VSTePによるソフトウェアテストの開発

ソフトウェアテストシンポジウム東北2016

2016/5/20（金）

電気通信大学 大学院情報理工学研究科
情報学専攻経営・社会情報学プログラム
西　康晴

自己紹介

- 身分
  - ソフトウェア工学の研究者
    - » 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
    - » 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／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
- もろもろ
  - TEF: テスト技術者交流会 / ASTER: テスト技術振興協会
  - WACATE: 若手テストエンジニア向けワークショップ
  - SESSAME: 組込みソフトウェア管理者技術者育成研究会
  - SQiP: 日科技連ソフトウェア品質委員会
  - 情報処理学会 ソフトウェア工学研究会 / SE教育委員会
  - ISO/IEC JTC1/SC7/WG26（ISO/IEC 29119 ソフトウェアテスト）

ASTER

SESSAME

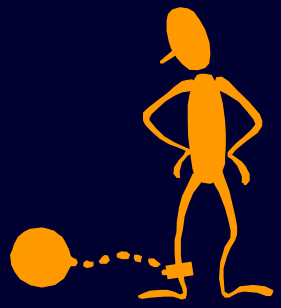
SQiP
Software Quality Profession

講演の流れ

- VSTePに取り組む価値のない組織・ある組織
- かんたんVSTeP
- 一歩踏み込んだVSTeP
- VSTePの応用と意義
- まとめ

VSTePに取り組む価値のない組織

- テストは仕様書通りに動かして証拠作りをする
（ので軽作業派遣でいい）と思っている組織
  - 仕様をなぞって画面キャプチャ、の無限ループにコストをかけなくてはいけない組織
    - » 開発由来の場合とお客様由来の場合がある
  - 考慮すべきことは上流で全て考慮しているので下流は確認すればいい、
というトップダウンな規格に疑問なく従っている組織
- 巨大なExcelの表を埋めることに凄まじい愛情を注げる組織
- パートナーにテストを丸投げしていて特に不自由のない組織
- なんでやるのか分からないテストケースでも
自動化して大量に実行すれば何かした気になれる組織
- テストが属人化していても、ずっと同じ組織でずっと同じアサインができるから
むしろもっと属人化させたい組織
- なにかコンサルの言うことに従っていれば
頭を使わなくてもよいテストができると期待している組織
  - VSTePは、自組織の誰も経験したことのないことを「予言」するような方法ではない



VSTePに取り組む価値のない組織

このままでいいや、と

どこかで思っている組織

VSTePに取り組む価値のある組織

- 仕様のコピペでテストケースが肥大化し収拾が付かなくなっている組織
- 技法やテストタイプなどのテストの社内標準はあるけど
  意味ないと思っている組織
- テストケース表のExcelの大中小項目がフリーダム過ぎる組織
- パートナーがダメ、もしくはパートナーに依存してしまっている組織
- テストの知恵を貯めてテストを改善したい組織、
  さらにはレビューや開発も改善したい組織
- 異なる部署や異なる組織、異なる国をまたいで
  効果的にテストを行いたい組織
- 説明責任の高いテストを行うことで監査や審査と闘いたい組織
- よいテスト設計とよいテスト自動化を融合させたい組織

VSTePに取り組む価値のある組織

このままじゃダメだ、と
強く思っている組織

講演の流れ

- VSTePに取り組む価値のない組織・ある組織
- かんたんVSTeP
- 一歩踏み込んだVSTeP
- VSTePの応用と意義
- まとめ

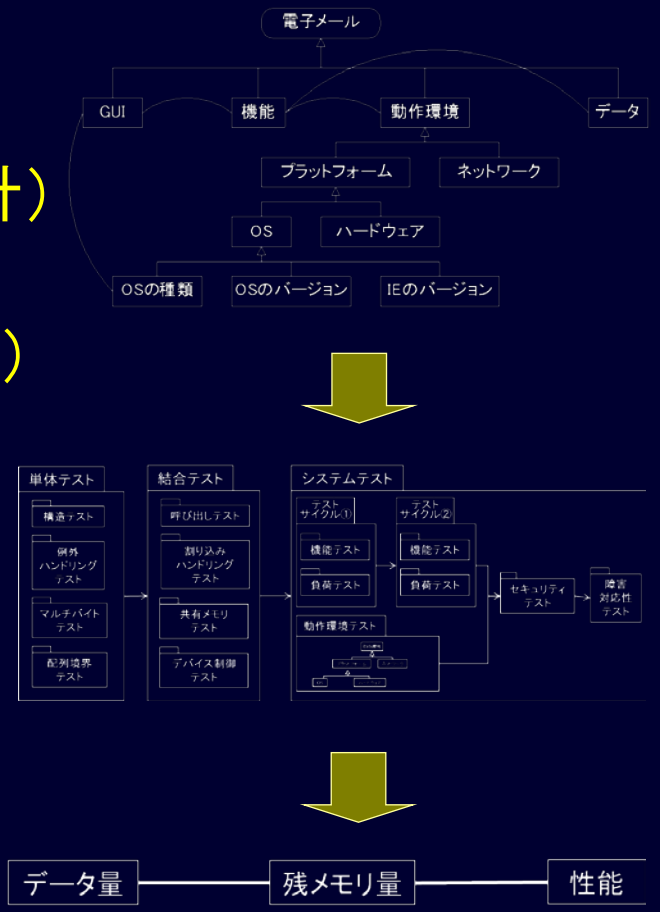
要するにVSTePってなに？

- テストケースをざっくり考えてから具体化する方法
  - 「○○機能に△△人のアクセスを与えて□□秒のレスポンスが…」
    といきなり細かく考えるのではなく
    「負荷をかけたいなー、でも他にテストすることないかなー」とまずはざっくり考える
    - » いきなり細かく考えると、大きな抜け漏れが発生しやすくなる
    - » ざっくりしたテストケース(「負荷」)を「テスト観点」と呼ぶ
    - » Excelの大項目・中項目・小項目のようなものだと考えてよい
  - Excelの表だと全体像が見えにくいので、UMLっぽい図に書いて全体像を把握する
    - » 全体像を把握できると、網羅してる実感を持ちやすくなったり、
      テスト計画やテスト戦略を立てたり改善しやすくなる
    - » 全体像を把握できると、みんながムダなくテストできるし、上流や顧客とも合意しやすい

ざっくり考えて図にすると
見やすいし整理しやすい！

VSTePの簡単な流れ

- テスト要求分析
  - テスト観点を思い浮かべて、線でつなげよう
- ・テストアーキテクチャ設計（コンテナ設計）
  - テストコンテナにまとめて並べよう
- テストアーキテクチャ設計（フレーム設計）
  - テストケースのひな形（テストフレーム）を作ろう
- テスト詳細設計
  - テストケースのひな形に具体的な値を入れよう
    - » 普通のやり方と同じでよい
- テスト実装
  - テストケースをテスト手順に具体化しよう
    - » 普通のやり方と同じでよい



テスト設計とテスト開発は何が違うの？

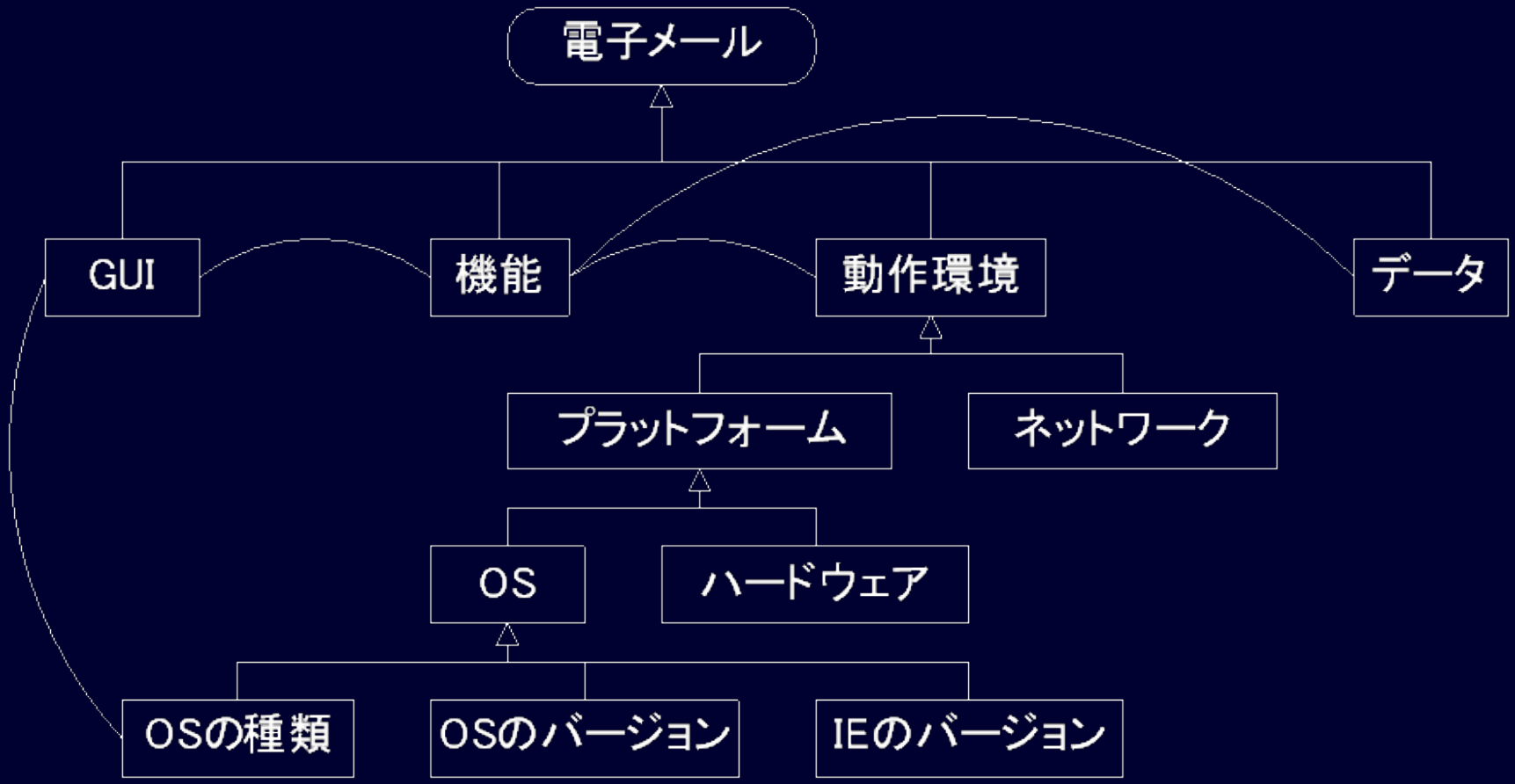
- いきなりコーディングすることと、
  ソフトウェア開発を行うことの違いと同じである
  - コーディングは、アルゴリズムとプログラミング言語を分かっていればできる
    - （従来の）テスト設計も、テスト設計技法と文書フォーマットを分かっていればできる
    - 経験豊富でセンスのある人が規模の小さいプログラムを書くのであれば別に構わない
  - ソフトウェア開発は、要求を把握し設計原則を理解・適用して、
    段階的に詳細化しながら順序立てて進めていく必要がある
    - テスト開発も、テスト要求を把握しテスト設計原則を理解・適用して、
      段階的に詳細化しながら順序立てて進めていく必要がある
    - これまで皆の経験とセンスを結集して規模の大きなソフトウェアを開発することである
    - 経験とセンスを結集するために、
      様々なソフトウェア工学上の概念や技術、方法論が必要となる
- テストの開発のための方法論の一つがVSTePである
  - ほかにもHAYST法やゆもつよメソッドなど色々あり、それぞれに強みがある

NGT/VSTePの概要

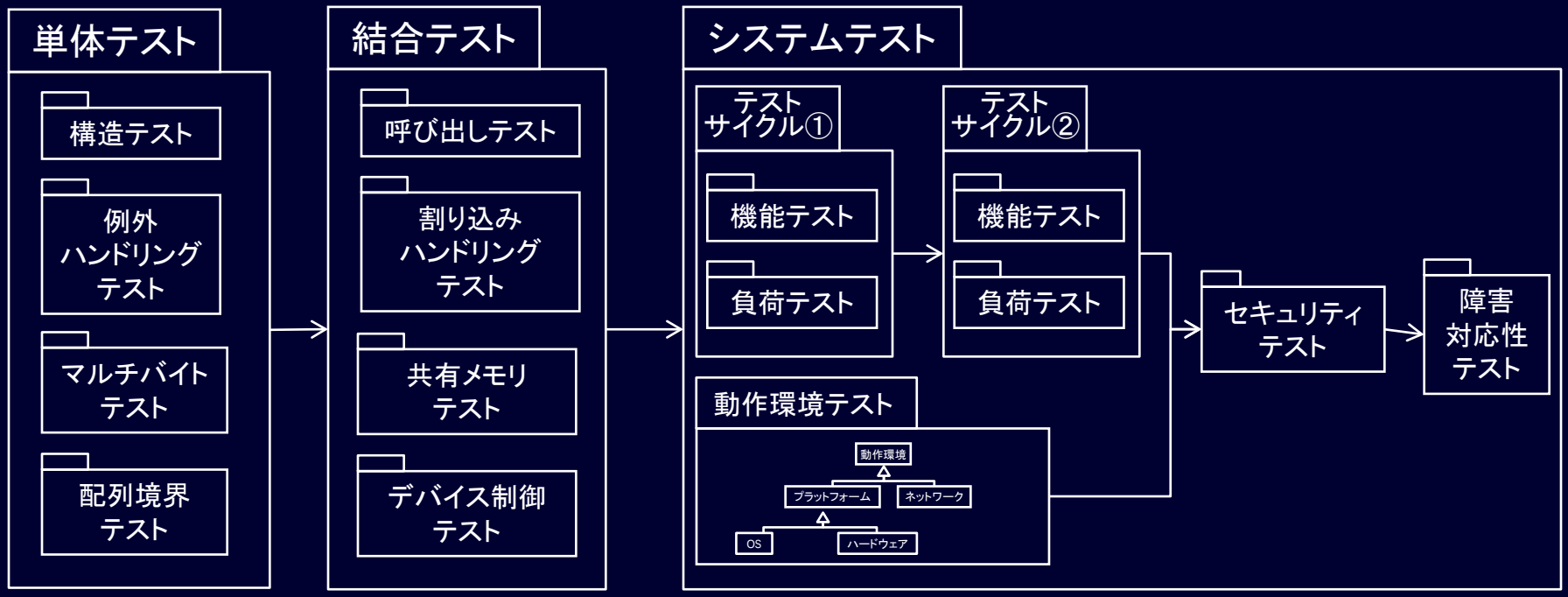
- NGT/VSTePとは何か
  - テスト観点を基盤としたテスト開発のための記法とテスト開発プロセス
    - » NGT（Notation for Generic Testing）：テスト開発のための記法
    - » VSTeP（Viewpoint-based Test Engineering Process）：テスト開発プロセス
  - テスト観点（図）、テストコンテナ（図）、テストフレームをつくることで、質の高いテストケースやテスト手順を作成する方法

