「バックスラッシュ工程でのOO設計とテスト実践」

小さな部品を積み重ねてものを作り上げるオブジェクト指向。テストが部品段階から磨き上げてこそ成功へとつながる。バグを元から絶つオブジェクト指向とテストのより積極的な付き合い方を紹介します。

～ 守りのテストから攻めのテストへ ～

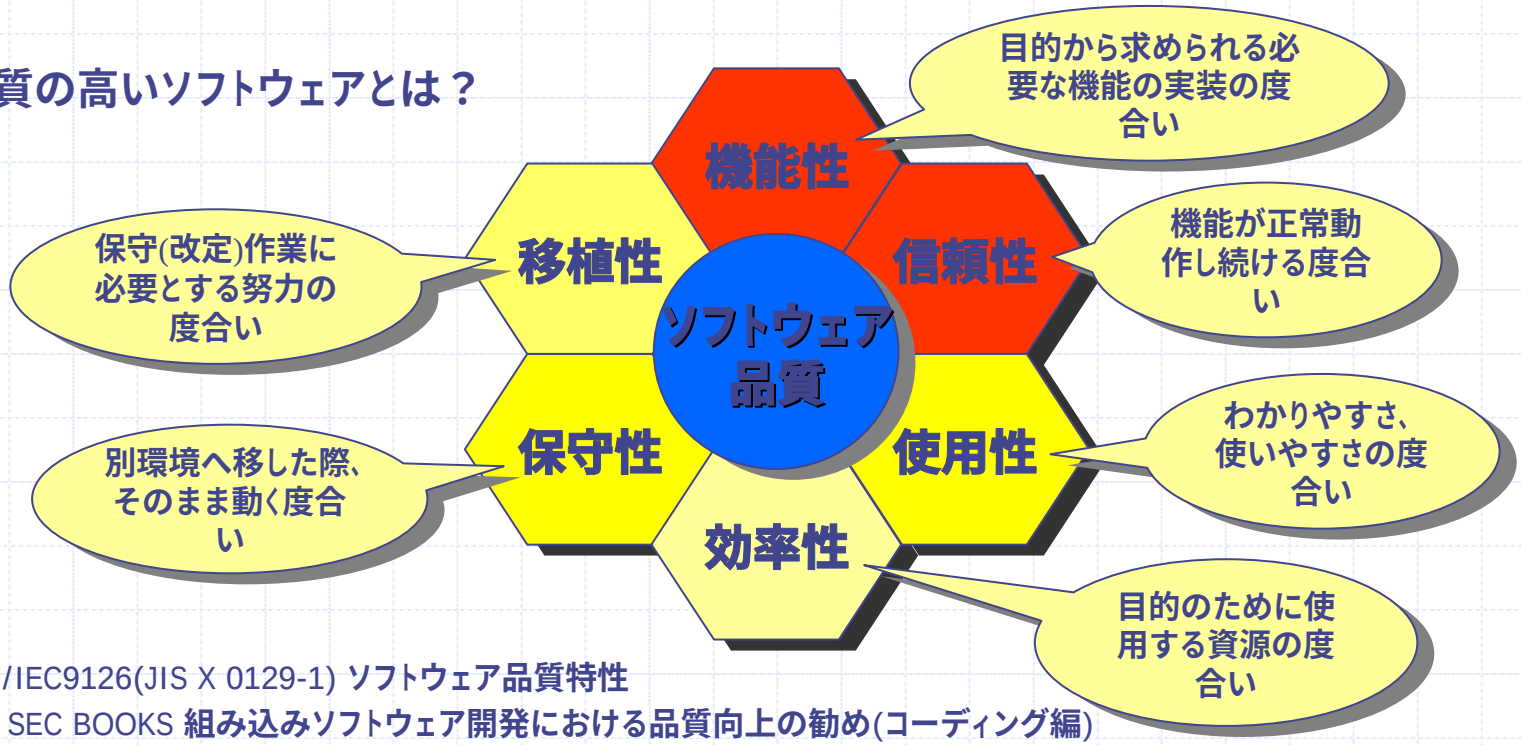
講演の趣旨

- ソフトウェア品質の危機
- ソフトウェアの大規模化･複雑化による品質管理の破綻
- 新たな手法の必要性
  - オブシェクト指向との関係
  - アジャイル的な開発との関係
- バグを捕るのではなくバグを混入させないプログラミングとテスト技法
- 継続的なバグの監視と開発の必要性

