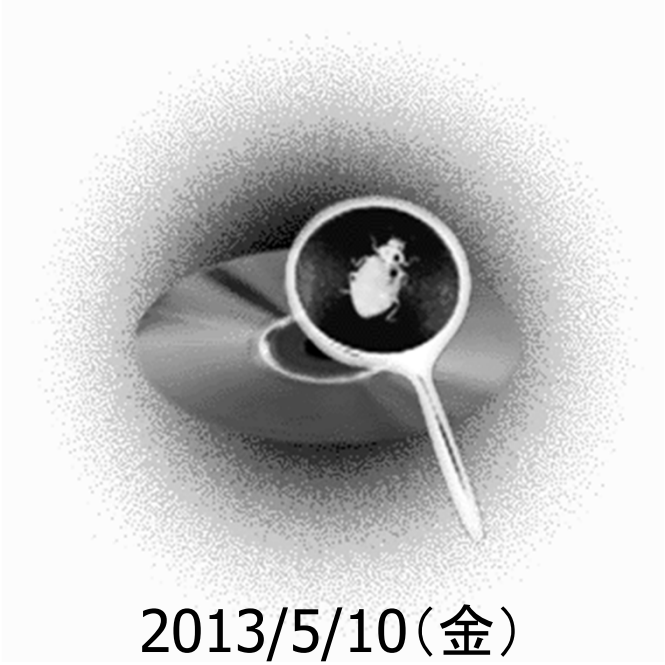
テスト観点に基づくテスト開発方法論
VSTePの概要



2013/5/10（金）

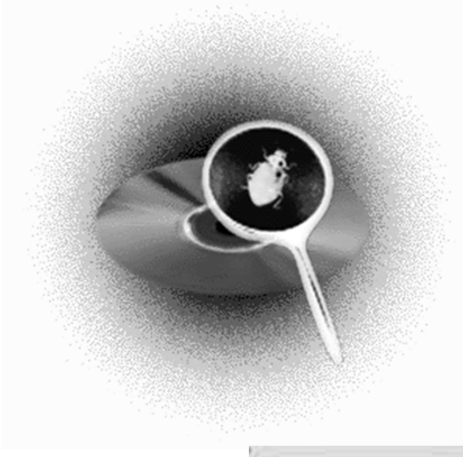
電気通信大学　大学院情報理工学研究科
総合情報学専攻 経営情報学コース
西　康晴（Yasuharu.Nishi@uec.ac.jp）

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/IEC29119 ソフトウェアテスト）

TEF: Software Testing Engineer's Forum

- ソフトウェアテスト技術者交流会
  - 1998年9月に活動開始
    - » 現在1800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
  - 「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
  - 技術別部会や地域別勉強会が実施されている
    - » プリンタ、Web、AVなど
    - » 東京、関西、九州、東海、札幌など
    - » TestLinkというオープンソースツールの日本語化部会もある



ASTER: Association of Software Test EngineeRing

- ソフトウェアテスト技術振興協会
  - テストを軸にして、ソフトウェア品質向上に関する教育や調査研究、普及振興を行うNPO法人
    - » 2006年4月に設立／理事・会員は無給
  - ソフトウェアテストシンポジウム（JaSST）を開催している
    - » 実行委員は手弁当／参加費は実費＋α
    - » 毎年4Qに東京で開催／今年はのべ約1,800名の参加者
    - » 今年は関西・四国・北海道・九州・東海・新潟・東北でも開催／会場はほぼ満席
  - ソフトウェアテストの資格試験（JSTQB）を運営している
    - » Foundation Levelは15,735名の受験者・8,227名の合格者
    - » Advanced Level（テストマネージャ）を2010年8月に開始・278/744名の合格
  - ソフトウェアテスト設計コンテストを開催している
    - » テスト設計の質の高さを競うコンテスト／今年は全国から21チームの参加
    - » 主要な成果物は無償で公開される／毎年テスト設計のレベルが向上している
  - 各地でソフトウェアテストの教育を行っている
    - » テストのスキル標準（test.SSF）をIVIAと共同で開発している
    - » セミナーや勉強会などを支援することで、地場の産業振興の定着を図る
  - アジア各国とテスト技術の交流（ASTA）を行っている
  - テストの先端技術を研究開発している：智美塾・バグ分析・ツール・Wモデルなど

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

SQiP：Software Quality Profession

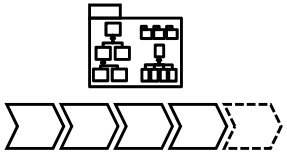
- 名称：
  - 日本科学技術連盟・ソフトウェア品質委員会
- 目的
  - SQiPは、ソフトウェア品質技術・施策の調査・研究・教育を通じて、実践的・実証的なソフトウェア品質方法論を確立・普及することにより、ソフトウェア品質の継続的な向上を目指す
- 3つの視点
  - ソフトウェア品質・実践・普及啓蒙
- 主軸とする活動
  - 1. 実践的・実証的なソフトウェア品質方法論の確立
  - 2. ソフトウェア品質方法論の普及促進・資格認定
  - 3. ソフトウェア品質向上のための国際協力の推進
- 活動方針
  - 1. ソフトウェア品質追究の重要性訴求
  - 2. 日本での実践的・実証的ソフトウェア品質方法論の形式知化
  - 3. グローバルな視野での活動
  - 4. 新しい課題へのチャレンジ

SQiP
Software Quality Profession

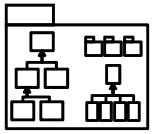
解説の流れ

- よくあるテストの現場の悩みのまとめ

- NGT/VSTePの概要と特徴



- NGTによるテスト観点図の概要

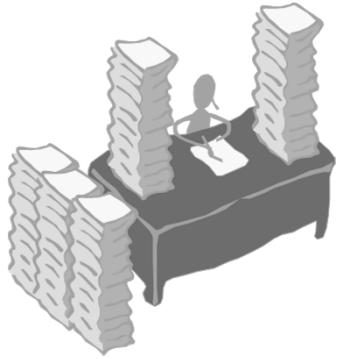


- VSTePによるテスト開発プロセスの概要

- NGT/VSTePを現場に適用する際のポイント

CPM法（コピー＆ペースト＆モディファイ法）の悩み

- 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

固定3レイヤー法の悩み

- 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

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

よくあるテスト技法（境界値テストなど）の悩み

- よくあるテスト技法とは
  - 「何をテストすればいいか」が分かっているという前提で、
    具体的なテスト条件を挙げる方法
- よくあるテスト技法の問題点
  - テスト技法そのものは理解したが、
    そもそも「何をテストすればいいか」が分からないので
    どこにどう適用すれば分からず、
    結局CPM法と固定3レイヤー法でテストケースを増殖させてしまう
    - » 機能の網羅はちゃんと出来たけど、
      動作環境が色々あるのは気づかなかった
    - » 境界値テストを設計したけど、
      そもそも同値クラスが浮かばなかった
    - » 直交表を使ってテストを設計したけど、因子が抜けてしまった

この「何をテストすればいいか」を
“テスト観点”と呼ぶ

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

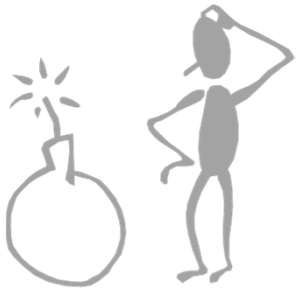
テスト設計には実に多くの観点が必要：組込みの例

- 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

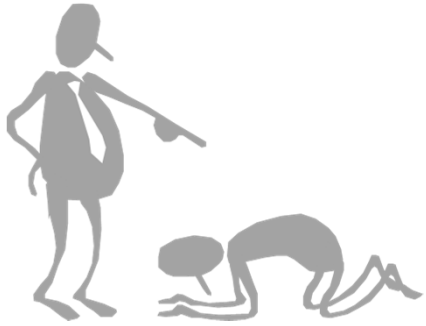
テストタイプ・テストレベルの悩み

- 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
- 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
- テストタイプやテストレベルの問題点
  - 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



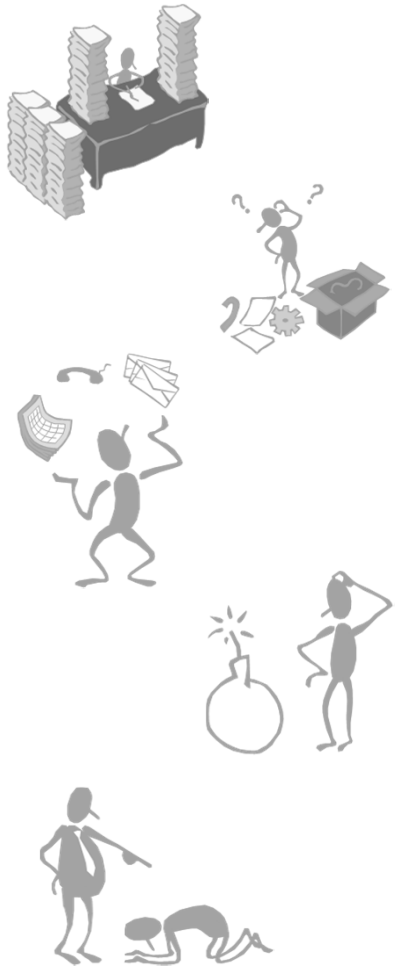
パートナー（協力会社）との悩み

- テストをパートナーに投げる理由
  - テストは誰でもできるので安い単価でパートナーに投げればよい、と判断する
  - テストは誰でもできるので丸投げしても問題無い、と判断する
- テストをパートナーに投げることの問題点
  - パートナーのテストがイマイチなので追求してみると、
    自分たちが作成しているテスト計画がスカスカだったことに気付く
    - » テストは誰でもできる、は幻想だったと後悔する
  - 丸投げしているのでパートナーがどんなテストをしているのか分からず、
    自分たちに全くテストの技術力が残らないので、
    気がつくとパートナーにテストを依存し抜け出せなくなってしまう
    - »「首根っこをつかまれた」状態になっている
  - 頭を使うところと機械的にできるところの区別がつかず、
    かつパートナーが工数精算のため自動化を嫌がるので、
    いつまで経ってもテストの自動化に取り組めない



よくあるテストの現場の悩みのまとめ

- CPM法：
  - テストケースの趣旨が分からない
- 固定3レイヤー法：
  - 詳細化が飛んでいる・揃わない・おかしい
- よくあるテスト技法：
  - テスト観点が列挙・整理されていない
- テストタイプ・テストレベル：
  - テストタイプ・テストレベルが整理されていない
- パートナー：
  - 自分たちにテストの技術は残しつつ
    コストダウンしたいし自動化もしたい



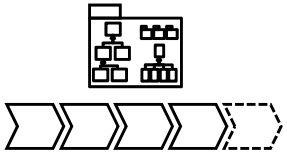
ではどんなテストの方法論があればよいのか

- 【悩み】テストケースの趣旨が分からない
  → 【解決策】テストケースとテスト観点とを必ず結びつけるようにテスト設計する
- 【悩み】詳細化が飛んでいる・揃わない・おかしい
  → 【解決策】親子関係をきちんと明示し網羅性を保証する
- 【悩み】テスト観点が列挙・整理されていない
  → 【解決策】テスト観点を列挙・整理し全体像を把握する
- 【悩み】テストタイプ・テストレベルが整理されていない
  → 【解決策】 テストタイプ・テストレベルをテスト観点でより詳細に表して図示し、
  テストタイプやテストカテゴリ間の重複や漏れ、
  不要な依存関係を減らす
- 【悩み】自分たちにテストの技術は残しつつ
  コストダウンしたいし自動化もしたい
  → 【解決策】テスト観点を列挙し整理する高度に頭を使う作業と、
  機械的に網羅するだけの単純作業とを峻別し、
  前者を自らに残し、後者を少しずつ自動化していく

