんなテストをどの程度実施したか
    - » 実施カバレッジ、テスト密度、テスト粒度
    - » テストサイクル数、テスト件数など
  - どんなバグがどの程度見つかったか
    - » バグの致命度、バグの数、発生頻度、発生条件
    - » 発生場所の偏り、類似のバグの数、原因特定時間など
  - どんな修正をどの程度直したか
    - » 修正にかかる時間、修正にかかる時間のばらつき
    - » 類似のバグをどの程度修正したか、デグレードの程度
    - » 未修正バグの数、未修正バグの重要度など
- **信頼度成長曲線を運用する前に管理図でプロセスを定量化してみるとよいだろう**
  - ただし出荷基準として用いるのではなく、パターン認識で「悪さ」の兆候を見抜くツールとするのがよいだろう

© NISHI, Yasuharu

バグ管理図によるテスト改善

- テスト項目の消化とバグの検出数のバランスを折れ線グラフで管理する手法
  - テストの改善が進むと実現できる
  - 右のグラフは以下を示唆している
    - » テストの質が悪い
    - » テストが偏っている
    - » テストが粗すぎる
    - » 品質がよい？
  - 定量的推定よりは形による パターン認識の方が効果的

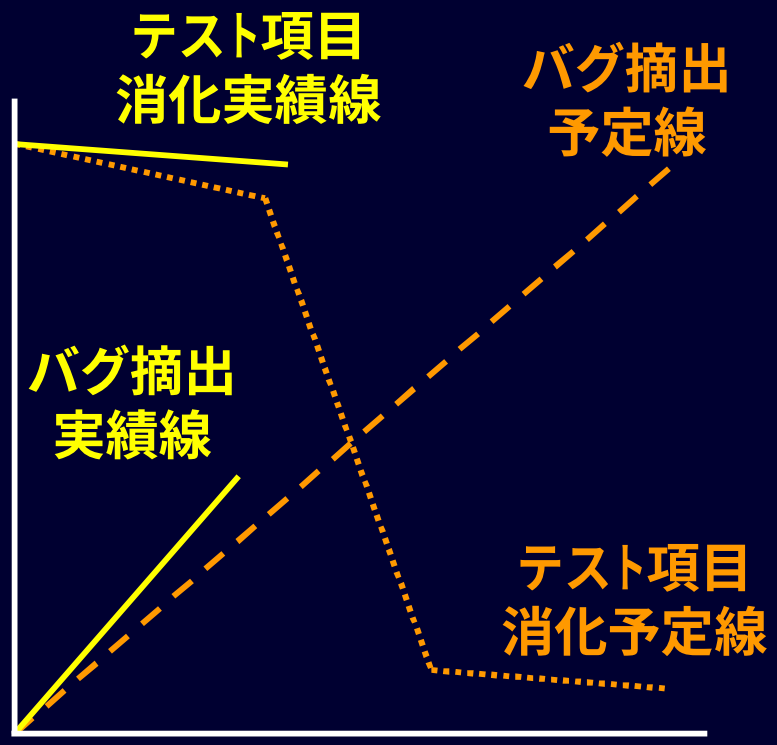
出展：「ソフトウェア品質保証の考え方と実際」
保田勝通著／日科技連出版社



© NISHI, Yasuharu

バグ管理図によるテスト改善

- テスト項目の消化とバグの検出数のバランスを折れ線グラフで管理する手法
  - 右のグラフは以下を示唆している
    - » 作り込んだバグの量が多くテストが進まない
    - » デグレードが多発している
    - » テストの質がよい？