✓ よいシステムとはテストをしなくても動作が確信できるシステム。しかし、よいシステムとは、十分なテストをされているもの。

テストの目的は品質の高いソフトウェアへの到達

品質の高いソフトウェアとは？



ISO/IEC9126(JIS X 0129-1) ソフトウェア品質特性
IPA SEC BOOKS 組み込みソフトウェア開発における品質向上の勧め(コーディング編)

- ✓ **テストの第一の到達目標は「機能性」と「信頼性」**
- ◆ より高い品質に関しても広義でのテストを通して到達されるべき→上質のイメージ

「テスト」に依存する品質維持

動作確認で品質を保証しようとしている

- **全ての検査→コンディションカバレッジ**

  確認可能な値域に対して状況によらず同じ動作をする場合など。

- **抜き取り検査→ブランチカバレッジ**

  確認可能なパスに対して状況によらず同じ動作をする場合など。

↓

機能の肥大化からテストケースが膨れ上がり“爆発”している

→ 検査では品質を十分維持できない。

簡単な分枝の組み合わせでも繰り返しがあり、各シークエンスが独立でない場合の動作パターンは事実上無限になる。

→ スパゲティ化した動作

他の産業での品質維持の例

HACCPは食品管理の先進的な方法 → バックスラッシュ工程での管理の重要性

- 従来の「食品の安全性」
  製造環境全体の衛生確保
  最終製品の抜取り検査による安全性の確認
- HACCP方式
  原材料から最終製品に至るまでの一連の工程における想定される危害の分析
  重要な点での重点的な管理・記録

1. 危害分析（危害物質を特定し、その防止措置を講じる）
2. 重点管理（重要な製造工程の連続的な監視の実施）
3. 工程別管理（原材料受入から出荷までの各工程ごとの管理）
4. 記録の重視（管理記録がなければその実施が認められない）

参考 福井県健康福祉部食品安全・衛生課
http://info.pref.fukui.jp/eisei/shokunoanzen/Haccp/index4.htm

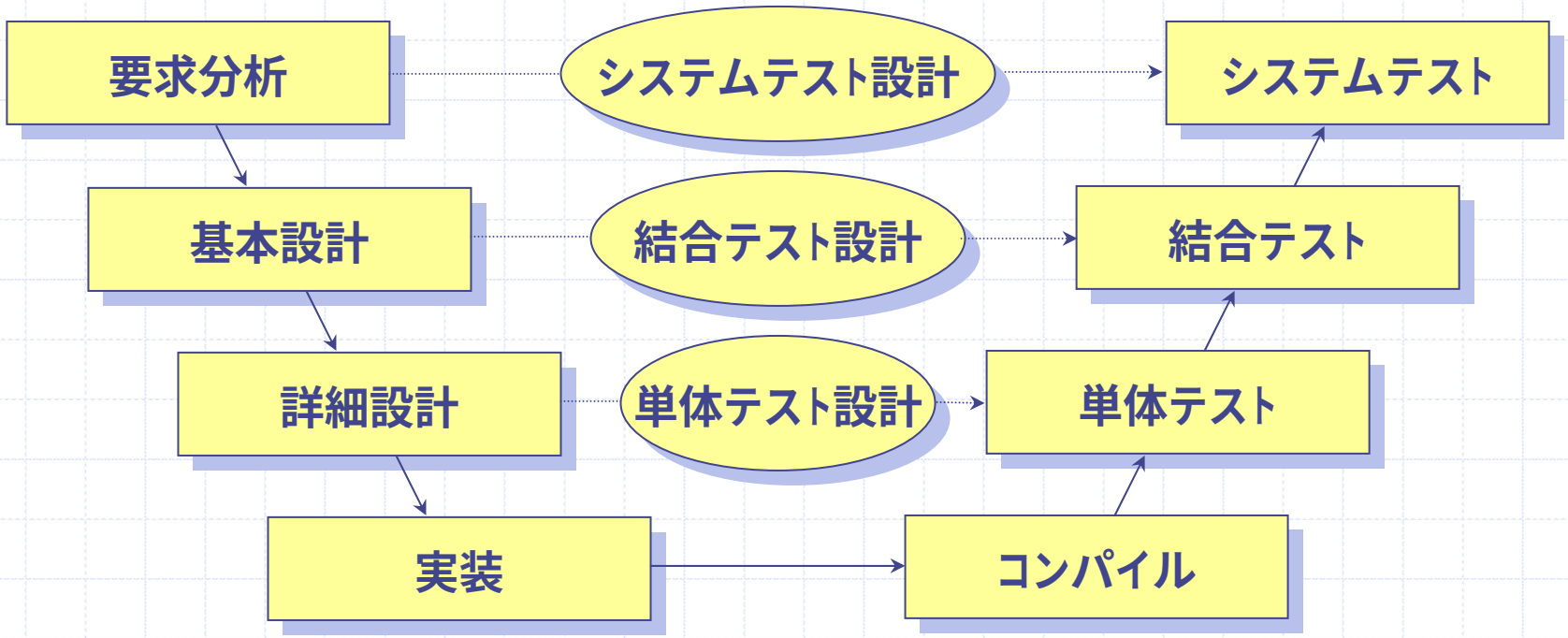
- 従来の「ソフトウェア品質維持」
  コーディングの一貫性
  全ての動作確認
- より積極的な「ソフトウェア品質維持」