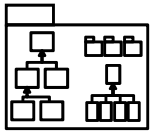
解説の流れ

- よくあるテストの現場の悩みのまとめ

- NGT/VSTePの概要と特徴



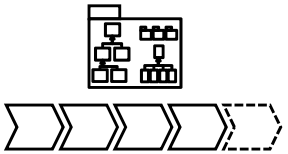
- NGTによるテスト観点図の概要



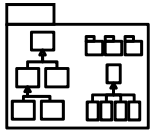
- VSTePによるテスト開発プロセスの概要

- NGT/VSTePを現場に適用する際のポイント

NGT/VSTePの概要

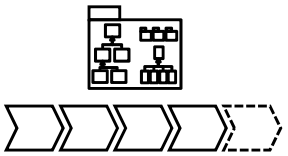


- NGT/VSTePとは何か
  - テスト観点を基盤としたテスト開発のための記法とテスト開発プロセス
    - » NGT（Notation for Generic Testing）：テスト開発のための記法
    - » VSTeP（Viewpoint-based Test Engineering Process）：テスト開発プロセス
- NGTによるテスト観点図
  - テストケースとテスト観点とを必ず結びつけるようにテスト設計する
  - 親子関係をきちんと明示し網羅性を保証する
  - テスト観点を列挙・整理し全体像を把握する

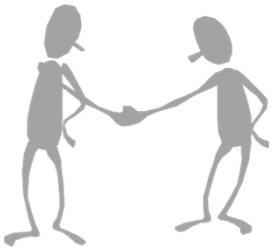


- VSTePによるテスト開発プロセス
  - テストタイプ・テストレベルをテスト観点でより詳細に表して図示し、テストタイプやテストカテゴリ間の重複や漏れ、不要な依存関係を減らす
  - テスト観点を列挙し整理する高度に頭を使う作業と、機械的に網羅するだけの単純作業とを峻別し、前者を自らに残し、後者を少しずつ自動化していく

NGT/VSTePの特徴

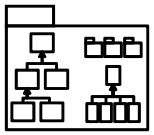


- NGT/VSTePが効果を発揮する点
  - 自分たちの技術や経験を十分発揮して網羅的に保証すること
    - » ステレオタイプを統一した詳細化や関連を使う
    - » 確定により、保証状況が不明というテスト観点を特定することができる
  - テスト規模に応じたやり方にテーラリングすること
    - » 例）小規模なテストではテストアーキテクチャ設計は少しでよい
  - 適用規模に応じたやり方にテーラリングすること
    - » 特定の機能群やテストタイプ、テストレベルだけに用いてもよい
  - 過去の無秩序なテストケース群（テストスイート）を整理すること
    - » Reverse VSTePでリバースエンジニアリングできる
  - テスト資産を蓄積して再利用すること
    - » 膨大なテストケースでなく、比較的少数のテスト観点で再利用を判断できる

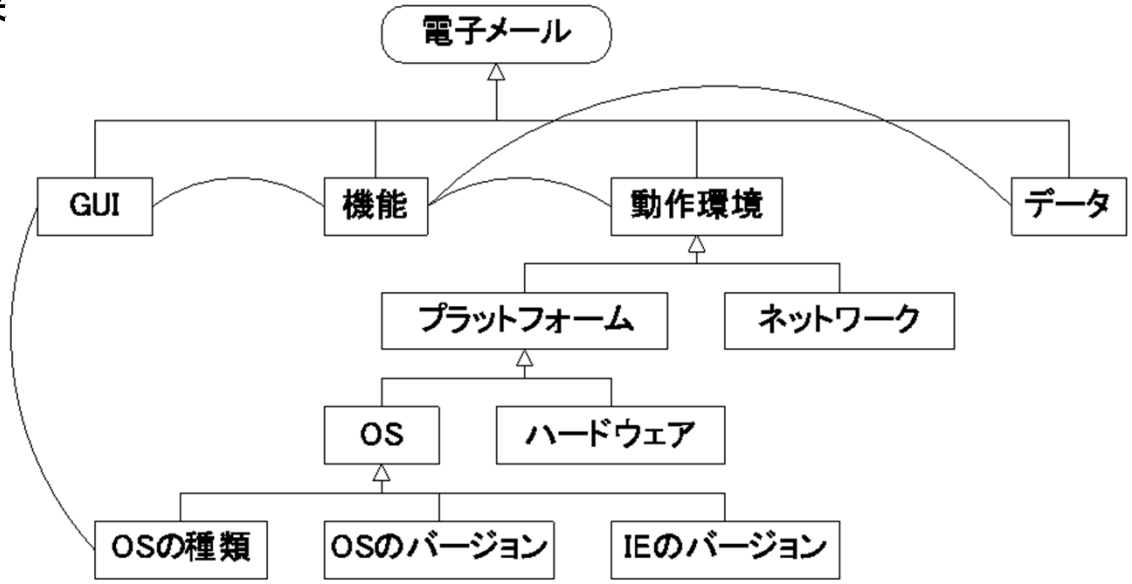


- NGT/VSTePだけでは効果を発揮できない点
  - 開発組織とテスト組織の叡智を超えて網羅的に保証すること
    - » 誰も経験したことのないことを「予言」するような方法ではない
  - ピンポイントテストや探索的テストでの狙いどころを新たに決めること
    - » 狙いどころを決めるには不具合モード分析などのパターン化技術が別に必要である
    - » 以前狙ったテスト観点はNGT/VSTePで整理して蓄積できる

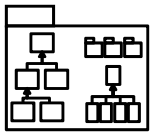
NGTによるテスト観点図の概要



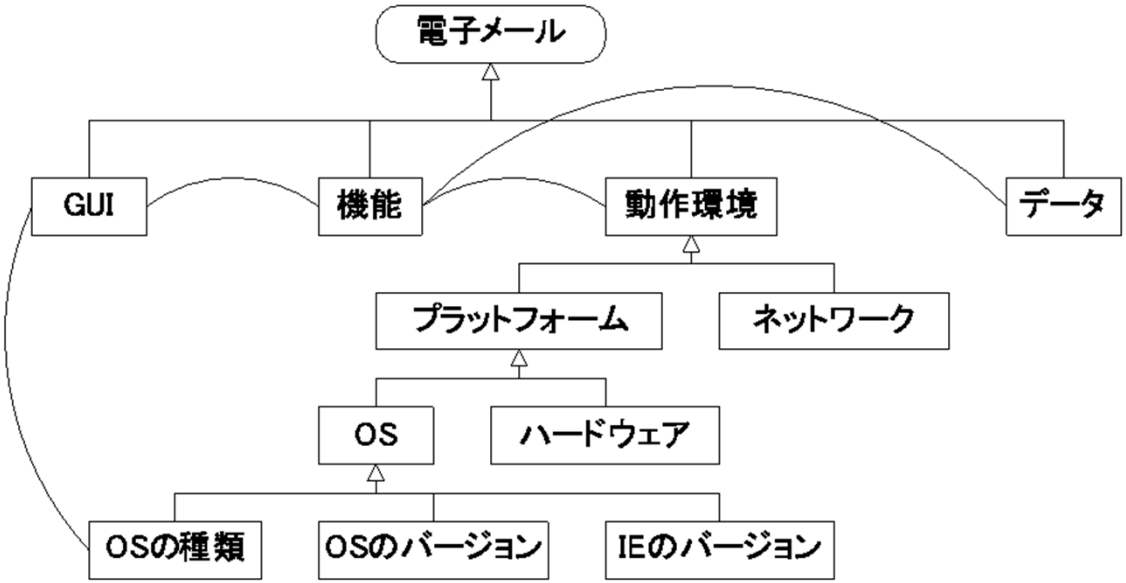
- テスト観点
  - テスト観点とは何か
  - なぜテスト観点と呼ぶのか
  - テスト観点とテストケースとテストスクリプト
  - 網羅基準と品質リスク
- 詳細化
  - ステレオタイプ
    - » 継承・部分・属性化・原因結果
  - ビューとビューポイント
- 関連
  - 組み合わせ関連
  - 順序依存関連
  - 関連ビューポイント
- テストフレーム
- テストコンテナ



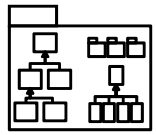
NGTによるテスト観点図の記述



- 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



NGT: テスト観点とは何か

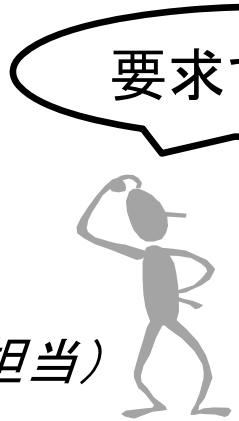


- テストには、様々な「観点」が必要だと言われている
  - 例）Ostrandの4つのビュー
    - » ユーザビュー、仕様ビュー、設計・実装ビュー、バグビュー
  - 例）Myersの14のシステムテスト・カテゴリ
    - » ボリューム、ストレス、効率、ストレージ、信頼性、構成、互換性、設置、回復、操作性、セキュリティ、サービス性、文書、手続き
  - 例）ISO/IEC 9126(ISO/IEC 25000s)の品質特性
    - » 機能性、信頼性、使用性、効率性、保守性、移植性

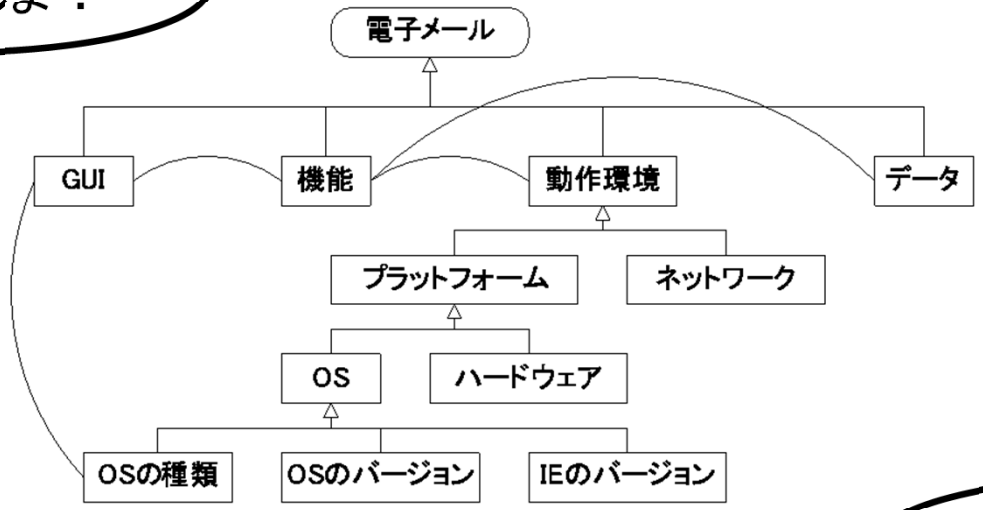
- テスト観点とは、テスト対象のテストすべき側面や、テスト対象の範囲やつくり、テスト対象が達成すべき性質である
  - 具体化してテストケースの一部になるものが基本的にテスト観点となる
    - » テスト条件（網羅すべきもの）：仕様、機能、データ、ユーザの使い方など
    - » テスト対象：機能、サブシステム、モジュール、H/Wなど
    - » ふるまい（期待結果）：品質特性など
    - » 狙いたいバグ（のパターン）：メモリリーク、バッファオーバーフローなど
  - テストタイプやテストレベル、テスト技法をヒントにして挙げることもできる
  - 開発成果物やその基となる考え方）をヒントにすることもできる

NGT: 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

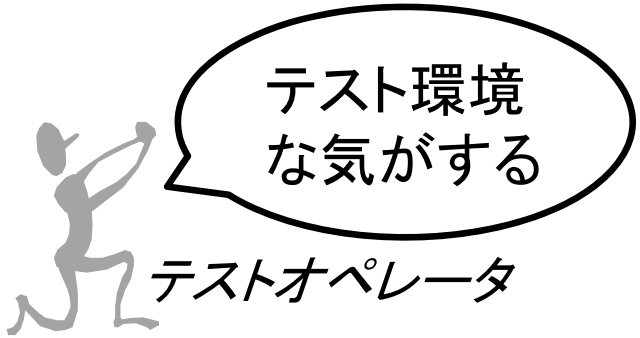


SE(要求担当)