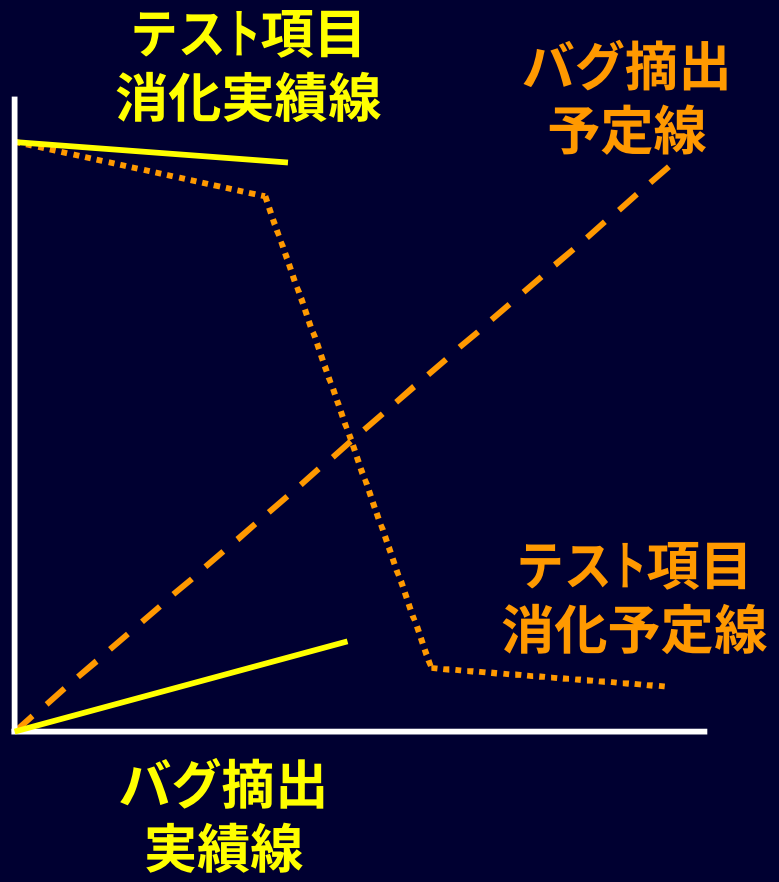
出展：「ソフトウェア品質保証の考え方と実際」
保田勝通著／日科技連出版社

© NISHI, Yasuharu

バグ管理図によるテスト改善

- テスト項目の消化とバグの検出数のバランスを折れ線グラフで管理する手法
  - 右のグラフは以下を示唆している
    - » テストの要員不足や環境未整備によりテストが進まない
    - » バグ修正が迅速に行われないのでテスト要員が待ち状態になっている
    - » 仕様が未整備のため結果判定に手間を取られテストが進まない