1. ソフトウェアの特性の分析からリスクを分析
2. リスクの高い機能を分離する
3. 機能を分離しそれぞれの部分で生じるリスクを隔離する
4. 問題の発生をモニタリングできる仕組みの整備

✓ **バグを元から封じ込め動作確認せずとも安全性を確信できる構造が必要。**

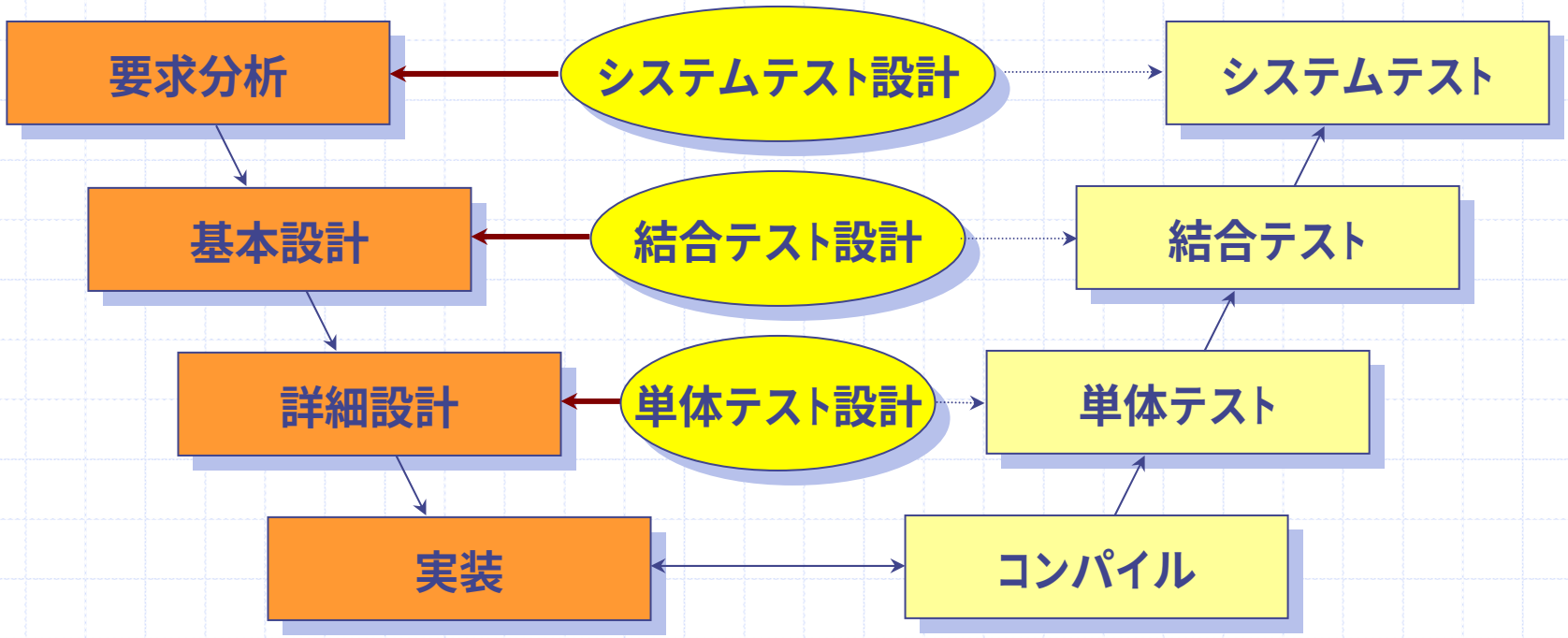
ウォーターフォールモデルでの開発における設計･実装･テストの関係

ソフトウェア開発のV字モデル



ウォーターフォールモデルでの開発における設計･実装･テストの関係

ソフトウェア開発のV字モデル



動作確認による品質の確認から証明的品質保証へ

現在のソフトウェアは機能が多くなりすぎてその組み合わせは指数的オーダーになったため実際の動作確認では品質を確認しきれなくなっている。

↓

設計段階から、動作の保証を証明できるかを意識した設計が必要。

↓

新たな開発手法が必要→オブシェクト指向、アジャイル的な開発

バグのないソフトは作れるか

■現在要求されるような高度なものであれば答えはNO

→人間はミスをする

■「バグのないプログラムは存在するか」であれば答えはYES

一般的に"Hello world"は紛れもなくバグのないプログラム

非常に単純な数十行で記述できるようなプログラムであれば、そこそこの注意力でもバグのないプログラムを書くことは可能

→ "Hello world"にバグがないのは「printfにバグがなければ」が前提条件

→ これが構造化の威力

→ さらにオブジェクト指向へ

✓ **バグの無い構造の積み重ねで全体としてバグの無いものが作られる**

参照 開発の現場 vol.006「ようこそ、組込みプログラミングの世界へ！」(添付資料)

規模の肥大によるさまざまな破綻

- プログラムそのものの生産性の低下
- テストケースの爆発的肥大