テストエンジニア

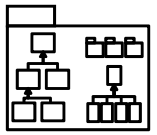
テストマネジャー



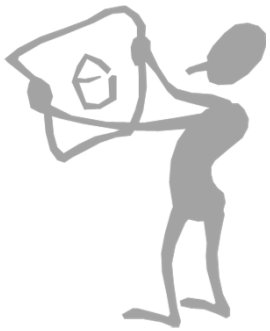
テストオペレータ

組み合わせ
テスト設計者

NGT: テスト観点とテストケースとテスト手順の関係

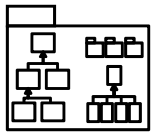


- テスト観点とテストケースとテストスクリプトをきちんと区別する
  - テスト観点
    - » 何をテストするのかのみを端的に記述したもの
      - ・ 例）正三角形
  - テストケース
    - » そのテスト観点でテストするのに必要な値などのみを特定したもの
      - ・ 例）(1,1,1), (2,2,2), (3,3,3)…
    - » 通常は、網羅基準に沿って特定される値などのみから構成される
    - » テストケースは基本的にシンプルになる
  - テストスクリプト（テスト手順）
    - » そのテストケースを実行するために必要な全てが書かれたもの
      - ・ 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

NGT: 網羅基準と品質リスク



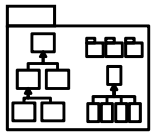
- テスト観点からテストケースを生成するために、網羅基準が必要となる
  - 例）制御パスを C1 基準で網羅する
  - 例）ボリュームの内部上限値と内部下限値でテストする
  - テスト観点が詳細に決まり網羅基準が決まれば、テストケースは基本的に機械的に生成できる
  - テスト実施以外でそのテスト観点を保証するなら、それを明記する

観点名	
テスト項目数	品質リスク
品質保証手段（網羅基準）	

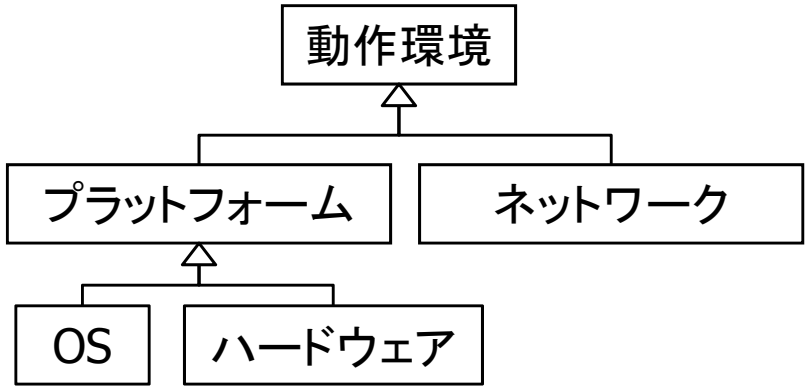
- テスト観点の重みを、品質リスクとして記述する
  - 品質リスクは通常、そのテスト観点の重要性と致命度から算出する
    - » 致命度は、そのテスト観点のテストを実施しなかった時にテスト対象に発生するダメージの確率や大きさから推測する
  - 品質リスクに応じて、テスト観点の実施優先度や厚み、詳細度を決める
    - » 厚みは、網羅基準の厳しさ（そのテスト観点のテストケースの多さ）となる
- 工程が進んでいくにしたがって網羅基準や品質リスクを記述していく
  - 最初はテスト観点名だけでモデリングし、後から網羅基準や品質リスクを追加して検討していく
    - » 最初からあらゆることを書こうとしてはいけない

WinNT系OS	
5（2K, XP, VS,7,8）	高（Imp:高 Sev: 中）
すべて	

NGT: テスト観点の詳細化

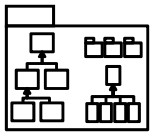


- テスト観点は階層的に詳細化できる
  - 詳細化は、近くからモノを見る（ズームインする）ようなことである
  - 詳しくモデリングする場合は、継承（is-a）、合成集約（部分：has-a）、属性化、原因－結果など詳細化すべき理由をステレオタイプで表す
    - » ステレオタイプごとに分けて詳細化を行うとMECE性（網羅性）を保証しやすくなる

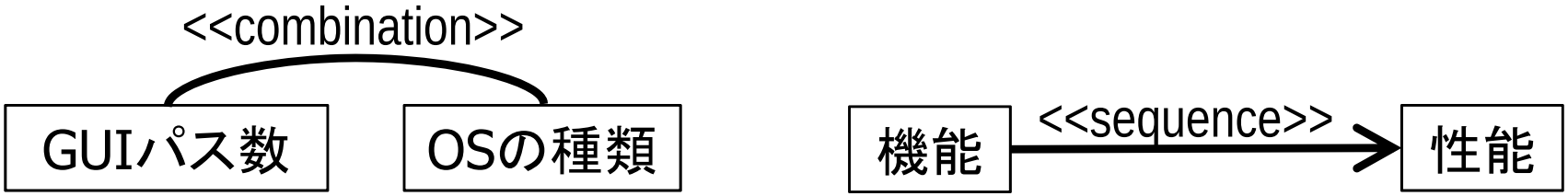


- 階層の最上位のテスト観点を「（トップ）ビュー」と呼ぶ
  - ビューは、そのサブツリーが全体として何をテストするのか、を表している
  - どのようなトップビューで構成するか、はテストエンジニアによってかなり異なる
- 階層の最下位のテスト観点を「ボトムビューポイント」と呼ぶ
  - そのテスト観点を見れば、何をどのように網羅すればよいかが一目で分かる詳細度、がボトムビューポイントである
  - ボトムビューポイントはテスト詳細設計でのテスト条件となる
    - » 例えば同値クラスになる

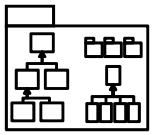
NGT: テスト観点間の関連



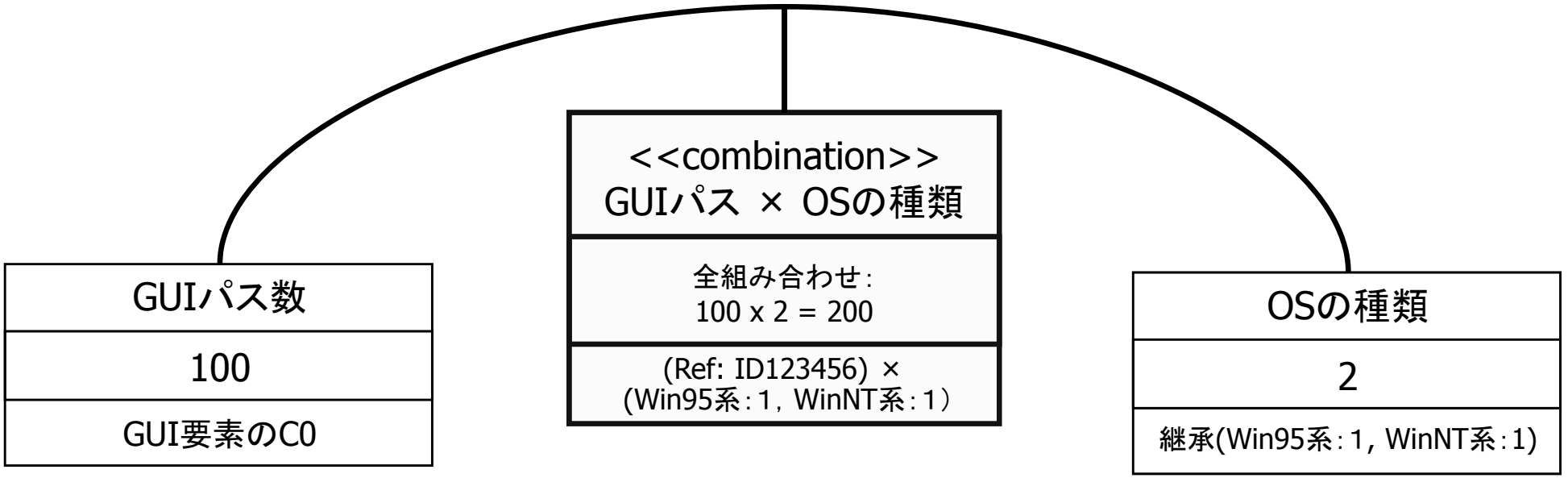
- 複数の観点を組み合わせるテストを「関連」で示す
  - 例）負荷テストでは、搭載メモリ量という観点と、
    投入データ量という観点を組み合わせてテスト設計を行う
    - » 搭載メモリ量と投入データ量のバランスによって、
      テスト対象のふるまいが変化するからである
  - 組み合わせテストの必要性のように順序依存性のない関連は、
    テスト観点間の矢頭の無い曲線で表す
    - » 順序依存性のある関連は → のように開き矢尻の矢印で表す
  - 新たな関係や詳細な関係を定義したり分かりやすくするために
    <<stereotype>>（ステレオタイプ）を使って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



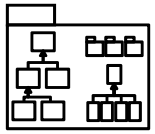
NGT: 関連ビューポイント



- 組み合わせテストのように関連そのものが
  テストケースを導く場合は、
  関連そのものに名前をつけて扱う方がよいことがある
  - テスト観点名に関連の種類をステレオタイプとして記述する
  - 実は単一のテスト観点かもしれない

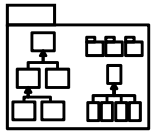


NGT: テストフレーム

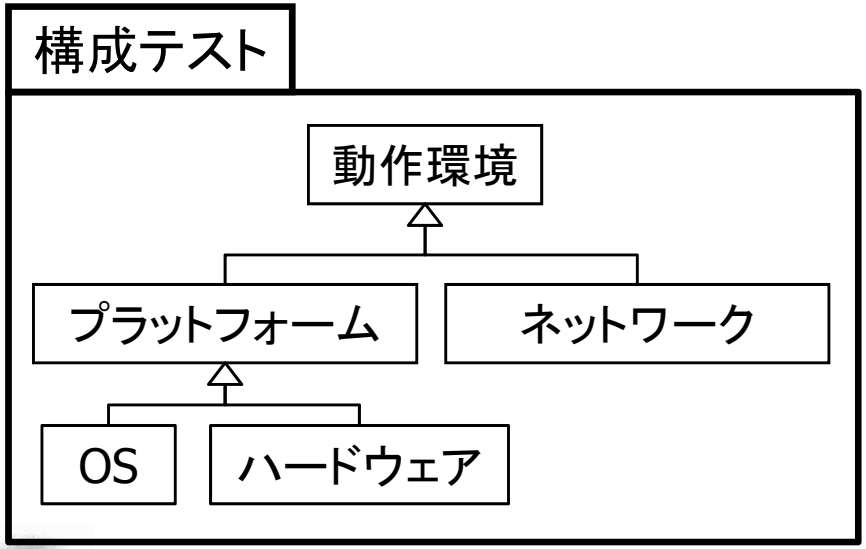


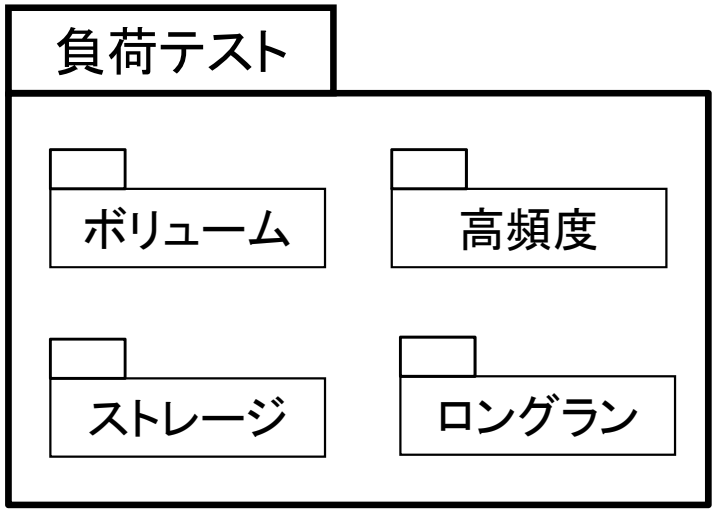
- 実際のテスト設計では、複数のテスト観点を組み合わせてテストケースを設計したい場合がある
  - この条件のテストは、テスト対象のどの部分に行うんだろう？
  - この部分に対するテストでは、どの品質特性を確認するんだろう？
  - この品質特性をテストするには、どの操作を行えばいいんだろう？
- テストケースの構造（スケルトン）をテスト観点の組み合わせで示すことで、複数のテスト観点によるテストケースを設計する
  - テストケースの構造を表すテスト観点の組み合わせを「テストフレーム」と呼ぶ
    - » <<frame>> というステレオタイプの関連を用いる
    - » シンプルなテ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：
    - » 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

