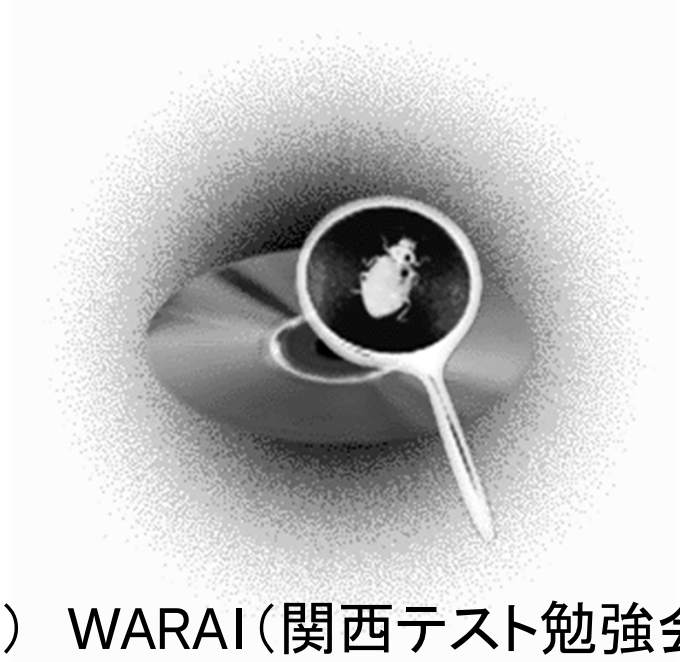
テスト観点に基づくテスト開発方法論
VSTePの概要



2013/4/3（水） WARAI（関西テスト勉強会）スペシャル

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）

本日のポイント

- 理解すべきポイント
  - テスト観点とは何か
  - テスト開発プロセスの考え方
    - » 特にテストアーキテクチャ設計の考え方
  - ツリー状のテスト観点図の描き方
    - » 詳細化と関連
  - テスト観点のモデリング
    - » リファインのためのモデリングパターン、剪定と確定
  - テストアーキテクチャ設計の考え方
    - » テストコンテナ分割、テストフレーム構築、テストスイートの品質特性
  - テスト詳細設計・テスト実装の考え方
  - 進んだ話題
    - » Reverse VSTeP、テストのプロダクトライン

