ちょっと明日のテストの話をしよう

2013/11/1（金）
ソフトウェアテストシンポジウム九州 2013

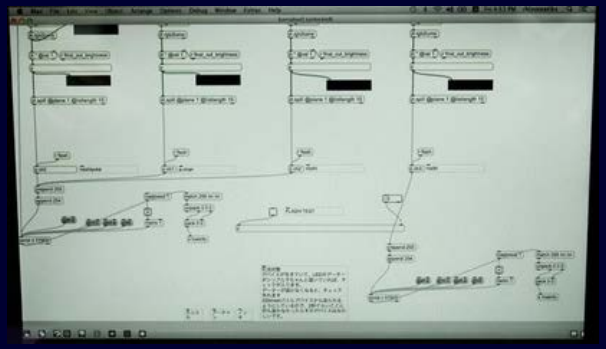
電気通信大学　大学院情報理工学研究科
総合情報学専攻 経営情報学コース
西　康晴（Yasuharu.Nishi@uec.ac.jp）

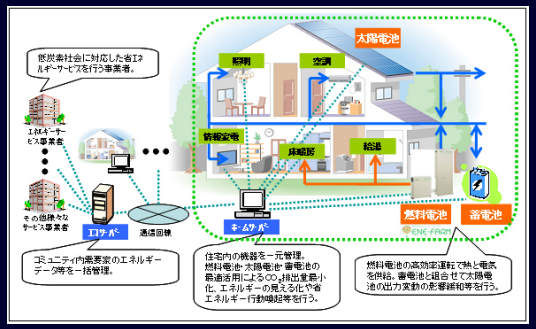
Profile

Assistant professor:
the University of Electro-Communications, Japan
(also providing consultancy service to industry on testing and TQM)
President:
Association of Software Test Engineering, Japan (ASTER)
President:
Japan Software Testing Qualifications Board (JSTQB)
National delegate:
ISO/IEC JTC1/SC7/WG26 Software testing
for ISO/IEC/IEEE 29119-1, 2, 3, 4, 5 and ISO/IEC 33063
Founder:
Japan Symposium on Software Testing (JaSST)
Founder:
Testing Engineers' Forum (Japanese community on software testing)
Founder and awards committee member:
Software Test Design Contest
Vice chair:
SQiP/Software Quality Committee of JUSE (promoting organization of TQM)
(SQiP has published the book of "SQuBOK: Software Quality Body of Knowledge")

今日、何を困ってますか？

- どうやってテストしますか？
  - 別に仕様さえあればテストできますけど何か？
    - » 自動販売機
    - » Matlabみたいな制御モデル
    - » サーバと通信する機器がたくさんある家
- しかし、本当に自信を持って「よいテスト」をしていますか？
  - 仕様を裏返すだけのテストしかしていない
    - » CPM法
  - テストの全体像を自分で納得できてない
    - » 固定3レイヤー法
    - » お仕着せテストタイプ・テストレベル法
  - いつまで経っても人海戦術で疲弊する
- そもそも、テストってなんでしょう？
  - 考えれば考えるほど分からなくなってくる





今日の悩みの例：CPM法（コピー＆ペースト＆モディファイ法）

- CPM法とは
  - 仕様書の文章をコピーしてExcelにペーストし、
    語尾を「～する」から「～すること」に変更することで
    テストケースを作成する方法
  - 複数のテストケースをコピー＆ペーストし、
    置換コマンドを使って機能名などをまとめて変更することで
    テストケースを増殖する方法

- CPM法の問題点
  - テストケースがどんどん増殖していき、収集がつかなくなる
  - テストケースの趣旨が分からないためテストの終了判定ができず、
    頑張って徹夜して消化できたところまでで勘弁してもらうことが常態化する
  - 何のために存在しているのか分からないテストケースが
    派生開発の度に増えていき、怖くて誰も減らせない
  - テストケース（テスト設計）のレビューがちっともできない

今日の悩みの例：固定3レイヤー法

- 固定3レイヤー法とは
  - テストケースを大項目・中項目・小項目のフォーマットに従って設計していく方法

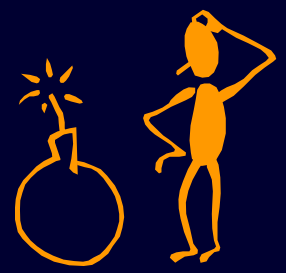
大項目	中項目	小項目	テストケース
機能レベル1	機能レベル2	機能レベル3	機能レベル10
機能レベル1	機能レベル8	機能レベル9	機能レベル10
機能	環境	データ	機能＋環境＋データ
機能	データ	GUI	機能＋データ＋GUI
機能	データ	前提条件	機能＋データ
機能	状態	イベント	状態＋イベント
テストカテゴリ	機能	データ	機能＋データ
組み合わせ	機能①	機能②	機能①×機能②

今日の悩みの例：固定3レイヤー法の問題点

- 詳細化のレベルが飛んでいる・揃わないので網羅できない
  - レイヤー間の距離が大きくなるほど漏れが発生しやすい
  - エンジニアによって偏った詳細化をしてしまう
  - テストケース群ごとに詳細化の偏りにばらつきがある
- 異なるテスト観点の組み合わせを詳細化だと考えてしまう
  - 機能＋環境＋データ、機能＋データ＋GUI
- テスト設計で考慮する必要の無いものが入ってしまう
  - 機能＋データ＋前提条件
- 異なるテスト観点にぶら下げているので網羅できない
  - 機能＋状態＋イベント
    - » 本来は状態遷移網羅をすべきであり、機能網羅で代用すべきではない
- テスト観点の詳細化を行わない
  - テストカテゴリ＋機能＋データ
    - » 負荷にも色々あるはず．．．。
- 組み合わせテストを押し込んでしまう
  - n元表を使うべきである

今日の悩みの例：お仕着せテストタイプ・テストレベル法

- テストタイプ・テストレベルとは
  - テストタイプ：性能テスト、動作環境変更テスト、負荷テストなど
  - テストレベル：単体テスト、結合テスト、システムテスト、受け入れテストなど
  - 多くの組織では、標準的なテストタイプやテストレベルを定義している
- お仕着せテストタイプ・テストレベル法とは
  - 組織標準など、誰かが決めたテストレベルやテストタイプを無批判に使う方法
- ある組織の例
  - 負荷テストとストレステストの両方のテストタイプが必要だとされていた
  - それぞれこんな記述になっていた
    - » 負荷テスト：ストレスをかける
    - » ストレステスト：負荷をかける
- お仕着せテストタイプ・テストレベル法の問題点
  - テストタイプやテストレベルの記述が粗いため、結局のところ一体何をテストしているのか誰も分かっていない
  - 隣接するテストタイプやテストレベルが依存しあっていたり、重なっていたり、両方から漏れていたりする



明日、目の前のテストを改善しましょう：Reverse VSTeP

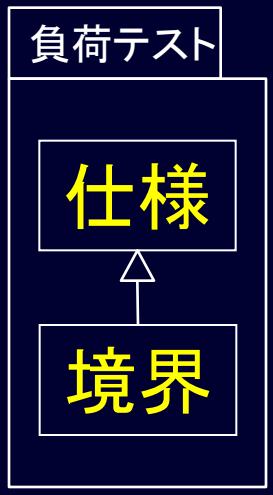
- 小難しい方法論を導入する前に、
  まず目の前のテストをじっくり見てちょっとだけ改善する
  - 目の前のテスト設計を改善する方法群にReverse VSTePがある
    - » http://qualab.jp/vstep/
  - テストケースやテスト設計のレビューも上手になる
  - できる範囲で少しずつ、しかし自分たちの頭で考えて納得しながら進めるのが大事
- カバレッジ分析による改善
  - 見逃したバグや検出できたバグ、実際のテストケースなどを分析して、
    テスト観点は特定できているのにカバレッジ基準・達成率が低すぎた場合や、
    特異点が存在してしまった場合、不適切なリスク値（工数不足）の場合と
    いった原因を明らかにし、カバレッジ基準・目標率、不足した・誤った前提、
    リスク値判定基準などを改善したり、テスト観点や関連を改善する
    - » 仕様で示された同値クラスが必ずしも設計・実装上の同値クラスと
      一致するとは限らないため、テスト設計時の「前提」が重要となる
    - » 関連よりもテスト観点として取り扱う方がよい場合もある
- 不具合モード分析による改善
  - 見逃したバグや検出されたバグをパターン化して、
    ピンポイント型のテスト観点を追加・整理し改善する
    - » 開発者が陥りそうな罠を皆で納得し共感するのが大事である