NGT: テストコンテナ

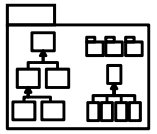


- 大規模なテスト設計では、
  複数のテスト観点をグルーピングして扱いたい時がある
  - 複数のテスト観点をグルーピングしたものをテストコンテナと呼ぶ
    - » テストタイプやテストレベル、テストサイクルなどを表現する
  - テストコンテナは包含関係を持つことができる
    - » テストコンテナに含まれるテスト観点を比べると
      テストコンテナ間の役割分担を明確に把握できる
      - ・ 負荷テストと性能テストなど、違いがよく分からないテストタイプも区別できる
      - ・ 単体テストと結合テストなど、役割分担がよく分からないテストレベルも区別できる

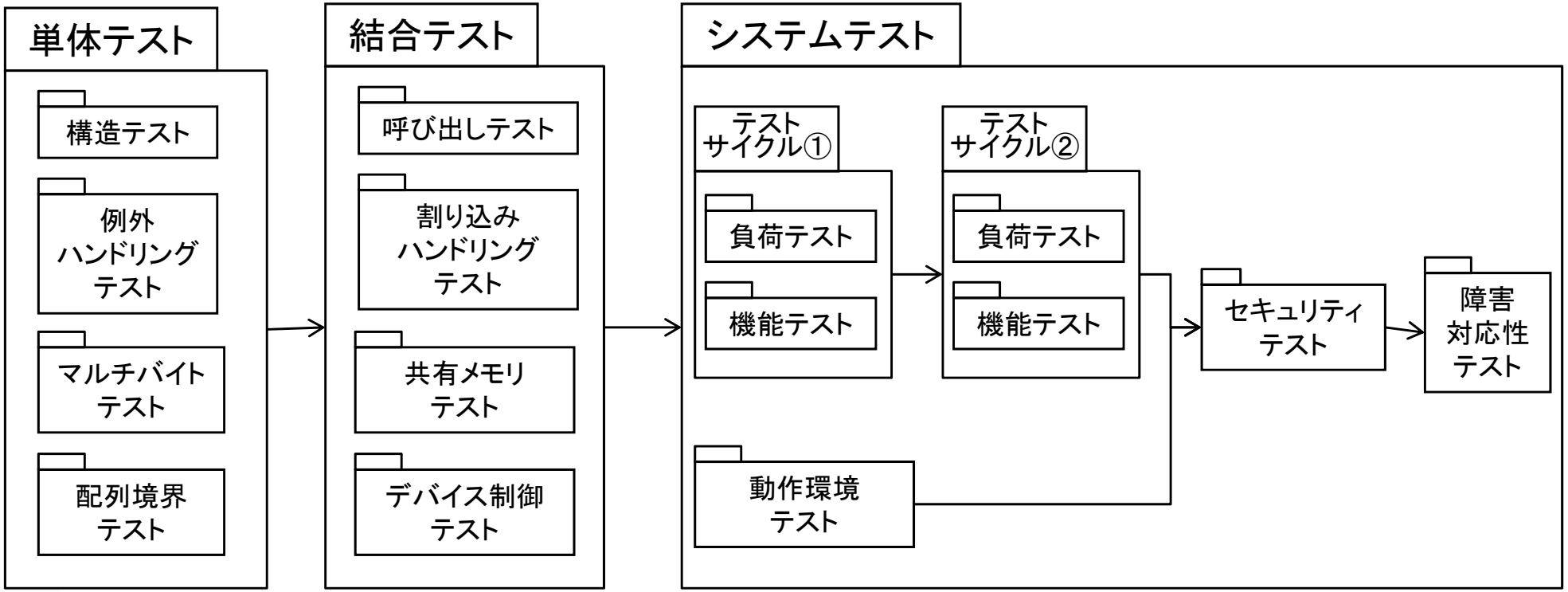




NGT: テストコンテナ

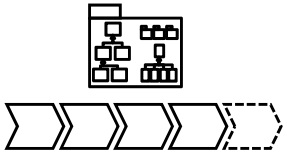
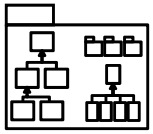


- 大規模なテスト設計では、
  テストコンテナで表すと全体像を把握しやすくなる
  - テストコンテナを用いるとテストアーキテクチャを俯瞰できる
    - » テスト全体をテストレベルやテストタイプ、テストサイクルの粒度で
      テストアーキテクチャを俯瞰する



解説の流れ

- よくあるテストの現場の悩みのまとめ
- NGT/VSTePの概要と特徴
- NGTによるテスト観点図の概要
- VSTePによるテスト開発プロセスの概要
- NGT/VSTePを現場に適用する際のポイント

VSTePによるテスト開発プロセスの概要

- テスト開発プロセス
- テスト要求分析
  - モデルのリファイン
  - ステレオタイプ分析
  - 確定
- テストアーキテクチャ設計
  - 剪定／ズームイン・ズームアウト
  - テストコンテナ分割
  - テストフレーム構築
- テスト詳細設計
- テスト実装
  - 集約

ソフトウェアテスト開発プロセスが必要である

- ソフトウェアの高品質化・大規模化・複雑化に伴い、ソフトウェアテストの高品質化・大規模化・複雑化も急務である
  - 10万件を超える様々な観点による質の高いテストケースを、いかに体系的に開発するか、が課題である
- しかしテスト計画やテスト戦略立案という工程は未成熟である
  - テストケースという成果物の開発と、テストのマネジメントとが混同されている
    - » テスト計画書にテストケースもスケジュールも人員アサインも全て記載される
  - テストケースの開発という工程が見えにくくなっており、そこで必要となる様々な工程が混沌として一体的に扱われている
    - » テスト(ケース)開発という用語はほとんど使われず、テスト計画、テスト仕様書作成、テスト実施準備といった用語が使われている
    - » テスト戦略という用語もイマイチよく分からない
- そこで「ソフトウェアテスト開発プロセス」という概念が必要となる
  - テストケースを開発成果物と捉え、開発プロセスを整備する必要がある
    - » 高品質・大規模・複雑なテストケースを開発するために必要な作業を明らかにする
    - » 本来はテスト実施以降やテスト管理のプロセスも必要だが、今回は省略する
  - ASTER／智美塾でエキスパートが議論している
    - » 我々はVSTeP(Viewpoint-based Software Test Engineering Process)を提案している

テスト観点を構成要素としてモデリングを行う

- テスト観点のモデリングを行ってテストを開発すべきである
  - モデリングを行うと、テストで考慮すべき観点を一覧でき、
    俯瞰的かつビジュアルに整理できる
    - » 10万件を超えるテストケースなど、テスト観点のモデル無しには理解できない
    - » モデルとして図示することで、大きな抜けや偏ったバランスに気づきやすくなる
    - » 複数のテストエンジニアでレビューしたり、意志を共有しやすくなる
  - テスト要求分析では、テスト要求モデルを作成する
    - » テスト対象、ユーザの求める品質、使われる世界などをモデリングする
    - » ボトムビューポイントが特定できるまで丁寧に段階的に詳細化する
  - テストアーキテクチャ設計では、
    テスト詳細設計・実装・実行しやすいようにテスト要求モデルをリファインする
    - » 適切なテストタイプやテストレベルそのものを設計する
    - » 見通しのよいテストアーキテクチャを構築する
    - » よいテスト設計のためのテストデザインパターンを蓄積し活用する
  - VSTeP(Viewpoint-based Software Test Engineering Process)では、
    NGT(Notation for Generic Testing)という記法でテスト観点のモデリングを行う
    - » ツリー型の記法を用いるため、Excelで表を埋めるよりも“エンジニアリング”っぽい

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

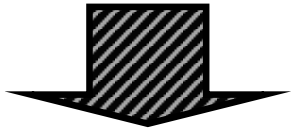
ソフト要求仕様（要求モデル）	＝	テスト要求仕様（要求モデル）
ソフトアーキテクチャモデル	＝	テストアーキテクチャモデル
ソフトモジュール設計	＝	テストケース
プログラム	＝	手動／自動化テストスクリプト（テスト手順）

旧来のテストプロセスでは粗すぎる

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



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

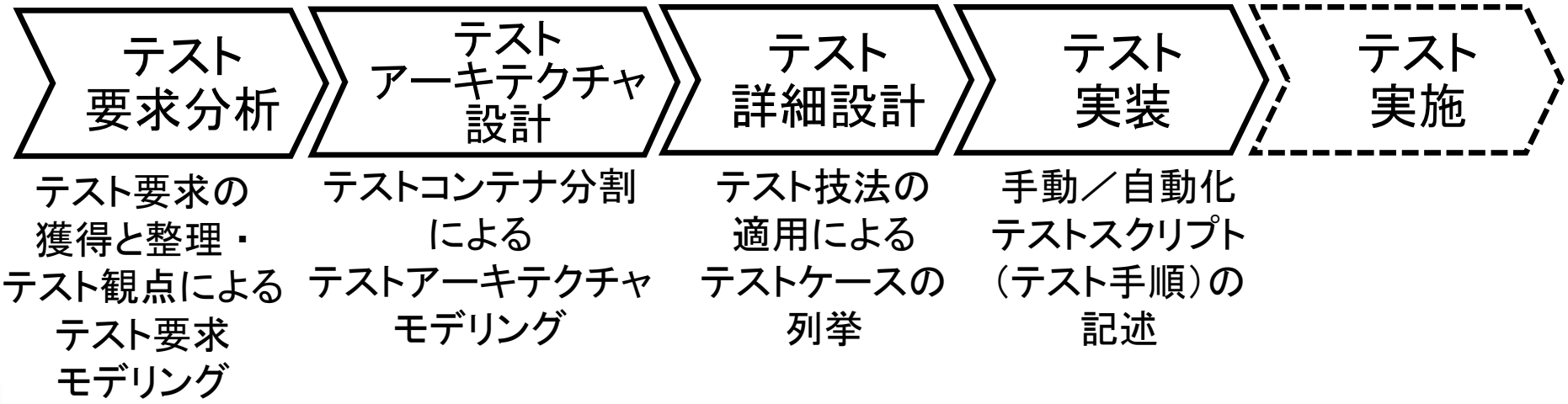
テスト技法の
適用による
テストケースの
列挙

手動／自動化
テストスクリプト
（テスト手順）の
記述

VSTePによるテスト開発プロセスの特徴

- VSTeP(Viewpoint-based Software Test Engineering Process)とは、テスト観点のモデリングを核としたテスト開発方法論である
  - テスト観点を核にして、テストの要求からテストケース、テスト手順までをシームレスに開発していく方法論である
    - » テスト要求分析やテストアーキテクチャ設計など、 軽視しがちなテストの上流工程に力を入れることができ、質の高いテスト設計やレビュー、経験の蓄積が可能になる
    - » ソフトウェア開発と同様にモデリングスキルを求められるが、パターンやSPLE、モデル駆動開発などソフトウェアエンジニアリング的な技術をテストに応用しやすい
    - » 既存のテストをリバースエンジニアリングできるのでテストの再利用や改善もしやすい