- テスト観点とは？
  - テストケースごとの「テストの意図」である
  - テストケースの（一部）を抽象化したものになる
- テストコンテナとは？
  - テストレベルやテストタイプ、テストサイクルをまとめたものである
    - » テストレベル： 単体テスト、結合テスト、システムテストなど
    - » テストタイプ： 負荷テスト、構成テスト（動作環境変更テスト）、セキュリティテストなど
    - » テストサイクル： 時間的な区切りでテストをまとめたもの
- テストフレームとは？
  - テストケースのひな形である

テスト観点図のイメージ

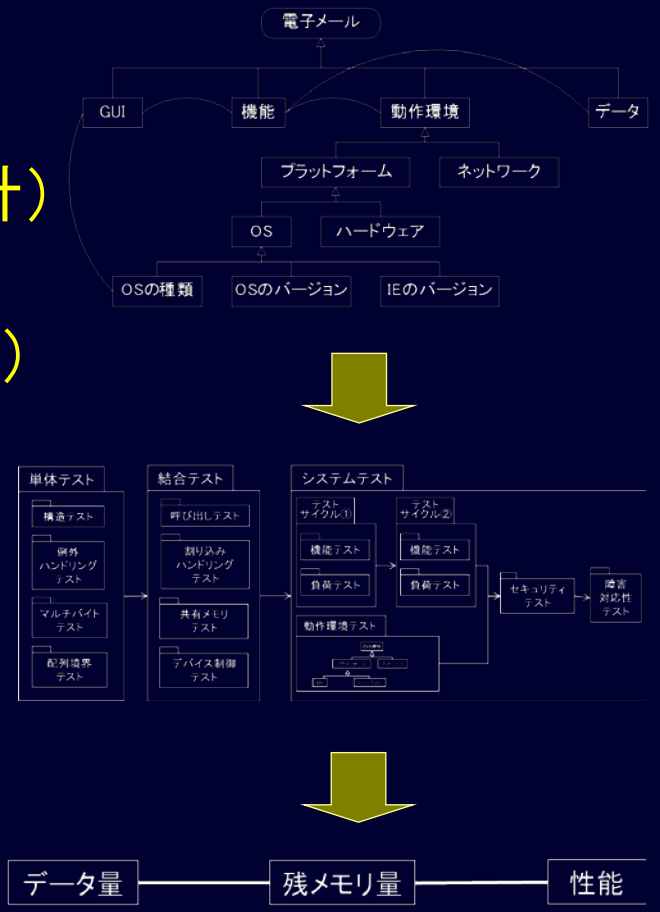


テストコンテナ図とテストフレームのイメージ



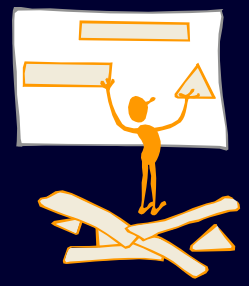
VSTePの簡単な流れ

- テスト要求分析
  - テスト観点を思い浮かべて、線でつなげよう
- ・テストアーキテクチャ設計（コンテナ設計）
  - テストコンテナにまとめて並べよう
- テストアーキテクチャ設計（フレーム設計）
  - テストケースのひな形（テストフレーム）を作ろう
- テスト詳細設計
  - テストケースのひな形に具体的な値を入れよう
    - » 普通のやり方と同じでよい
- テスト実装
  - テストケースをテスト手順に具体化しよう
    - » 普通のやり方と同じでよい



テスト開発プロセスの基本的考え方

- テストケースを開発成果物と捉え、
  ソフトウェア開発プロセスと
  ソフトウェアテスト開発プロセスを対応させよう



ソフト要求分析　＝　テスト要求分析
ソフトアーキテクチャ設計　＝　テストアーキテクチャ設計
ソフト詳細設計　＝　テスト詳細設計
ソフト実装　＝　テスト実装

- テストケースがどの工程の成果物かを考えるために、
  各プロセスの成果物を対応させよう

ソフト要求仕様（要求モデル）　＝　テスト要求仕様（要求モデル）
ソフトアーキテクチャモデル　＝　テストアーキテクチャモデル
ソフトモジュール設計　＝　テストケース
プログラム　＝　手動／自動化テストスクリプト（テスト手順）

旧来のテスト設計を細かく分けたのがテスト開発プロセス

【旧来の流れ】

テスト設計（or テスト計画 or テスト実施準備）　→　テスト実施

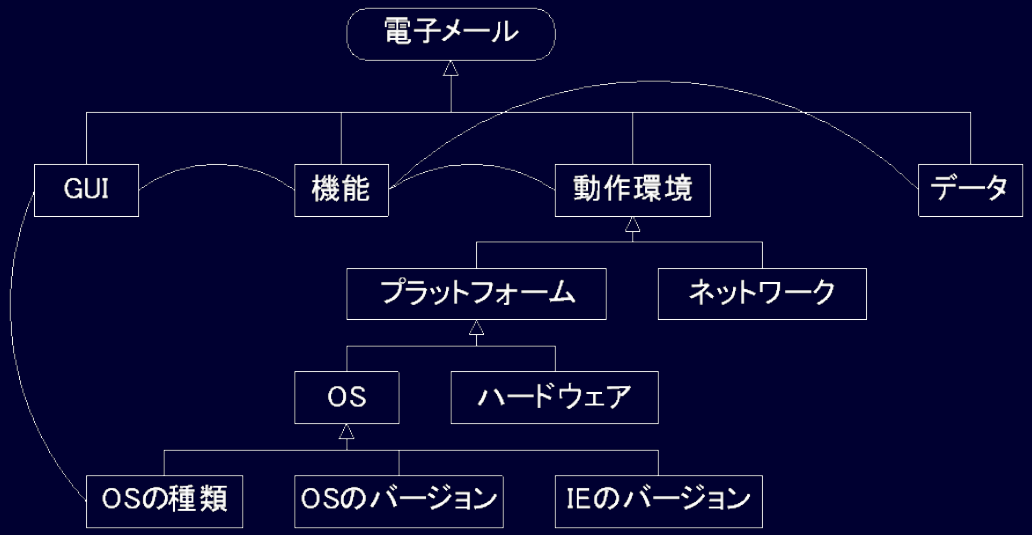
テスト項目の抽出

【VSTePの流れ】

テスト要求分析	テストアーキテクチャ設計	テスト詳細設計	テスト実装	テスト実施
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	

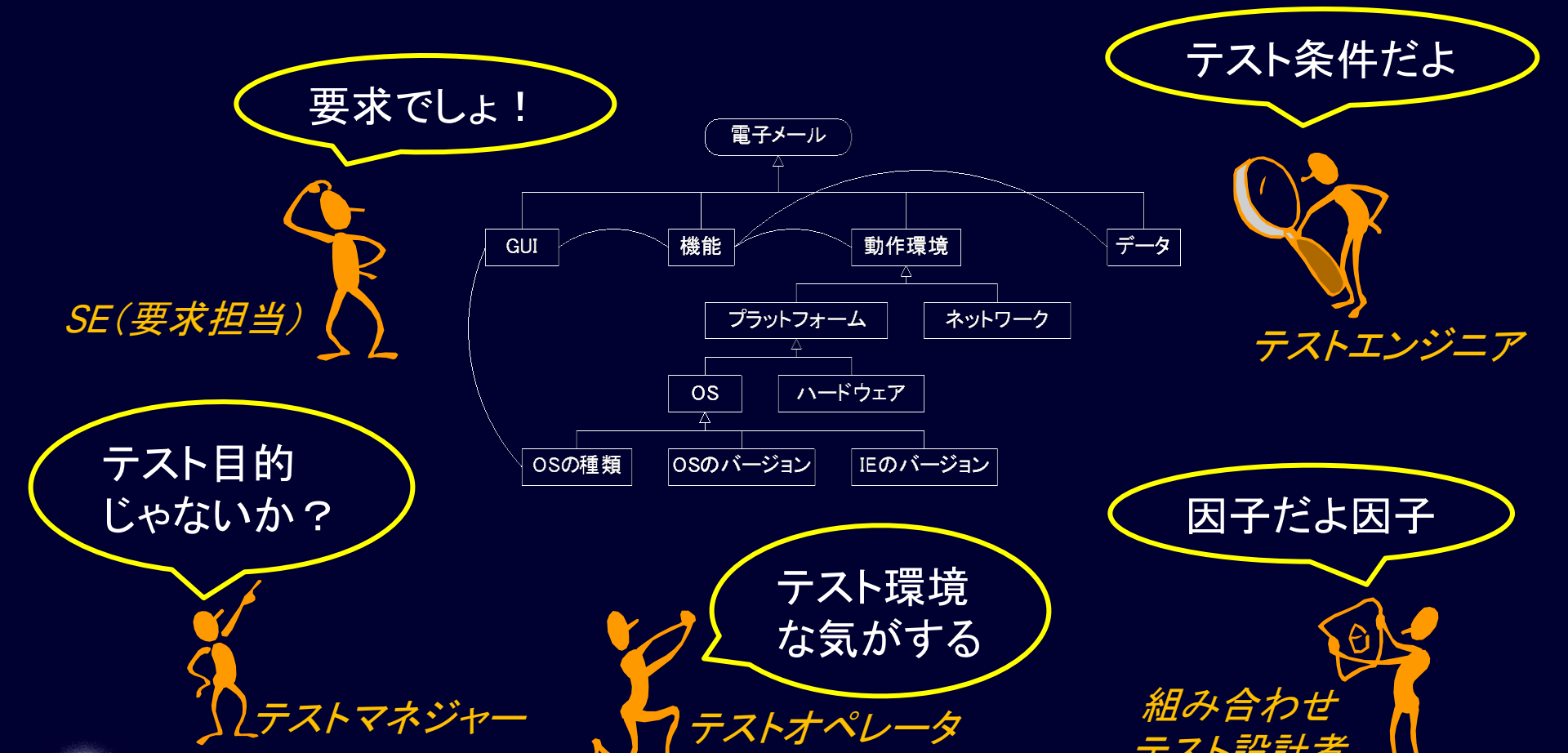
NGTによるテスト観点図の記述

- NGT/VSTePではテスト観点図を中心にして進めていく
  - NGT (Notation for Generic Testing) という記法を定義している
  - テスト観点を階層的に記述していく
  - □ はテスト観点、⬭ はテスト対象を表す
  - → は詳細化関係を表す
  - ⌒ は組み合わせテストの必要性などテスト観点間の関連を表す
  - ⟶ は順序依存関係を表す
  - 新たな関係や詳細な関係を定義したり分かりやすくするために **<<stereotype>>**（ステレオタイプ）を使ってもよい



なぜ「テスト観点」と呼ぶのか？

- テスト観点という用語はロールに依存しないからである



テスト観点の例：組込みの場合

- 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

テスト観点の例：組込みの場合

- 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
- 性能
  - 最も遅そうな条件は何か
- 信頼性
  - 要求連続稼働時間
- セキュリティ
- 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

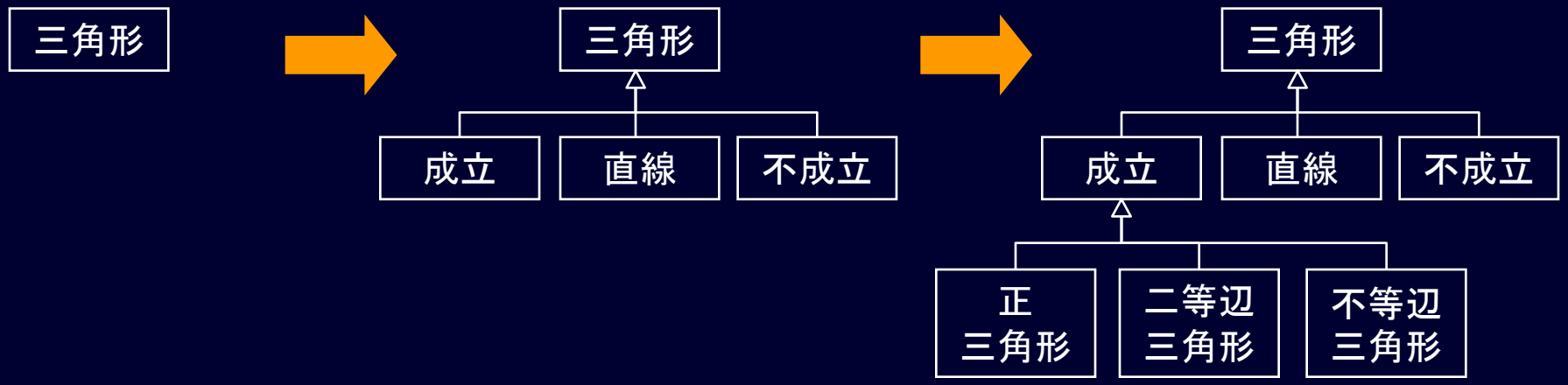
テスト観点はテストケースの「意図」を表している

テスト観点とテストケースとテスト手順を区別する

- テスト観点とテストケースとテストスクリプトをきちんと区別する
  - テスト観点
    - » 何をテストするのか（テストケースの意図）のみを端的に記述したもの
      - 例）正三角形
  - テストケース
    - » そのテスト観点でテストするのに必要な値などのみを特定したもの
      - 例）(1,1,1), (2,2,2), (3,3,3)…
    - » 通常は、網羅基準に沿って特定される値などのみから構成される
    - » テストケースは基本的にシンプルになる
  - テストスクリプト（テスト手順）
    - » そのテストケースを実行するために必要な全てが書かれたもの
      - 例）1. PCを起動する 2. マイコンピュータからC:¥sample¥Myers.exeを起動する…
    - » 手動でのテスト手順書の場合もあれば、自動テストスクリプトの場合もある
    - » 複数のテストケースを集約して一つのテストスクリプトにすることもある
- これらを区別し、異なる文書に記述し、
  異なる開発工程に割り当てることによって、
  テスト観点のみを検討することができるようになる