} 今日はここまで

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網羅的に設計する必要がある

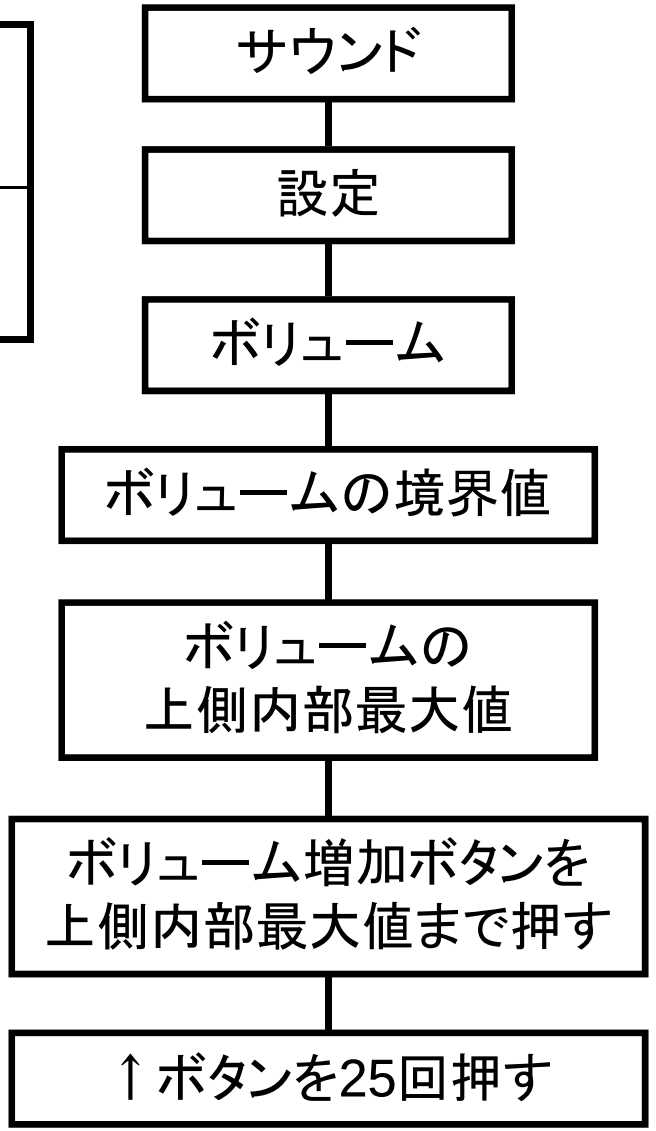
よくある大・中・小項目型テスト設計の書き方

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

丁寧に詳細化しないと漏れが発生する

大項目	中項目	小項目	テスト ケース
サウンド	設定	ボリューム	↑ボタンを 25回押す

- 大・中・小項目型テスト設計の場合
  「ボリューム」と「↑ボタンを25回押す」との
  間に抽象度の差があるため、
  列挙する際に漏れが発生しやすい
  - ボリュームの下側内部最大値が漏れる
  - 他のウィンドウでのボリューム増加が漏れる
- 何も見ずに都道府県の名前を
  漏れなく思い出せますか？
  - 地方を列挙し、地方ごとに思い出すと網羅が簡単



大・中・小項目型テスト設計では問題が多い

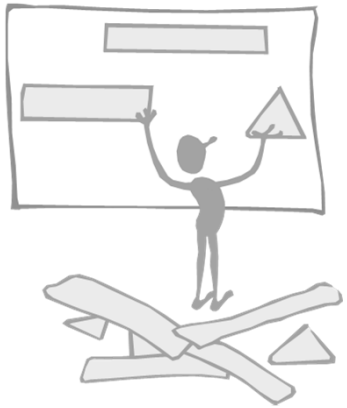
- 詳細化のレベルが揃わない
  - 偏った詳細化をしてしまう
  - テストケース群ごとに詳細化の偏りにばらつきがある
- 異なるテスト観点の組み合わせを詳細化だと考えてしまう
  - 機能＋環境＋データ、機能＋データ＋GUI
- テスト設計で考慮する必要の無いものが入ってしまう
  - 機能＋データ＋前提条件
- 異なるテスト観点にぶら下げているので網羅できない
  - 機能＋状態＋イベント
    - » 本来は状態遷移網羅をすべきであり、機能網羅で代用すべきではない
- テスト観点の詳細化を行わない
  - テストカテゴリ＋機能＋データ
    - » 負荷にも色々あるはず．．．。
- 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
- どんな観点に着目しているのかを俯瞰できない

非常に多くの観点を網羅的に設計できる記法ではない

テスト計画やテスト戦略をどうやって立てる？

- テストの専門書を開いてみると
  - いろいろなテスト設計の技法が書いてある
    - » 境界値テスト、制御パステスト、状態遷移テスト．．．。
  - 言っていることは分かるし、正しいと思うのだが、
    どう使ってよいのやら、よく分からない
    - » 果たして、どの技法をいつ使えばよいのだろう？
  - だから適切なテスト計画やテスト戦略を立てなければいけない
- テストの「観点」は、テスト計画やテスト戦略の構成要素である
  - 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