明日、目の前のテストを改善しましょう：Reverse VSTeP

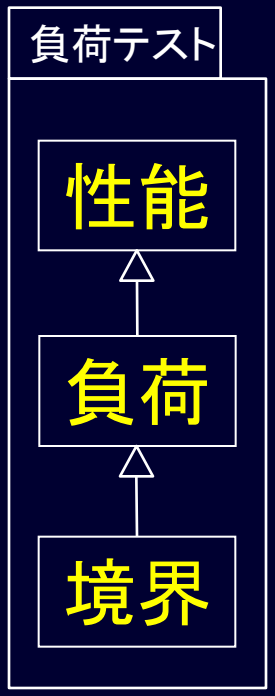
- テスト観点モデルのリバースモデリングによる改善
  - 実際のテストケースを分析して、
    どういうテスト観点や関連でテストしようとしているかを列挙・整理し改善する
  - 実際に見逃したバグや検出されたバグ、テストケース外で検出されたバグを
    分析して、どういうテスト観点や関連が足りなかったかを列挙・整理し改善する
    - » テスト観点を網羅したつもりでも、解釈が曖昧だったり不適切な場合や、
      剪定に失敗してしまった場合もある
- ステレオタイプ分析による改善
  - テスト観点モデルの各詳細化関係の種類やMECE性を分析し、
    テスト観点が適切かつ網羅的に詳細化されるように
    子テスト観点を追加したりモデルをリファクタリングして改善する
- ディレイ分析によるテストアーキテクチャの改善
  - 見逃したバグや検出されたバグを分析して、
    実際のテストレベルやテストタイプを
    リバースされたテスト観点モデルにマッピングし、
    テストアーキテクチャ設計（テストレベルの設計）をレビューし改善する
- 傾向分析によるリスク値の改善
  - 見逃したバグや検出されたバグを分析して、各テスト観点のリスク値
    （利用頻度、致命度、欠陥混入確率など）を改善する

明日、目の前のテストを改善しましょう：スキルの可視化

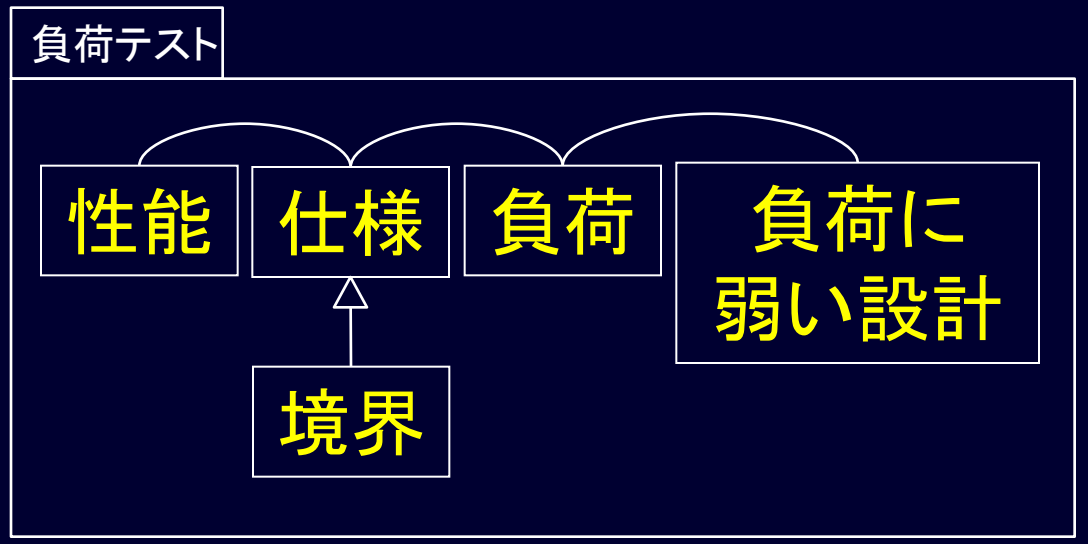
- テスト観点をリバースすると
  テスト設計者の意図が透けて見えてしまうので、
  テスト設計者の技術力が分かってしまう



様々な
負荷のかけ方が
漏れる



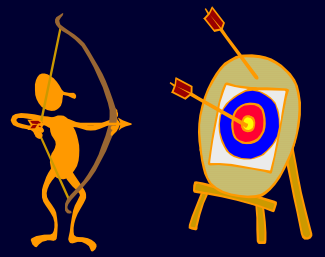
様々な状況で測るべき
性能が漏れる



技術力が必要

明日、目の前のテストを改善しましょう：不具合モード

- 不具合が作り込まれそうな「弱点」（不具合モード）を狙ってピンポイントでテストを設計する方法
  - 過去の不具合を蓄積・分析し、不具合が作り込まれそうな箇所を推測することで、ピンポイントにテスト設計を行う
    - » よく発生する、同じようなメカニズムで作り込まれたと思われる不具合を集めて分析することで、そのメカニズムを推定し、仕様や設計、コードのパターンを導き出してテストを設計する
    - » 当該工程だけでなく、後工程でバグを作り込む原因となりうる部分もパターン化する
  - 開発者がどう間違えるかに着目して開発者が陥りやすい「罠」をパターン化する
    - » 人間の思考の誤謬と、誤謬を引き起こしやすいパターンをセットで明らかにする
      - 例）後から追加された例外的な仕様は、設計漏れを引き起こしやすい
      - 例）優先順位の表現で1（高い）～5（低い）と　5（高い）～1（低い）が混在すると混同しやすい
      - 例）異なるサブシステムでリソースの確保・解放を行うとリークしやすい
      - 例）「備考」にはバグが多い
    - » 同じパターンのバリエーションをツリー状に整理する
    - » 我々は誤謬を引き起こしやすいパターンを「ロジカル・アフォーダンス」と名付けている
      - 認知科学・認知心理学分野との学際領域になる



明日、目の前のテストを改善しましょう：ガイドワード

強度	強い	弱い	
数量	多い	少ない	
	増加	減少	
	余分	不足	
	ある	ない	ゼロ
種類	多種	小種	
規模	大きい	小さい	
距離	遠い	近い	
速度	速い	遅い	
	高速	低速	
	緩	急	
設定	許容	禁止	
	必須	任意	
範囲	長い	短い	
	上限	下限	
	広い	狭い	
	最大限	最小限	
頻度	高い	低い	
	多い	少ない	
金額	大きい	小さい	

時期	早く	遅く	
時間	長く	短く	
間隔	広く	狭く	
期間	長く	短く	
順序	さきに	あとで	
	同時に	並行に	
	早く	遅く	
	逆転	反復	挿入
タイミング	早く	遅く	
拡張	広く	狭く	
	多く	少なく	
頻度	多く	少なく	
	高く	低く	
程度	いつも	たまに	
回数	多く	少なく	
接続	つなぐ	切り離し	切り替え
方向	上へ	下へ	
負荷	高い	低い	
	超過	不足	限界
位置	高く	低く	
	遠く	近く	

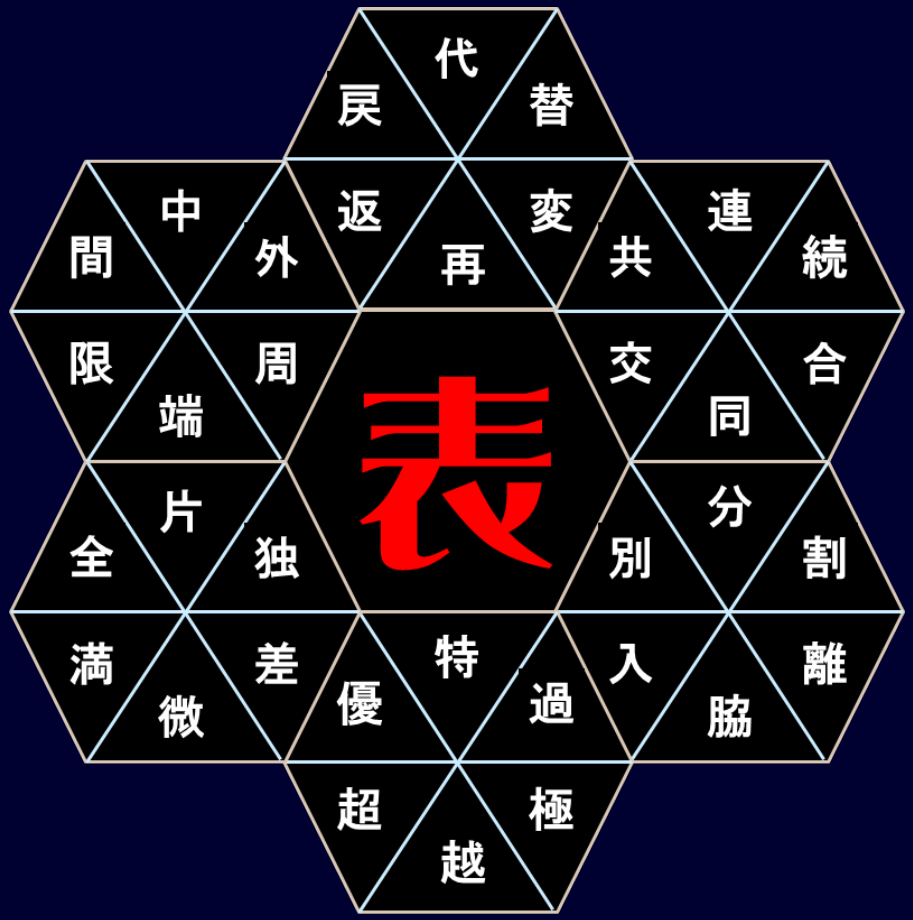
鈴木三紀夫さん（MRTコンサルティング）、秋山浩一さん（富士ゼロックス）がご作成