テスト観点図の描き方の例:トップダウン型

- 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

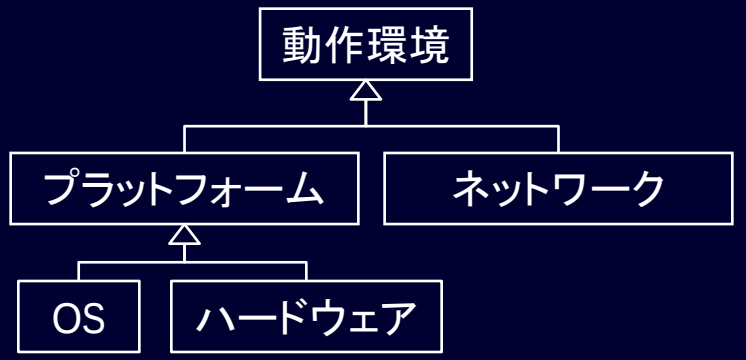


テスト観点図の描き方の例：ボトムアップ型

- 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

テスト観点の詳細化：親観点と子観点

- テスト観点は階層的に詳細化できる
  - 詳細化は、近くからモノを見る（ズームインする）ようなことである
  - 踏み込んでモデリングする場合は、継承（is-a）、合成集約（部分：has-a）、属性化、原因ー結果など詳細化すべき理由をステレオタイプで表す
    - » ステレオタイプごとに分けて詳細化を行うとMECE性（網羅性）を保証しやすくなる



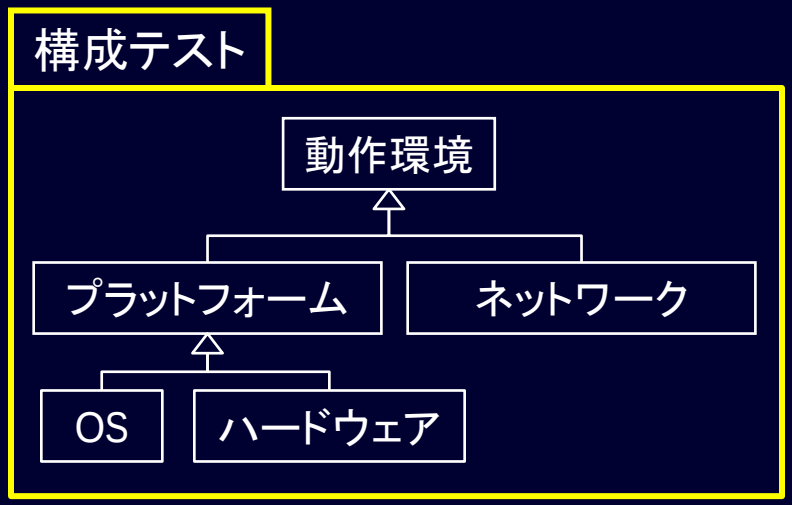
- 階層の最上位のテスト観点を「（トップ）ビュー」と呼ぶ
  - ビューは、そのサブツリーが全体として何をテストするのか、を表している
  - どのようなトップビューで構成するか（テスト要求モデルの全体像）はテストエンジニアによってかなり異なる
- 階層の最下位のテスト観点を「ボトムビューポイント」と呼ぶ
  - そのテスト観点を見れば、何をどのように網羅すればよいかが一目で分かる詳細度、がボトムビューポイントである
  - ボトムビューポイントはテスト詳細設計でのテスト条件となる
    - » 例えば同値クラスになる

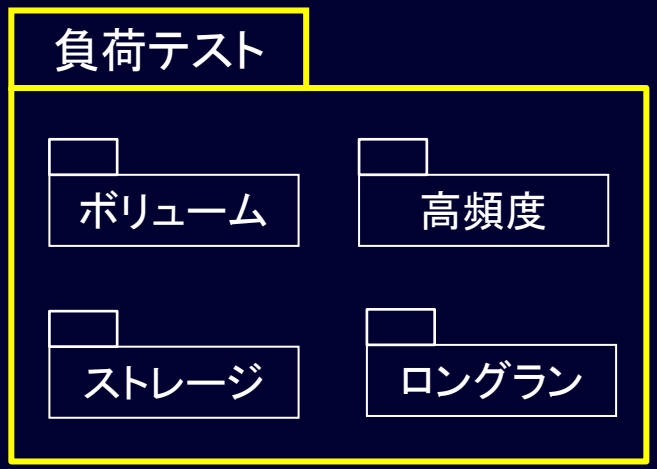
組み合わせテストが必要なら観点同士を関連で結ぶ

- 複数の観点を組み合わせるテストを「関連」で示す
  - 例）負荷テストでは、搭載メモリ量という観点と、
    投入データ量という観点を組み合わせてテスト設計を行う
    - » 搭載メモリ量と投入データ量のバランスによって、
      テスト対象のふるまいが変化するからである
  - 組み合わせテストの場合のように順序依存性のない関連は、
    テスト観点間の矢頭の無い曲線で表す
    - » 順序依存性のある関連は → のように開き矢尻の矢印で表す
  - 新たな関係や詳細な関係を定義したり分かりやすくするために
    <<stereotype>>（ステレオタイプ）を使ってもよい

テストコンテナとは

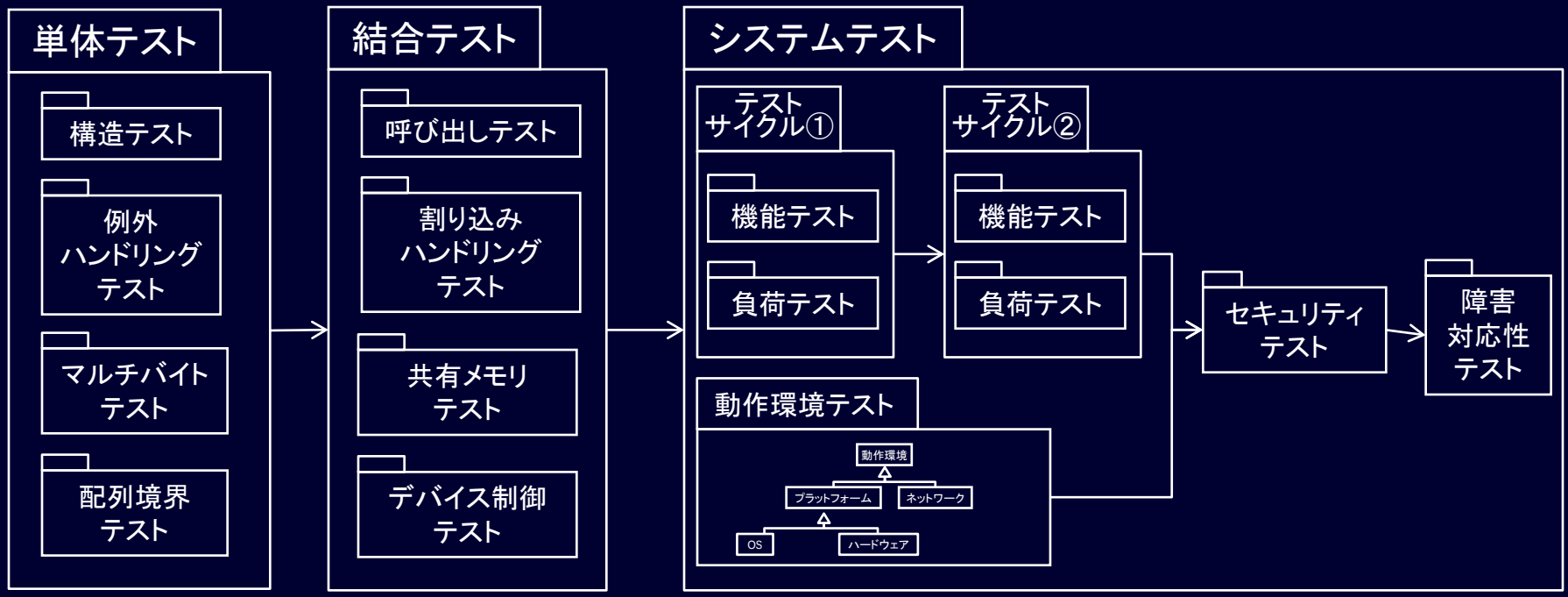
- 大規模なテスト設計では、
  複数のテスト観点をグルーピングして扱いたい時がある
  - 複数のテスト観点をグルーピングしたものをテストコンテナと呼ぶ
    - » テストタイプやテストレベル、テストサイクルなどを表現する
  - テストコンテナは包含関係を持つことができる
    - » テストコンテナに含まれるテスト観点を比べると
      テストコンテナ間の役割分担を明確に把握できる
      - 負荷テストと性能テストなど、違いがよく分からないテストタイプも区別できる
      - 単体テストと結合テストなど、役割分担がよく分からないテストレベルも区別できる





テストコンテナ図

- 大規模なテスト設計では、
  テストコンテナの図で表すと全体像を把握しやすくなる
  - テストコンテナ（レベルやテスト、サイクル）を用いるとテストアーキテクチャを俯瞰できる
    - » 先に設計/実施しておくべきコンテナを後に回してしまう、といったトラブルが防げる
    - » 保守や派生しやすいテストが可能になる



テストフレームとは

- 実際のテスト設計では、複数のテスト観点をまとめて
  テストケースを設計したい場合が多い
  - この条件のテストは、テスト対象のどの部分に行うんだろう？
  - この部分に対するテストでは、どの品質特性を確認するんだろう？
  - この品質特性をテストするには、どの操作を行えばいいんだろう？
- テストケースの構造（スケルトン）をテスト観点の組み合わせで示すことで、
  複数のテスト観点によるテストケースを設計する
  - テストケースの構造を表すテスト観点の組み合わせを「テストフレーム」と呼ぶ
    - <<frame>> というステレオタイプの関連を用いる
    - シンプルな構造のテストケースで十分であれば、テストフレームを組まなくてもよい
  - テストフレームの例：
    - 下の図では、テスト条件＋テスト対象＋振る舞いをテストフレームとしている
    - 組み合わせテストを設計しようとしているわけではない点に注意する

テスト詳細設計

- テスト観点をボトムビューポイントまで詳細化し、
  テストフレームを組んだら、テストケースを生成する
  - ボトムビューポイントに対応するテスト詳細設計モデルを定め、
    網羅アルゴリズムと網羅基準にしたがって
    具体的な値や機能、組み合わせを列挙し、テストケースとする
    - » 例）状態モデルに対して、C0の遷移網羅基準で、
      深さ優先探索法を用いて生成する
    - » テストフレームの各テスト観点を列見出しにしたExcelの表を埋めてもよいだろう
    - » ボトムビューポイント、網羅アルゴリズム、網羅基準の3者が明確であれば、
      モデルベースドテストとしてテストケース生成を自動化できる可能性が高い
  - テストケースに対応するテスト観点を常に明確にすることができる
    - » 目的のよく分からない漫然と設計されるテストケースを撲滅できる
  - テスト詳細設計は機械的に行っていくのが基本である
    - » 可能な限り自動化すると、Excelを埋める単純作業を撲滅できる
    - » テスト詳細設計を機械的にできないところは、網羅基準が曖昧だったり、
      テスト観点が隠れていたりする

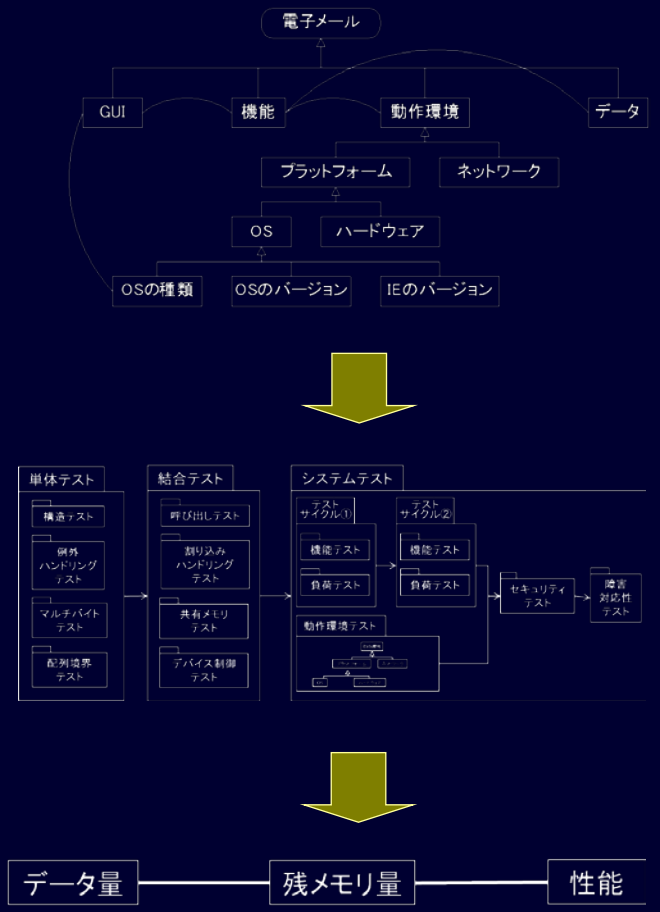
テスト実装

- テストケースが生成できたら、テスト実装を行う
  - テスト対象のシステムやソフトウェアの仕様、テストツールなどに合わせて
    テストケースをテストスクリプトに具体化していく
    - » 手動テストスクリプトの場合もあるし、自動化テストスクリプトの場合もある
  - テスト実装は、テスト対象のUI系の仕様や実行環境、ユーザのふるまいに関する
    ノウハウが必要である
- 集約
  - テスト実施を効率的に行うため、
    複数のテストケースを1つのテストスクリプトにまとめる作業を集約と呼ぶ
  - 同じ事前条件や同じテスト条件、同じテストトリガのテストケース、
    あるテストケースの実行結果が他のテストケースの事前条件やテスト条件に
    なっている場合は集約しやすい
    - » テストトリガとは、テスト条件を実行結果に変えるためのきっかけとなるイベントである
- キーワード駆動テスト
  - 自動化テストスクリプトを“クリック”といった自然言語の「キーワード」で
    記述することにより、保守性を高めたり自動生成をしやすくする技術である
  - テスト観点とキーワードを対応させると、キーワード駆動テストを実現しやすくなる
    - » NGT/VSTePをベースにして自動テストスクリプトの自動生成を目指すことができる