【VSTePの流れ】



テスト要求分析

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

テスト要求分析：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

テスト要求分析: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

テスト要求分析：3) モデルの構築と納得

3) モデルの構築と納得

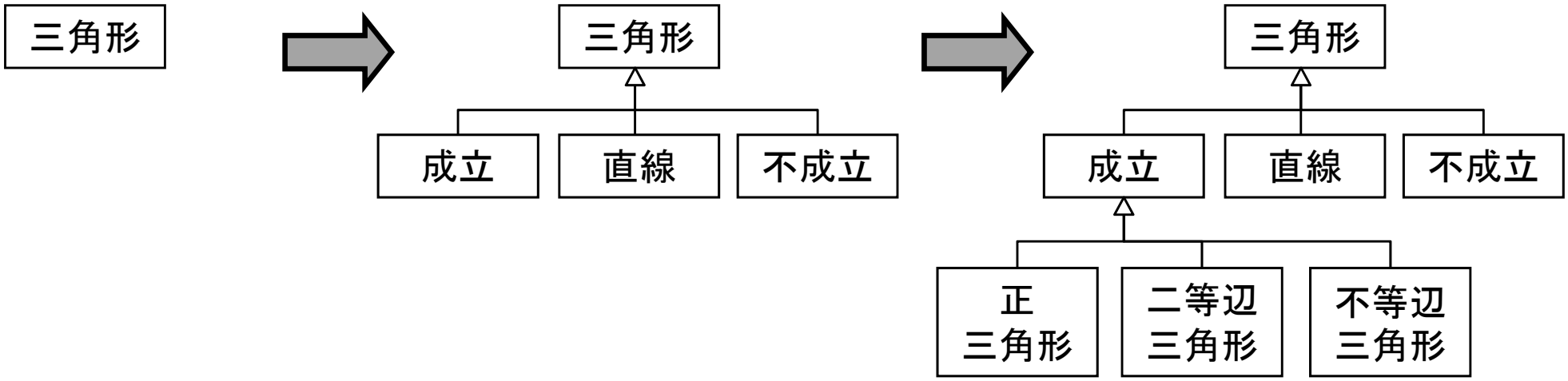
- テスト観点図を用いてテスト要求モデルを構築し、
  自分たちとステークホルダーが納得するまでリファインする

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

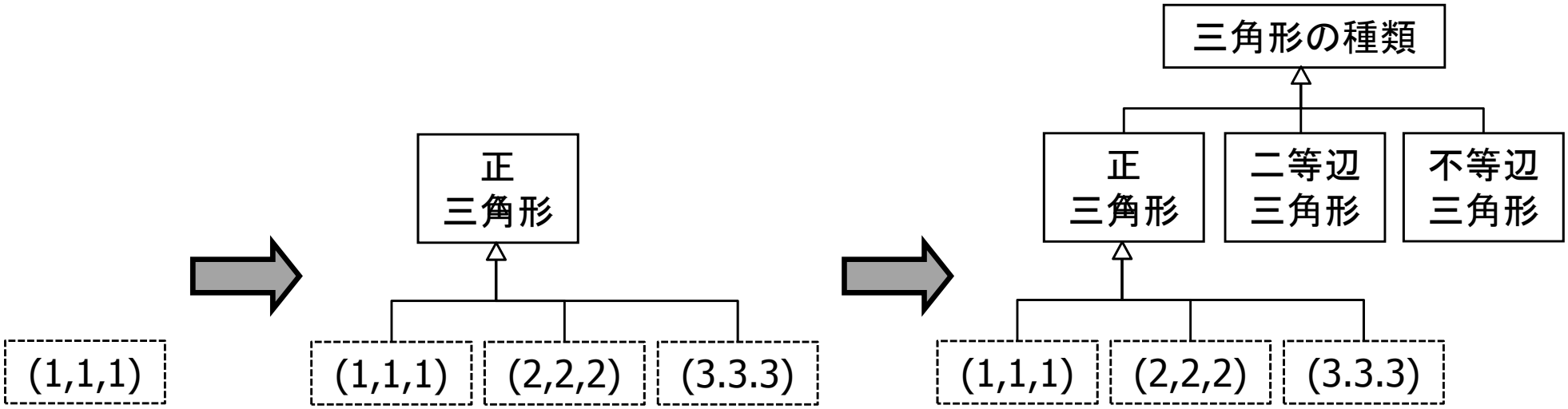
テスト要求分析：モデリングの2つのアプローチ

- 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



テスト要求分析：モデリングの2つのアプローチ

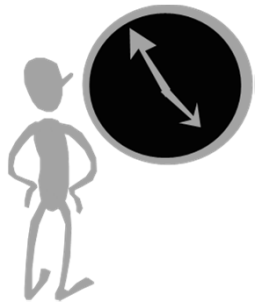
- ボトムアップ型
  - まず思いつくところからテストケースを具体的に書き、
    いくつか集まったところで抽象化し、テスト観点を導き出していく
    - » (1,1,1) なんてテスト、どうかな
    - » (2,2,2) とか、(3,3,3) もあるな
    - » これって要するに「正三角形」だな
    - » 他に似たようなのあるかな、う～ん、二等辺三角形とかかな
  - 実際にはトップダウンとボトムアップを循環させながらモデリングする



テスト要求分析: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



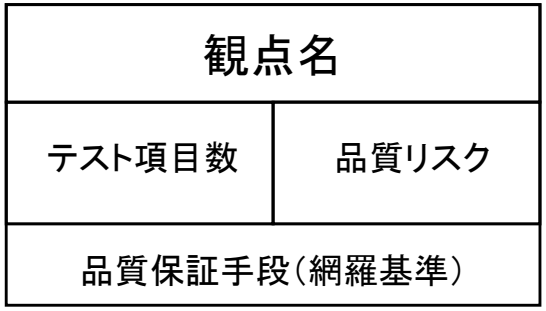
- 自分たちで要求が達成でき品質リスクが受け入れられる
  テスト観点図ができたと実感できたら、そのテスト観点図を囲んで、
  ステークホルダーが納得するまで一緒にリファインする
  - » 剪定前と剪定後を根拠つきで比較し、確定の結果と合わせて検討することで、
    ステークホルダー自身が品質リスクを受け入れたと認識することが重要である
  - » あくまで顧客の納得を深めるのがテスト要求モデリングのリファインの
    主目的であり、その後の工程でテストしやすくするのが目的ではない
  - » リファインした結果、総テスト項目数から予想されるテスト工数が不足すると
    見積もることができる場合、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

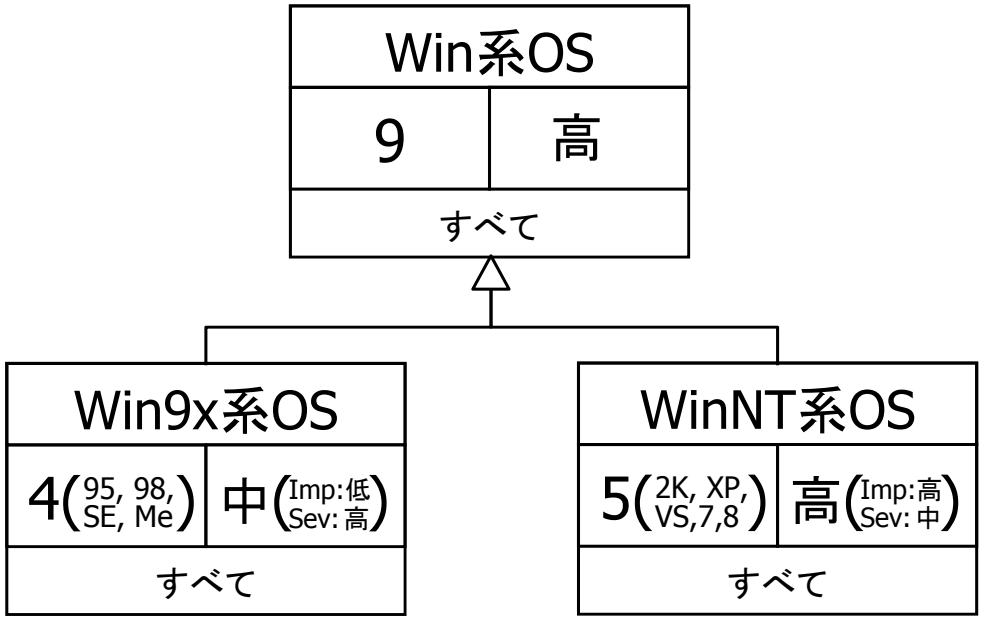
テスト要求分析：モデルのリファイン

- 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

テスト要求分析：項目数と品質リスクの概算

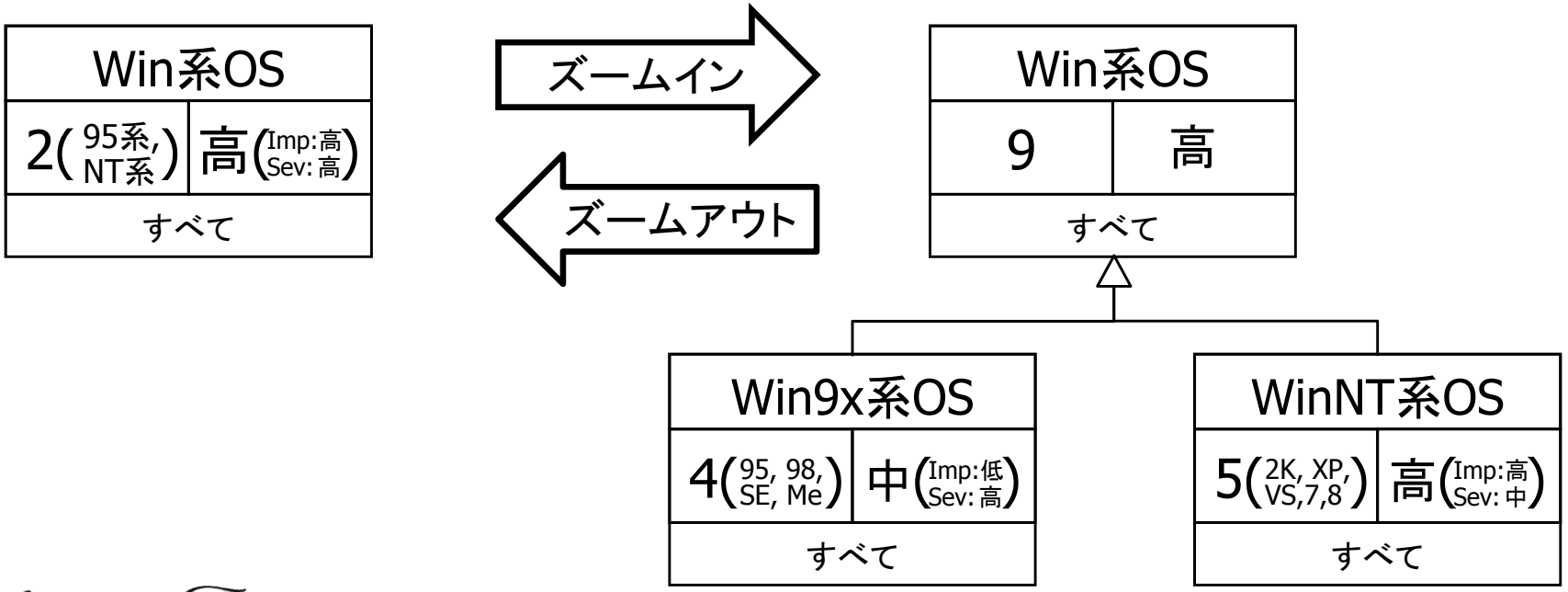
- モデリングが進んできたら、テスト観点ごとにテスト項目数と品質リスクの概算を行う
  - モデリングの進み方やテストプロセスの成熟度などに応じて、どちらかだけでもよい
  - テスト項目数を概算しやすいようにテスト観点のメンバを記述してもよい
  - 親テスト観点のテスト項目数は、子テスト観点のテスト項目数を足し合わせる
  - 親テスト観点の品質リスクは、継承した子テスト観点の品質リスクの高い方を採る