テスト観点に着目した
テスト計画やテスト戦略立案が必要



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
    - » 我々はを提案している

ソフトウェアテスト開発プロセスの基本的考え方

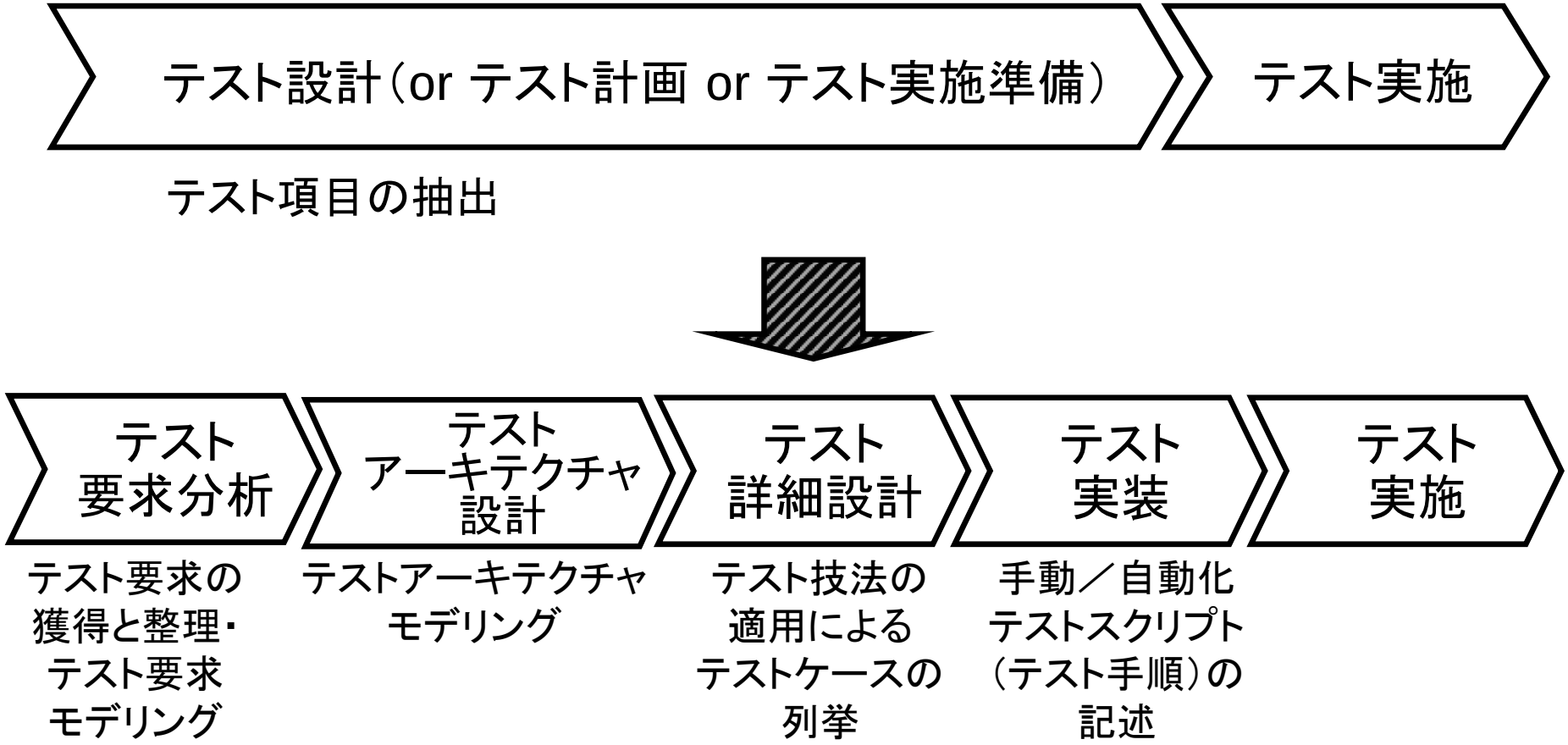
- テストケースを開発成果物と捉え、
  ソフトウェア開発プロセスと
  ソフトウェアテスト開発プロセスを対応させよう