まとめ：かんたんVSTeP

- テスト要求分析
  - テスト観点を思い浮かべて、線でつなげよう
- ・テストアーキテクチャ設計（コンテナ設計）
  - テストコンテナにまとめて並べよう
- テストアーキテクチャ設計（フレーム設計）
  - テストケースのひな形（テストフレーム）を作ろう
- テスト詳細設計
  - テストケースのひな形に具体的な値を入れよう
    - » 普通のやり方と同じでよい
- テスト実装
  - テストケースをテスト手順に具体化しよう
    - » 普通のやり方と同じでよい

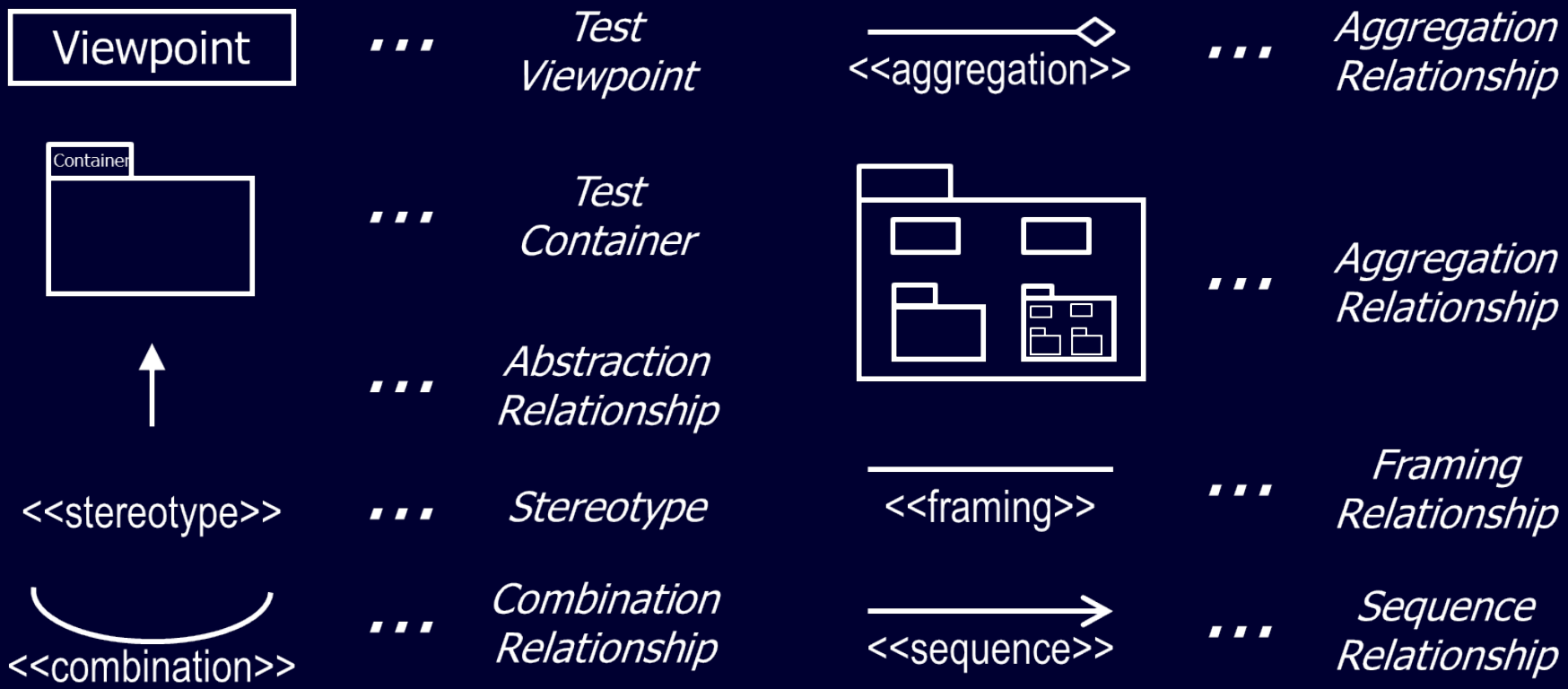
ざっくり考えて図にすると
見やすいし整理しやすい！



講演の流れ

- VSTePに取り組む価値のない組織・ある組織
- かんたんVSTeP
- 一歩踏み込んだVSTeP
- VSTePの応用と意義
- まとめ

NGT: 記法のまとめ



VSTePにおけるモデリングの基本

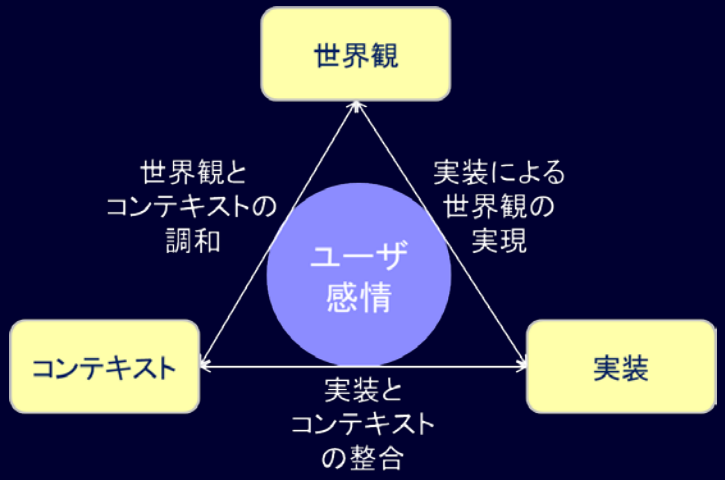
モデリングに正解は無い、
納得の共感による
説明責任の確保があるだけだ

VSTePにおけるモデリングの基本

- モデリングに正解は無い、
  納得の共感による説明責任の確保があるだけだ
  - 納得感の得られないモデルや、共感できないモデルは、往々にして質が低い
  - その組織のエンジニアが皆「こんなもんだ」と思い、
    それで過去に問題がなかったのなら、モデリングはその程度で終わりでよい
  - 仕様書とのトレーサビリティにばかり囚われるとよいモデルにならない
- 各（中間）成果物の抽象度をどこまで落とすべきか、
  は意外に難しい問題である
  - テスト観点、テストコンテナ、テストフレーム、テストケース、テスト手順を
    それぞれどこまで具体的に書けばいいのか、は一概には言えない
- 類似した構造を見抜き、テストデザインパターンとして
  パターン化して蓄積することが重要である
  - VSTePは抽象化されているのでパターン化しやすいが、
    パターンを掘り出そうという意志を常に持つこと一番重要である
- 複数の派生製品のテスト開発をする際は、派生製品全体をまたいで
  （テストの）プロダクトラインモデルを構築するとよい

テスト要求モデルの全体像の種類

- 仕様ベースモデル
- 機能ベースモデル
- 画面ベースモデル
- 正常系異常系モデル
- 品質特性モデル
- 一般テストタイプモデル
- CIBFWモデル
  - Condition / Item / Behaviour / Fault / World
- 三銃士モデル
  - 世界観・コンテキスト・実装＋ユーザ感情

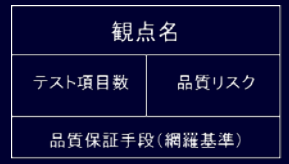


テスト観点モデルのテクニック

- 仕様書を書き写すことで網羅性を高めようとしない
  - 仕様書に書いてないテスト観点をテストでも漏らしてしまうことになる
    - » 仕様書がそんなに質や完成度が高かったら苦労しない
- 同じテスト観点名を皆が同じように理解しているとは限らない
  - 子観点が明示すると理解の齟齬は解消しやすい
- 観点が通り一遍で関連が多いモデルは理解度が低いことが多い
  - 不安だから／責任を取りたくないから組み合わせておかなきゃ、と発想していることが多い
  - なぜその関連をテストすべきだと思ったのか、という掘り下げを行い、可能なら関連をテスト観点に変換すると、よりテストの意図が明確になる
- 詳細化のステレオタイプを明示すると網羅しやすい
  - 異なるステレオタイプの詳細化が混ざっていると網羅しにくい
- 抽象度は丁寧に落として（具体化して）いく
  - 認知的マジックナンバーの範囲内にできるとよい
- 網羅できた確信が持てないところにはノートを貼っておく
  - 抜け落ちの危険性が消失することが一番怖い
- チャーターの記述とテストの（気づいた点の）記録はテスト観点モデルを用いると表現しやすい

VSTePにおける網羅の考え方

- 基本的な考え方：まず全体像を網羅して、次に明示的に減らす
  - 網羅とは証明することではなく、納得感を共感することである
    - » ソフトウェアのテストは、現実的には無限or工数オーバーになるので網羅はできない
    - » しかしまず全体像を網羅しておいて、何がどのように無限or工数オーバーになるので
      こういう考え方でここを減らす、ということを整理して明示的に把握しておけば
      現実的に問題の無いテストが可能になる
    - » テスト観点を用いて抽象化しているので、網羅する工数はそんなに増えない
    - » テスト漏れが怖いのではなく、漏れがどこにあるのか分からないのが怖いのである
- VSTePにおける網羅は以下の4つから構成される
  - テスト観点とそれらの組み合わせの網羅 – テスト観点図の納得感の共感で担保する
    - » 納得感を上げるためにテスト要求モデル（テスト観点図）のリファインを繰り返すことが肝要
  - テストケースの網羅 – テストフレームの明示で担保する
    - » 網羅を確実に行うためにモデルベースドテスト（テスト詳細設計の自動化）に取り組むとよい
  - テスト手順の確実な遂行 – テストケースとテスト手順の分離によって促進する
- VSTePにおける網羅は以下の考え方で行われる
  - 全網羅 – 規模が非常に小さいときのみ可能 -「確定」で網羅性を担保する
  - トレードオフ –「剪定」やペアワイズ/直交配列表などで行う
  - ピンポイント – バグのパターン化（ex. 不具合モード）など別途技術が必要
  - リスクベースドによる優先度付け - 今回の講演では意図的に省略している



テスト要求モデルのリファインの例

- 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

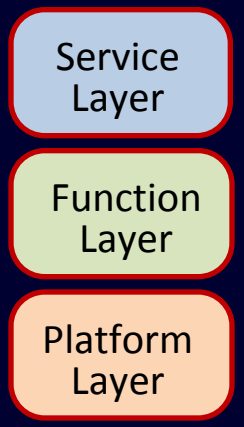
テストコンテナモデリングのテクニック

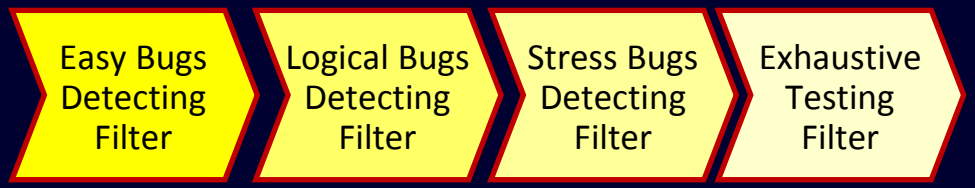
- テストアーキテクチャ設計方針／テストアーキテクチャの品質特性を明示的に考慮し、複数案を作れるとよい
  - テストアーキテクチャ設計のどの部分を取っても、そこがそうなっている理由や意図を説明できるようになっていなければならない
  - 複数案を作ると、コンテナモデリングの重要性が腹落ちできる
- コンテナの前後依存関係は最も基本的なテストアーキテクチャである
  - （リスクベースドテストの）リスクは、後先だけでなくコンテナごとの規模にも影響するので、リスクだけでコンテナの前後依存関係を考えるのはよくない
- テスト設計方針／テストアーキテクチャの（内部）品質特性の例
  - 結合度と凝集度、保守性、自動化容易性、環境依存性、開発との一貫性、想定欠陥修正容易性、記述容易性、設計方向性、リズムなど色々ある
- トップダウンのモデリングとボトムアップのモデリング
  - 必ず一度はボトムアップでコンテナモデルを構築してみること
  - 探索的テストは観点が動的に決まる／変わるため、コンテナとして出し忘れやすいので気をつける

