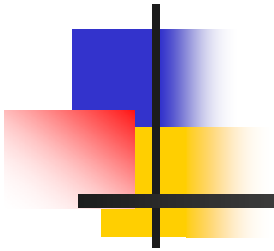
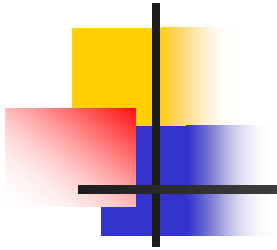
第6回システム検証セミナー
2006.9.8



ソフトウェアテストに求められている役割と課題

宮崎大学 工学部 情報システム工学科
片山 徹郎



目次

- 組込みソフトウェアについて考える
  - 品質の現状
- ソフトウェアテストとは？
  - テストの目的、テストの工程、テストの視点
  - テスト作業に付きまとう問題点
  - 代表的なテスト技法
- やるべきことをやってますか？
  - テストマトリクスの作成、不具合分析、メトリクス、テスト計画書
- 「テスト対策」してますか？
  - テスト容易性(Testability)、テスト設計の前倒し
- テスト戦略の心得
  - 「正しい手の抜き方」、リスクを考える
- テストの今後
  - モデル指向、テストの自動化、組込みシステムの場合

組込みソフトウェアについて

- 急激に大規模化が進んでいる

携帯電話	500万LOC（Lines of Code)
通信機能搭載型カーナビ	300万LOC
薄型テレビ	60万LOC
HDD内蔵DVDレコーダ	100万LOC

- 開発規模：2.4兆円
  - 情報サービス産業全体14.0兆円の17.1%
  - 組込みシステムの開発費5.9兆円の40.6%
- 開発者数：17.5万人
  - 情報サービス産業全体51万人の34.3%
  - 組込みシステムの技術者総数38万人の46.3%

(「2005年度版組込みソフトウェア産業実態調査」経済産業省 より)

組込みソフトウェア品質の現状

- 組込みソフトウェア不具合の事例(2004年のみ)

(「組み込みソフトの巨大化に立ち向かう」日経コンピュータ,2004.12.27,No.616,pp.118-123 より)

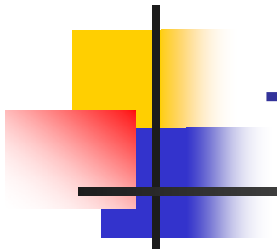
公表日	製品	不具合の内容
3月9日	携帯電話	メモリカードの画像を縮小加工後、電話着信や特定の操作で再起動する
4月28日	無線LANカード	無線LANカードを対応のADSLモデムに取り付けてもアクセスポイントとして使えない
5月13日	HDDビデオレコーダ	250Gバイト容量のハードディスクの112Gバイト以降が録画/再生できない
6月8日	HDDビデオレコーダ	ハードディスクに録画した番組を分割して編集する際に操作できなくなる
8月16日	DVDレコーダ	電源がオン状態の時に録画予約メールを受信できない
8月17日	携帯電話	メールの題名を入力後、すばやく本文入力に移ろうとすると操作できなくなる
9月13日	携帯電話	定額制サービスの接続先として指定するサーバの設定を誤り、料金を誤請求した
10月1日	携帯電話	内蔵カメラで撮影した映像が青みがかるため、出荷日に発売を延期した
10月4日	デジタルカメラ	連続撮影中やオートパワーオフ状態でレンズを着脱後、本体が操作不能になる
10月21日	携帯電話	電話を発信するとほぼ同時に着信があった場合、電話帳の内容が消失する

組込みソフトウェア品質の現状

- 組込みソフトウェアの品質向上は急務
  - 製品の不具合の大きな部分をソフトウェアが占めている
  - ソフトウェア開発において品質の課題が最も重要視されている

- 製品出荷後に生じた設計品質問題の主な原因の割合
  1. ソフトウェアの不具合　34.2%
  2. ハードウェアの不具合　23.3%
  3. 製品仕様の不具合　22.2%
  4. その他　20.4%
- 組込みソフトウェア開発に課せられている課題
  1. 設計品質の向上
  2. 開発効率の向上
  3. 開発期間の短縮
  4. 従業員の能力向上

(「2005年度版組込みソフトウェア産業実態調査」経済産業省 より)



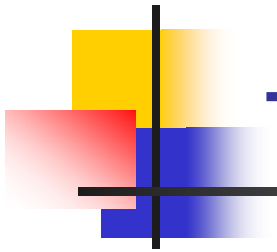
テストの目的

＜突然ですが問題です＞

2002年6月に、アメリカのNIST(国立標準技術研究所)が下記のレポートを出しました。

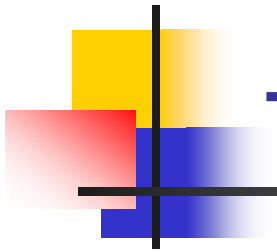
「ソフトウェアのバグが、アメリカに年間？？？円の損害を与えている。」

- テストをするのは、何のため？
  - プログラムの誤りを見つけること
  - 誤りを見つけないまでも、プログラムの正しさに対する確信を増すこと



テストの目的

- 先人たちのお言葉
  - 「テストでプログラム中のバグの存在は示せても、バグが存在しないということは示しえない。」by E.W. Dijkstra
  - 「バグ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
  - 「テストとは、プログラムを、既知の環境下で選ばれた入力により実行した結果に基づいて、その動作特性を推論する過程である。」by J.B. Goodenough
  - 「テストとは、サンプルデータの集合でプログラムを実行することにより、プログラムの動作を調べること。」by W.R. Adrion
  - 「テストとは、エラーを明らかにするために、プログラムコードを管理しながら実行すること。」by M.S. Deutch