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）

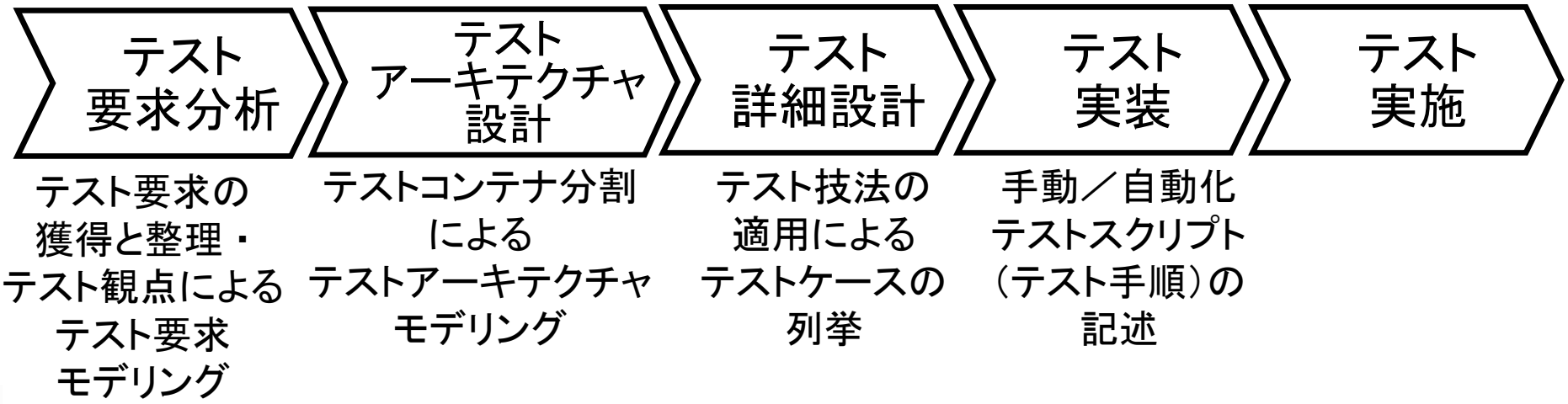
旧来のテストプロセスでは粗すぎる



テスト観点に基づくテスト開発方法論：VSTePの概要

- 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】



一般的なテスト要求/アーキテクチャモデルの例

- 一般的なテスト要求モデル・テストアーキテクチャモデルの例
  - 仕様と同じ構造なので、モデルは不要
    - » もしくは仕様書の章立てそのまま
  - 機能の一覧
  - 品質特性や非機能特性の一覧
  - 自然言語によるテストの概要記述
  - 経験的なテスト項目表の大項目の一覧
  - テストタイプやテストカテゴリの一覧
  - テストフェーズやテストレベルの一覧
  - テスト詳細設計技法の一覧
    - » もしくはテスト詳細設計モデルの一覧（制御パスモデルや状態遷移モデルなど）

テスト要求/アーキテクチャモデルの例:ゆもつよメソッド

- HPの湯本さんが使っている手法

テストカテゴリ / 機能(群)		入力・表示		計算処理			エラー処理	排他処理	インタフェース		状態	
		アドレスの入力	アドレスの変換	暗号化	エンコード	文字数カウント	タイムアウト	既受信メールの編集	SMTP	IMAP4	サーバ未接続	サーバ接続中
メール送受信系機能群	メール送信											
	メール返信											
	メール転送											
メール編集系機能群	文字の入力											
	文字の削除											
メール管理系機能群	メールの削除											
	フォルダの作成											
	フォルダの移動											