テストアーキテクチャの例

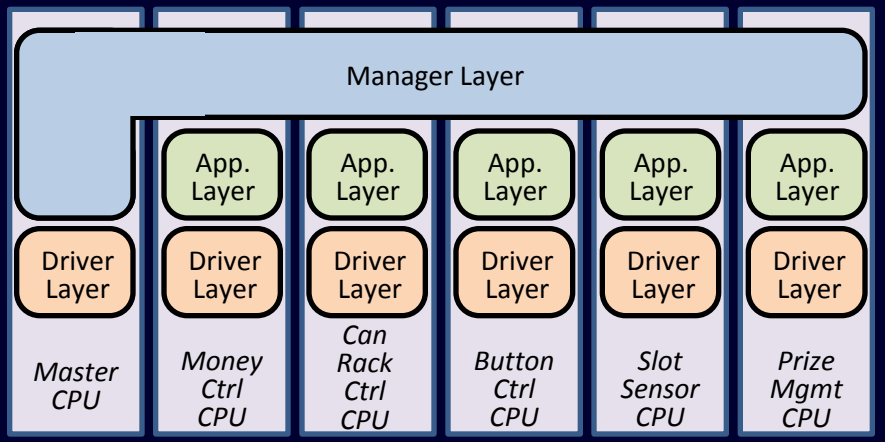
- テスト設計コンテストの応募作を分析すると、様々なテストアーキテクチャが構築されていた
  - テスト対象構造型テストアーキテクチャ
  - 伝統的テストタイプ型テストアーキテクチャ
  - 階層型テストアーキテクチャ
  - フィルタ型テストアーキテクチャ
  - 複合型テストアーキテクチャ

階層型テストアーキテクチャ





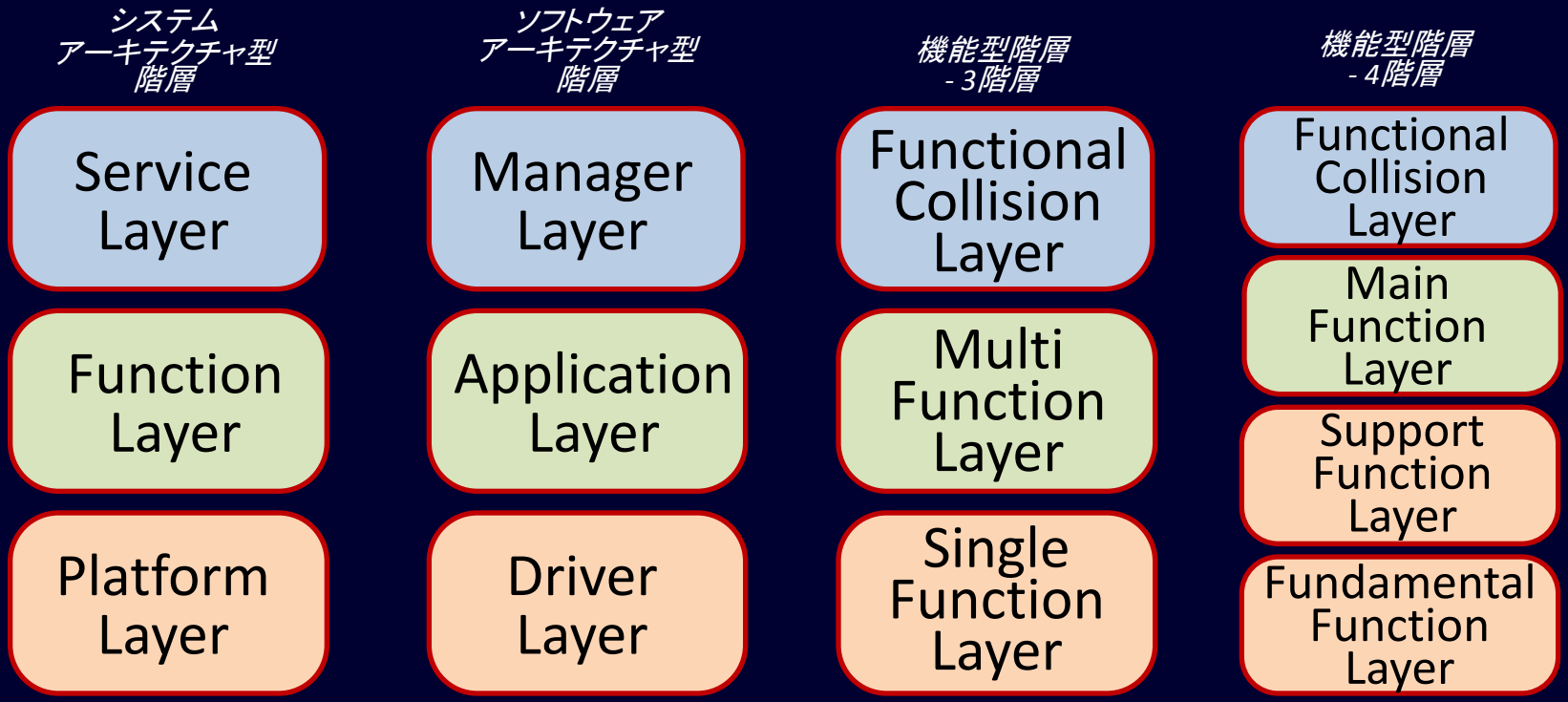
フィルタ型テストアーキテクチャ

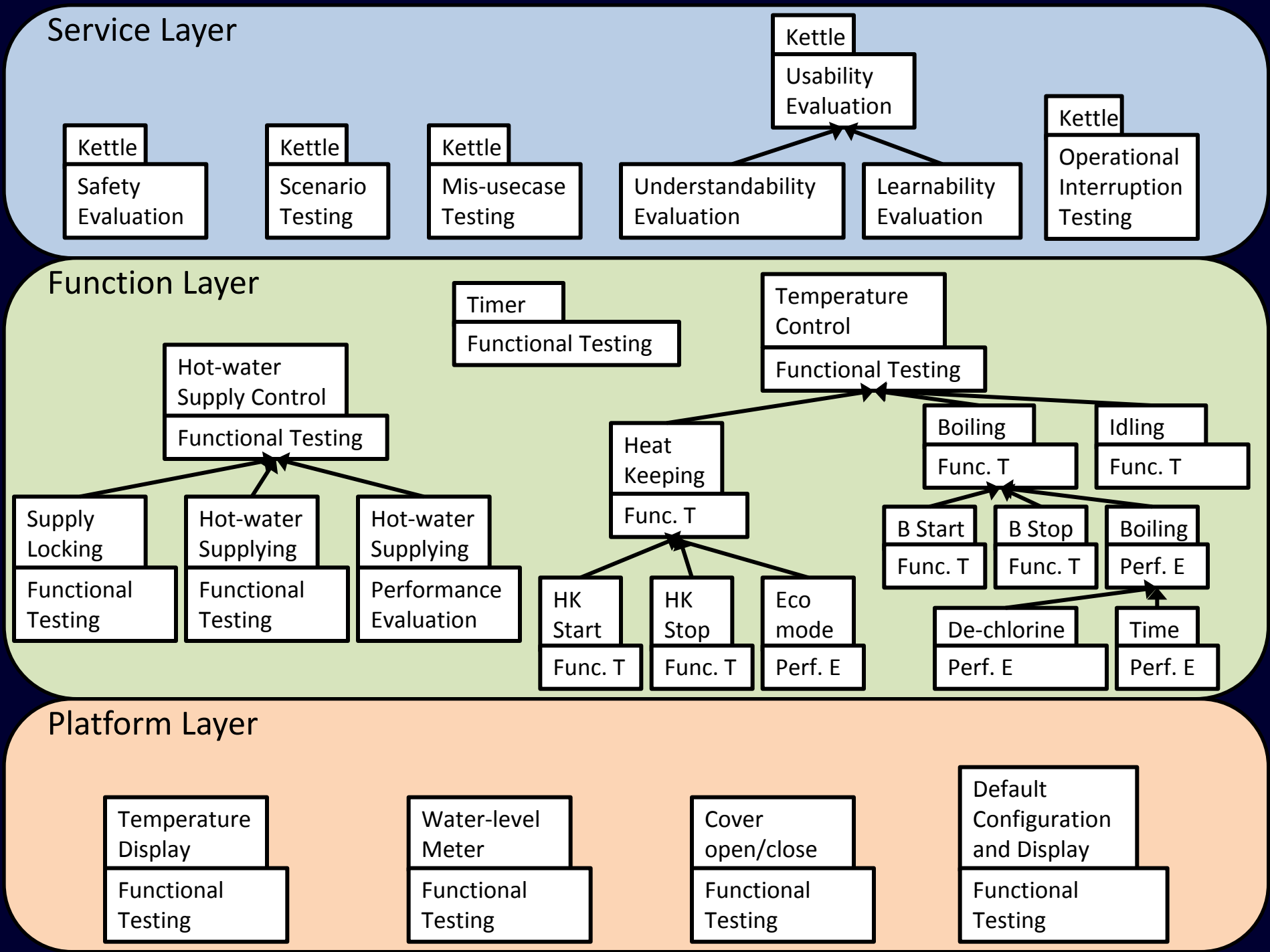


複合型テストアーキテクチャ

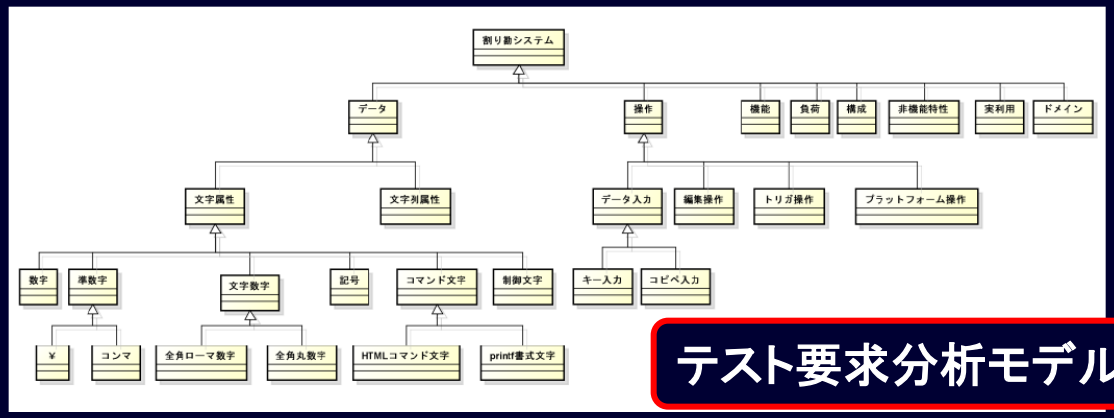
テスト設計コンテストの応募作のテストアーキテクチャ

- 一見似たようなレイヤードアーキテクチャだが、実は異なっていた
  - 階層の切り口もレイヤ数も様々だった

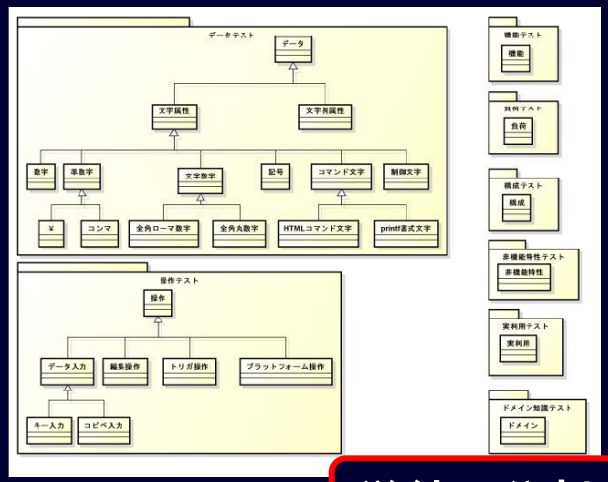



Service Layer
Kettle
Safety Evaluation
Kettle
Scenario Testing
Kettle
Mis-usecase Testing
Kettle
Usability Evaluation
Understandability Evaluation
Learnability Evaluation
Kettle
Operational Interruption Testing
Function Layer
Timer
Functional Testing
Hot-water Supply Control
Functional Testing
Supply Locking
Functional Testing
Hot-water Supplying
Functional Testing
Hot-water Supplying
Performance Evaluation
Temperature Control
Functional Testing
Heat Keeping
Func. T
HK Start
Func. T
HK Stop
Func. T
Eco mode
Perf. E
Boiling
Func. T
Idling
Func. T
B Start
Func. T
B Stop
Func. T
Boiling
Perf. E
De-chlorine
Perf. E
Time
Perf. E
Platform Layer
Temperature Display
Functional Testing
Water-level Meter
Functional Testing
Cover open/close
Functional Testing
Default Configuration and Display
Functional Testing