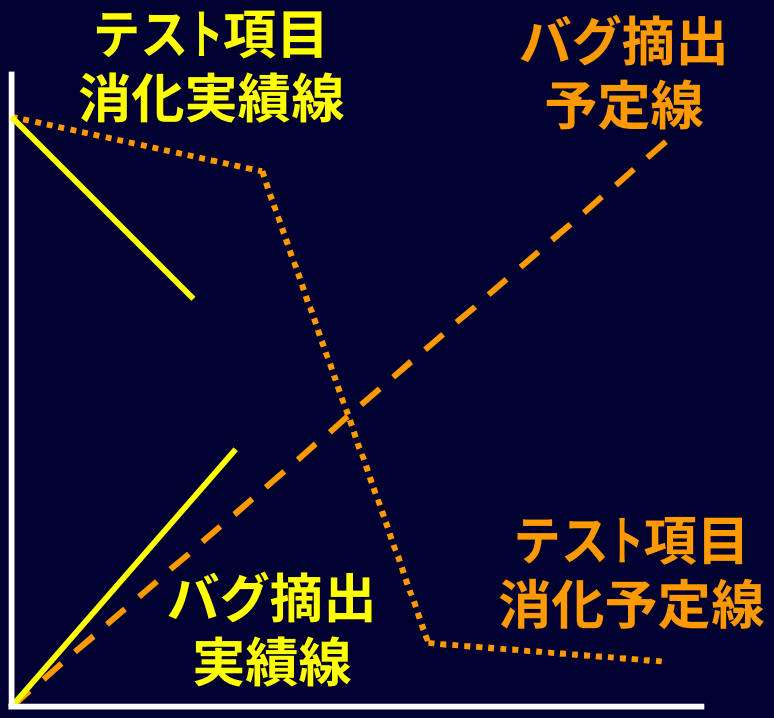
出展:「ソフトウェア品質保証の考え方と実際」
保田勝通著／日科技連出版社



© NISHI, Yasuharu

バグ管理図によるテスト改善

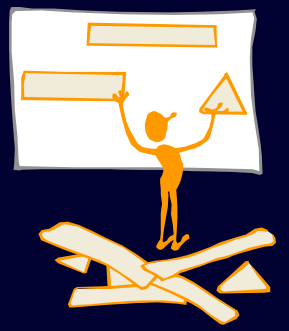
- テスト項目の消化とバグの検出数のバランスを折れ線グラフで管理する手法
  - 右のグラフは以下を示唆している
    - » 同じようなバグを何度も摘出している
    - » 理想的なパターン？



出展：「ソフトウェア品質保証の考え方と実際」
保田勝通著／日科技連出版社

© NISHI, Yasuharu

テスト改善の技法の例

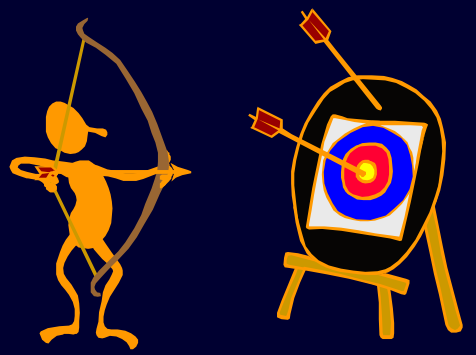


- ディレイ分析
  - 見つかった（見逃した）バグが、
    本来はどのレビューやテストのフェーズで見つかるべきだったか、
    を分析することで問題点を把握する
  - 本来見つかるはずのレビューやテスト工程を改善する
  - 問題点のあるレビューやテスト工程で見逃しただろうバグを狙って
    テストを設計する
- カバレッジ分析
  - 見逃したバグが、どのカバレッジを網羅しなかったことによるものか、
    を分析することで問題点を把握し、改善する
    - » カバレッジは制御パステストだけの概念ではない
    - » 実施カバレッジの問題か、設計カバレッジの問題か
    - » 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、
      テスト設計の抜けか
    - » 工数がかかりすぎるための間引きの失敗か
    - » 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

© NISHI, Yasuharu

テスト改善の技法の例

- **不具合モード分析**
  - バグは偏在する
    - » 似たようなバグが何度も検出される
  - 過去の不具合のパターンを蓄積しておき、そこを狙ってテストする
    - » ヒット率の改善
    - » 組み合わせテストの補完
    - » 回帰テストの改善
    - » 開発上流の改善
  - パターンの蓄積の観点がキモとなる
    - » キーワードで整理する
    - » 機能ツリーで整理する
    - » 構造パターンで整理する
    - » バグを作り込みやすい特徴で整理する



© NISHI, Yasuharu

講演の流れ

- 最近のソフトウェアテストの動向
- テスト今昔物語
- テストツールのよい使い方
- テストの基本の「キ」
- テストプロセス
- テスト改善
- ブラックボックスによる組み合わせテスト
- リスクベースドテスト

© NISHI, Yasuharu

ブラックボックスによる組み合わせテスト

- **例題**
  - レストランのシェフが、コース料理を考案中している
  - 各料理には自信がありますが、相性の悪い組み合わせ（バグ）を無くすために味見をしたいと考えている
  - コストを抑えるため、味見するコースの組み合わせの数（テスト項目数）は最小にしたい
- **料理の種類**
  - 前菜
    - » いんげん豆のスープ(豆/1)、小海老のテリーヌ(海老/2)
  - メイン
    - » カジキマグロの香草焼き(魚/1)、牛ほほ肉の煮込み(肉/2)
  - ワイン
    - » シャブリ（白/1）、ボルドー（赤/2）
- **相性の悪い組み合わせを検出するテスト**
  - 魚料理と赤ワイン、肉料理と白ワインを組み合わせてはいけない（バグ）
  - 相性の悪い組み合わせを味見できればOK、味見できなければNG