- プロジェクトに関わる人間の増大による生産性の低下
- 設計内容の肥大化

人間の対応可能なキャパシティの限界の低さが問題

→ 今に始まったことではない

→ 物事の単純化と整理が人間の文化の歴史

プログラミング言語でのオブジェクト指向とは何か

- ✓ **手続き型言語**
  全てのステートメントが他のステートメントに影響を与えうる。
  → 全ての記述に独立性が保障されない。スパゲティ化

  ↓ プログラム構造を分割して独立性を保障した部分に整理しよう

- ✓ **構造化言語**
  グローバル変数や関数そのものなどグローバルに宣言されるものがお互いに影響しあう
  → グローバル変数、関数の宣言のスパゲティ化

  ↓ ソフトの機能を分割して、より独立性の高いものに整理しよう

- ✓ **オブジェクト指向へ**

オブジェクト指向の特徴

■ 抽象化の概念
　最も基本的な概念で、大規模なプログラムをより単純に記述できるようにします。
■ カプセル化の概念
　プログラムの変更とメンテナンスを簡単にします。
■ クラス階層の概念
　プログラムを簡単に拡張できるようにします。

「Microsoft Visual C++ Version 1.0 C++ チュートリアル」より

- ✓ 抽象化は全ての階層で単純な構造を維持しコード上のバグを作らないための要
- ✓ カプセル化が大規模化への流れに対応するための要

カプセル化が問題の切り分けの要

- **カプセル化はそれぞれの問題の独立性を保障し、バグなどによる影響を小さな領域に封じ込める殻**
  - 独立であるモジュールはお互いに自由に設計･実装できる
    → 設計を容易にする
  - 独立であるモジュールはお互いに自由に修正できる
    → メンテナンスを容易にする
  - 独立な事象のテストは組み合わせによるテストを必要としない
    → テストケースが縮小する → 品質保証が現実的になる