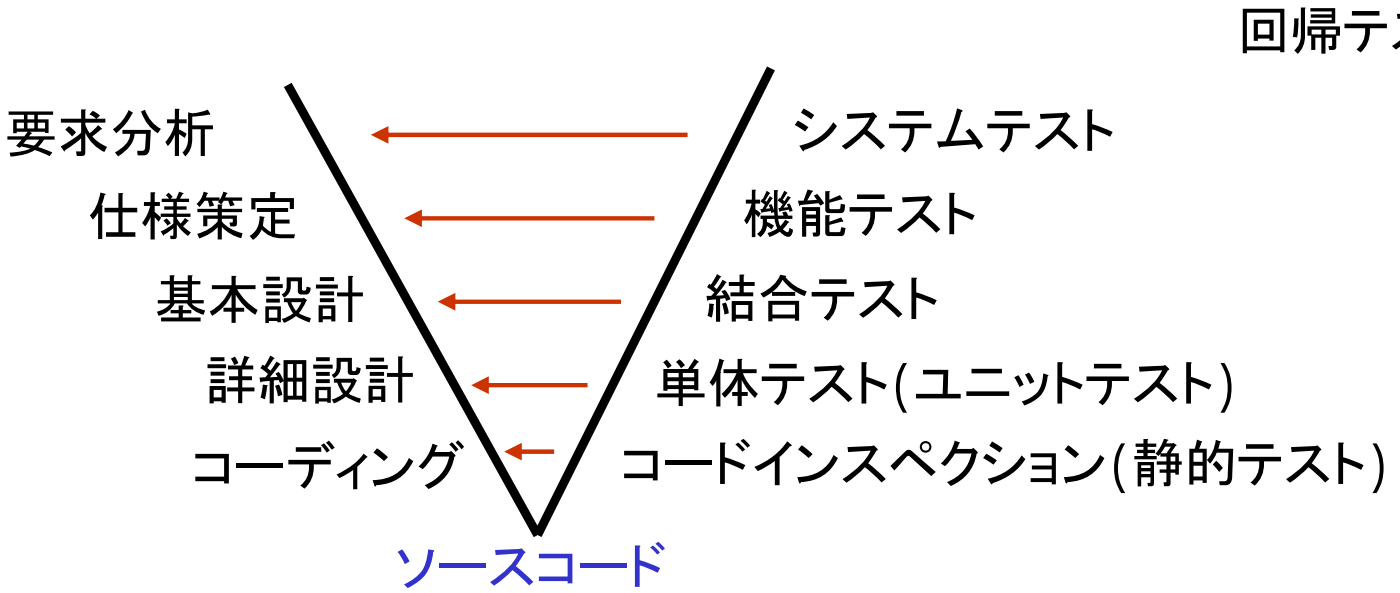
テストの目的

- 先人たちのお言葉
  - 「エラーは見つからないだろうという仮定のもとに、テストの計画を立ててはいけない。」by G.J. Myers
  - 「プログラム開発グループは、自分たちのプログラムをテストしてはいけない。」by G.J. Myers
  - 「プログラムのある部分でエラーが存在している確率は、すでにその部分で見つかったエラーの数に比例する。」by G.J. Myers

  - 「ソフトウェアテスティングとは、期待されるビヘイビアを求めて、通常は無限に大きいと考えられる実行ドメインから適正に選定された有限のテストケース集合を用いて、プログラムビヘイビアの動的検証を行なうことである。」

    by IEEE SWEBOK2004(Guide to the Software Engineering Body Of Knowledge 2004 Version)

テストの工程(V字モデル)



- コードインスペクション
  - バグになりやすいコーディングをしていないかチェックするコード監査や、複雑すぎないかを測定するメトリクス分析などを行なう。
- 単体テスト
  - モジュール内のコードが正しいロジックかどうかをチェックし、予期しない例外についても確認する。

テストの工程(V字モデル)

- 結合テスト
  - 関数の呼び出しなど、モジュール間のつながりが正しいかをチェックする。
- 機能テスト
  - ソフトウェアに実装された機能が仕様書と合っているかどうかをチェックする。
- システムテスト
  - ユーザが満足しているかどうかをチェックする。
- 回帰テスト
  - 修正確認用のテストを実施する。

ソフトウェア開発におけるテストにかかる費用

- 各工程での費用

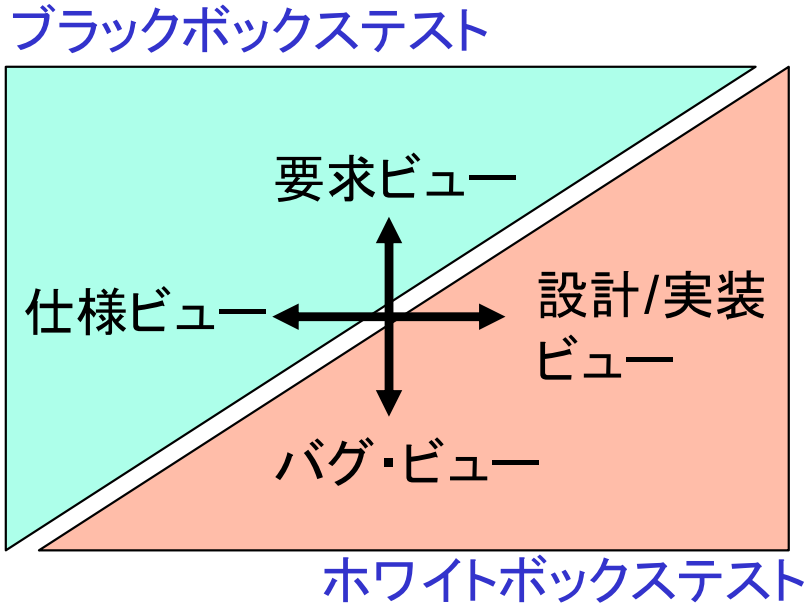
	コストの割合	「運用と保守」を除いた割合
要求分析	3%	9%
仕様書	3%	9%
設計	5%	15%
コーディング	7%	21%
テスト	15%	46%
運用と保守	67%	—