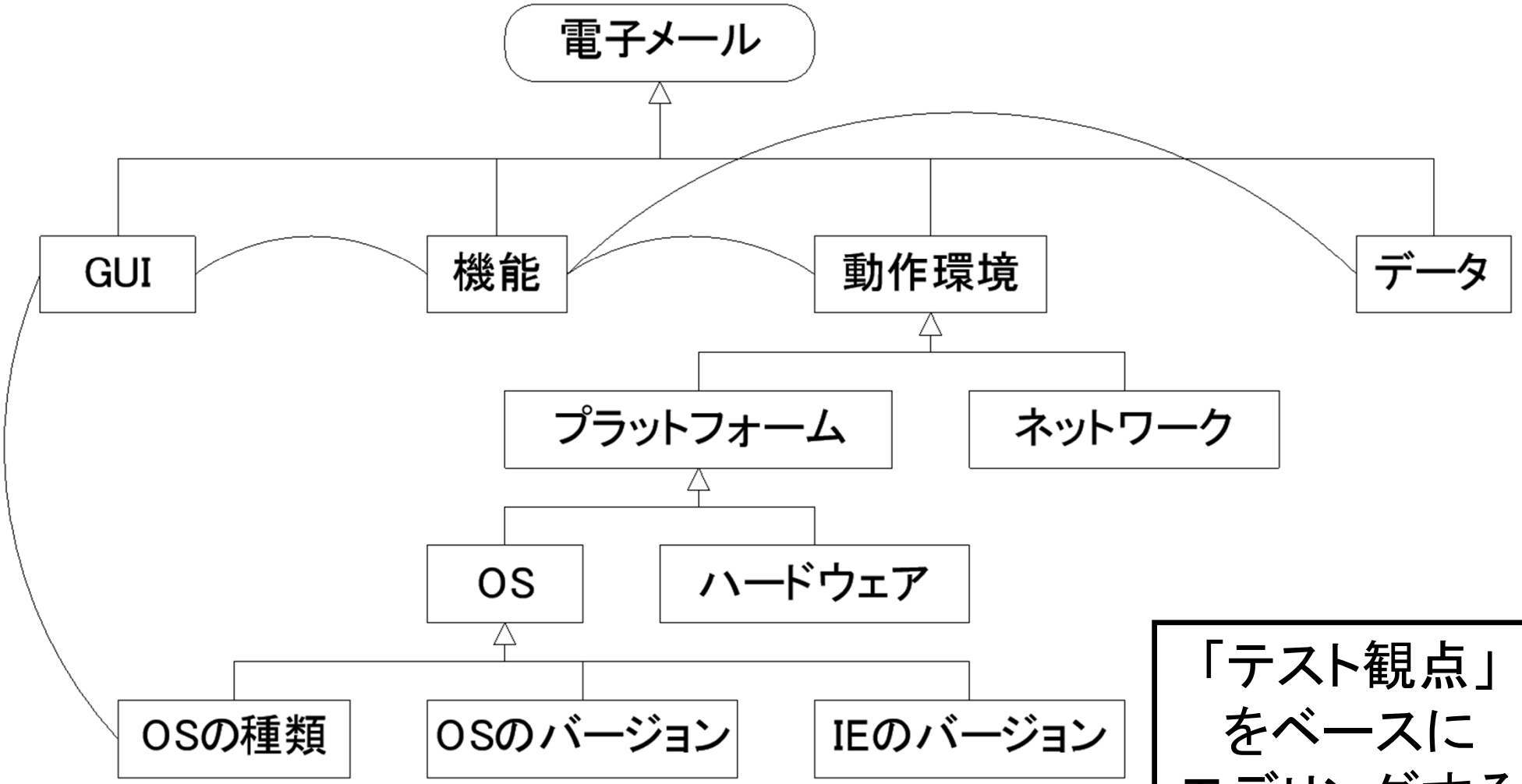
テスト要求/アーキテクチャモデルの例：HAYST法-FV表

- FV表：Function Verification Table
  - 「ソフトウェアテストHAYST法入門」より
    - » 吉澤 正孝/秋山 浩一/仙石 太郎, 日科技連出版社, ISBN4-8171-9228-3

番号	機能	検証内容	使用するテスト技法
1	商品を検索する	・希望する商品が速やかに 検索できること	
1.1	フリーワードで検索	・存在する商品名で検索できること ・存在しない商品名を入力したときに エラーとなること	・商品名をHAYST法で設計
1.2	商品カテゴリから検索	・リンク切れがないこと	・総探索を自動化処理で実施
2	数量を選択し購入する	・数量を入力し計算ボタンを押すことで 価格表示が変わること ・ラッピングの指示が反映されること ・購入ボタンで画面が遷移すること ・途中でブラウザが閉じられても 購入が継続できることを確認すること	・HAYST法