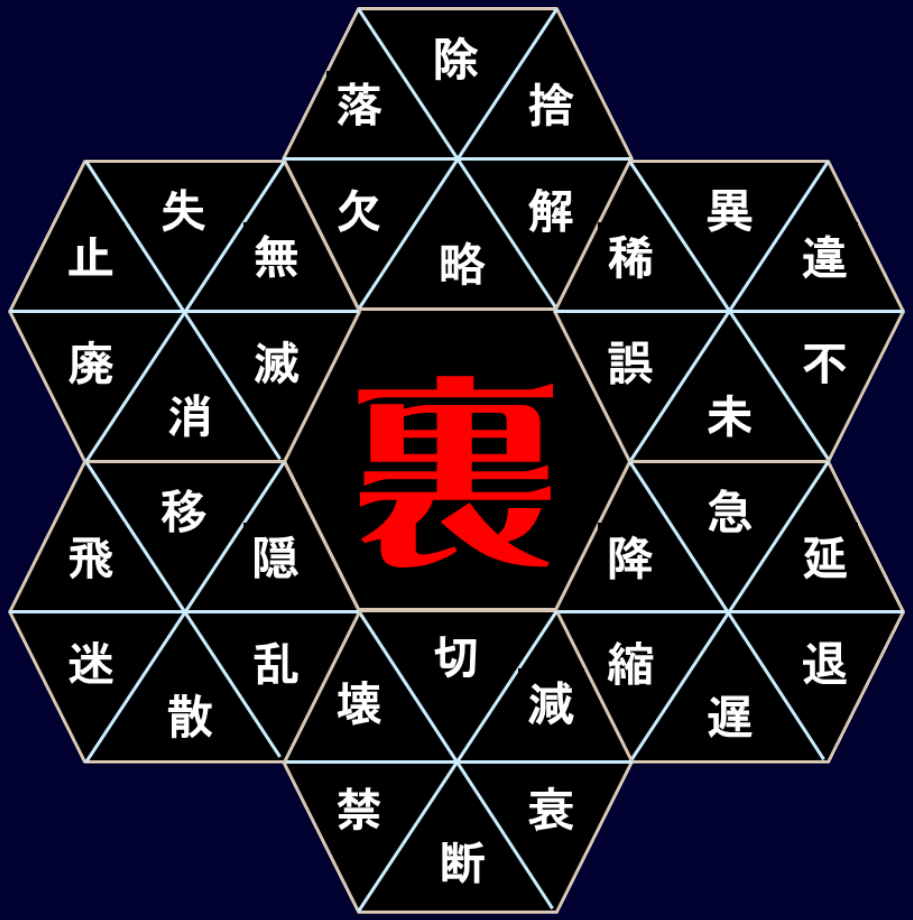
明日、目の前のテストを改善しましょう：ガイドワード

	原則	例外			正常	異常	準正常
	全体	部分			通常	非通常	
	全部	一部			通常	例外	代替
	主	従			定期	臨時	
	正	誤			通例	異例	常例
	無事	有事			平時	非常時	緊急時
	公正	不正			定常	可変	
	任意	強制			尋常	非常	
	基本	詳細	主要		一般	特別	
	完了	未完			普通	特殊	
	有効	無効			並	特異	
	起動	停止					

鈴木三紀夫さん
（MRTコンサルティング）、
秋山浩一さん
（富士ゼロックス）
がご作成

明日、目の前のテストを改善しましょう: 意地悪漢字





鈴木三紀夫さん(MRTコンサルティング)がご作成

明日、目の前のテストを改善しましょう:
富士通の7つの設計原理

①単純原理
- 可能な限りシンプルにすること

②同型原理
- 同じものは同じように実現すること
- 例外を設けないこと

③対称原理
- 対称にすること
- 対になるものは対にすること
- バランスを崩さないこと

④階層原理
- 階層関係、主従関係、前後関係、本末関係などを崩さないこと
- 層に穴を空けないこと

⑤線形原理
- 特異点や閾値、組み合わせによる特殊な振る舞いが無いこと

⑥明証原理
- ロジックは直感的に分かりやすくすること
- 分からないものやふるまいがないこと
- おかしなものを受け取らない、先に送らない、無視しない、勝手に処理しないこと

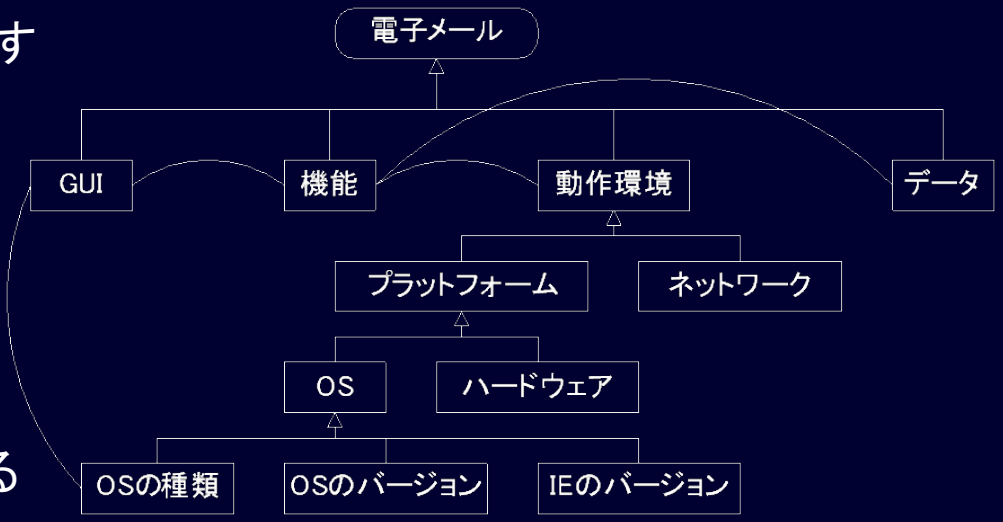
⑦安全原理
- 必然性のないところや曖昧なところは安全サイドに設計しておくこと

明日、目の前のテストを改善しましょう: バグ分析のアンチパターン

- Vaporization（その場限りの簡単なミスとして片付けてしまう）
  - コーディングミス: スキルを向上させるよう教育を行う
  - 考慮不足: しっかり考慮したかどうかレビュー時間を増やし確認する
  - うっかりミス: うっかりしていないかどうかレビュー時間を増やし確認する
- Exhaustion（不完全な実施や単なる不足を原因としてしまう）
  - しっかりやらない: しっかりやったかどうかレビュー時間を増やし確認する
  - スキル不足: スキルを向上させるよう教育を行う
  - 経験不足: 経験を補うためにスキルを向上させるよう教育を行う
- Escalation（上位層に責任を転嫁してしまう）
  - マネジメントの責任: トップを含めた品質文化を醸成する
- Imposition（外部組織に責任を転嫁してしまう）
  - パートナー企業の責任: パートナー企業を含めた品質文化を醸成する
- Culturalization（文化的問題に帰着してしまう）
  - 品質意識／品質文化の欠如: 全社的な品質文化を醸成する

１週間くらいでテスト観点モデルを作ってみましょう

- テスト観点とは？
  - テストケースの狙いのこと
    - » 網羅して動作保証したいものや、検出したいバグなど
    - » テスト条件とかテスト対象とか期待結果（ふるまい）とか狙いたいバグとか
  - 抽象化/詳細化関係、すなわち親子関係を持ちます
    - » 関係の種類をステレオタイプと呼び、継承・部分・属性化・原因結果などがあります
- 関連とは？
  - 組み合わせてテストしたい
    テスト観点同士を曲線で結びます
- どうやって作るの？
  - ボトムアップで作る
    - » 要するにリバースする感じ
  - トップダウンで作る
    - » 要求分析をするのに近い感じ
- 注意点は？
  - 仕様書を書き写さないようにする
  - 「正解」を求めない
  - 最初から全部作ろうと思わない



テスト設計には実に多くの観点が必要：組込みの例

- 機能：テスト項目のトリガ
  - ソフトとしての機能
    - » 音楽を再生する
  - 製品全体としての機能
    - » 走る
- パラメータ
  - 明示的パラメータ
    - » 入力された緯度と経度
  - 暗黙的パラメータ
    - » ヘッドの位置
  - メタパラメータ
    - » ファイルの大きさ
  - ファイルの内容
    - » ファイルの構成、内容
  - 信号の電気的ふるまい
    - » チャタリング、なまり
- プラットフォーム・構成
  - チップの種類、ファミリ
  - メモリやFSの種類、速度、信頼性
  - OSやミドルウェア
  - メディア
    - » HDDかDVDか
  - ネットワークと状態
    - » 種類
    - » 何といくつつながっているか
  - 周辺機器とその状態
- 外部環境
  - 比較的変化しない環境
    - » 場所、コースの素材
  - 比較的変化しやすい環境
    - » 温度、湿度、光量、電源