© NISHI, Yasuharu

全組み合わせ網羅テスト

- **全ての組み合わせを考えると、2x2x2=8通りになる**
  - バグは必ず見つかるが、テスト項目数は爆発的に増えていく
  - パステスト風に言うと、C∞ (全パス網羅) になる

	前菜	メイン	ワイン
1	豆	魚	白
2	豆	魚	赤
3	豆	肉	白
4	豆	肉	赤
5	海老	魚	白
6	海老	魚	赤
7	海老	肉	白
8	海老	肉	赤

© NISHI, Yasuharu

単一網羅テスト

- **個々の料理を単一で網羅するだけでは、**
  相性の悪い組み合わせは発見できるとは限らない
  - パステスト風に表現すると、C0（命令網羅）になる

	前菜	メイン	ワイン
1	豆	魚	白
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
8	海老	肉	赤

© NISHI, Yasuharu

直交配列表

- **直交表を使うと、「2つの料理に関する」相性の悪い組み合わせを少ない工数で網羅的に検出できる**
  - 3因子では4項目（／8項目）、7因子では8項目（／128項目）
  - 3つ（以上の）相性に関しては抜けが生じる
    - » テストしたい組み合わせが分かっていれば割り付けられる可能性が高い

	因子1	因子2	因子3
①	1	1	1
②	1	2	2
③	2	1	2
④	2	2	1

L_4直交配列表

	因子1	因子2	因子3	因子4	因子5	因子6	因子7
①	1	1	1	1	1	1	1
②	2	2	1	2	2	1	1
③	1	2	2	1	2	2	1
④	2	1	2	2	1	2	1
⑤	1	1	1	2	2	2	2
⑥	2	2	1	1	1	2	2
⑦	1	2	2	2	1	1	2
⑧	2	1	2	1	2	1	2

L_8直交配列表

© NISHI, Yasuharu

直交表によるテスト

- **直交表を使うと、「2つの料理に関する」相性の悪い組み合わせを少ない工数で網羅的に検出できる**
  - 3つ(以上の)相性に関しては抜けが生じる
    - » 豆・魚・赤など

	因子1	因子2	因子3
①	1	1	1
②	1	2	2
③	2	1	2
④	2	2	1

L_4直交配列表

	前菜	メイン	ワイン	
1	豆	魚	白	①
-	-	-	-	
-	-	-	-	
4	豆	肉	赤	②
-	-	-	-	
6	海老	魚	赤	④
7	海老	肉	白	②
-	-	-	-	

© NISHI, Yasuharu

直交表と禁則

- 魚料理には海老のテリーヌが付いてくるので、そのコースは元々出さないような場合は？
  - テストケース No.6 がテストできない

	前菜	メイン	ワイン
1	豆	魚	白
-	-	-	-
-	-	-	-
4	豆	肉	赤
-	-	-	-
~~6~~	海老	魚	赤
7	海老	肉	白
-	-	-	-

© NISHI, Yasuharu

禁則項目の補完

- 禁則以外の組み合わせを埋めるように
  テストケースを補完する

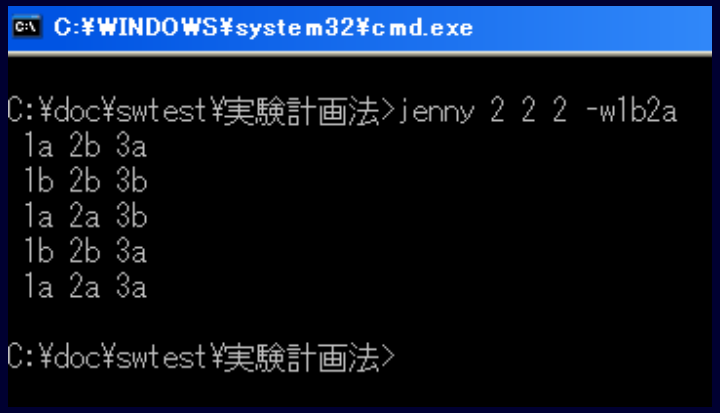
	前菜	メイン	ワイン
1	豆	魚	白
-	-	-	-
-	-	-	-
4	豆	肉	赤
-	-	-	-
~~6~~	~~海老~~	~~魚~~	~~赤~~
6a=8	海老	肉	赤
6b=2	豆	魚	赤
7	海老	肉	白
-	-	-	-