テスト要求/アーキテクチャモデルの例：NGT

NGT: Notation for Generic Testing



「テスト観点」
をベースに
モデリングする

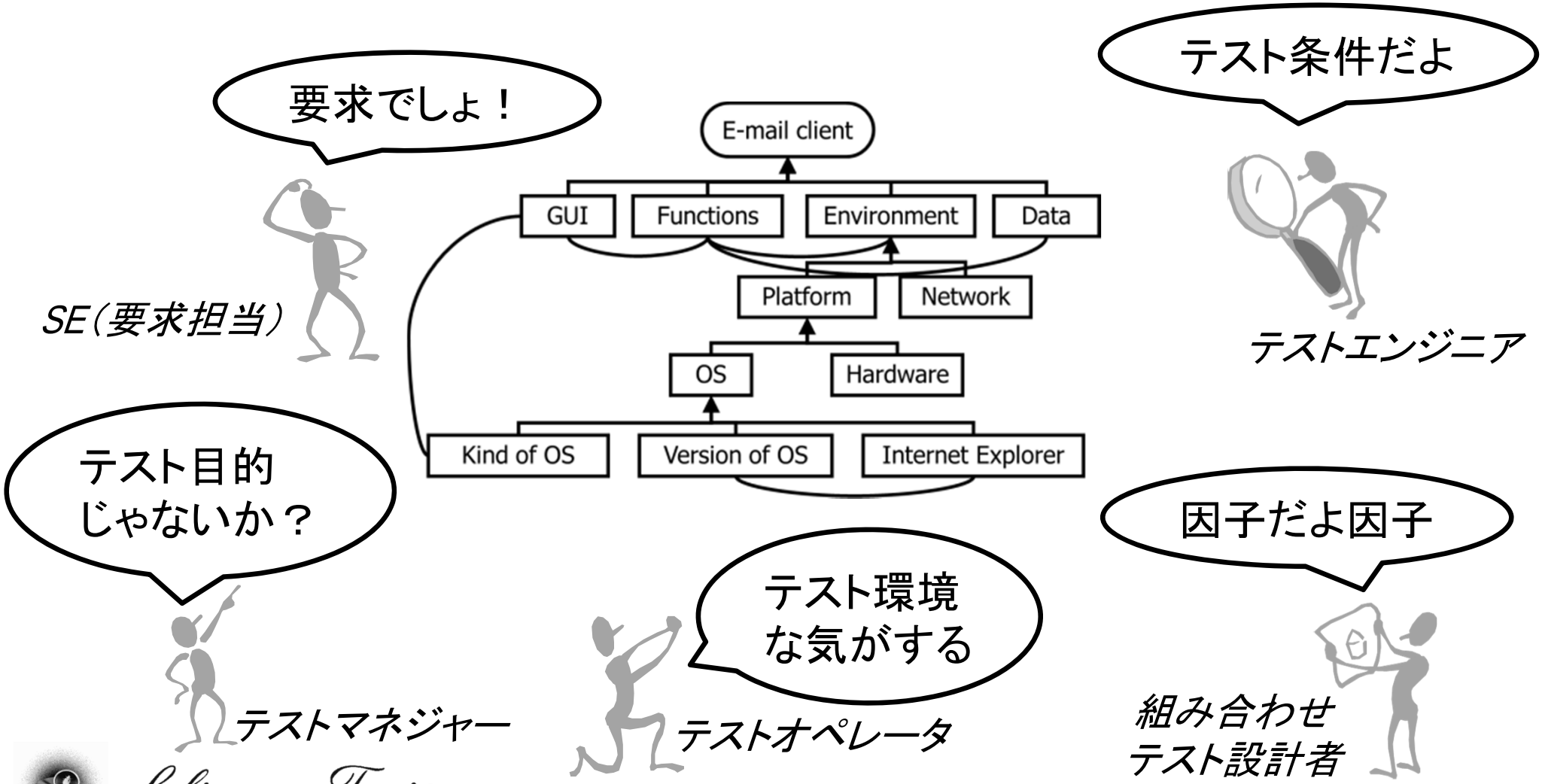
テスト観点とは何か

- 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、テスト対象が達成すべき性質であり、階層構造を持つ
  - ある側面から抽象化されたテストケースと言うこともできる
    - » 網羅すべきもの
      - ・ 仕様、機能、データ、ユーザの使い方など
    - » 達成すべき特性
    - » テストアイテム（テスト対象）の一部
      - ・ 機能、サブシステム、モジュール、H/Wなど
    - » バグ（のパターン）
    - » テストタイプや技法をヒントにすることもできる
    - » テストレベルをヒントにすることもできる
    - » ソフトウェア開発時に記述する図表をヒントにすることもできる
  - 具体化してテストケースにならないものは基本的にテスト観点ではない