- **カプセル化の殻の存在が状態の有効性を保証する関所になる**

  → バグの混入の防止、例えるなら防疫の仕組み

  食品管理に例えるなら要所要所での消毒に相当する

  データとして扱えないものの排除 → 例:改行文字を含む文字列を混入させないなど

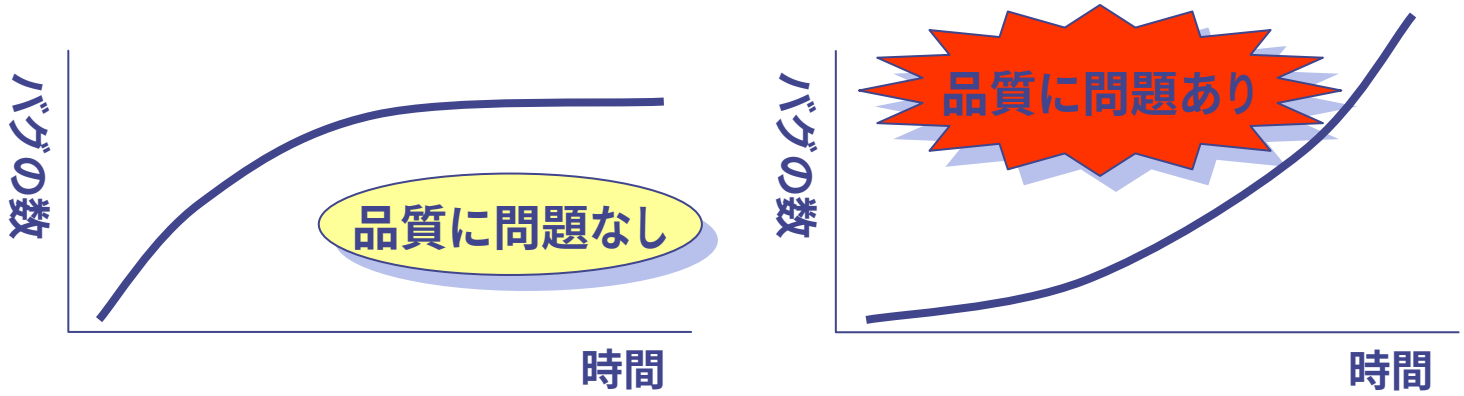
「機能性」と「信頼性」への到達

- バックスラッシュ工程での品質維持はモジュールの適切な分割とその独立性の確認が要になる
- それぞれのモジュールの十分な動作テストで品質保証の証拠を積み重ねる

いかにバグをなくすか

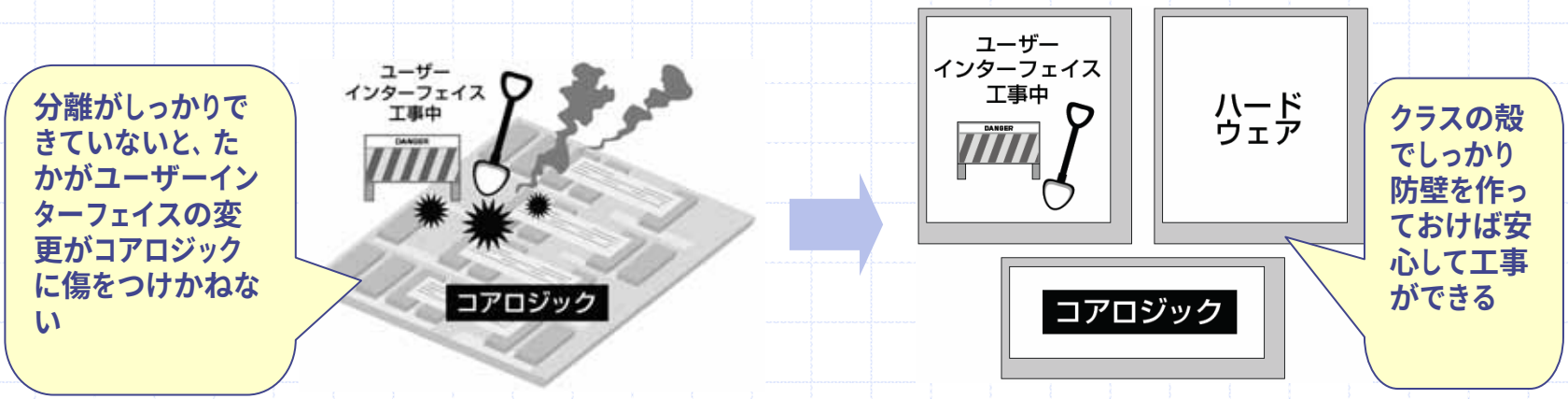
- バグを作らない
  - バグを発見する労力よりもバグを作らない労力の方が結果的には低いことが多い
    - → 構造化 → オブジェクト指向
- バグを発見する
  - 十分なテスト
    - → テストケースを増やさない工夫→構造化→オブジェクト指向
- バグの修正が容易に出来るようにする
  - → バグの修正が新たなバグを生みにくい構造→オブジェクト指向
- バグに気が付ける
  - → 例外処理・デバッグのテクニック

テスト工程で見つかるバグの数



日経BP「ステップアップのためのソフトウェアテスト実践ガイド」より

■ 発散していくのは構造化が不十分でデバッグが新たなバグを生み出している



開発の現場 vol.006「ようこそ、組込みプログラミングの世界へ！」(添付資料)