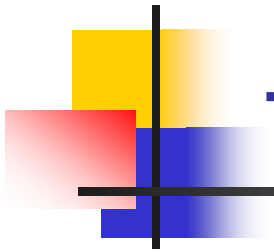
(「基本から学ぶソフトウェアテスト」Cem Kaner, Jack Falk, Hung Quoc Nguyen著,テスト技術者交流会訳, 日経BP社, ISBN4-8222-8113-2より)

テスト作業は開発におけるボトルネック！

テストの視点

- 要求ビュー
  - ユーザの要求が満足しているかどうかをチェック
- 仕様ビュー
  - 仕様どおり実装しているかどうかをチェック
- 設計/実装ビュー
  - ロジックが正しいか、実行されないコードは無いかなどをチェック
- バグ・ビュー
  - プログラムの特徴などから推測されるバグがきちんと見つかったかどうかをチェック





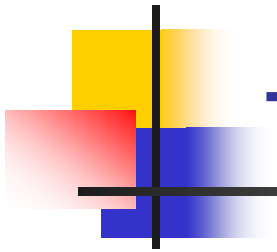
テスト作業に付きまとう問題点

- 今見つけたバグがソフトウェアに潜む最後のバグかどうか判断できない。
  - テストにかかるコストの問題
  - テスト作業をどこで終わらせるかという問題
- テストによって、

「このプログラムにはまったくバグが存在しない」
＝「このプログラムは正しい」

ことを示すことはできない。

  - ソフトウェアに対して、考えられるすべての入力をテストで実行させようとしても、テストに要する時間が膨大になる！
    - 可能な入力の組み合わせが多すぎる
    - プログラムが取りうる実行パスの数が多すぎる
    - ユーザインタフェース（およびその設計）が複雑すぎる

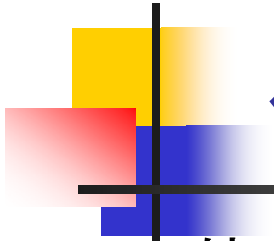


テストの設計

- テスト作業は、時間とお金の都合を考えて、結局どこかで妥協するしかない。

  ≠「納期までの時間を優先し、残った時間でテストする」

- そこで、テストの設計が重要となる。つまり、
  - より少ないテストケースで、
    - 同値クラス
    - 全ペアテスト
  - より多くバグが見つかるようにして、
    - 境界値
  - テスト対象を漏れがないように網羅する。
    - 制御パステスト
    - カバレッジ(被覆率)
    - データフローパステスト
    - 状態遷移テスト



代表的なテスト技法

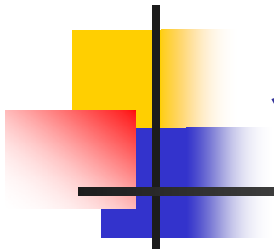
- 同値クラス
  - テストに使う入力値が、同様の結果をもたらす場合、その入力値を「同値」と呼ぶ
- 境界値テスト
  - 同値クラスの中から代表を選ぶときに「端っこ」、つま り、境界の値を選ぶ
- 制御パステスト
  - 関数やメソッドのロジックを網羅する
- データフローパステスト
  - すべて変数について、定義(definition)から参照(use)までのパスを網羅する
- 状態遷移テスト
  - イベントを起こして、プログラムの状態が正しく変化することをチェックする
- デシジョンテーブル
  - プログラムロジックを条件と動作に分け、マトリクスで表現する
- 全ペアテスト（直交表）
  - 2つの変数の組み合わせを最低限テストする

テストにおける他のキーワード

- テストの順番
  - テストをいざ実施する際には、テストの順番を決定する(優先順位をつける)必要性がある。
    - テスト作業が途中で打ち切られることが多い
      - 納期まで時間が無い！
  - 重要なテストをやり残さないようにする必要

- 実際の環境を意識
  - テストデータや環境を、本番に近い形で準備する。

- ユーザの視点に立つ
  - 使う人の立場で、どういう使い方があるのかを考える。

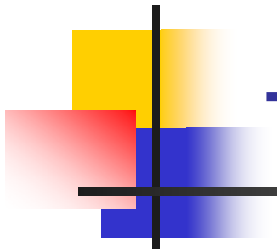


やるべきことをやってますか？

- ソフトウェア開発者は甘えている？
  - 「ソフトは、ハードとは違うんだから…」
  - 「バグ0なんて無理だよ！」

- テストにおいてやるべきこととは？
  - テストマトリクスの作成
  - 不具合分析
  - メトリクス
  - テスト計画書

「地道な作業を根気強く努力し続ける」ことが重要！

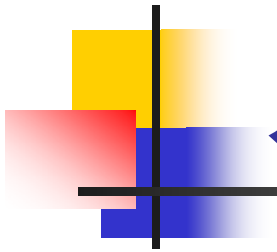


テストマトリクスの作成

- テストマトリクス
  - さまざまなマトリクス
    - テスト要件とテスト要件に対して、どのようなテストを実施するかを表にする
    - 状態とイベントを表にする（状態遷移テストにおける状態遷移図）
    - 入力値と入力フィールドを表にする
    - テストケースと仕様項目を表にする
  - まずテスト漏れを防ぐことに注視し、その後で、網羅する考え方を適用
  - テスト実施の全体像が俯瞰でき、計画が立てれる。また、進捗状況も見れる。