テスト設計には実に多くの観点が必要：組込みの例

- 状態
  - ソフトウェアの内部状態
    - » 初期化処理中か安定動作中か
  - ハードウェアの状態
    - » ヘッドの位置
- タイミング
  - 機能同士のタイミング
  - 機能とハードウェアのタイミング
- 組み合わせ
  - 同じ機能をいくつカブせるか
  - 異なる機能を何種類組み合わせるか
- 性能
  - 最も遅そうな条件は何か
- 信頼性
  - 要求連続稼働時間
- GUI・操作性
  - 操作パス、ショートカット
  - 操作が禁止される状況は何か
  - ユーザシナリオ、10モード
  - 操作ミス、初心者操作、子供
- 出荷先
  - 電源電圧、気温、ユーザの使い方
  - 言語、規格、法規
- 障害対応性
  - 対応すべき障害の種類
    - » 水没
  - 対応動作の種類
- セキュリティ
  - 扱う情報の種類や重要度
  - 守るべきセキュリティ要件

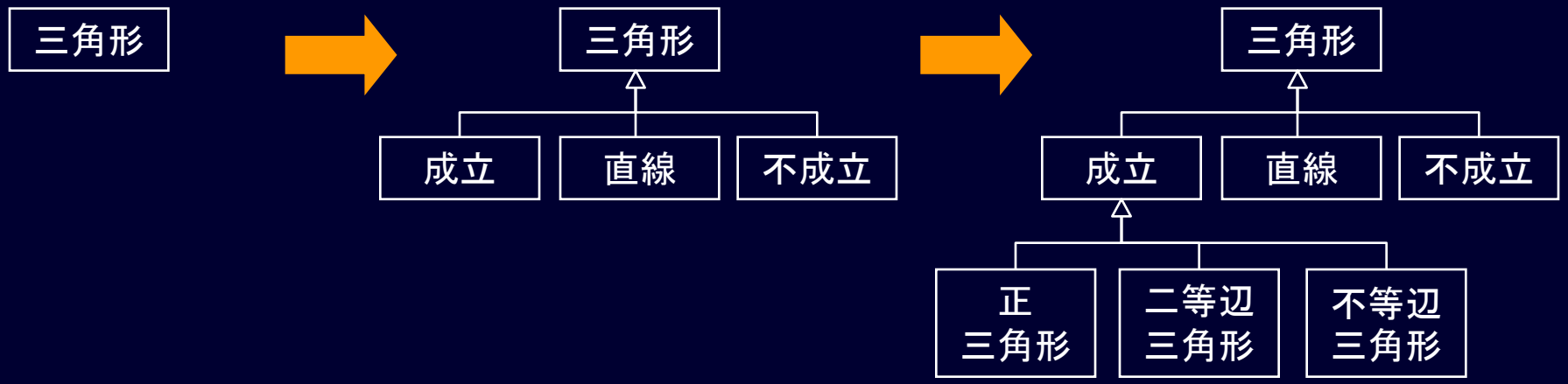
非常に多くの観点を整理して扱う方法が必要である

１週間くらいでテスト観点モデルを作ってみましょう

- ボトムアップ型
  - まず思いつくところからテストケースを具体的に書き、
    いくつか集まったところで抽象化し、テスト観点を導き出していく
    - » (1,1,1) なんてテスト、どうかな
    - » (2,2,2) とか、(3,3,3) もあるな
    - » これって要するに「正三角形」だな
    - » 他に似たようなのあるかな、う～ん、二等辺三角形とかかな
  - 実際にはトップダウンとボトムアップを循環させながらモデリングする

１週間くらいでテスト観点モデルを作ってみましょう

- トップダウン型
  - おもむろにテスト観点を挙げ、詳細化していく
    - » 三角形のテストには、三角形の構造に関する観点が必要だ
    - » 三角形の構造には、成立、直線、不成立の3つがあるな
    - » 成立する場合には、正三角形、二等辺三角形、不等辺三角形があるな
  - 実際にはトップダウンとボトムアップを循環させながらモデリングする



数週間かけてテスト観点モデルをリファインしましょう

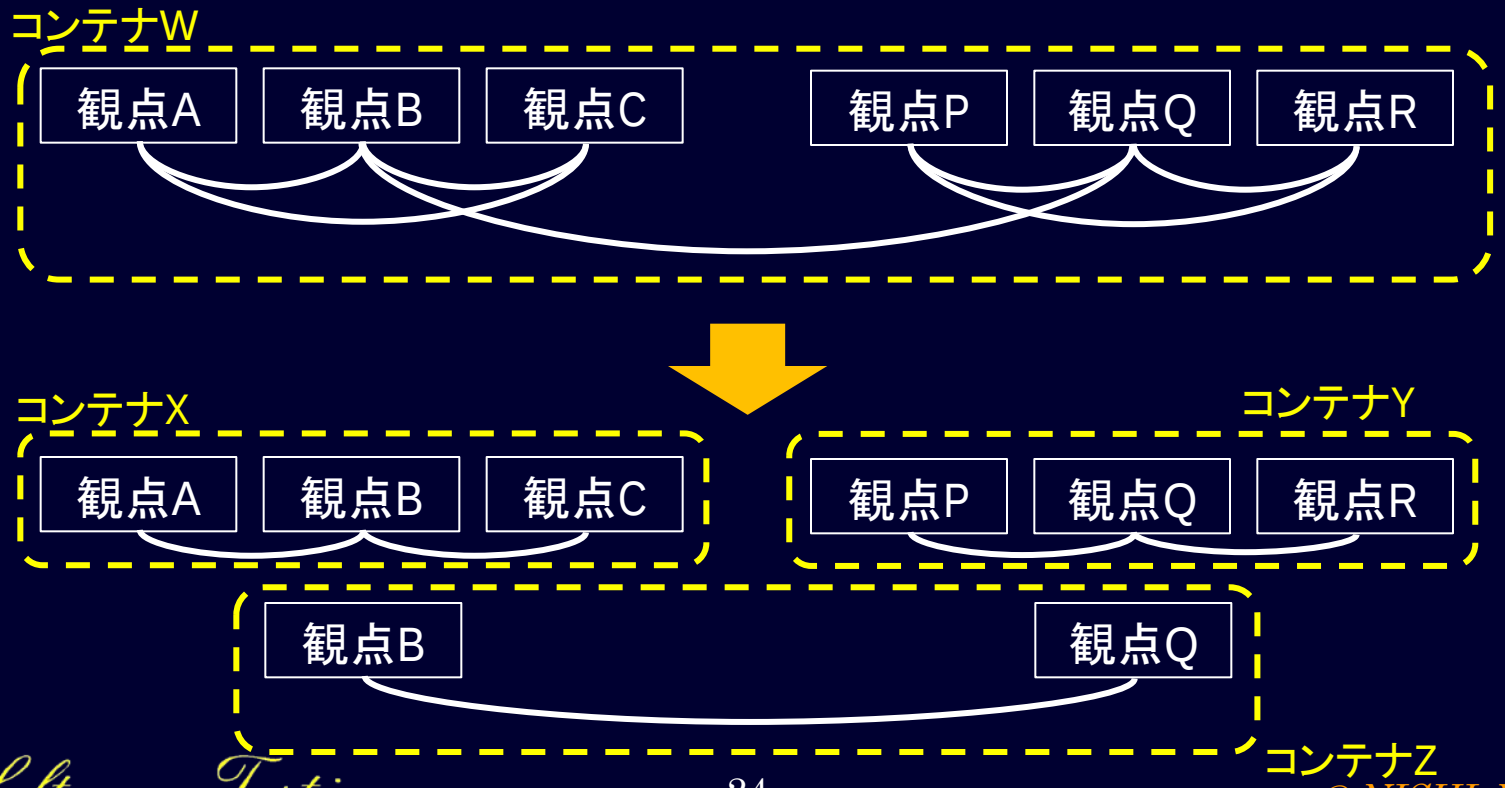
- 質の高いモデルにするために様々なリファインを行う
  - 網羅化：MECE分析
    - » 子観点がMECEに列挙されているかどうかをレビューし、不足している子観点を追加する
    - » MECEにできない場合、必要に応じて「その他」の子観点を追加し非MECEを明示する
    - » 子観点をMECEにできるよう、適切な抽象度の観点を親観点と子観点との間に追加する
  - 網羅化：ステレオタイプ分析
    - » MECE性を高めるために、詳細化のステレオタイプを明示し子観点を分類する
  - 整理
    - » 読む人によって意味の異なるテスト観点を特定し、名前を変更する
    - » テスト観点や関連の移動、分割、統合、名前の変更、パターンの適用、観点と関連との変換、観点と網羅基準との変換などを行う
    - » 本当にその関連が必要なのかどうかの精査を行う必要もある
  - 剪定
    - » ズームイン・アウト、観点や関連の見直し、網羅基準や組み合わせ基準の緩和によって、テスト項目数とリスクとのトレードオフを大まかに行う
  - 確定
    - » 子観点および関連が全て網羅的に列挙されているかどうかをレビューすることで、テスト要求モデル全体の網羅性を明示し、見逃しリスクを確定する