規模の肥大によるさまざまな破綻

- プログラムそのものの生産性の低下
- テストケースの爆発的肥大
- プロジェクトに関わる人間の増大による生産性の低下
- **設計内容の肥大化**

人間の対応可能なキャパシティの限界の低さが問題

→ 今に始まったことではない

→ 物事の単純化と整理が人間の文化の歴史

ウォーターフォールモデルから
アジャイル(機動)的モデルへ

ウォーターフォールモデルでは設計が肥大化して維持できなくなる

→設計の考慮不足による修正の多発→手戻りの増加

↓修正を容易にしたい

オブジェクト指向

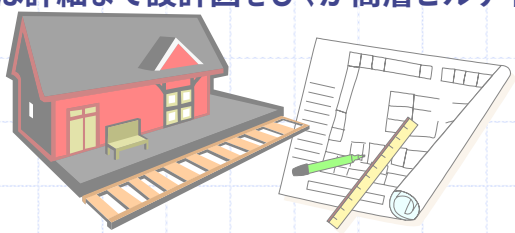
オブジェクト指向により修正・メンテナンスが容易

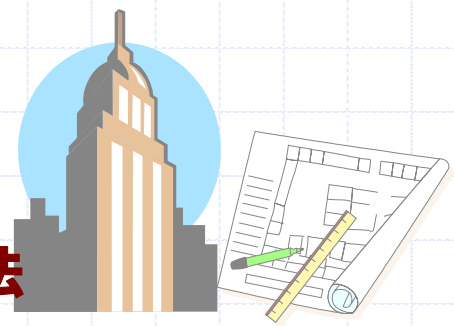
↓はじめからプロジェクトの進行過程での軌道修正を前提に設計ができる

アジャイル的モデルへ

アジャイルモデルは他の産業では珍しいことではない

例：戸建住宅は詳細まで設計図をひくが高層ビルディングを建てる場合などはフロアの詳細な設計は後回し





✓ **アジャイルモデルとオブジェクト指向は双子の手法**

より高い次元での品質の到達

保守性 → オブジェクト指向の目的そのもの

移植性 → 移植の障害になる可能性のある部分をあらかじめ小さくクラスの殻に封じ込められるのもオブジェクト指向の特徴。保守性に通じる。

使用性 → 現実的には、実際に動作(テスト)させてみて具体的に検証される要素。修正を前提とするオブジェクト指向・アジャイル的モデルではここの磨きがいが一番ある。 → **最も大きなアドバンテージ**