	前菜	メイン	ワイン
1	豆	魚	白
2	豆	魚	赤
-	-	-	-
4	豆	肉	赤
-	-	-	-
-	-	-	-
7	海老	肉	白
8	海老	肉	赤

© NISHI, Yasuharu

jennyを使って禁則付き組み合わせを生成する

- jenny: **組み合わせテスト項目を自動生成するツール**
  - http://burtleburtle.net/bob/math/jenny.html
  - **禁則を考慮した組み合わせ項目を自動生成する**
  - **最適もしくは最適に近いテスト項目を生成する** (All pair**法？**)



	前菜	メイン	ワイン	
1	豆	魚	白	1a 2a 3a
2	豆	魚	赤	1a 2a 3b
3	豆	肉	白	1a 2b 3a
-	-	-	-	
-	-	-	-	
-	-	-	-	
7	海老	肉	白	1b 2b 3a
8	海老	肉	赤	1b 2b 3b

© NISHI, Yasuharu

水準が多い場合

- デザートが4種類ある場合はどうしたらよい？
  - デザートは4種類、飲み物が2種類あるとする
    - » クレープシュゼット／スフレ／ブランマンジェ／ソルベ、コーヒー／紅茶
  - 多水準は、2水準の複数の因子の組み合わせで表現する
  - 多水準の直交表もある

	因子1	因子2	因子3	因子4	因子5	因子6	因子7
①	1	1	1	1	1	1	1
②	2	2	1	2	2	1	1
③	1	2	2	1	2	2	1
④	2	1	2	2	1	2	1
⑤	1	1	1	2	2	2	2
⑥	2	2	1	1	1	2	2
⑦	1	2	2	2	1	1	2
⑧	2	1	2	1	2	1	2

	因子IV	-	因子 I	因子 II	因子III	因子 V
①	1 (1x1)	/	1	1	1	1
②	4 (2x2)	/	2	2	1	1
③	2 (1x2)	/	1	2	2	1
④	3 (2x1)	/	2	1	2	1
⑤	1 (1x1)	/	2	2	2	2
⑥	4 (2x2)	/	1	1	2	2
⑦	2 (1x2)	/	2	1	1	2
⑧	3 (2x1)	/	1	2	1	2

© NISHI, Yasuharu

ブラックボックスによる組み合わせテストのまとめ

- **全ての組み合わせを網羅すると、組み合わせバグは必ず検出できるが、テスト項目数が爆発する**
  - 単一で網羅すると、テスト項目数は少なく済むが、組み合わせバグを検出できるとは限らない
- **直交表を使うと、2因子に関する組み合わせバグを少ないテスト項目数で必ず検出できる**
  - 3因子以上の組み合わせバグについては検出できるとは限らない
    - » テストしたい組み合わせが分かっていれば割り付けられる可能性が高い
  - 複数の水準は、複数の因子を割り付けるか、多水準直交表を用いる
  - 禁則については補完を行うか、All pair法などを用いる
- **あくまでブラックボックスの保証型テストに有効**
  - グレーボックスで内部構造を考慮した方が効率的になる場合もある

© NISHI, Yasuharu

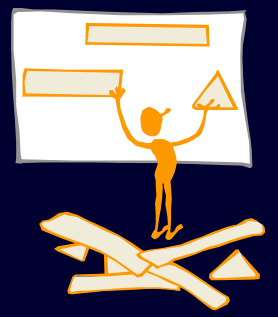
講演の流れ

- 最近のソフトウェアテストの動向
- テスト今昔物語
- テストツールのよい使い方
- テストの基本の「キ」
- テストプロセス
- テスト改善
- ブラックボックスによる組み合わせテスト
- リスクベースドテスト

© NISHI, Yasuharu

リスクベースドテストの考え方

- 「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
- 「リスク」とは？
  - 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