数週間かけてテスト観点モデルをリファインしましょう

- 例えばこんなルールでリファインしてみる
  - ルール1：親と子の関係がおおむね納得できるまで、ノードを移動させる
    - » 子は親の一種（継承／is-a関係）、子は親の一部（合成集約／has-a関係）、子は親の性質（属性化関係）が代表的である
      - 継承関係の例：環境（親）←OS（子）、合成集約関係の例：サーバ（親）←メモリ（子）、属性化関係の例：OS（親）←OSのバージョン（子）
    - » なるべく、親を評価するために子という手段が必要（目的手段関係）、親の条件で動かすと子というふるまいをする（原因結果関係）、の関係を減らす
      - 目的手段関係の例：セキュリティ（親）←ログイン（子）
      - 原因結果関係の例：負荷（親）←性能（子）
    - » 親子が離れていると感じたら、その間にノードを増やす
      - 大中小の3階層ではなく、必要に応じて階層を増やして構わない
  - ルール2：親との関係が同じ子を兄弟と考える
    - » 親と子の間の線に、親子関係を示す名前をつけると便利である
      - 親子関係の名前は<<親子関係>>のように書く
      - 同じ親に異なる兄弟群がぶら下がることになることも多い
  - ルール3：兄弟はなるべく漏れなくダブりないようにする
    - » 漏れてるな、と思ったらノードを追加してよい
    - » 漏れがありそうなところは「その他」という兄弟をつくったりして、そのノードに漏れがありそうなことを明示する
    - » 絶対に漏れがないと確信できるノードは囲みをつけたりマーカーをつけるとよい

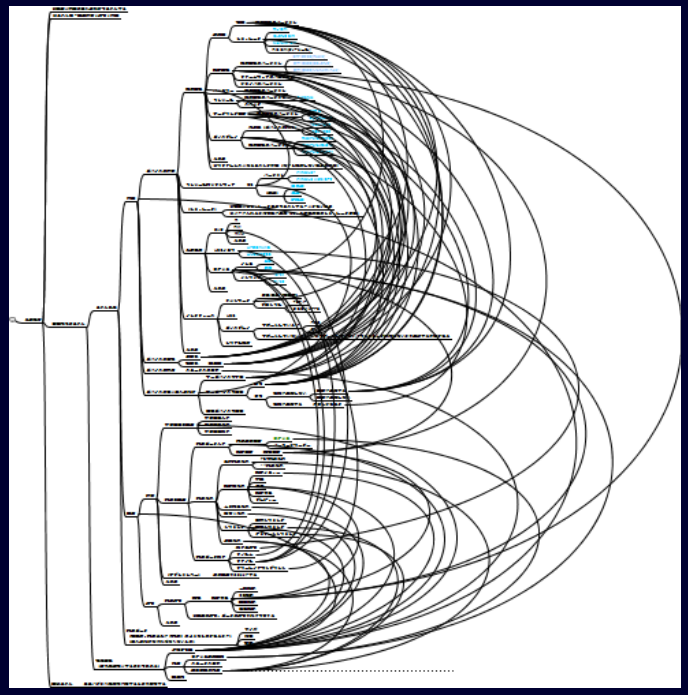
数週間かけてテスト観点モデルをリファインしましょう

- パターンを上手に使うと、すっきりしたテストアーキテクチャになる
  - 例）クラスタ分割：
    - » ごちゃごちゃしたテストコンテナを、凝集度の高いテストコンテナと、コンテナ間の組み合わせテストを表すコンテナに分割して整理するパターン

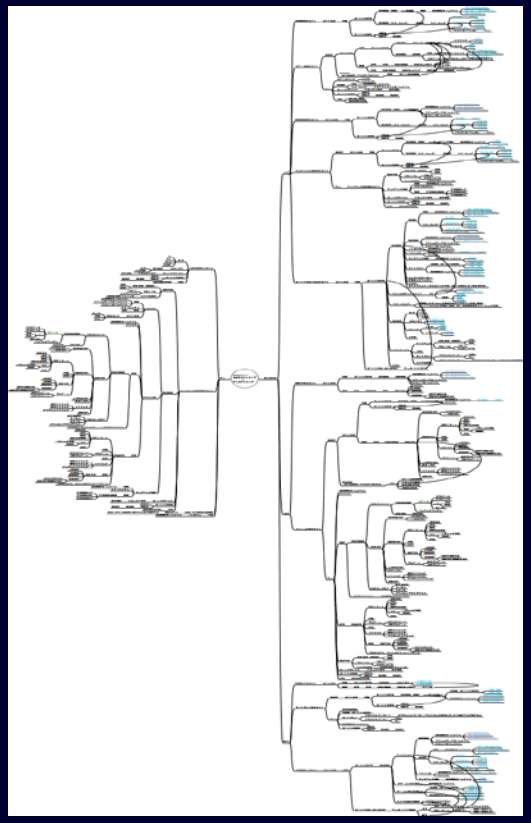


数週間かけてテスト観点モデルをリファインしましょう

- いくつかのテストデザインパターンを用いてリファインを行った例
  - 左右の図は両者とも準ミッションクリティカルシステムのソフトウェアである
    - » マインドマップを使っているので記法は厳密にはNGTに従っていない
  - 左図をリファインして右図のテストアーキテクチャモデルにした
  - 右図のテストアーキテクチャは全体の把握やテスト詳細設計がしやすくなっていると感じられる