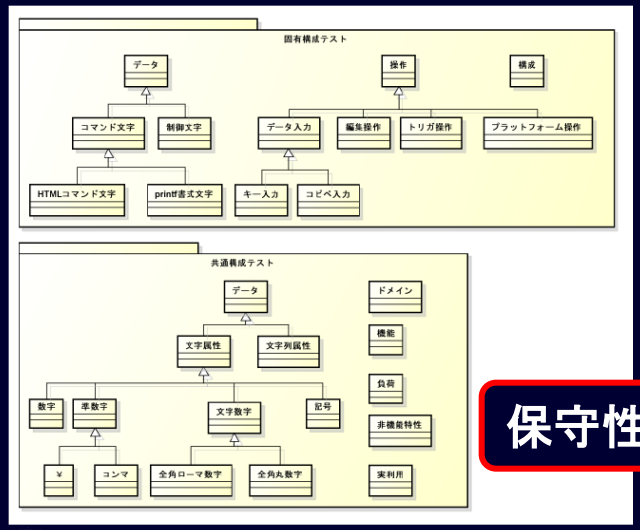
設計方針／(内部)品質特性によって異なるテストアーキテクチャの例



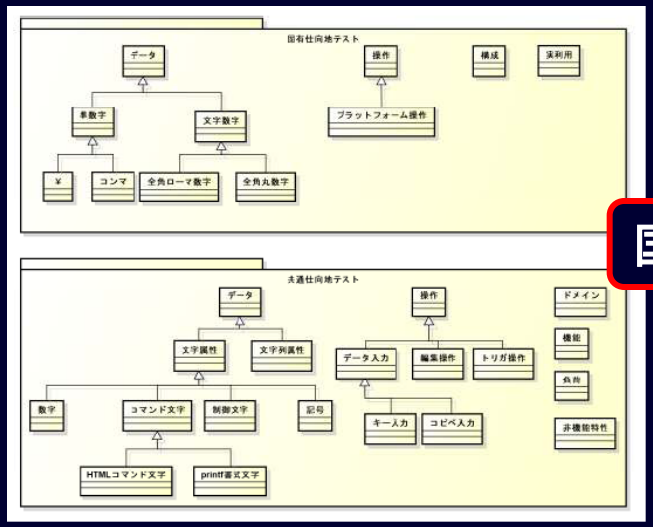
テスト要求分析モデル



単純に分割



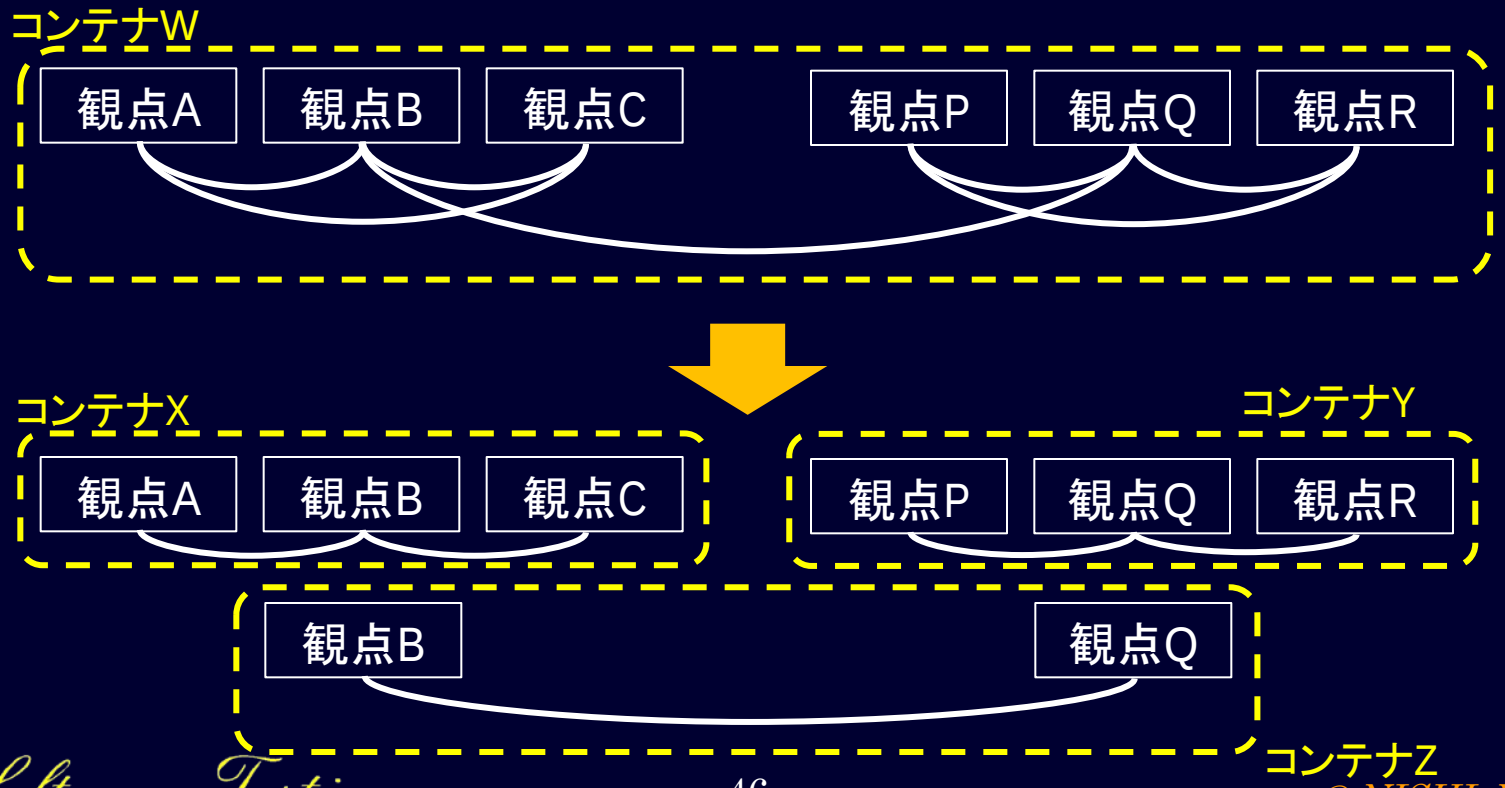
保守性重視



国際化重視

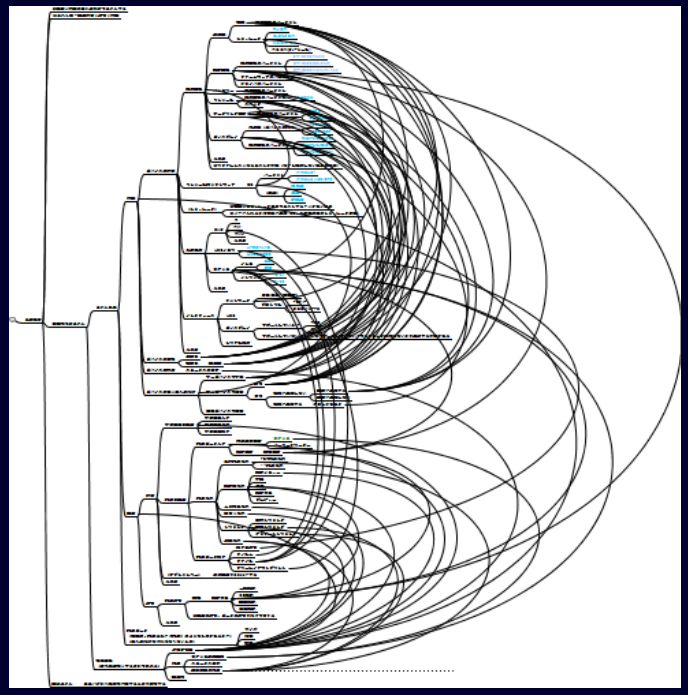
テストアーキテクチャレベルのテストデザインパターン

- 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

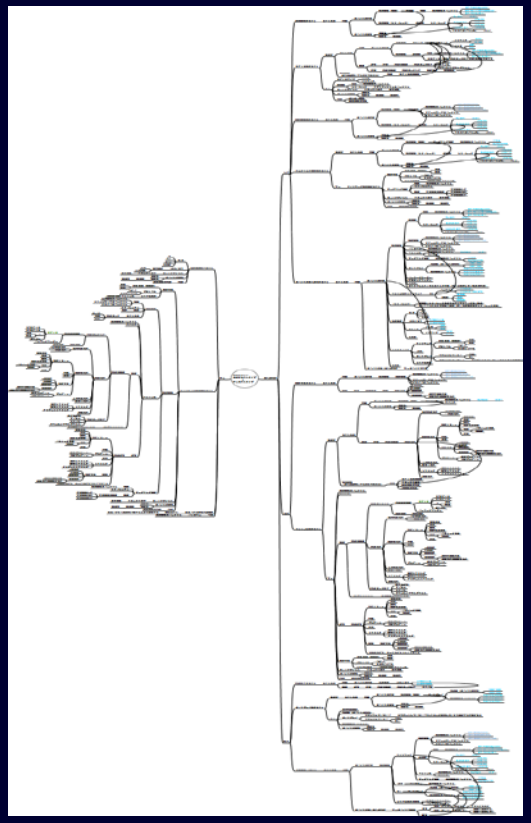


パターンによるテストアーキテクチャのリファインの例

- 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
  - 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

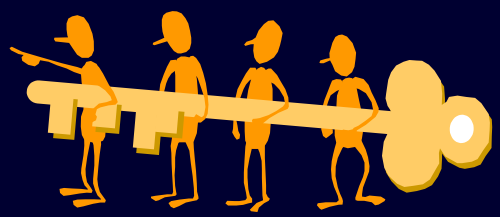


リファイン



ウォーターフォールプロセス？反復型プロセス？

- VSTePはウォーターフォール型の開発やテストが向いているのか？
  - VSTePに限らず体系立ててプロセスの説明をすると
    説明の順がウォータフォールになりがちなので、
    プロセスの中身もウォーターフォール限定ではないかと勘違いされることがある
  - テスト対象の要求やテストプロジェクトの制約、メンバのスキルや理解度によって、
    ウォーターフォール的が適しているか反復的マネジメントが適しているかは異なる
    - » 要求信頼性
    - » 耐監査度
    - » 開発全体が反復的かどうか
    - » テストオペレータのテスト対象の理解度
    - » ソフトウェア設計やコードの質
    - » など
  - VSTePを反復型開発に適用する際にはコツがある



反復開発におけるVSTePのコツ

- イテレーションごとに
  テスト要求モデルとテストアーキテクチャモデルを改善していきながら、
  イテレーションナビゲーションを行う
  - イテレーションごとに何をテストするのかを、テストコンテナ図を使ってナビゲーションしていく
    - » 保証できたテスト観点の種類と程度によって次に何をするかを動的に決める
    - » コンテナに色をつけていくと直感的になる
  - 最初は曖昧なコンテナ図になっても構わないが、進むに従って地図の精度を上げていく
  - 後半は横串を通すようなテストが増えてくるため、イテレーションごとの開発規模を減らすか、テストチームによるテスト用のイテレーションをつくるか、などの選択が必要になるので、コンテナ図を用いるなどして早めに予想して事前に仮判断しておく
- 1回のイテレーションコンテナに、スクリプトテストと探索的テストを両方入れる
  - スクリプトテストの深掘り的な探索ばかりすると仕様の後追いになりがちでよくない
  - イテレーションの時期に応じてスクリプトテストと探索的テストのバランスを変える
    - » バグの状況が把握しやすく余裕がある時は“スクリプト重→探索重→スクリプト重”のパターンで、把握しにくく余裕が無い時は“探索重→スクリプト重→探索重”のパターンになる
- 1回のイテレーションで必ずテスト観点リフレクションの時間を取る
  - リフレクションによって観点図やテストコンテナ図を書き換えて（書き加えて）いく
  - モデルの質を高く保っておかないとすぐに何をテストしているのか分からなくなる

VSTeP導入のコツ

- 組織文化を変えようとせずにVSTePを導入しても上手くいかない場合がある
  - 何かに従っていれば仕事をしたことになる文化
  - 頭を使わない文化
    - 「テスト開発方法論 ＝ 概念／記法＋プロセス＋モデリング技術」なので、記法とプロセスを文書化して整備しても、モデリング技術を鍛えないと使いこなせない
- 導入推進者はVSTePのTIPSだけではなく、VSTePを適用している現場の悩みの理解も必要である
  - 導入そのものの改善をマネジメントする必要がある
- プロセス改革的なトップダウンのアプローチよりも、改善型のボトムアップのアプローチの方が、時間がかかるが上手くいくことが多い
  - リーンスタートアップで現状の問題点や改善の成果を実感してもらいながら進める方がよい
    - Reverse VSTeP
  - リバース→フォワード→リバース…という導入ができるとよい
- テストの改善だけで終わらせず、レビューの改善や開発の改善まで長期的視野に入れる方が根付きやすい

Reverse VSTePによるテストの改善

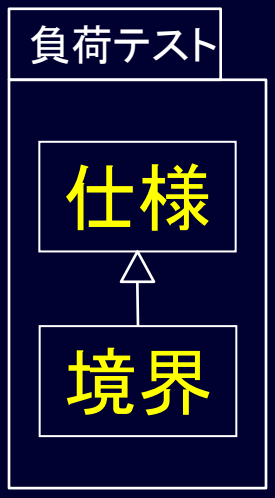
- Reverse VSTeP: テスト設計をリバースモデリングする手法
  - 要求からテストスクリプトを作成していく流れをForward VSTeP、逆をReverse VSTePと呼ぶ
  - Reverse VSTePとは、既存の（十分設計されていない）テストケース群から
    テスト観点モデルをリバースモデリングして改善する手法群である
    - » 実際のテストケースを分析して、
      どういうテスト観点や関連でテストしようとしているかを列挙・整理し改善する
    - » 実際に見逃したバグや検出されたバグ、テストケース外で検出されたバグを
      分析して、どうい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
  - リバースしないVSTePをForward VSTePと呼ぶこともある
- 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

Reverse VSTePによるテストの改善

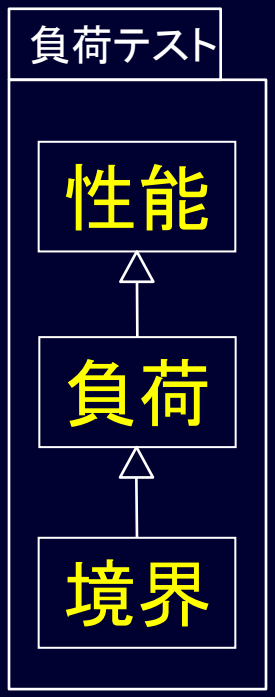
- ステレオタイプ分析による改善
  - テスト観点モデルの各詳細化関係の種類やMECE性を分析し、
    テスト観点が適切かつ網羅的に詳細化されるように
    子テスト観点を追加したりモデルをリファクタリングして改善する
- 不具合モード分析による改善
  - 見逃したバグや検出されたバグをパターン化して、
    ピンポイント型のテスト観点を追加・整理し改善する
- ディレイ分析によるテストアーキテクチャの改善
  - 見逃したバグや検出されたバグを分析して、
    実際のテストレベルやテストタイプを
    リバースされたテスト観点モデルにマッピングし、
    テストアーキテクチャ設計（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