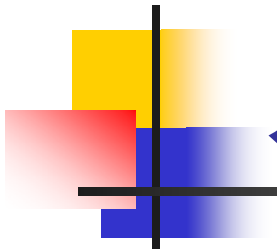
不具合分析

- 不具合分析
  - 「また出た！このバグ！」
  - 不具合を、収集し、観察し、分析する。
  - 自分たちが作りこんでしまった不具合の傾向や原因を、きちんと把握する。
  - 製造業の品質管理の原点
    - 製品の不良の山の中で一つ一つ不良の原因を掘り下げて改善を積み上げてきた成果
    - PDCA(Plan-Do-Check-Action)の継続の努力の成果
    - 発生した問題点の分析から再発防止策の実施



メトリクス

- ソフトウェアメトリクス
  - プログラムの構造や内容を測るための数値や指標
    - コードの複雑さ
    - 健全性
    - 信頼性、などなど…
  - 一般的には、技術者とそのチームの進化(改善)の助けとなることが多い。
    - 数字を使って、感覚的な品質や生産性などの項目を表現すると、より実感がわく。
  - メトリクスの限界も知っておくこと！
    - 2つ以上のものを比較して傾向を確かめる場合は役に立つが、数字だけですべての現象を判断することは危険
  - メトリクスを目標として使うことは意味があるが、目的になってはいけない！

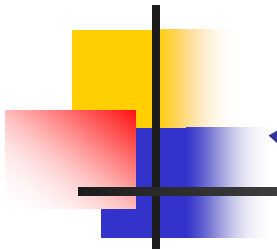


メトリクス

- Microsoftが使っているメトリクス

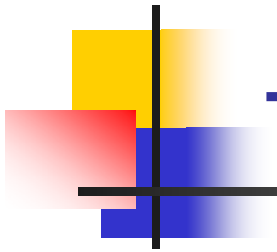
  (“Microsoft Metrics: Low Cost, Practical Metrics for Software Development,” James Tierney, Annual Oregon Workshop Software metrics, 1997 より)

  - バグのメトリクス
    - 時間軸でのバグの発見数
    - コンポーネントごとのバグの発見数 → 多すぎ？少なすぎ？
  - テストのメトリクス
    - コードカバレッジ(Coverage:網羅率)
    - テスト担当者以外のバグの発見数
    - テストケースの数、テストの自動化率
  - ソースコードのメトリクス
    - 追加、削除、変更されたコードの行数
    - KLOCs: コードの行数 → コードの行数がどのくらい増加しているか
    - コードの複雑度：McCabeのサイクロマチック数



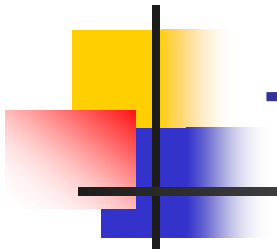
メトリクス

- Microsoftが使っているメトリクス（続き）
  - ソフトウェアの信頼性メトリクス
    - 実際の顧客が最も使うと思われるオペレーションをした際の平均故障時間(MTTF:Mean Time to Failure)
    - 信頼性成長曲線
    - ストレステストを行なった際のMTTF
  - ビルドのメトリクス
    - ビルドにかかる時間
    - ビルドで見つかった問題
    - ビルドが失敗した原因
  - スケジュールのメトリクス
    - はじめに立てたスケジュールと実際のずれ



テスト計画書

- IEEE 829のテストプランテンプレート
  - テスト計画書識別番号(Test plan identifier)
  - リファレンス(参考資料)(References)
  - はじめに(概要)(Introduction)
    - プロジェクトまたはリリースの範囲、テスト計画の範囲
  - テスト項目(Test items)
    - テスト計画の範囲内で何をテストするのか
  - ソフトウェアのリスクについての課題(Software risk issues)
    - 大きな金額を扱う機能、多くの顧客に影響を及ぼしうる機能、欠陥歴のあるモジュール、多くのまたは複雑な変更が加えられたモジュール、変更またはテストすることが困難な機能、など
  - テストの対象とする機能(Features to be tested)
  - テストの対象とはしない機能(Features not to be tested)



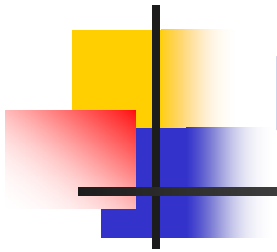
テスト計画書

- IEEE 829のテストプランテンプレート(続き)
  - アプローチ(戦略)(Approach)
    - テストの目的、リソース、テスト要件、テスト環境、テストツール、ミーティング、など
  - テスト項目の合否判定基準(Item pass/fail criteria)
    - エラーが検出されなかったテストケースの割合、欠陥の数、テストケースカバレッジ、ユーザによるテストの成功、パフォーマンス基準、など
  - 一時停止基準と再開要件(Suspension criteria and resumption requirements)
    - クリティカルパス上のタスクの未完、大量のバグ、重大なバグ、リソースの不足、など
  - テスト成果物(Test deliverables)
    - テスト計画、テスト設計仕様、テストケース、テスト手順、テストログ、テスト不具合レポート、テストデータ、ツール、など