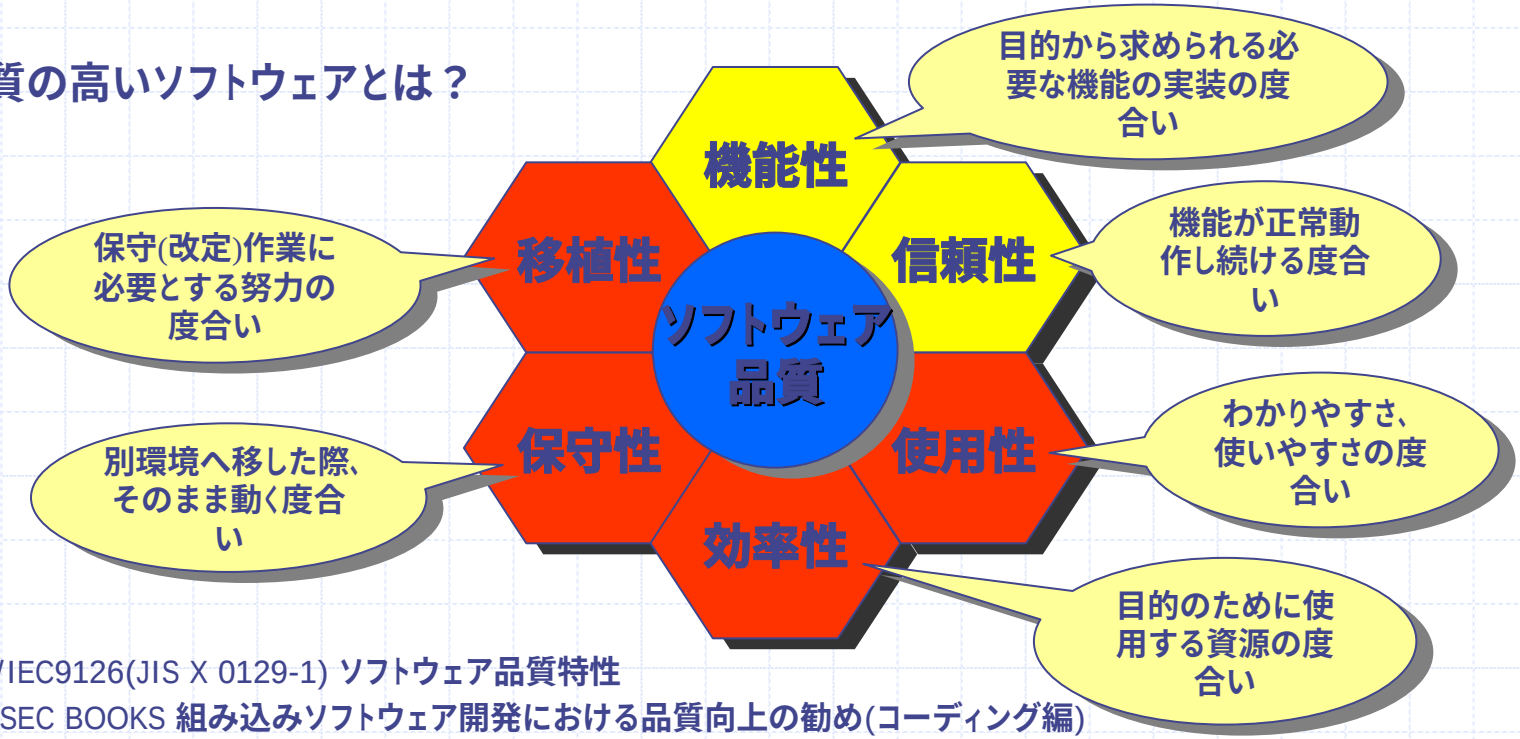
効率性 → オブジェクト指向はモノリシックな構造に比べオーバーヘッドが大きい

しかし、単純化・改良の得意なオブジェクト指向では効率性の向上のための作業も効率がいい

→この作業をバックアップするのもテストの重要な役割

テストの目的は品質の高いソフトウェアへの到達

品質の高いソフトウェアとは？



ISO/IEC9126(JIS X 0129-1) ソフトウェア品質特性
IPA SEC BOOKS 組み込みソフトウェア開発における品質向上の勧め(コーディング編)

- ✓ テストの第一の到達目標は「機能性」と「信頼性」
- ✓ **より高い品質に関しても広義でのテストを通して到達されるべき** → 上質のイメージ

より高い次元での品質の到達

保守性 → オブジェクト指向の目的そのもの

移植性 → 移植の障害になる可能性のある部分をあらかじめ小さくクラスの殻に封じ込められるのもオブジェクト指向の特徴。保守性に通じる。

使用性 → 現実的には、実際に動作(テスト)させてみて具体的に検証される要素。修正を前提とするオブジェクト指向・アジャイル的モデルではここの磨きがいが一番ある。 → **最も大きなアドバンテージ**