テスト観点のリバースは技術力把握にも有効

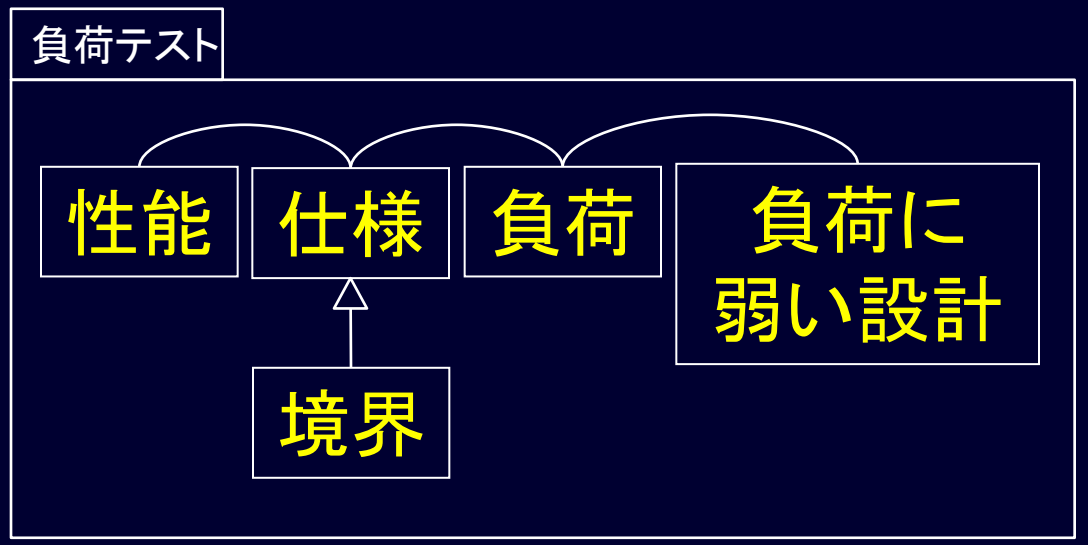
- テスト観点をリバースするとテスト設計者の意図が透けて見えてしまうので、テスト設計者の技術力が分かってしまう
  - この3つのモデルは、同じ組織・同じ経験・同じテストスキルを持っていると主張する3名のエンジニアの負荷テストの設計モデルである



様々な負荷のかけ方が漏れる



負荷に弱い設計の性能低下が漏れる



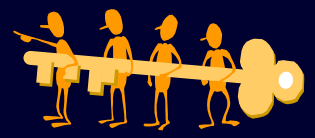
技術力が高い

講演の流れ

- VSTePに取り組む価値のない組織・ある組織
- かんたんVSTeP
- 一歩踏み込んだVSTeP
- VSTePの応用と意義
- まとめ

VSTePの応用

- VKDT：VSTeP ＋ KDT（キーワード駆動テスト）
  - VSTePをベースにテストの自動化を行うと、
    テストケースの自動生成からKDTを通じて自動実行までシームレスにつながるので、
    （さほど大きくない）仕様変更や設計変更に対する確認テストをターンキーにできる
- レビュー観点や開発考慮事項へのフロントローディング（VSPI／Wモデル）
  - レビューの改善というと会議術の話ばかりで、レビューの中身の質につながらない
    - » 一回一回のレビューがやりっ放しになっていて、知恵の蓄積がされていない
    - » 参加者が思いつきを言うだけの会になってしまったりするので、ちっともまとまらない
    - » 全体としてどんなレビュー（群）をすべきなのか、というレビュー（群）の全体像を誰も掴んでいない
  - VSTePを応用して、レビュー観点をベースにレビューケースの設計や
    レビューアーキテクチャの可視化を行うことができる
  - さらにフロントローディングすると、開発で考慮すべき事項をモデリングできるようになる
    - » Wモデルの一形態である
- グローバル開発におけるテストの組織化
  - ソフトウェア開発のグローバル化は進んでいるが、テストやQAはこれからという組織が多い
  - VSTePをベースにすると、日本HQ、海外開発センター、
    現地ソフトハウスの役割分担の見通しがよくなる

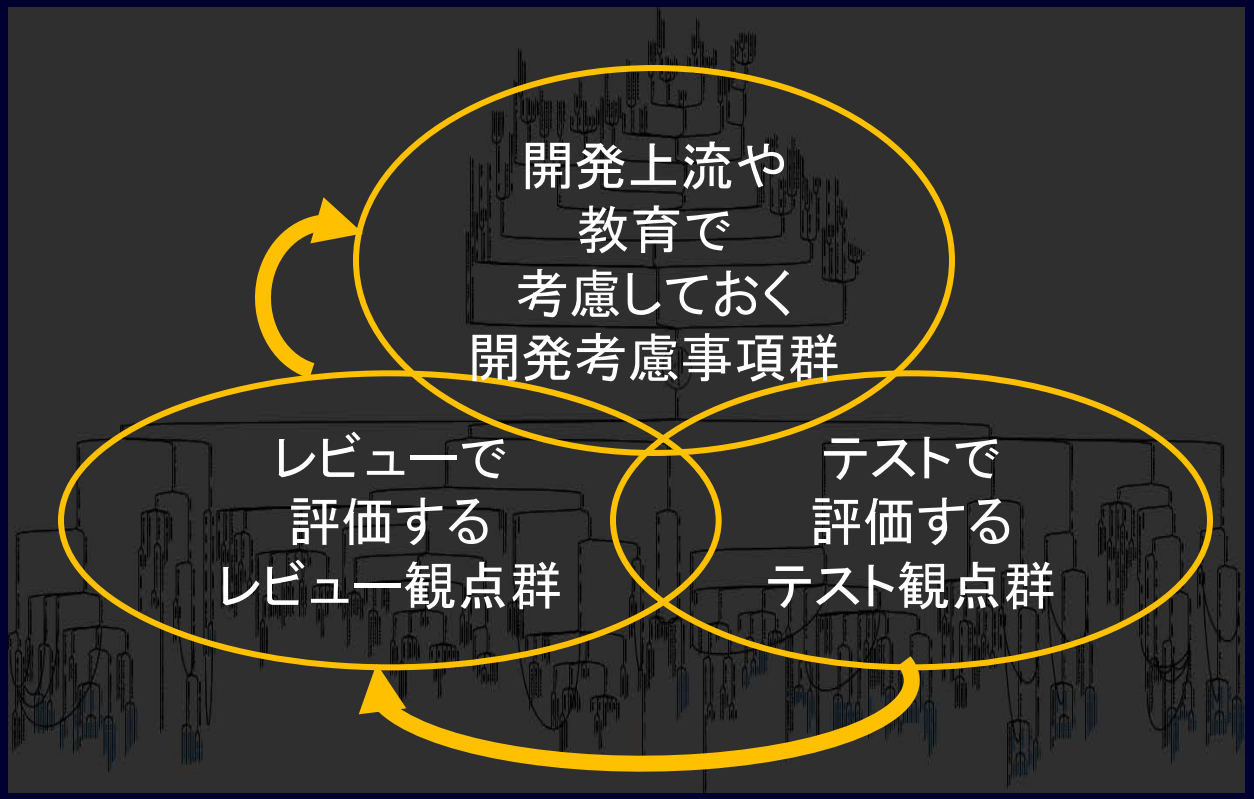


VSTePによるテストの自動化：VKDT(KDT+MBT)

- テストの自動化は、人間が頭を使わないといけない作業を明確にしてくれる
  - こうした作業を明確にしないと、どこまで網羅できたのかを説明しにくくなる
  - 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）世代
  - 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）世代
  - 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

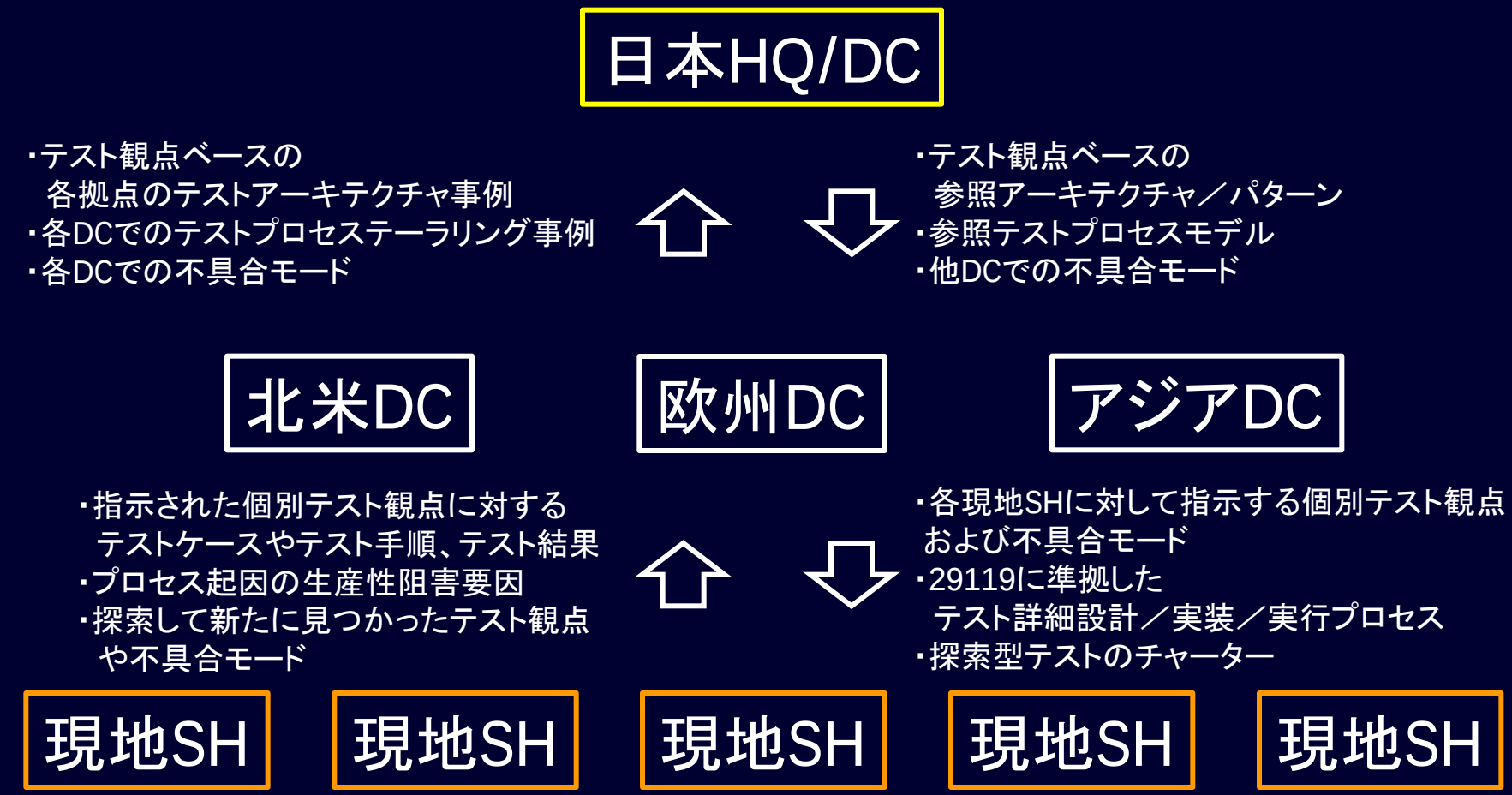
- キーワード駆動テスト（KDT）世代
  - 単純な操作（キーワード）をスクリプトライブラリと関連づけておくと、テストケースをキーワード名で書くだけで、自動化テストスクリプトを意識しなくてもツールがテストケースのバリエーションを自動で作成し実行してくれる
    - » 操作レベルの詳細なテストケースが必要なため、仕様変更にすぐに追従できない
- KDT+MBT（モデルベースドテスト）世代：VKDT
  - テストケースの自動生成からKDTを通じて自動実行までシームレスにつなげることで、（さほど大きくない）仕様変更や設計変更に対する確認テストがターンキーになる
  - 自動化すべきところとすべきでないところ、しやすいところとしにくいところの弁別が重要
  - 自動化を前提とした開発上流の仕様調整やモデル化とテストプロセスの整備も重要

テストの知見を活かして開発全体を改善する：VSPI



上流へのテスト観点の
フロントローディングによって改善を行う

グローバル開発におけるテストの組織化



グローバル開発におけるテストの組織化

日本HQ/DC

・テスト観点ベースの
各拠点のテストアーキテクチャ事例
・各DCでのテストプロセステーラリング事例
・各DCでの不具合モード

・テスト観点ベースの
参照アーキテクチャ／パターン
・参照テストプロセスモデル
・他DCでの不具合モード

北米DC　欧州DC　アジアDC

・指示された個別テスト観点に対する
テストケースやテストデータの充足度
・プロセス起因の生産性阻害要因
・29119に準拠した
テスト詳細設計・実行
・探索型テストのチャーター

拠点の責務と
取り扱う設計情報・プロセス情報の粒度を
適切に設計することで、
ノウハウを掘り出して活きた標準に落とし込み、
組織全体で改善サイクルを回すことができる

現地SH　現地SH　現地SH　現地SH　現地SH

VSTePの意義

- 丸や四角と線で絵を描くとエンジニアのように見える
  - エンジニア魂を刺激して
    「自分は軽作業派遣ではなくクリエィティブなエンジニアなんだ」
    と自覚してもらいモチベーションを上げてもらえる
- テスト条件（や期待結果）を抽象化することで、
  全体像をコミュニケーションしやすくなったり知恵を蓄積しやすくなる
  - プロジェクト個別の情報を削除して、知恵づくりに必要な情報だけを残すことができる
  - 全体像を把握しやすいように可視化されるのでコミュニケーションしやすくなる
  - 知恵の質がモデルによって比較可能になる
  - ソフトウェアエンジニアリングの様々な技術を応用しやすくなる
  - テストだけでなく、レビューや開発上流も含めた開発考慮事項を蓄積できるようになる
- 頭を使うところと手を動かせばいいところを明確に分離できる
  - 違う頭を使うところは異なる工程になっている
  - 頭を使うことを鍛えるという意識が無いと、テストがよくならない
    - » 銀の弾丸を求めている組織には向いていない
  - 単純作業は自動化することに頭を使うようにする