テスト計画書

- IEEE 829のテストプランテンプレート(続き)
  - テストタスク(Remaining test tasks)
  - 実行環境条件(Environmental needs)
    - ハードウェア、ソフトウェア、施設、要員、インタフェース、文書、など
  - 責任範囲(Responsibilities)
  - 人員計画とトレーニング(Staffing and training needs)
  - スケジュール(Schedule)
  - プランニングリスクとその対策(Planning risks and contingencies)
    - 非現実な納期、予算、リソース、製品の要件の欠如、機能のなし崩し的な追加、など
  - 承認(Approvals)
  - 用語集(Glossary)

必要に応じてカスタマイズすることが重要！

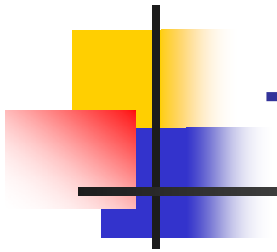


「テスト対策」してますか？

- ソフトウェア開発者はテストを軽視している？
  - 「テストは新人の仕事」
  - 「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！」

- 「テスト対策」とは？
  - テスト容易性(Testability)を考える
  - テスト設計の前倒し

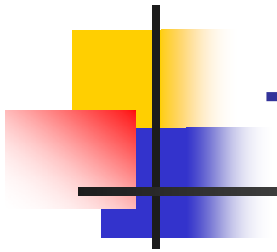
「テストありき」を前提にすることが重要！



テスト容易性を考える

- テスト容易性(Testability)
  - テストの実施し易さ
  - テスト容易化設計(DFT:Design For Testability)

- テスト容易性 (by J. Bach)
  - 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。

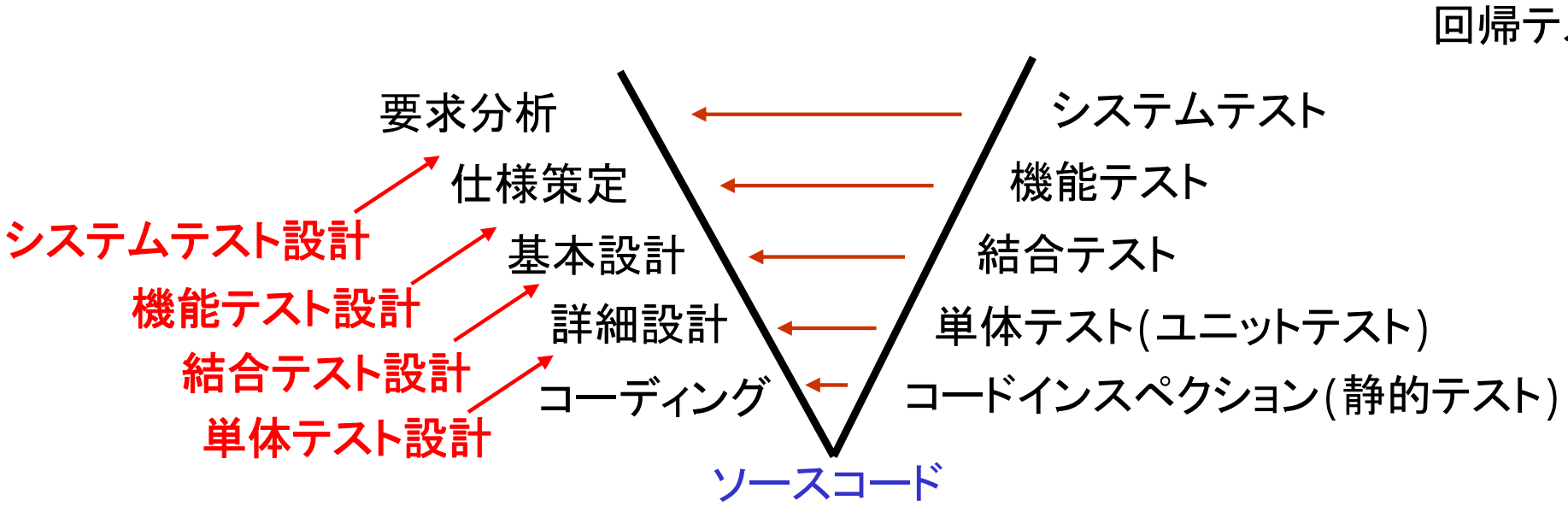


テスト容易性を考える

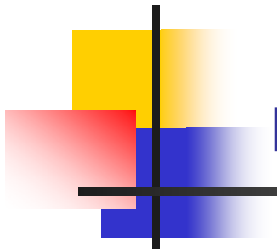
- テスト容易性 (続き)
  - 分解容易性(Decomposability)
    - テストの対象範囲を制限することにより、速やかに問題を切り分け、手際よく再テストを行える。
  - 単純性(Simplicity)
    - テストする項目が少なければ少ないほど、テストを速やかに行える。
  - 安定性(Stability)
    - 変更が少なければ少ないほどテストへの障害が少なくなる。
  - 理解容易性(Understandability)
    - より多くの情報があれば、それだけ手際よくテストができる。

テスト設計の前倒し

- テストの設計をいつやるか？
  - テストの設計とテストの実施は分けることが可能



テスト設計を、作りこみの段階で実施する
ことによって、それぞれの工程を見直すことができる



品質の作りこみ

- 「品質を上げるとコストがかかる」？
  - レビュー、ドキュメント、テスト、…
- コストがかかるのは、不具合の作成、検出、修正が原因
  - テストでバグが見つかると、デバッグ作業が必要。
  - テスト工程は開発の大部分を占めるが、デバッグ作業が多い。
- 「デバッグは手戻り作業である」という認識を！

品質の作りこみ

- 製品の品質は、作りこみの段階で確保することが基本
- テストの実施で品質は上がる？
  - 間違ってはいない。だが、同時にデバッグ作業(手戻り)が発生。
  - テストの目的は「誤りを見つけること」