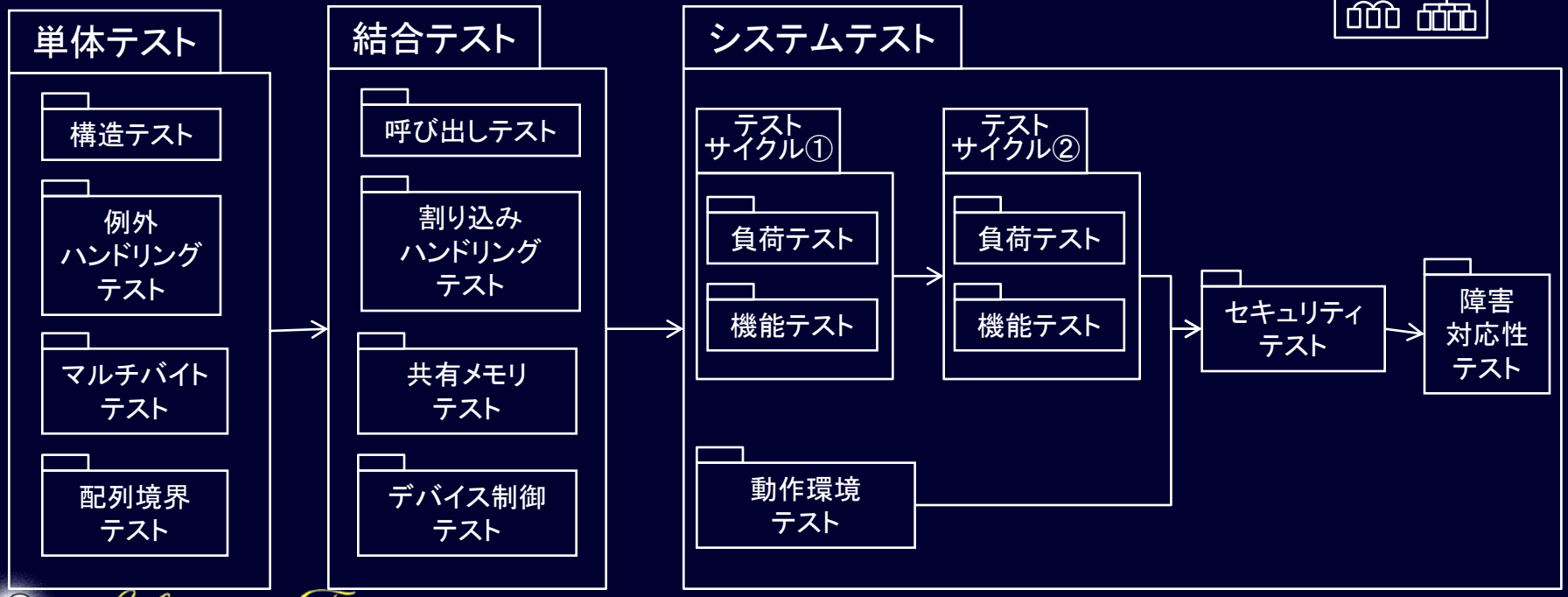


リファイン

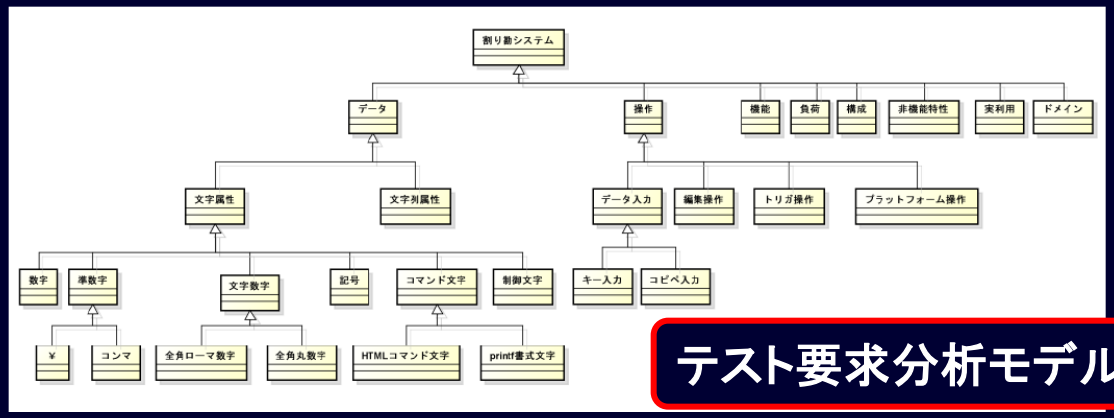


2～3ヶ月かけてテストアーキテクチャを構築しましょう

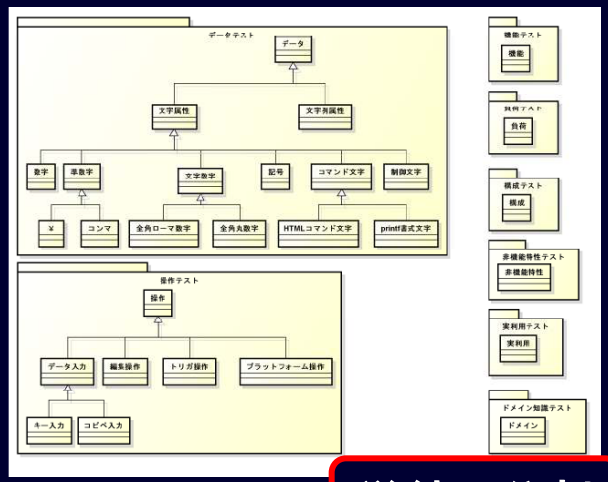
- 自分たちが納得できるテストタイプやテストレベルによって
  テストアーキテクチャを構築し、テストの全体像を俯瞰する
  - テスト観点をつなげて「テストフレーム」をつくる
    - » テストフレームはテストのスケルトンである
  - テスト観点やテストフレームをまとめて「テストコンテナ」をつくる
    - » テストコンテナはテストレベル、テストタイプ、テストサイクルを表す



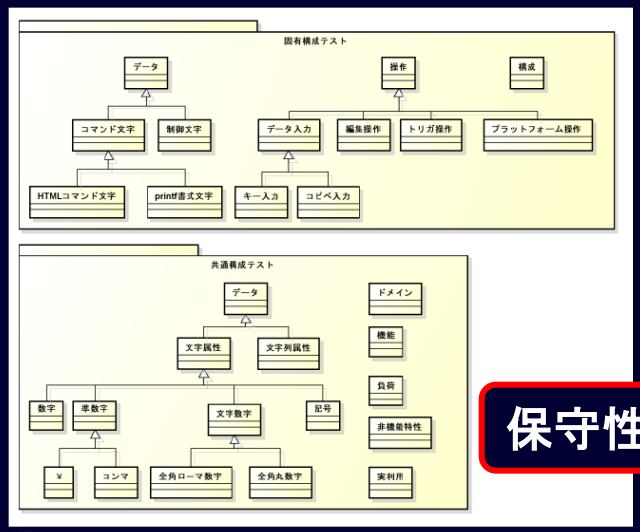
2～3ヶ月かけてテストアーキテクチャを構築しましょう



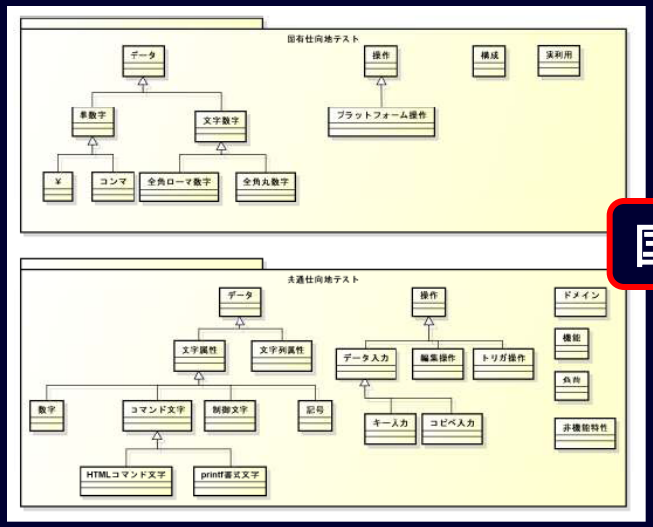
テスト要求分析モデル



単純に分割



保守性重視



国際化重視

3～4ヶ月かけてテスト開発プロセスを確立しましょう

- テストエンジニアが頭を使うべきところを区別・凝縮する
  - 頭を使わないところはなるべく自動化する

テスト項目の抽出

テスト要求の獲得と整理・テスト要求モデリング

テストアーキテクチャモデリング

テスト技法の適用によるテストケースの列挙

手動／自動化テストスクリプト（テスト手順）の記述

５～６ヶ月かけてテストマネジメントを機動的にしましょう

- テスト要求モデルやテストアーキテクチャを地図に見立てて、テスト詳細設計・テスト実装・テスト実行という「運転」をする
  - ハンドルやアクセル、ブレーキを固定せず、いつも調整しながら運転する
    - » テスト詳細設計・実装・実行・バグ報告・品質状況報告・改善を短いピリオドで繰り返す
  - どんどんスピードを上げていく
    - » ピリオドごとに改善を行い、テストのベロシティ（速度）を継続的に向上する
    - » テスト側受け入れテストを漸進的に複雑化し、リリースの品質を継続的に向上する
  - ドライバーのセンスを活かし、バランスを取る
    - » 1人時～数人日程度のテストスロットごとにテスト観点やテストフレーム、テストコンテナ、チャーターなどのスロットミッションを決める
    - » 記述的なテストだけでスロットを埋めず、踏み込んだテストをする余地を必ず残す
    - » 探索的テストはそれだけで一つのスロットを使う
    - » 保証型テストと検出型テストのスロットのバランスを取り、時期に応じて変化させる
  - ゴールへのルートを機動的に判断してナビゲーションを行い、地図に色を塗っていく
    - » ピリオドごとに、次のコンテナ／フレーム／観点に進んで実施済みエリアを拡げるか、そのコンテナ／フレーム／観点に留まって品質リスクを減らすか、を判断する
      - そこに留まる場合は、通過できないエリアを明示してトレードオフを行う
    - » きちんとしたテスト要求モデルやテストアーキテクチャを基に、ステークホルダーに頻繁に状況の説明やルートの提示、懸念点の提示を行い、常に納得してもらっているという状態を保つ

1～2年かけて腰を据えて自動化に取り組みましょう

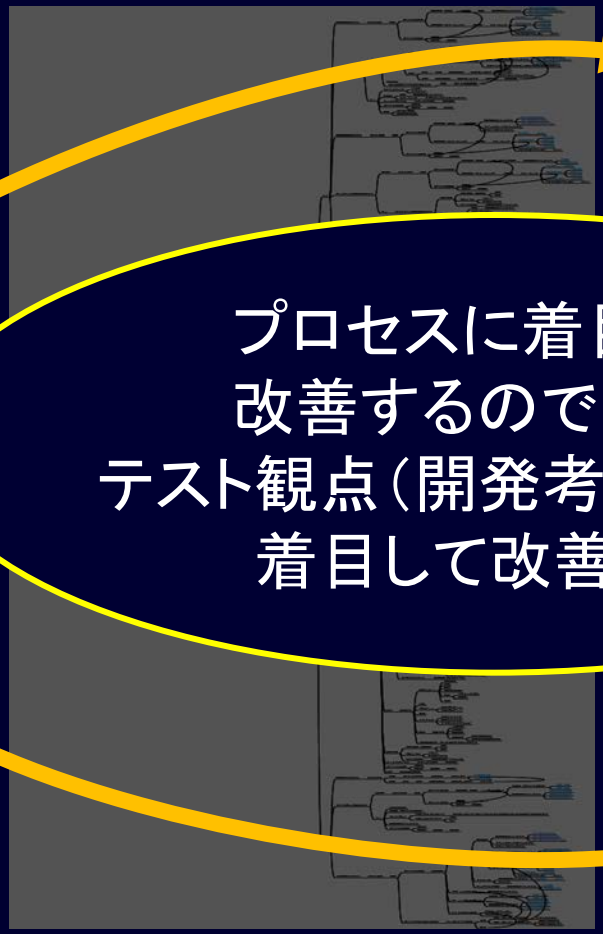
- 自動化には4つの技術レベルがある
  - よいツールはKDTまでカバーしているが、何でも自動化できると思ってはいけない
- 自動操作ツールによるキャプチャ＆リプレイ（C&R）
  - キャプチャが必要となるため、多くのバリエーションをテストするには手間がかかる
- データ駆動テスト（DDT）
  - 全く同じ操作でデータだけが異なる場合は、ツールが自動でデータの書かれたExcelファイルを読み込んでバリエーションを自動で作成し実行してくれる
    - » しかし単純な操作の組み合わせのようなバリエーションは、自動化テストスクリプトそのものを変更しなくてはならないため依然として手間がかかる

- キーワード駆動テスト（KDT）
  - 単純な操作（キーワード）をスクリプトライブラリと関連づけておくと、テストケースをキーワード名で書くだけで、自動化テストスクリプトを意識しなくてもツールがテストケースのバリエーションを自動で作成し実行してくれる
    - » 操作レベルの詳細なテストケースが必要なため、仕様変更にすぐに追従できない
- テスト観点を用いたキーワード駆動テスト（VKDT）
  - テストアーキテクチャやテストフレームをVSTePで作成しておくとテストケースが自動生成できるので、仕様変更をすると自動でKDTを行うことができる
    - » ケースレベルフレームとスクリプトレベルフレームの作成と対応が必要