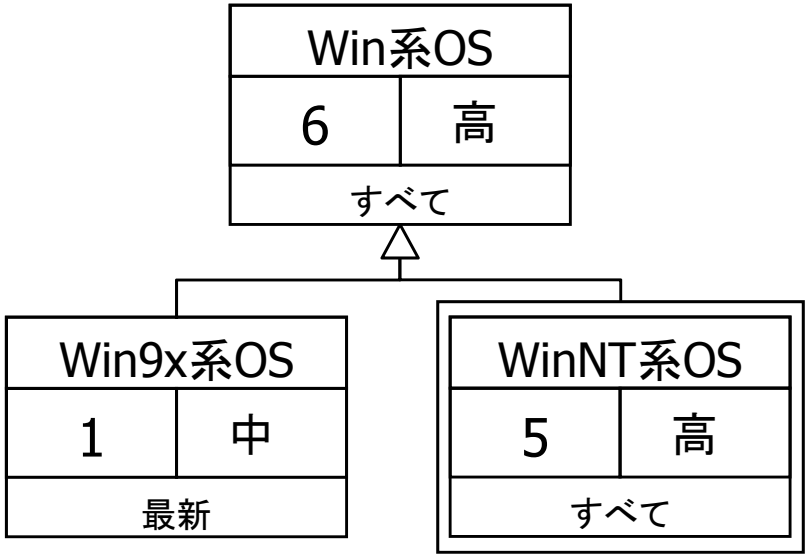
テスト要求分析：モデルの剪定

- 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



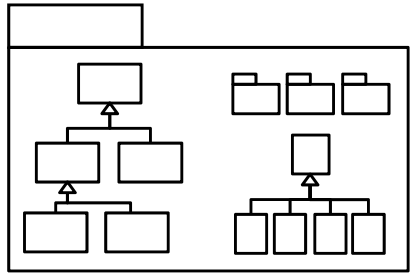
テスト要求分析：（見逃しリスクの）確定

- 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

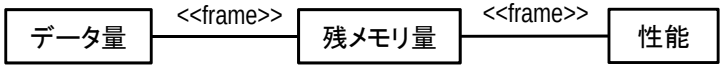


テストアーキテクチャ設計

- テストアーキテクチャ設計とは、
  テストスイートの全体像を把握しやすくしつつ
  後工程や派生製品、後継プロジェクトが作業しやすくなるように
  テスト要求モデルを整理する工程である
  - 1) テストコンテナモデリング
    - » テスト要求モデルを分割し、
      同じまとまりとしてテストすべきテスト観点を
      同じテストコンテナにまとめることで整理する
      - ・ テストコンテナ：テストレベルやテストタイプ、テストサイクル
      - ・ テストレベルやテストタイプそのものを、テスト観点を用いて設計する
      - ・ モデルのリファイン（整理）も行っていく

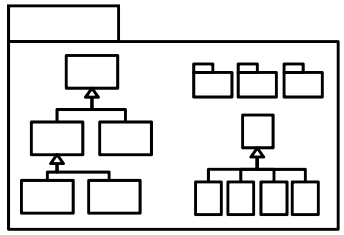


    - » テスト観点よりも粒度の粗いテストコンテナを用いるので
      テストスイートの全体像の把握や全体に関わる設計がしやすい
    - » 自社で定められているテストレベルやテストタイプの趣旨を理解し、
      きちんと全体像を設計できるようになる
    - » テストにも保守性など特有の「テスト品質特性」があるが、
      どのようなテスト品質特性を重視するかによって異なるテストアーキテクチャとなる
  - 2) テストフレームモデリング
    - » （同じテストコンテナ内で）複数のテスト観点をまとめて
      テストケースの構造（スケルトン）を定義し、
      テスト詳細設計をしやすくする



テストアーキテクチャ設計：1) コンテナモデリング

- 同じ時期にテストすべきテスト観点をテストレベルとしてまとめる
  - テスト観点には、テスト観点同士の順序依存関係や欠陥特定の絞り込みのための順序依存関係、プロジェクトの制約としての時期依存関係がある
    - » 例）機能テスト観点の実施情報を用いて負荷テストを設計したい
    - » 例）モジュール内設計のテストを実施した後にモジュール間設計のテストを実施しないと欠陥特定の工数がかかりすぎてしまう
    - » 例）特殊なテスト環境が揃うのが後半なので構成テストは最後に回したい
- 同じ趣旨を持ってるテスト観点をテストタイプとしてまとめる
  - 例）負荷テストタイプは複数のテスト観点からなるが、負荷をかけるという同じ趣旨を持っている
- 繰り返しのテストや回帰テストが必要なテスト観点をテストサイクルとしてまとめる
  - 同じテスト観点を繰り返しているようでいて、実は意味が異なったりすることがあるので注意すること
- 複数のテストタイプからなるテストレベル、サブレベルを持つテストレベルなどのように、他のテストコンテナを包含しながらまとめることもできる

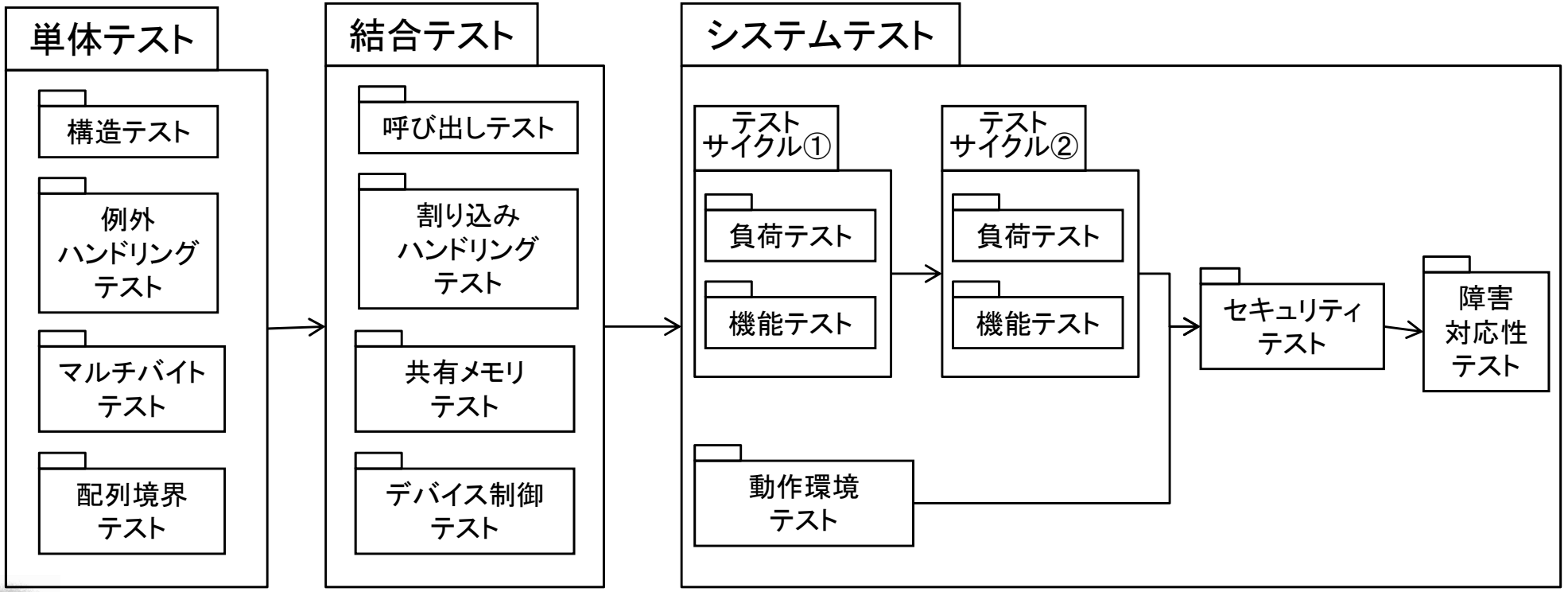


テストアーキテクチャ設計：1) コンテナモデリング

- テストコンテナを用いてテストアーキテクチャを表現するメリット
  - テスト観点よりも粒度の粗いテストコンテナを用いるので、俯瞰して把握しやすい
  - テストコンテナ間に含まれるテスト観点を比べると、役割分担を明確に把握できる
    - » 現場で経験的に用いているテストコンテナの妥当性を評価したり改善できるようになる
      - ・ 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
  - サブレベルなどを階層的に表現できる
  - テストスイートの品質特性（保守性や自動化容易性など）を意識して
    テスト設計に反映できる
- テストコンテナを設計する際のポイント
  - テストが小規模で単純な場合、テスト要求モデルのビュー（サブツリー）を
    そのままテストコンテナにしてしまってよい場合も多い
  - テストコンテナの趣旨が把握しやすいようにモデリングする
  - テストコンテナ間の関係がなるべく少なくなる（疎になる）ようにモデリングする
  - テスト要求モデルでは単一のテスト観点を分割することもある
  - マネジメント的要求からテストスイートの品質特性を抽出し、
    それらを達成できるようにモデリングする
    - » 例）保守性を高めたいというテストスイートの品質特性がある場合、
      変更があまり入らないテストコンテナと変更が頻繁に入るテストコンテナに分割する

テストアーキテクチャ設計：1) コンテナモデリング

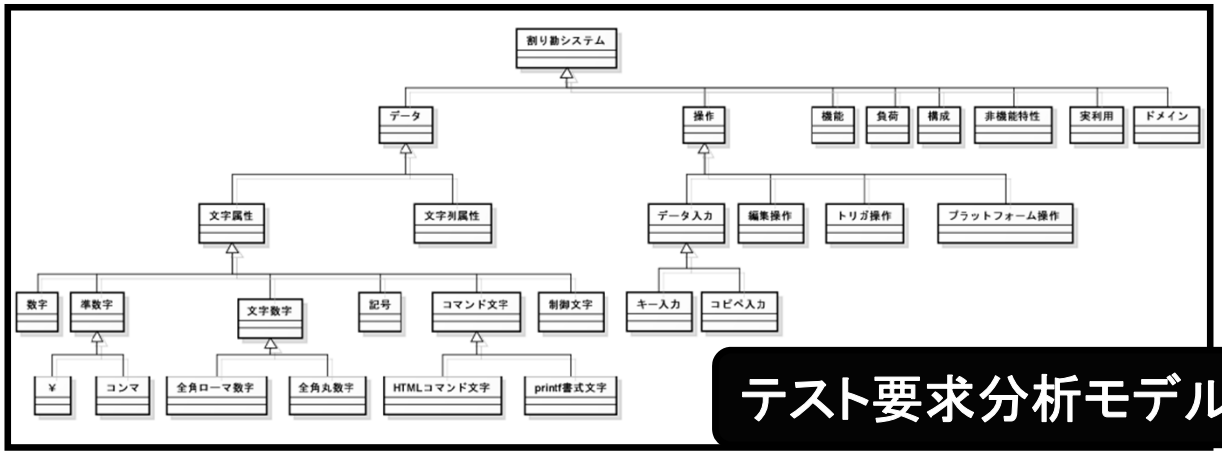
- テストアーキテクチャ設計では、ソフトウェア設計のような設計技術やモデリング技術が必要になる
  - » テストコンテナ間の関係がなるべく少なくなるように分割する ＝ 結合度を下げる
  - » テストコンテナの表している意味が明確になるように分割する ＝ 凝集度を高める
  - » テストスイート（テストケース群）の品質特性を高めるように設計する
    - ・ 高い保守性が必要、自動化しやすい、など
  - » テストデザインパターンを抽出、蓄積、活用するとよい