- スケジュールの立て方にも問題がある

テスト作業がコストの大半を占めている現状

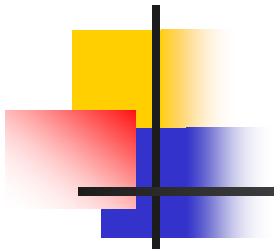
その現状に基づいて、計画を立てる

「設計後にレビューなんてやってる暇がない！」

テスト戦略の心得

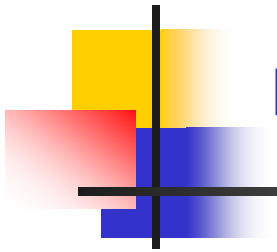
(「ソフトウェアテスト293の鉄則」Cem Kaner, James Bach, Bret Pettichord著, テスト技術者交流会訳,日経BP社, ISBN4-8222-8154-Xより)

- テストをする理由は、つまるところただ1つしかない。
  - 「重要な部分が正しく動かない恐れがあるから」
- テスト戦略に関する3つの質問を、自分たちに繰り返し問いかける必要がある。
  - 本当に必要か？
    - テストには時間やコストがかかる。明らかにすべきリスクが製品にない限り、テスト戦略に含めるべきではない。
  - 誰のためか？
    - 誰の役にも立たないことは、テスト戦略に含めるべきではない。
  - どこまでか？
    - どこまでテストする必要があるかを、テスト戦略に含めるべき。



「正しい手の抜き方」

- 「正しい手の抜き方」(by 山浦 恒央(東海大学) JaSST'06 in Tokyoにて)
  - 従来の考え方では、全テストケースを実行することが重要
    - しかし、リソースも時間も限られている状況において、膨大なテストケースをこなすことは、もはや不可能
    - また、全てのテストケースを実行したとしても、バグは埋め込まれてしまう
  - 「手を抜く」とは、このような現状を踏まえて、「バグは埋め込まれてしまうという」ことを認識した立場で、数あるテストケースの中からサンプリングしてテストを行うこと
  - ただし、この考え方には前提がある
    - 現状の品質レベルを把握すること
    - 最低限度品質の設定
    - テスト実行の際もサンプリングし、実行したテストケースのバグ数や影響度をしっかりと分析すること



リスクを考える

- バグがどうやっても発生してしまうならば、発生してしまうバグのインパクトを小さくさせることが鍵
  - どんな壊れ方をするのか。

- リスクベースドテストの考え方
  - システムの機能に対し、

    障害発生の可能性 と

    障害発生時の影響

    とを段階化することで、リスク度を算出し、リスク度の高い順に集中的にテストする。

機能安全規格

- 機能安全規格
  - IEC 61508
    - リスク（災害の頻度と規模）を評価
    - 故障時に安全性を確保、品質規格を補完
    - 製品＋マネージメント（人・組織・プロセス）規格
    - 4段階の確率または頻度で定義された安全度水準(SIL)
  - 安全度水準(SIL:Safety Integrity Level)

E/E/PE安全関連系に割り当てられる安全機能に対する目標機能失敗確率

SIL	低頻度作動要求モード運用（注1）	高頻度作動要求又は連続モード運用（注2）
4	10^{-5}以上10^{-4}未満	10^{-9}以上10^{-8}未満
3	10^{-4}以上10^{-3}未満	10^{-8}以上10^{-7}未満
2	10^{-3}以上10^{-2}未満	10^{-7}以上10^{-6}未満
1	10^{-2}以上10^{-1}未満	10^{-6}以上10^{-5}未満

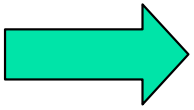
注1：作動要求あたりの設計上の機能失敗確率
注2：単位時間あたりの危険側失敗確率（1／時間）

モデル指向の考え方

- MDA (Model Driven Architecture)
  - ツール上で、モデルからモデルへの変換や、モデルからコードへの変換、モデルのチェックを行っていく。
- MDD (Model Driven Development)
  - MDAをベースにシステムを開発する手法
- 代表的なモデルとしては、UML(Unified Modeling Language)が有名

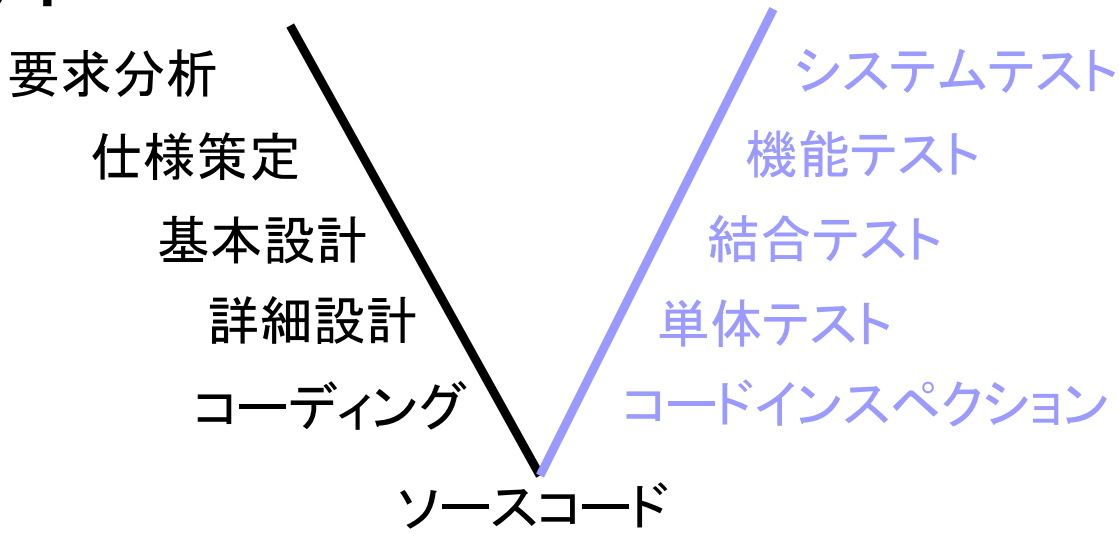
```
class Oven_c
{
   private:
        int m_SelectID;
        int m_JuicePrice;

   public:
        void setm_SelectID(int id);
        int getm_SelectID();
        void setm_JuisePrice(int price);
        int getm_JuisePrice;
};
```