2～3年かけて開発上流と協調していきましょう

- VSPI：Viewpoint-based Software Process Improvement
  - テスト観点図を基にして上流の改善を行う
    - » 上流で検討されていないテスト観点を特定し、検討するよう上流を改善する
- Wモデル：テストのフロントローディング
  - プログラミングの後にテストを行うVモデルには限界がある
    - » 手戻りが多くなり不具合の影響も大きくなってしまうため、品質も生産性も頭打ちとなる
  - Wモデルでテストの設計をフロントローディングして上流工程から品質を作り込む
    - » 上流の作業と同時に、もしくは直後にテストの設計を行うと、作り込んだ欠陥をすぐ直せるため、手戻りが減り影響も小さくなり、品質も生産性も向上する
    - » （特にCPM法による）テストケース記述をフロントローディングするのは危ない
    - » テスト設計の際にテストエンジニアが用いている「知恵」をフロントローディングする
  - Wモデルにより、ソフトウェア開発組織やソフトウェア開発者が「賢く」なる
    - » Wモデルでノウハウの抽出・蓄積・共有・駆使・改訂がプロセスに埋め込まれるようになるため、ソフトウェア開発組織が全ての工程でノウハウを生みだし共有し駆使すべく進化できるようになる
    - » ソフトウェア開発者が「最もよいやり方で考える」状態に近づくようになる

2～3年かけて開発上流と協調していきましょう：VSPI



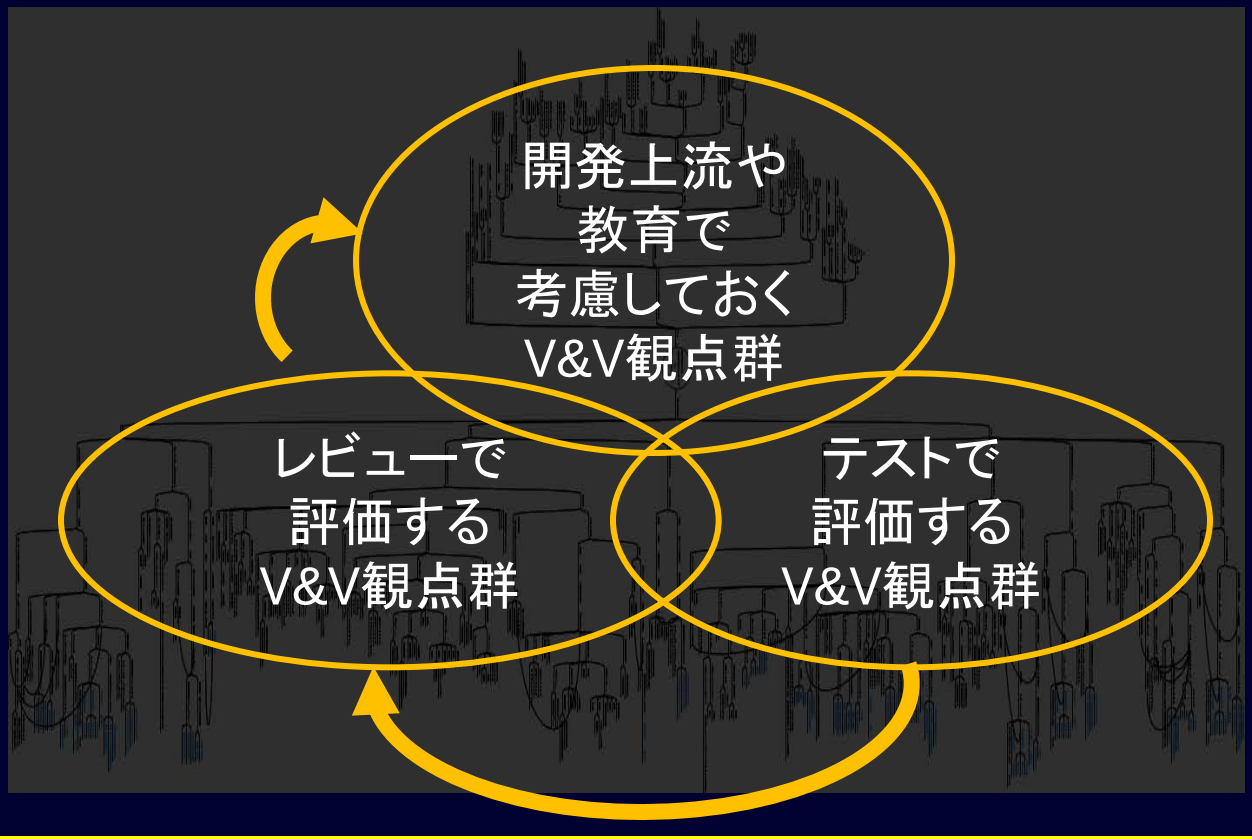
仕様書や設計書に
網羅的に書かれている
テスト観点群

網羅テストを基に
探索して得られたり
バグを分析して得られた
テスト観点群

テスト工程で
リバースしないと
分からない
テスト観点群

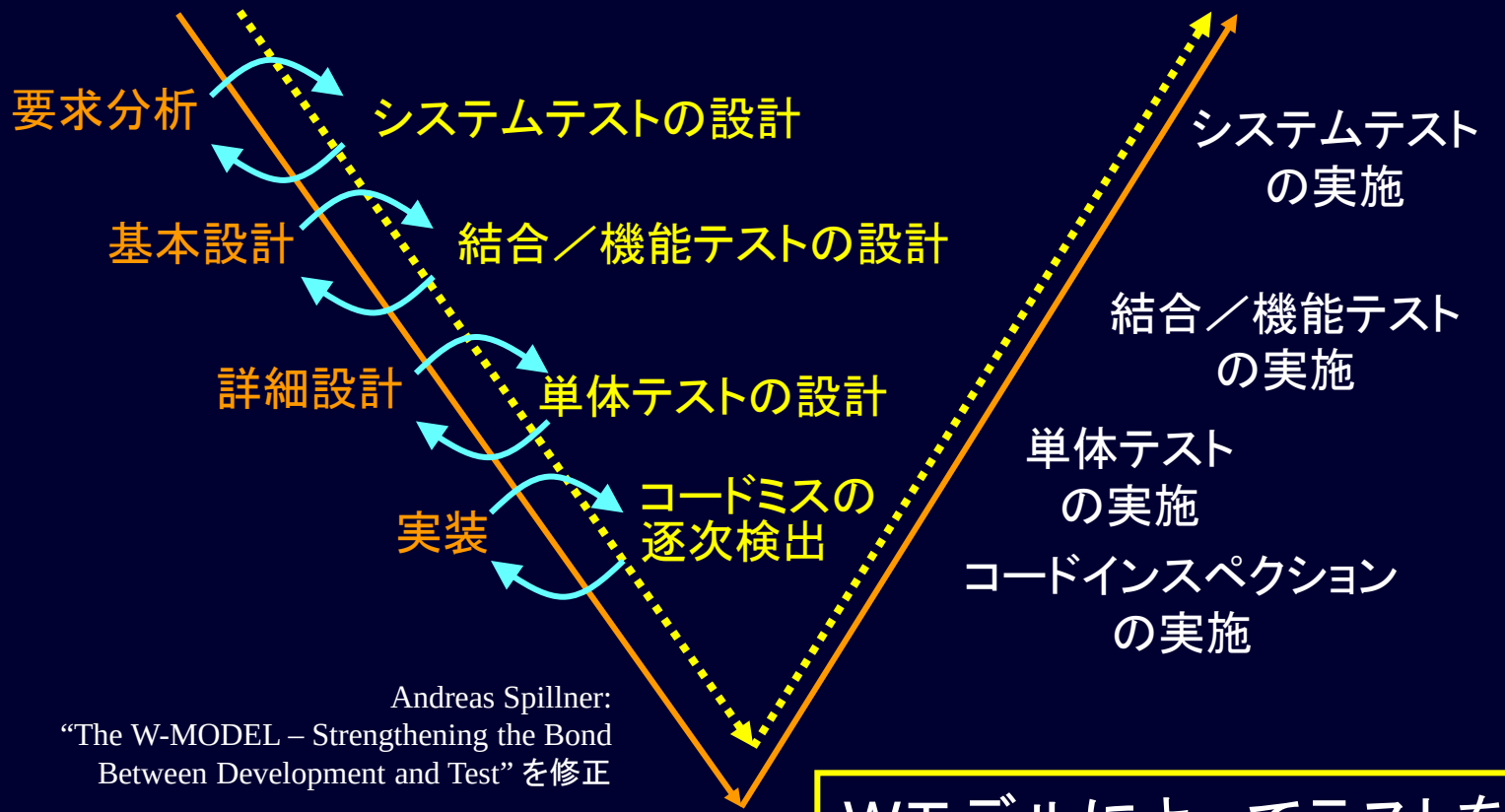
プロセスに着目して
改善するのではなく
テスト観点（開発考慮事項）に
着目して改善する