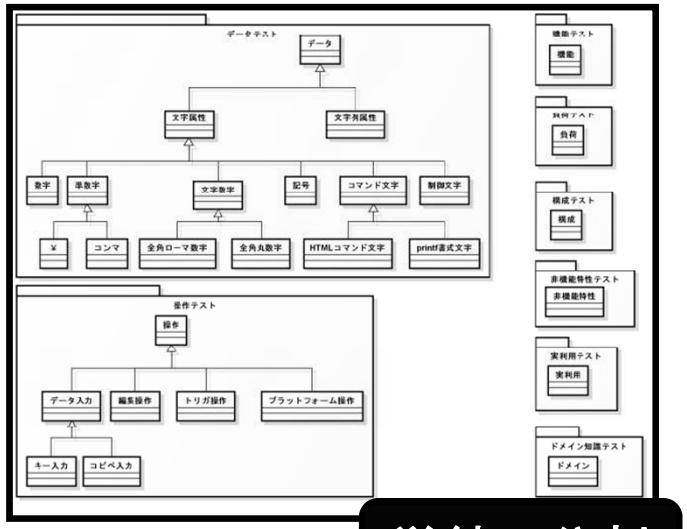
テストアーキテクチャ設計：テストスイートの品質特性

- テストスイートにも品質特性が考えられる
  - アーキテクチャを検討するほど大規模で複雑なテストスイートには、テストスイート自身の品質特性を考慮しなければならない
  - しかしテストスイートの品質特性についてはほとんど議論されていない
  - ソフトウェアの品質特性は十分議論されている
    - » ISO/IEC9126sの品質特性：機能性、信頼性、使用性、効率性、保守性、移植性など
    - » ソフトウェアの品質特性はテストスイートの品質特性と同じではないが、参考程度にはできるかもしれない
- テストスイートの品質特性の違いによって、異なるテストアーキテクチャを設計する必要が生じる
  - 保守性、派生容易性、自動化容易性、網羅保証性、ピンポイント性などがあるかもしれない

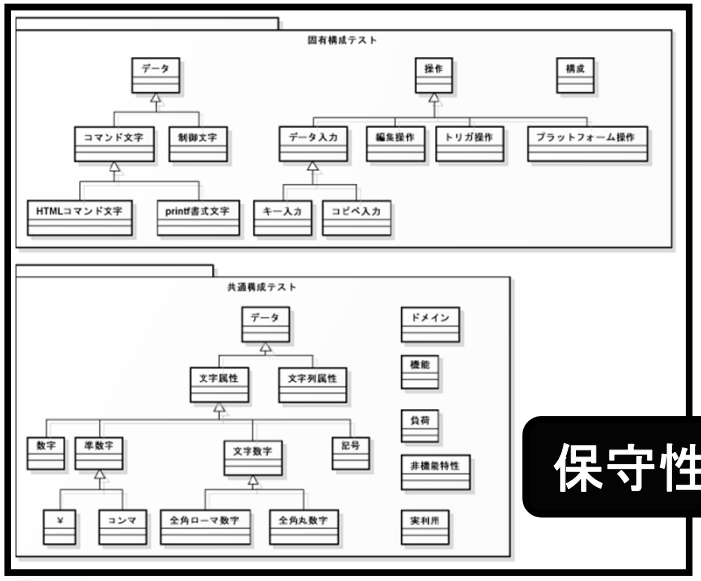
テストスイートに求められる品質特性によって異なるテストアーキテクチャの例



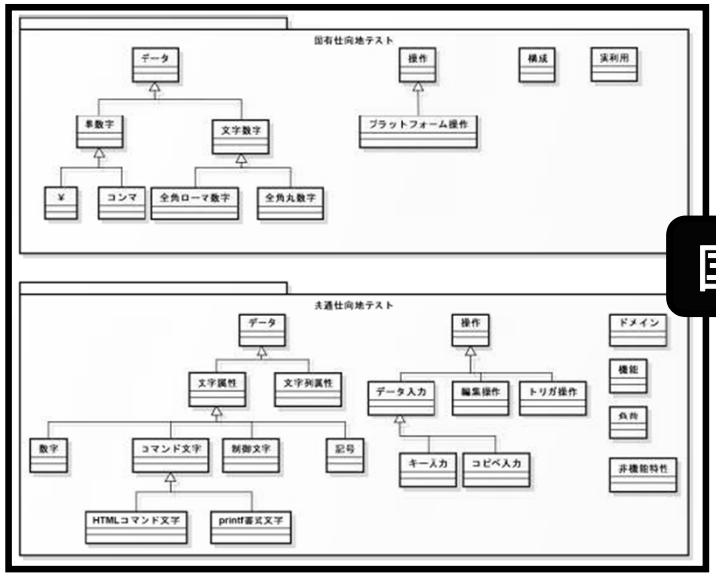
テスト要求分析モデル



単純に分割



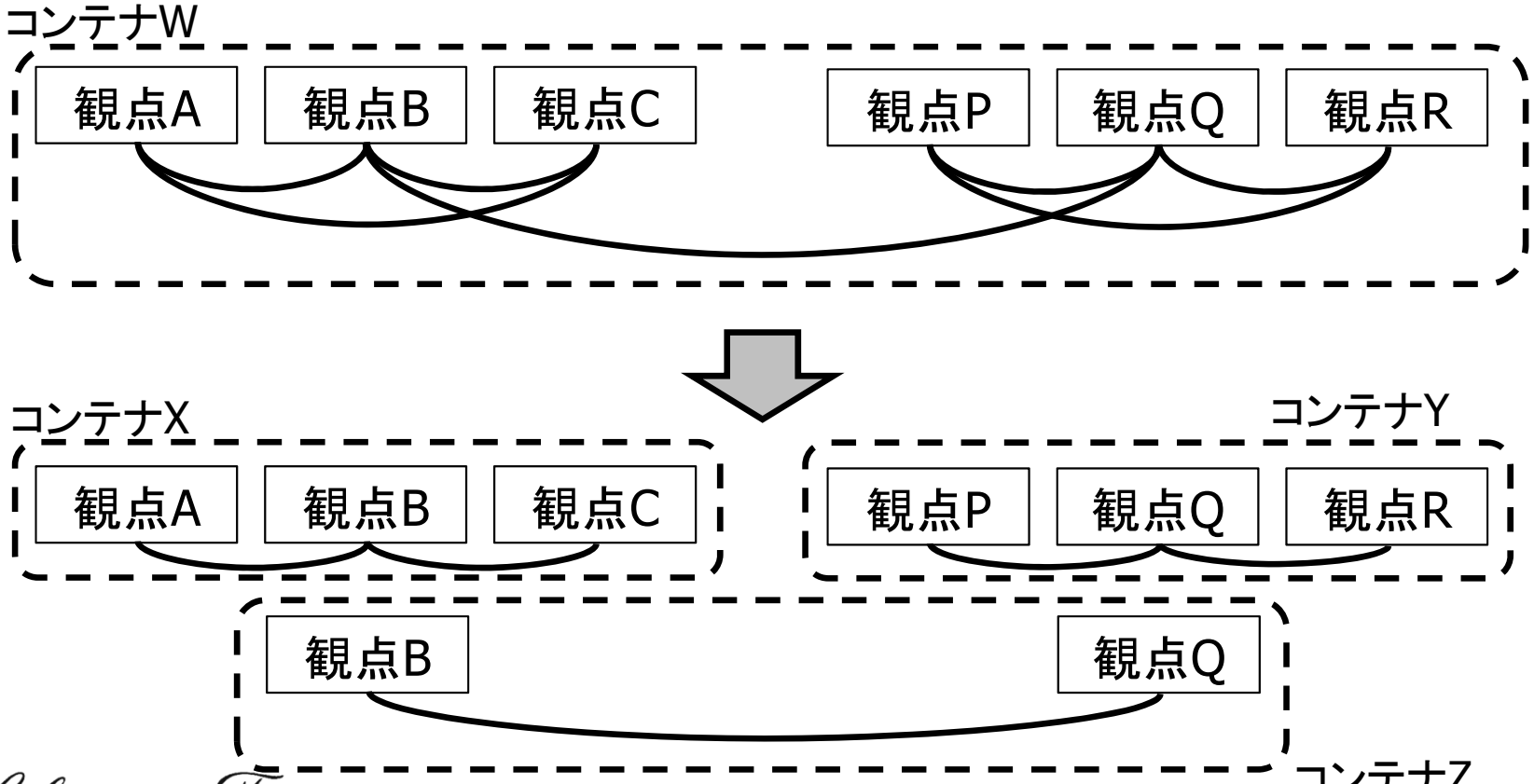
保守性重視



国際化重視

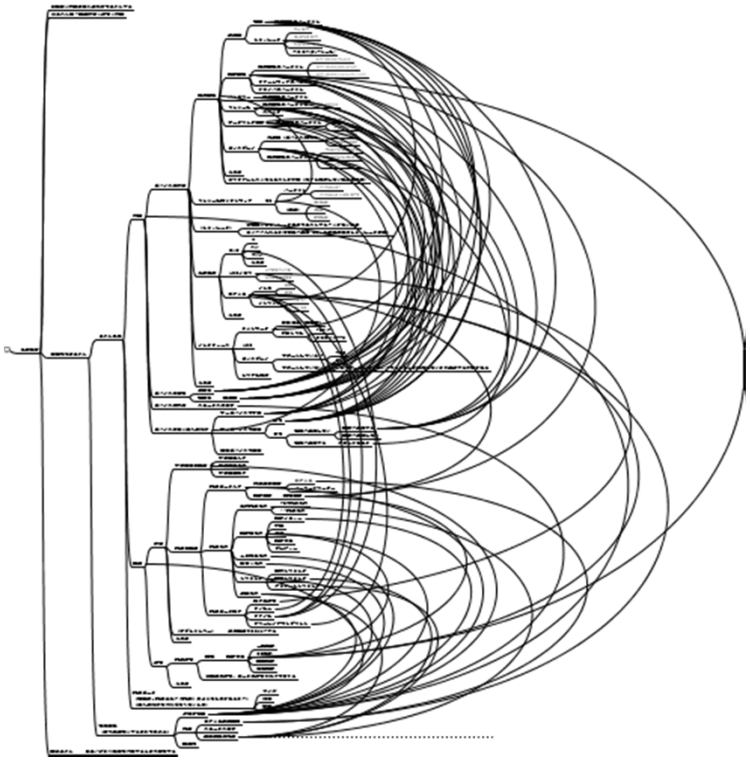
テストアーキテクチャ設計：テストデザインパターン

- 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

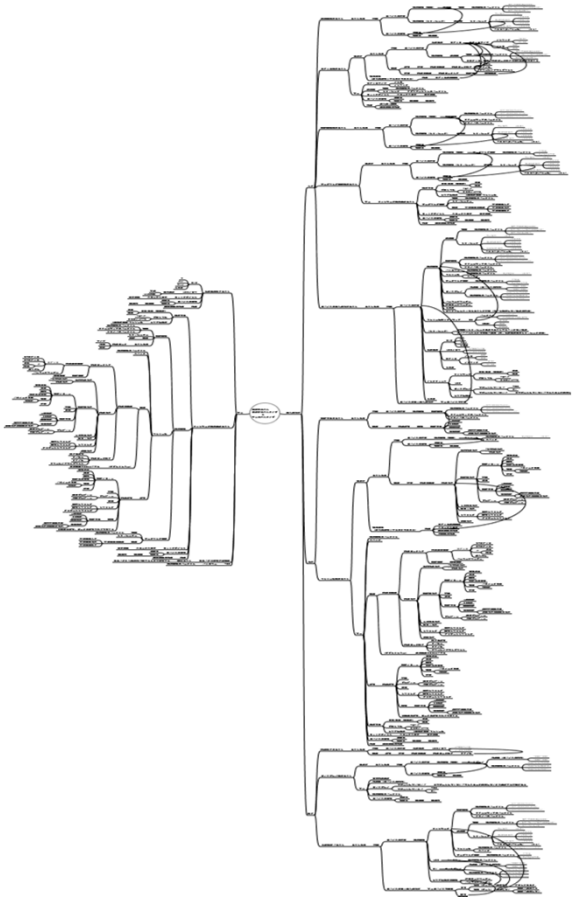


テストアーキテクチャ設計：リファインの例

- 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



テストアーキテクチャ設計：2) フレームモデリング

- 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ある
  - 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：
    - » 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