モデル指向の考え方

- この考え方をつきつめると、開発のバックスラッシュモデルが完成する？

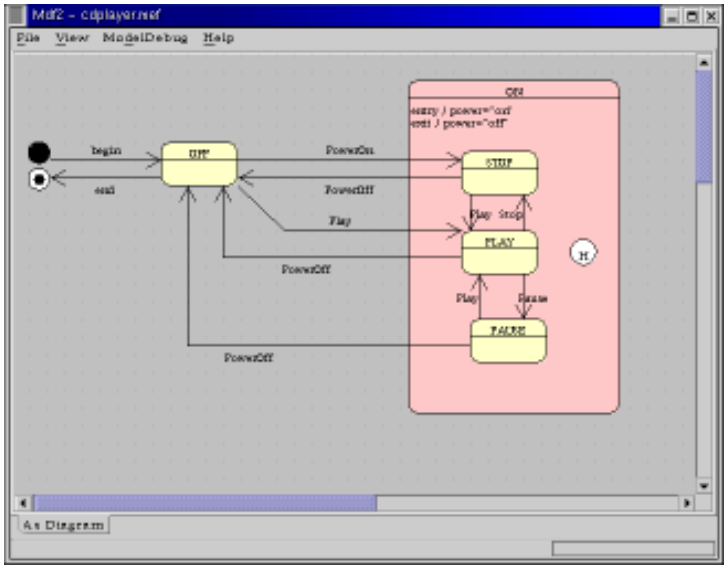


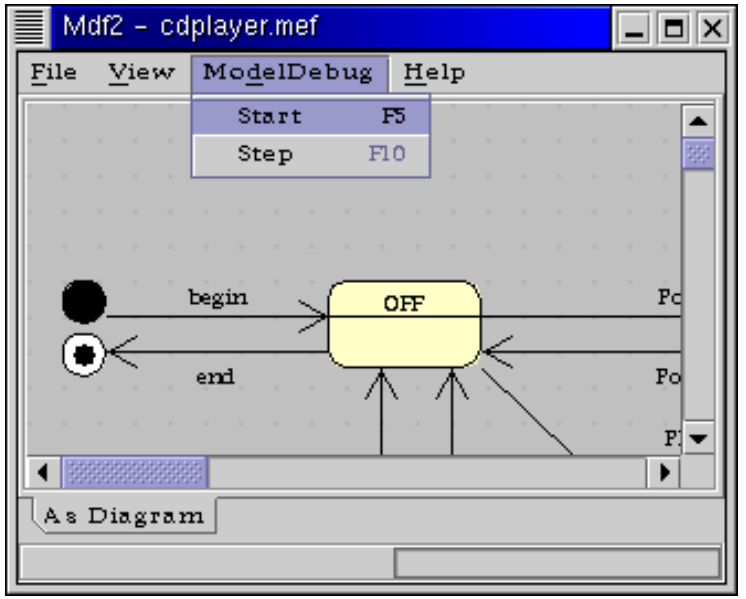
- ここまでいくと、テストの現場からは懐疑的
  - 仕様をモデルで全部表すことが可能か？
    - 実際の開発作業では、コードの約8割は正常ケースよりイレギュラーケースの記述にエネルギーを費やしている現実
    - 性能やセキュリティなどの非機能要件の取り扱い

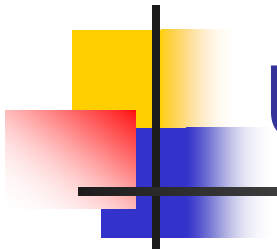
モデル指向の考え方

- モデル上で何をどこまで検査できるのかが、今後の課題
  - モデル上でできることが、まだまだたくさんありそうだという期待
  - テストに限らなければ、かなりの恩恵があるのでは？

＜下記の画面はIIOSS(Integrated Inter-exchangeable Object-modeling and Simulation System)より＞







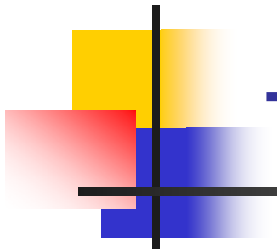
U2TP(UML2.0 Testing Profile)

- テスト仕様をモデル化するための記述言語として、UMLをプロファイル拡張したもの
- UML2.0のSuperstructure, Infrastructure仕様をベース
- U2TPの構造
  - Test Architecture
    - Test Context, Test Configuration, Test Component, SUT(System Under Test), Arbiter, Scheduler, Utility Part
  - Test Behavior
    - Test Control, Test Case, Test Invocation, Test Objective, Stimulus, Observation, Coordination, Default, Verdict, Validation Action, Log Action, Test Log
  - Test Data
    - Wildcard, Data Pool, Data Partition, Data Selector, Coding Rule
  - Time Concepts
    - Timer, Timezone

U2TP(UML2.0 Testing Profile)