2～3年かけて開発上流と協調していきましょう：VSPI



上流へのV&V観点の
フロントローディングによって改善を行う

2～3年かけて開発上流と協調していきましょう：Wモデル

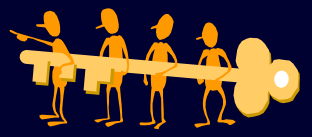


Andreas Spillner:
“The W-MODEL – Strengthening the Bond Between Development and Test” を修正

Wモデルによってテストを前倒しし、
テストエンジニアの「知恵」を
フロントローディングする

2～3年かけて開発上流と協調していきましょう: Wモデルの目指すべき姿

- 全ての工程でノウハウを生みだし共有し駆使する開発プロセス
  - ノウハウの抽出・蓄積・共有・駆使・改訂がプロセスに埋め込まれるようになる
  - 開発チームが開発進行中に自分達でプロセス改善をするようになる
  - 幅と遠さの両方で、開発者の見通しをよくすることができるようになる
    - » 開発者は一般的に分割統治・詳細隠蔽の原則で発想することが多いので、テスト設計者と一緒に設計検討をすると見通しがよくなることが期待できる
  - ソフトウェア開発者の多能工化が進む
- 開発者が「最もよいやり方で考える」ことに近づけていくプロセス
  - テストについて深く洞察することで、テストを実施することなく品質を高められるようになる
  - 開発者の気づきのフィードバックサイクルが極限まで小さくなり、開発予測力が高まっていく
  - よい開発をするという点において、開発者とテスト技術者のゴールは同じである



いつも、テストってなんなのかを心に刻みましょう

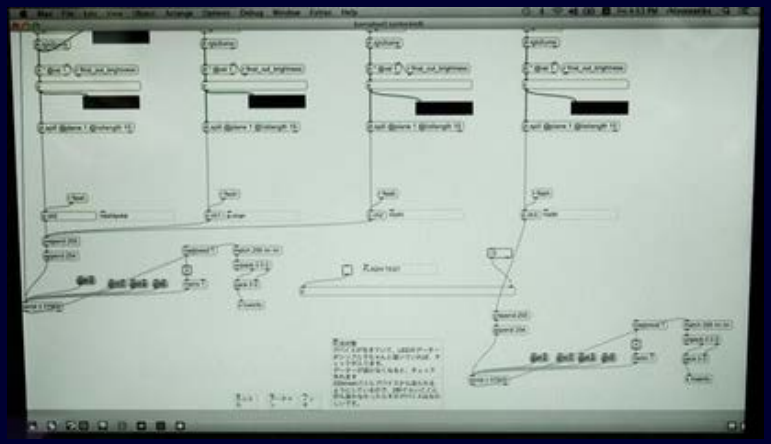
- スタンス1：テストとは行動である
  - 「テストでは何も証明できない、ただバグを見つけるだけである」
    - » 行動の内容と結果だけを提示する
      - 何時間テストしました、何千件テストしました
    - » 見つけたバグが全てだと思うあまり、探索型などという言葉を隠れ蓑にして“食い散らかす”だけのテストを導いてしまったりする
- スタンス2：テストとは説明である
  - 「テストとは仕様通り動くことを実証することである」
    - » 行動の根拠や行動の妥当性も提示する
      - この仕様書の網羅性は100%です
    - » 自分達はこれに従ってこれをやりました、だから問題はないでしょう、という“説明無責任”のテストを導いてしまったりする
- スタンス3：テストとは納得し不安を減らすことである
  - 自分と相手の双方が納得し不安を減らすために技術と知性（とまごころ）を尽くす
    - » 納得し不安を減らしやすいように理解しやすく提示する
      - これがテストの全体像です、一緒に検討して頂けませんか
      - こっちのテストはこういう狙いでここまで網羅し、そっちはこういう狙いで網羅を放棄しました
      - いくつか選択肢とメリットデメリットがあるので、一緒に検討していきましょう

ずっと「テストは何を評価するのか」を考え続けましょう

販売機能のテストから
「買い物の素晴らしさ」の評価へ

≪画像の出典≫
http://blogs.yahoo.co.jp/toshio_yamaguchi_papa/20277585.html
http://img.youtube.com/vi/aLL2BpTqzaY/hqdefault.jpg

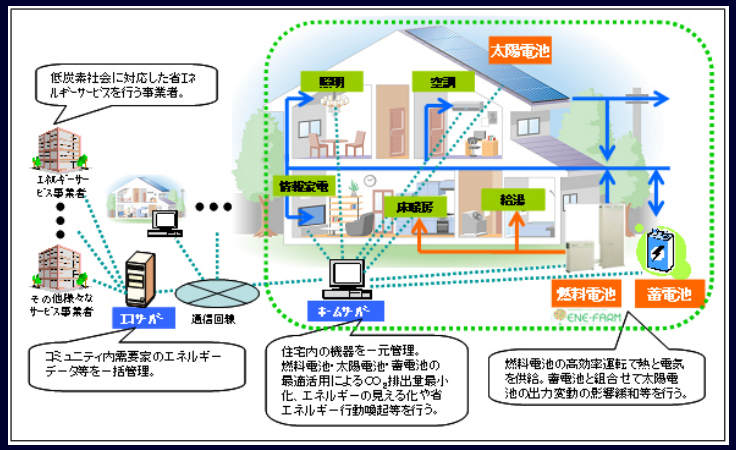
ずっと「テストは何を評価するのか」を考え続けましょう

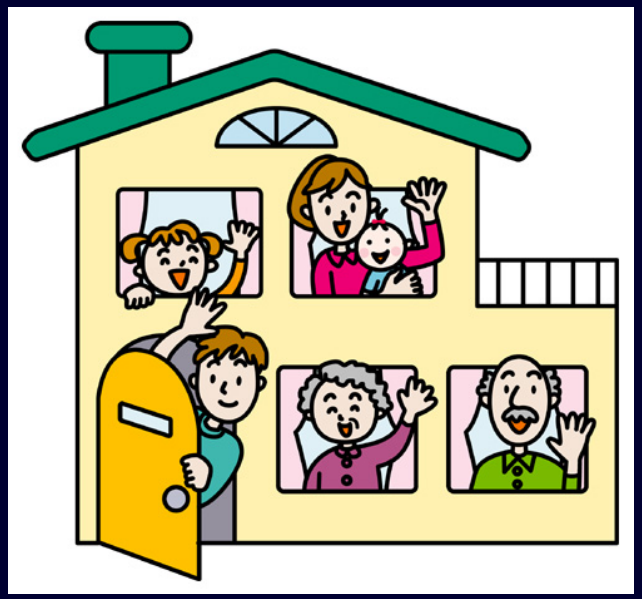


LEDの点滅制御テストから
「エンターテイメントの素晴らしさ」の評価へ

≪画像などの出典≫
http://www.fashion-headline.com/article/img/2013/09/20/3366/35058.html
http://minkara.carview.co.jp/userid/1015107/blog/26079145/
http://www.youtube.com/watch?v=hjoomb_dWtI

ずっと「テストは何を評価するのか」を考え続けましょう



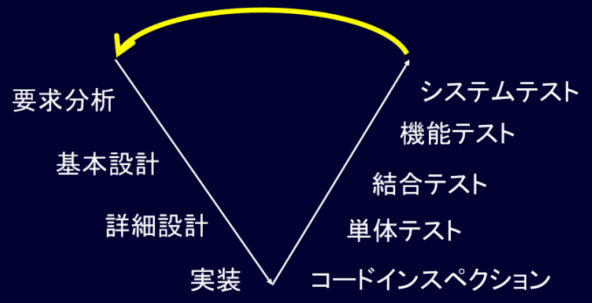


通信・制御機能のテストから
「よい暮らし」の評価へ

≪画像の出典≫
http://www.osakagas.co.jp/company/press/pr_2009/1177857_1256.html
http://picasaweb.google.com/lh/photo/lyKW-7rj9S6vPiCHNe2bXIq8i5aFaOBvuykzi7JB9L4

ずっと「テストは何を評価するのか」を考え続けましょう

- テストとは、対象の「良さ」と「悪さ」を本質的に評価すること
  - 「良い」ソフトウェアとは何か
    - » 仕様適合？非機能要求？ユーザ経験？ビジネス価値？ユーザの達成？社会の変革？
  - 「悪い」は「良くない」ではない
    - » 人はなぜ間違えるか、何を気持ち悪いと思うか、嫌いとはどういう感情か
- 本質的な評価に必要なことは何か
  - 感受性
  - 知識・教養・たくさんのものを知って見て触れて嗅ぐこと
  - まっすぐものを見ること
  - 自分のものの見方・考え方に気づき、色々なものの見方・考え方を身につけること
  - 本質を掴み、本当はどうすべきかを導けること
- テストエンジニアから品質クリエイターへ
  - きちんと評価できるテストエンジニアは、
    素晴らしいシステムを生み出す創造性を持っている
    - » Vモデルからクレープモデルへ
    - » 評価の軸を常に自分で創造する気概を持とう



ずっと「テストは何を評価するのか」を考え続けましょう

きちんとテストできるエンジニアは
素晴らしい未来を描くことができる

ちょっとした明日をちょっとずつ積み重ねて素晴らしい未来を実現していきましょう

電気通信大学　西　康晴
Yasuharu.Nishi@uec.ac.jp
http://qualab.jp/