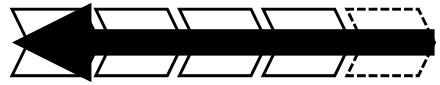
テスト詳細設計

- 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
    - » 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

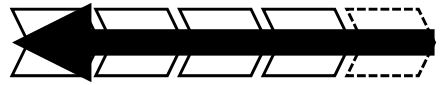
- テストケースが生成できたら、テスト実装を行う
  - テスト対象のシステムやソフトウェアの仕様、テストツールなどに合わせて テストケースをテストスクリプトに具体化していく
    - » 手動テストスクリプトの場合もあるし、自動化テストスクリプトの場合もある
  - テスト実装は、テスト対象のUI系の仕様や実行環境、ユーザのふるまいに 関するノウハウが必要である
- 集約
  - テスト実施を効率的に行うため、 複数のテストケースを1つのテストスクリプトにまとめる作業を集約と呼ぶ
  - 同じ事前条件や同じテスト条件、同じテストトリガのテストケース、 あるテストケースの実行結果が他のテスト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

Reverse VSTePによるテストの改善



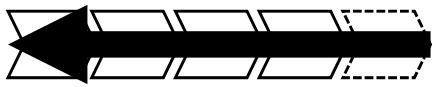
- Reverse VSTeP: テスト設計をリバースモデリングする手法
  - 要求からテストスクリプトを作成していく流れをForward VSTeP、逆をReverse VSTePと呼ぶ
  - Reverse VSTePとは、既存の(十分設計されていない)テストケース群から
    テスト観点モデルをリバースモデリングして改善する手法群である
- テスト観点モデルのリバースモデリングによる改善
  - 実際のテストケースを分析して、
    どういうテスト観点や関連でテストしようとしているかを列挙・整理し改善する
  - 実際に見逃したバグや検出されたバグ、テストケース外で検出されたバグ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

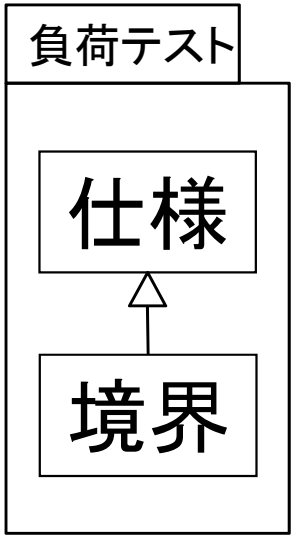


- ステレオタイプ分析による改善
  - テスト観点モデルの各詳細化関係の種類やMECE性を分析し、
    テスト観点が適切かつ網羅的に詳細化されるように
    子テスト観点を追加したりモデルをリファクタリングして改善する
- 不具合モード分析による改善
  - 見逃したバグや検出されたバグをパターン化して、
    ピンポイント型のテスト観点を追加・整理し改善する
- ディレイ分析によるテストアーキテクチャの改善
  - 見逃したバグや検出されたバグを分析して、
    実際のテストレベルやテストタイプを
    リバースされたテスト観点モデルにマッピングし、
    テストアーキテクチ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

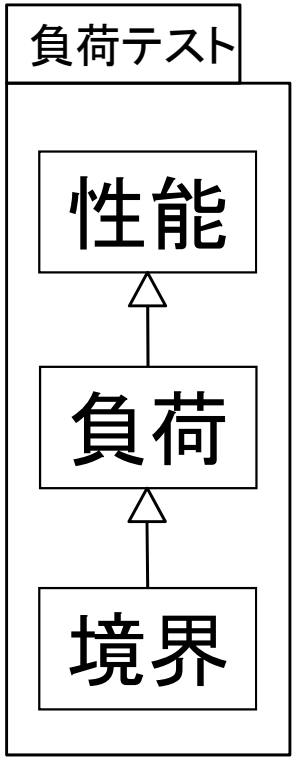
テスト観点のリバースは技術力把握にも有効



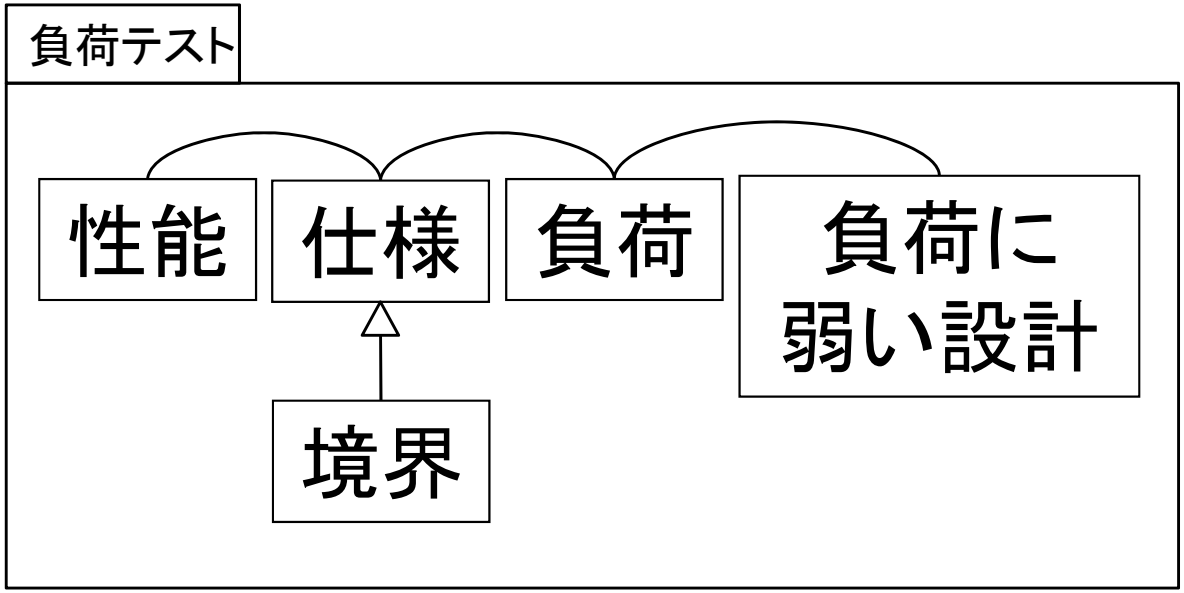
- テスト観点をリバースすると
  テスト設計者の意図が透けて見えてしまうので、
  テスト設計者の技術力が分かってしまう



様々な
負荷のかけ方が
漏れる



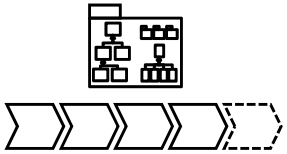
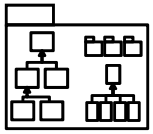
様々な状況で測るべき
性能が漏れる



技術力が必要

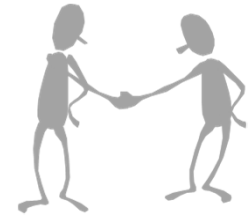
解説の流れ

- よくあるテストの現場の悩みのまとめ
- NGT/VSTePの概要と特徴
- NGTによるテスト観点図の概要
- VSTePによるテスト開発プロセスの概要
- NGT/VSTePを現場に適用する際のポイント

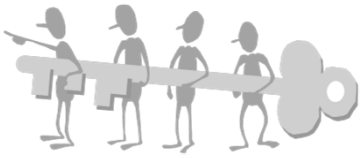
NGT/VSTePを現場に適用する際のポイント

- テストエンジニアのマインドを、
  表を埋める単純作業からクリエィティブなモデリングに変えていく
  - 単純作業だと思ってしまっている限り、モチベーションもスキルも向上しない
  - 単純作業の部分は自動化するように技術を高めていく
- できるところから小さく始めて小さく回し、
  効果を実感しながら進めることが重要
  - まずは腕の良いエンジニアに、テストスイートのごく一部から適用してもらうとよい
    - » いきなりVSTePというプロセスを全て天下り的に導入しない方がよい
  - まずはあまり網羅性を意識せず、テスト要求モデル（全観点網羅モデル）をつくり
    自分たちのテストの良さや悪さを実感してみるとよい
  - 常にNGT/VSTePのよさをエンジニアが実感しながら進めるようにする
  - 自分たちに必要だができていないことを明らかにして、さらなる導入を進める
- テスト観点図をコミュニケーションの道具として使う
  - 同じプロジェクトの異なるテストエンジニア同士で、役割の違うエンジニアと、
    そして顧客とのコミュニケーションの道具としてテスト観点を用いるとよい
    - » 皆でテスト観点図を囲んで議論し、納得感を深めていく
  - 網羅しているかどうか、質の高いモデルかどうかなどを
    皆で「納得」することがとても重要である



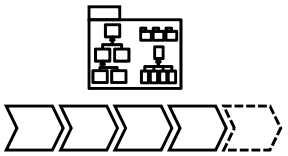
NGT/VSTePを現場に適用する際のポイント

- テスト要求分析・テストアーキテクチャ設計での
  モデリングスキルを高める
  - 観点の意味をブレないようにする／詳細化の種類を明示し整理する
  - テスト観点の抽象度をきちんと意識し、詳細度が丁寧に落とし込まれるようにする
    - » トップダウンとボトムアップを循環させながら詳細度を整えていく
  - 単に適当に組み合わせるのではなく、関連が発生する要因を推測する
  - テストデザインパターンを抽出し蓄積する
  - テストアーキテクチャスタイル（どんなビューがあるか）を蓄積する
  - 自分たちが気にしている・気にすべきテスト品質特性を明らかにする
- Forward VSTePの導入と合わせて
  Reverse VSTePで現状のテストを改善できるとよい



  - テストプロセスそのものの改善や、テストをきっかけとした開発プロセスの
    改善の道具としてNGT/VSTePを活用できるとよい
    - » 例）テストタイプやテストレベルの定義を見直す
    - » 例）固定3レイヤー法は継続するが、大項目・中項目・小項目のレビューをきっちり行う
    - » 例）上流工程で経験的に（アドホックに）行っている工程を文書化する
    - » 例）アドホックな派生開発をテスト観点図で整理していく

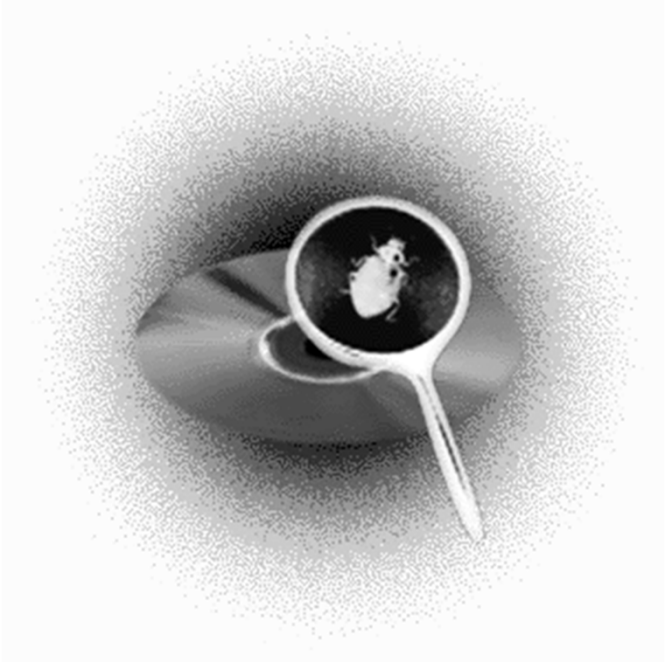
NGT/VSTePのまとめ



- 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である
    - » 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
- NGTによるテスト観点図
  - テスト観点を列挙・整理し全体像を把握するための記法である
- VSTePによるテスト開発プロセス
  - テストケース、テスト手順を区別してテスト観点をモデリングするプロセスである
  - 網羅性が高く重複が低く、俯瞰して全体像を把握しやすいテスト開発が可能になる
  - いきなり全てを導入せずに、できるところから小さく導入したり、Reverse VSTePを使って自分たちのテストを改善するとよい

NGT/VSTePは万能の道具ではなく、
現場の知恵を最大化するテスト開発方法論である

まずは小さく適用してみて下さいな



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