効率性 → オブジェクト指向はモノリシックな構造に比べオーバーヘッドが大きい

しかし、単純化・改良の得意なオブジェクト指向では効率性の向上のための作業も効率がいい

→この作業をバックアップするのもテストの重要な役割

アジャイルモデルで求められる部品のクォリティ

作られたモジュールは当初の設計以上の広範囲な使われ方をされるものだ

→ 要求された動作だけを満足できるモジュールでは十分でない。

→ ソフトがユーザーに想定外の使われ方をするのと同じ

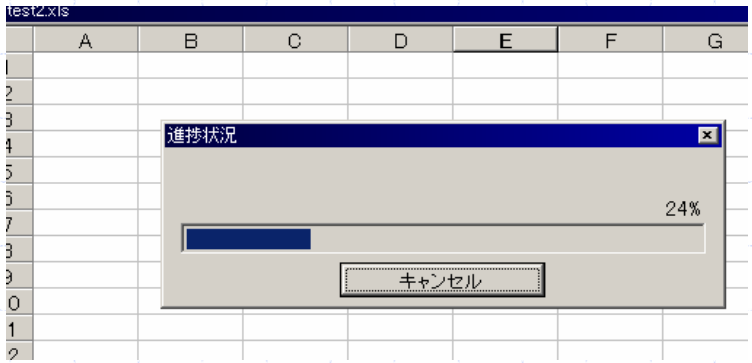
オブジェクト指向ではコンポーネントそのものが完成した独立のソフトウェア

→完成度の高いコンポーネントで組み上げることが重要

✓ 部品レベルから非機能要件まで吟味したテストが求められる

例：0除算の可能性をどう扱うか

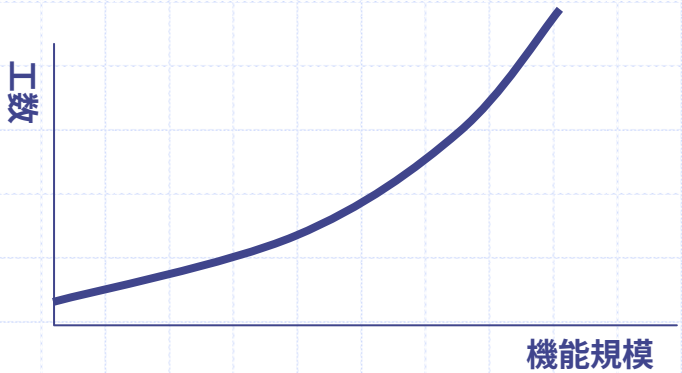
プログレスバーで 進捗具合/全体 を表示するコンポーネントがほしい



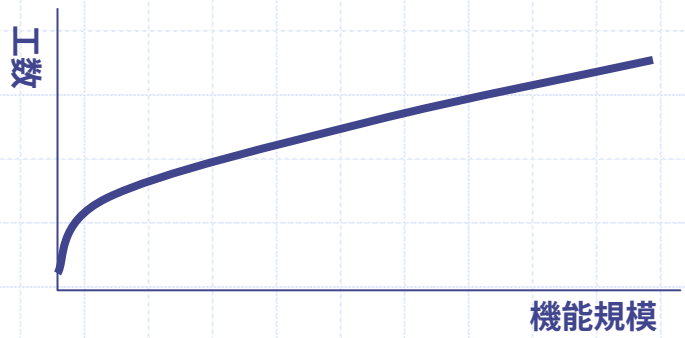
- 全体に0を入れたときの挙動
  - エラーにすべきか？
  - 100%にするべきか？
  - ーや100%以上にも対応しているか？

アジャイルモデルは不利？

- すでにウォーターフォールモデルは限界に来ている
  - → 手戻りが多くなるのは結果的にはアジャイルモデルとかわらない
- バージョンアップのような作業はアジャイルモデルと同じ
  - → 初期バージョンから十分な作りこみになることは今の規模の開発では非現実的
    - ↑ ユーザビリティは現実的には、実際に動作(テスト)させてみて具体的に検証される要素。
- ✓ 手法としてウォーターフォールモデルを採用していたとしても部品が設計上の要求以上の品質が必要な時代になっていることは同じ



ウォーターフォールモデルでの機能規模と工数(イメージ)



管理されたオブジェクト指向･アジャイル的モデルでの機能規模と工数(イメージ)

懐の狭い部品の再利用に関する危険性の例

Webシステムでのデータの取得や表示

- HTMLタグが含まれた時の挙動 → クロスサイトスクリプティングの危険性
- SQLのステートメントが含まれた時の挙動 → SQLインジェクションの危険性

↓

サニタイズ？

↓

狭義でのサニタイズではなく無条件に正しく処理が出来るシステムの工夫が大切

✓ **全てのコードは常に別の用途で再利用される可能性を意識しないと潜在的なバグを生産しつづける**

→それではバグを元から立つことは出来ない

まとめ

- ○○がテストを現実的なものにし、テストが品質を高める。
- ○○は高品質の小さな部品の積み重ね。テストが十分部品を磨かなくては部品はきれいに積み重ならない。

→ **上質とは目に見えない部分での出来のよさ**

使い込めば使い込むほど味が出てくるようなソースコードが上質なプログラムの本質。
上質なプログラムはバクを元から絶ち、より高次元での品質を追求すること。

- **より積極的なテストを目指そう**
  - ✓ バグを洗い出す → バグを作らせない
  - ✓ バグがないことを確認した → バグが無いことを証明した
  - ✓ バグをなくす → より良い製品への検証･研究の場とする
  - ✓ 製品になってからの全体としてだけの磨き上げ → 部品レベルからの磨き上げ

それでも小さな不具合やなんやとつぶしきれるものではない

- 認めたくないことではあるが、最終的にはリリース後の実際のユーザーの使用も事実上テストの延長と言わざるを得ない。
- バグが無いことが理想だが、発生頻度の非常に低い特殊なケースでのバグはユーザーの使用で発見されるものと考えたほうがいい。

■ 使いこまれたソフトがなにより質の高いソフト

→ バグや不具合は実際の使用から洗い出されていく

→ 機能や動作も実際の使用の要望から洗練されていく

→ ユーザーからのフィードバックが大切

→ サポートの窓口･窓口と開発の連携

→ デバッグのための情報の確保

→リリース後も続ける継続的開発が大切

→ 息の長い商品開発がよいソフトに育てる

→ 現在のリソース･機能の枠にとらわれない将来を見通したスケーラブルなソフト設計

✓**HACCPの「4 記録の重視」に通じる→ 常に慢心してはいけない**