VSTePの意義

- 現場でのリーンスタートアップがしやすくなっている
  - 段階的／改善的導入や一部導入がしやすい
  - 自組織がいま使っているフォーマットをベースにできる
  - ダメなテスト設計をしている組織は
    当初ダメなモデルが可視化されてやる気を失うかもしれない
    - » VSTePは万能の道具ではないので、頭を使わずにダメなテスト設計を改善することはできない
- VSTePはメソドロジ・プラットフォームである
  - 自社のテストのやり方を尊重したまま、VSTePで問題点を整理・改善することができる
    - » ・例）改良型ゆもつよメソッド powered by VSTeP
  - 様々なプロジェクトのテストのバリエーションを統一的に整理することができる
    - » 自社の各部門や開発子会社、海外拠点などを統一的にマネジメントできるようになる
  - 知恵を貯めてVSTePを自社内で成長させ、
    自分たちの身体に馴染む「自社版VSTeP」に仕立て上げることが重要である

VSTePは何でないか

- VSTePは頭を使わずにテストできる魔法の道具ではない
  - 「この表を埋めればテストケースが完成です！」とか
    「とにかく探索的テストをすればバグが見つかります」みたいなまやかしではない
  - 「あなたの組織で誰も想定できなかった想定外を想定できます」みたいな詐欺ではない
- VSTePはエンジニアを孤立させるものではない
  - ひたすら一人でフォーマットを埋めるようなぼっちめしではない
- VSTePは思考停止してテストできるプロセスマニュアルではない
  - 分厚い手順書とフォーマットのような無能コンサルの納品物ではない
  - 膨大な文書群とトレーサビリティを必要とするプロセススパゲッティではない
- VSTePはエンジニアを奴隷にするものではない
  - 意図の分からないテストケースをひたすら書いて実行して
    誰も見ないスクリーンショットを積み上げる賽の河原ではない
  - 役に立っている実感の無い施策をやらされ感満載で強制される
    経営者の自己満足の道具ではない
- VSTePは全部入りではない
  - バグのパターン化や各種テスト技法などは別途アドオンしていく必要がある

まとめ

VSTePは
エンジニアがクリエィティブな作業に集中し
納得感を共感することで
よりよいテストを継続的に行っていくための
道具立てである

ではワークショップで手を動かしてみましょう

電気通信大学　西 康晴
Yasuharu.Nishi@uec.ac.jp
@YasuharuNishi

付録：VSTePによる厳密なテスト要求モデルの構築

1) 要求の源泉の準備：
   - 入手できるなら、要求の源泉を準備する
2) 要求の獲得と分割
   - 要求の源泉から要求を獲得する
   - テストケースを導くエンジニアリング的要求と
     テストケースを導かないマネジメント的要求に分ける
3) モデルの構築と納得
   - テスト観点図を用いてテスト要求モデルを構築し、
     自分たちとステークホルダーが納得するまでリファインする

   3-1) 要求を充実化する
   - » 全観点網羅モデルを構築する

   3-2) 要求を実質化する
   - » 実施可能観点モデルを構築する

注）厳密とは、信頼性要求が高いのでテストをたくさんやらなきゃいけないという意味では無く、
テスト漏れを可能な限り減らすために込み入ったことをやるという意味である

付録: 1) 要求の源泉の準備

- 入手できるなら、要求の源泉を準備する
  - 要求の源泉の例）
    - » 動いているテスト対象
    - » 要求系文書、開発系文書、ユーザ系文書、マネジメント系文書など各種文書
    - » テストに関して従うべき社内・社外の標準や規格
    - » ヒアリングできるステークホルダー
    - » など
  - 一つも入手できないなら、自分たちでステークホルダーになりきって
    ブレーンストーミングや仮想ヒアリングなどを行う
    - » あくまで入手できないなら仕方が無い、という意味合いであり、
      要求の源泉が無くても大丈夫、という意味ではないことに注意する

付録: 2) 要求の獲得と分割

- 要求の源泉から要求を獲得する
- テストケースを導くエンジニアリング的要求と
  テストケースを導かないマネジメント的要求に分ける
  - エンジニアリング的要求:システムの完成像とテスト対象の途中経過
    - » システムの完成像への要求の例)
      - システム要求、ソフトウェア要求
      - 機能要求、非機能要求、理想的な使い方、差別化要因、目的機能
    - » テスト対象の途中経過に関する情報の例)
      - 良さに関する知識:テスト対象のアーキテクチャや詳細設計、実装、自信があるところ
      - 悪さに関する知識:バグが多そうなところ
        - 構造上問題が起きそうなところ
        - 前工程までの検証作業(レビューやテスト)が足りなかったり滞ったところ
        - 類似製品や母体系製品の過去バグ、顧客クレームから分析した知識
        - スキルの足りないエンジニアが担当したところ、設計中に不安が感じられたところ
        - 進捗が滞ったりエンジニアが大きく入れ替わったりしたところ
  - マネジメント的要求
    - » 工数、人数、スキル分布、作業場所、オフショアか否か、契約形態など
    - » 機材利用可否(シミュレータや試作機など)、
      ツール利用可否(ツールの種類とライセンス、保有スキル)など
    - » 目標残存バグ数、信頼度成長曲線など
    - » テストスイートの派生可能性や保守性など

付録：3) モデルの構築と納得

3) モデルの構築と納得

- テスト観点図を用いてテスト要求モデルを構築し、
  自分たちとステークホルダーが納得するまでリファインする

3-1) 要求を充実化する：全観点網羅モデルを構築する

- エンジニアリング的要求を基に、テスト観点を列挙していく
- エンジニアリング的要求を基に、観点を詳細化したり関連を追加していく
- 充実系リファインを行う
  - » MECE分析・ステレオタイプ分析による網羅化
  - » 整理
  - » 確定
- 自分たちで十分に充実したと実感できたら、そのテスト観点図を囲んで、
  ステークホルダーが納得するまで一緒にリファインする
  - » ステークホルダーに不十分・不完全・不正確な把握が無いようにする
  - » ステークホルダーに、本当はこれだけテストしなきゃいけないんだ、
    という実感を持ってもらうことが重要である

付録：3) モデルの構築と納得

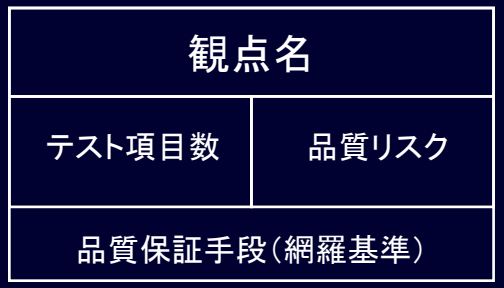
3-2) 要求を実質化する：実施可能観点モデルを構築する

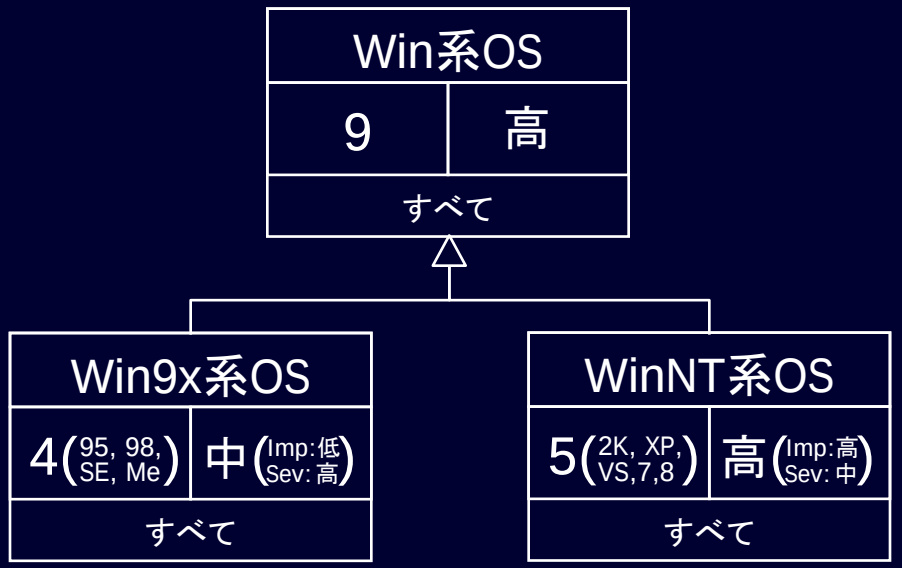
- マネジメント的要求を基に、テスト観点図に品質リスクの情報を加えていく
- 要求および過去の経験から、テスト観点図にテスト項目数の情報を加えていく
- 実質化系リファインを行う
  - » 剪定
    - 剪定したら（根拠が無いことも含めて）根拠を記録しておく
  - » 確定
- 自分たちで要求が達成でき品質リスクが受け入れられる
  テスト観点図ができたと実感できたら、そのテスト観点図を囲んで、
  ステークホルダーが納得するまで一緒にリファインする
  - » 剪定前と剪定後を根拠つきで比較し、確定の結果と合わせて検討することで、
    ステークホルダー自身が品質リスクを受け入れたと認識することが重要である
  - » あくまで顧客の納得を深めるのがテスト要求モデリングのリファインの
    主目的であり、その後の工程でテストしやすくするのが目的ではない
  - » リファインした結果、総テスト項目数から予想されるテスト工数が不足すると
    見積もることができる場合、自動化やテストプロジェクト編成見直し、工数追加、
    品質目標修正など、テスト要求（およびその基となる製品要求・プロジェクト要求）を
    見直さなければならない場合もある

3-3) 見逃しリスクを確定する

付録：項目数と品質リスクの概算

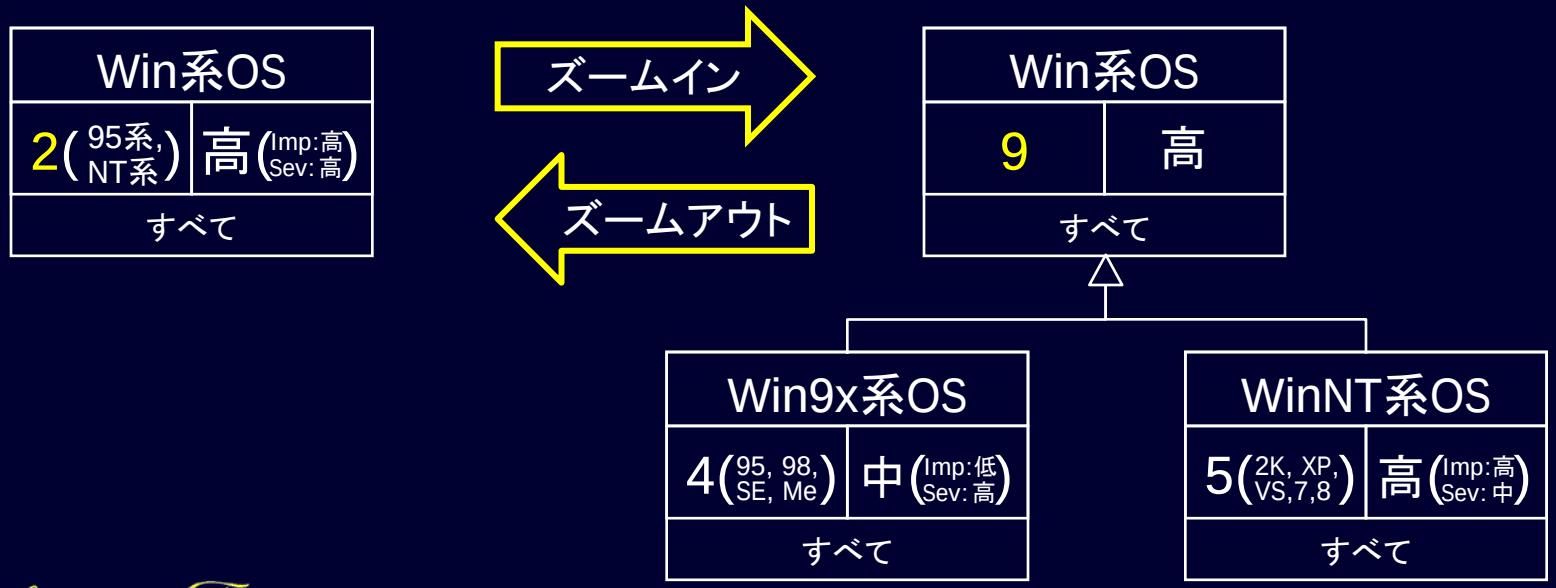
- モデリングが進んできたら、テスト観点ごとに
  テスト項目数と品質リスクの概算を行う
  - モデリングの進み方やテストプロセスの成熟度などに
    応じて、どちらかだけでもよい
  - テスト項目数を概算しやすいように
    テスト観点のメンバを記述してもよい
  - 親テスト観点のテスト項目数は、
    子テスト観点のテスト項目数を
    足し合わせる
  - 親テスト観点の品質リスクは、
    継承した子テスト観点の
    品質リスクの高い方を採る
    - » 規模が大きい場合はあれもこれも
      品質リスクが最高になるので、
      分布を示す計算方式を採ってもよい





付録：モデルの剪定

- 3つの方法でテスト観点モデルをアレンジすることで、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
  - ズームイン・アウト（テスト観点の抽象度の調節）
    - » ズームアウトするとテスト項目数は減少することが多い
  - 観点や関連の見直し
  - 網羅基準や組み合わせ基準の緩和



付録：（見逃しリスクの）確定

- ビューに漏れが無いという前提で
  テスト要求モデルの網羅性を評価する作業を「確定」と呼ぶ
  - 全て網羅できているという評価結果が重要なのではなく、
    どの観点に見逃しリスクが存在する（しない）のかを明示することが重要である
    - » 怖いのは、見逃してしまうことではなく、見逃すかどうか分からないことである
      - 見逃すことが分かっていれば、他の工程で対策を講じることもできる
- テスト観点ごとにボトムアップで見逃しリスクを確定していき、
  全体の見逃しリスクを確定する
  - そのテスト観点や関連でテスト漏れが発生しないと言い切れるならば確定する
    - » 子観点と関連に漏れが無く、
      網羅基準と組み合わせ基準が適切な観点は
      確定できる
      - 例）特異点も存在せず実行コードまで
        同値性を確認した同値クラスは確定できる
    - » 見逃しリスクが高いものを「暫定テスト観点」、
      十分低いものを「確定テスト観点」と呼ぶ
    - » 確定テスト観点は実線で囲む
    - » 子観点と関連が全て確定できたら、
      親テスト観点も確定できる