© NISHI, Yasuharu

リスクの例（機能ベーステストの場合）

- **機能が動作した場合の価値**
  - 売りの機能か
  - 重要なユーザシナリオに含まれているか
  - リリースしなくてもよい機能か
  - リリースを遅らせてもよい機能か
- **バグが存在する確率**
  - バグの多そうな仕様の機能か
  - バグの多そうな設計の機能か
  - バグの多そうな開発環境か
  - バグの多そうな開発条件か（レガシーマイグレーションなど）
  - 新規技術はどの程度使っているか
  - 新規開発分はどの程度か
  - 流用部の品質はどの程度か

© NISHI, Yasuharu

リスクの例（機能ベーステストの場合）

- **バグが存在する確率**
  - どの程度、何回変更が行われたか
  - 以前のフェーズのテスト（単体テストなど）の結果はどうか
  - 過去のレビューの結果はどうか
  - 過去のコードインスペクションの結果はどうか
  - 過去にどのくらい修正の難しい不具合があったか
  - デグレードの回数や頻度はどの程度か
  - プロジェクトはその頃混乱していたか
  - 開発者の能力、疲労度やモチベーション、家庭環境はどうか
- **バグを検出できる確率**
  - 期待結果はちゃんと導出できるか
  - チェックポイントは明確にできるか（多すぎないか）
  - テスト項目は複雑になりすぎないか
  - テスト担当者の能力、疲労度やモチベーション、家庭環境はどうか

© NISHI, Yasuharu

リスクの例（機能ベーステストの場合）

- **バグの致命度**
  - ユーザにどう認識されるか
  - 復旧までの時間や手間
  - データや処理の保全度
  - ハードウェアに影響は無いか
  - セキュリティに影響しないか
  - 性能低下はどの程度か
  - スケーラビリティ低下はどの程度か
  - 規格･法規制･業界慣習は守れるか
  - ユーザビリティ低下はどの程度か
- **テストプロセスへの影響度**
  - ショーストッパーになりうるか
  - 順序依存関係はどの程度か

© NISHI, Yasuharu

リスクの例（機能ベーステストの場合）

- **テストプロセスへの影響度**
  - 不具合報告書の記述時間はかかるか
  - 仕様の問い合わせの時間はかかるか
  - 腕っこきがどの程度張り付かないといけないか
  - 切り分けはできるか、時間がかかるか
  - 社外リソースの契約期間に間に合うか
  - テスト設備の利用可能期間・リース期間に間に合うか
- **プロジェクト全体への影響度**
  - 出荷はどの程度遅れるか
  - 修正できない場合は誰にダメージを与えるか
  - 保守は難しくなりそうか
  - サポートは難しくなりそうか
  - 再利用できなくなりそうか

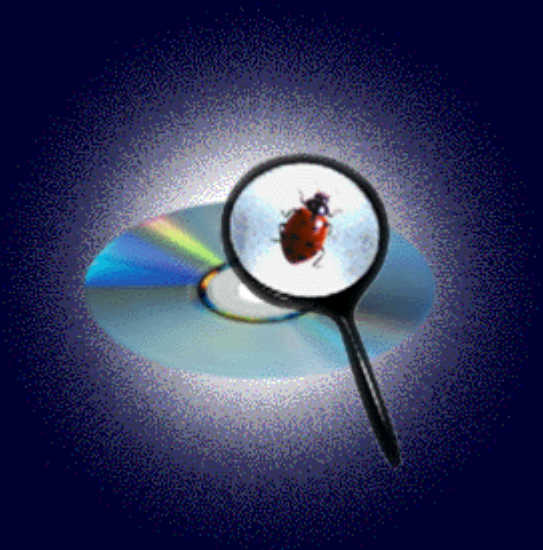
© NISHI, Yasuharu

まとめ：動向は動向、自分は自分

- 最近のソフトウェアテストの動向
- テスト今昔物語
- テストツールのよい使い方
- テストの基本の「キ」
- テストプロセス
- テスト改善
- ブラックボックスによる組み合わせテスト
- リスクベースドテスト

© NISHI, Yasuharu

ご静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

© NISHI, Yasuharu