- さらに、非機能要求に対して、UML表記にスケジューリングやパフォーマンスに関する要求をモデル化するための拡張
  - UML Profile for Schedulability
  - Performance and Time
  - UML Profile for Systems Engineering

- マイクロソフト社など数社は、ドメイン固有言語(DSL:Domain Specific Language)を設定するアプローチを提唱



テストの自動化

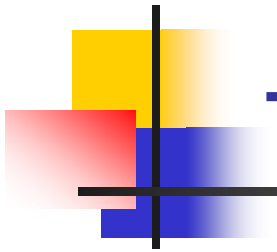
- 導入理由誘惑は、実際いくらでもある
  - テストなんて面倒くさいんだから、自動でやろう！
  - 膨大な数をこなしたい！
  - テストにかかるコストを削減したい！

- 自動テストの罠

("Systematic Software Testing," Rick D. Craig and Stefan P. Jaskiel, Artech House Publishers, 2002 より)

  - なんら戦略なく自動化をしようとしている
  - 期待が大きすぎる

    （→テストの自動化は、問題をすべて解決してくれるものではない）
  - トレーニングコースの欠如もしくはレベルの低さ
  - 誤ったツール選択

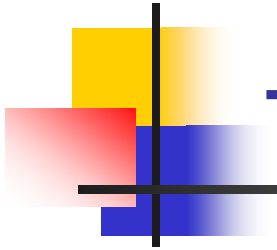


テストの自動化

- 手動でテストする必要がなくなる？
  - 向き不向きがあるので、自動テストと手動テストは別と考える方が良い
  - 自動化すべき項目と手動で行うべき項目を整理することが大事

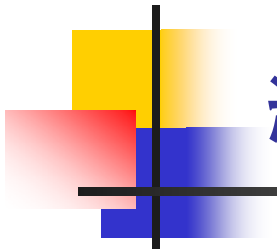
  - まずは、何をテストすべきか明確にする。
  - 自動化、および、手動で、できることとできないことを考える。
  - 自動化できそうで、かつ、手動でもできそうな箇所が議論になる！
  - 時間と費用を考える。

何をテストするのかが明確でないなら、自動化は無理！



テストの自動化

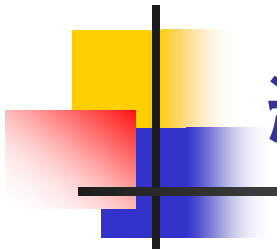
- 自動テストが向いているテストとは？
  - ストレステスト（特に過負荷）
  - タイミングのテスト
  - ビルド検証テスト（スモークテスト）
  - 夜通しテスト
  - トレース（実行履歴）
  - 回帰テスト？
  - グラフィックスやサウンド？



組込みシステム

- 今後「組込みソフトウェア」が鍵となる
  - 組込みソフトウェアの課題
    - 信頼性、生産性、応答性、資源節約、…
  - 「組込みシステムが多機能になり、大規模になって複雑になってきた。このまま同じ手法で開発を続けていくと破綻しないかな。そうならないように、何かもっといい手は無いの？」という疑問
    - ソフトウェア工学が生まれた背景そのもの
  - 「組込みシステムにはさまざまな要求があって、どれも厳しいという特色がある」という主張
    - 物事を解決する際には、問題を小さくするアプローチが一般的。

品質を大切に考える日本が、強みを発揮できる分野

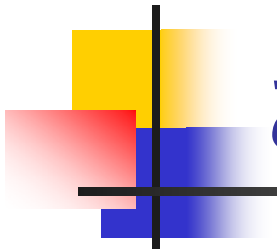


組込みシステムのテスト

- リアルタイム性、並行処理、割り込み、…
- 安全保護系のテストが必要
  - 安全保護系

    安全性を損なう恐れのある異常な状態や誤作動が生じた場合に、それを感知してシステムを自動停止させたりする装置。
    - どのような条件下でも、この安全保護系が正しく動作することを確認する必要がある。
      - 他のタスクとの競合
      - 割り込み禁止時、など

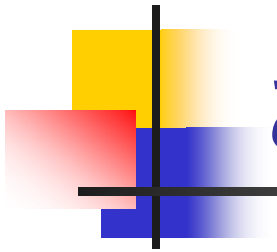
組込みシステムに特化したテスト法とは？



おわりに

- ソフトウェアテストの役割と課題について
  - ソフトウェアの品質を決定付ける「最後の砦」！
  - テストの目的は、プログラムの誤りを見つけること。
  - テスト技術に対してかつてないほどの注目が！
    - ちまたで叫ばれる「テスト不足」！
    - 最近、テストに関する書籍が次々出版されている。
    - 組込みシステムの品質保証の観点は、今後鍵に。

- テストは、ソフトウェア開発工程の一部の作業。だが、
  - 将来的には、ソフトウェア技術者全員に共通する基礎知識に！
  - テストでの考え方をマスターすれば、その知識は設計やコーディングなどの他の工程でも、きっと生きるはず！



おまけ

- テスト技術者交流会(TEF:Testing Engineer's Forum)
  - 電気通信大学の西康晴先生が世話人
  - swtest@blues.se.uec.ac.jp
  - http://www.swtest.jp/
- ソフトウェアテストシンポジウム (JaSST:Japan Symposium on Software Testing)
  - JaSST'06 in Sapporo　2006年10月17日　　小樽商科大学札幌サテライト
  - JaSST'07 in Tokyo　　2007年1月30, 31日　目黒雅叙園
  - http://www.jasst.jp/
- テスト技術者資格認定(JSTQB:Japan Software Testing Qualification Board)
  - 日時:　2006年8月28日
  - 会場:　東京, 大阪
  - http://www.swtest.jp/certification-pre.html