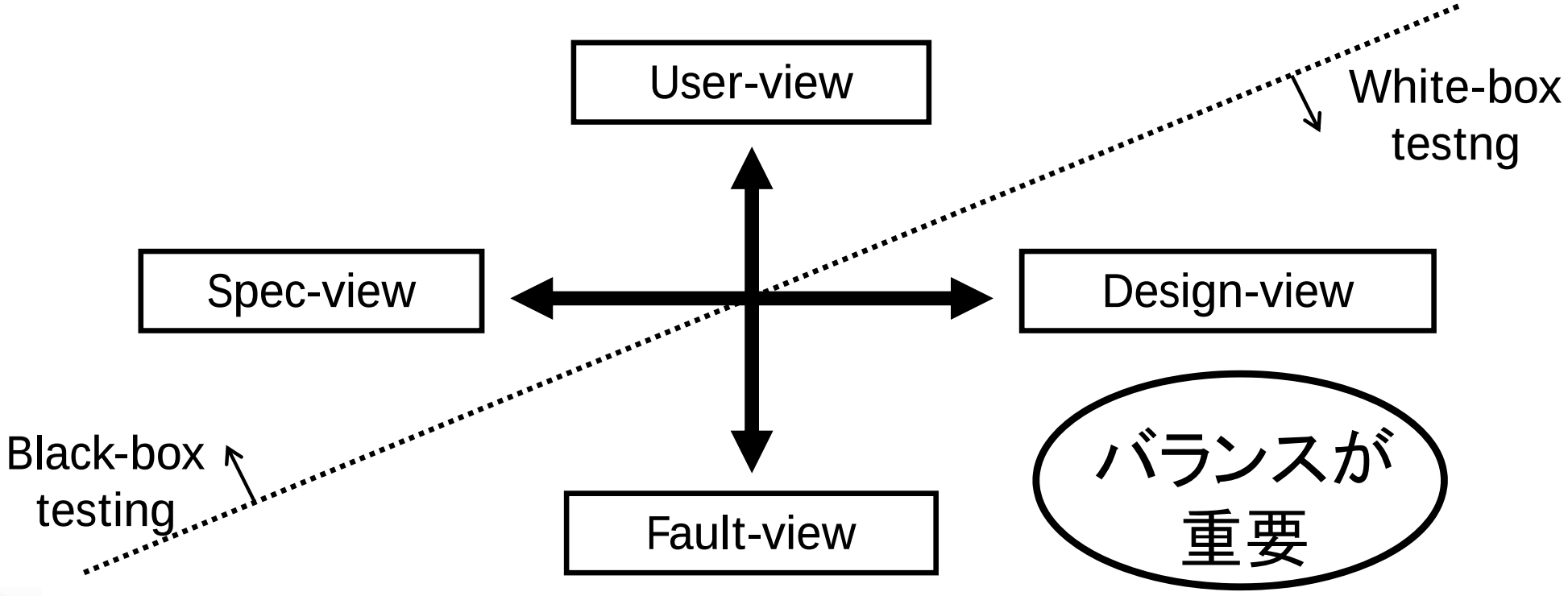
なぜ「テスト観点」と呼ぶのか？

- 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



Ostrandの4つの主なテスト観点

- User-view
  - ユーザが何をするかを考える
- Fault-view
  - 起こしたいバグを考える
- Spec-view
  - 仕様を考える
- Design-view
  - 設計やソースコードを考える



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、
    テスト設計・実装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

テスト要求分析

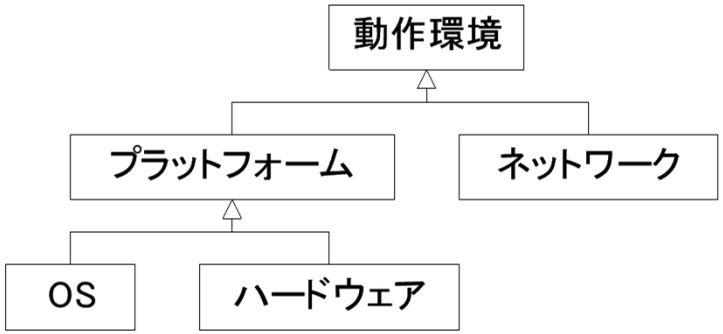
- テスト要求分析のインプットとなる情報を獲得し、テスト開発に必要な情報を選り分ける
  - テストマネジメントに必要な要求を選り分ける
    - » 後で「リスク」としてまとめていく
  - システムの完成像への要求と、システムの「出来」の情報を選り分ける
    - » ピンポイントテストやリスクとしてまとめていく
  - 顧客がトップダウンに指定してきた制約（テストツールなど）を選り分ける
    - » その制約が本当に必要なのかを検討し顧客に確認する
- テスト要求をモデリングする
  - 顧客から獲得した要求を、顧客が把握しているようにモデルとして記述する
  - 顧客の把握が不十分・不完全・不正確などの部分についてモデルを囲んで顧客と合意を取りながらリファインしていく
    - » あくまで顧客の要求の理解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、テストしやすくするのが目的ではない
  - モデルは図であっても表であってもリストであっても構わない
    - » NGT/VSTePでは、テスト観点図というツリー状の図を用いる

テスト要求分析のインプットとなる情報

- ソフトウェア開発で達成すべき顧客からの要求
  - システムの完成像への要求
    - » 例）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
- 顧客からの要求をソフトウェア開発で達成するための テスト開発プロセスへの制約や（他から与えられた）方針
  - テストプロジェクトへの要求
    - » 例）コスト、納期、人数、スキルや経験
  - （顧客から適用を要求された）テスト手段
    - » 例）テスト技法、シミュレータ、テスト環境など
- 顧客からの要求を達成するための前提
  - テスト対象の情報
    - » テスト対象の実現手段
      - 例）テスト対象のアーキテクチャ
  - バグが多そうなところの知識
    - » テスト対象の出来
      - 例）システムの不安な点、弱点、バグの多い点
  - 過去の類似製品のバグの知識
  - 上流のプロジェクトの様子
    - » 例）進捗状況、人の入れ替わり

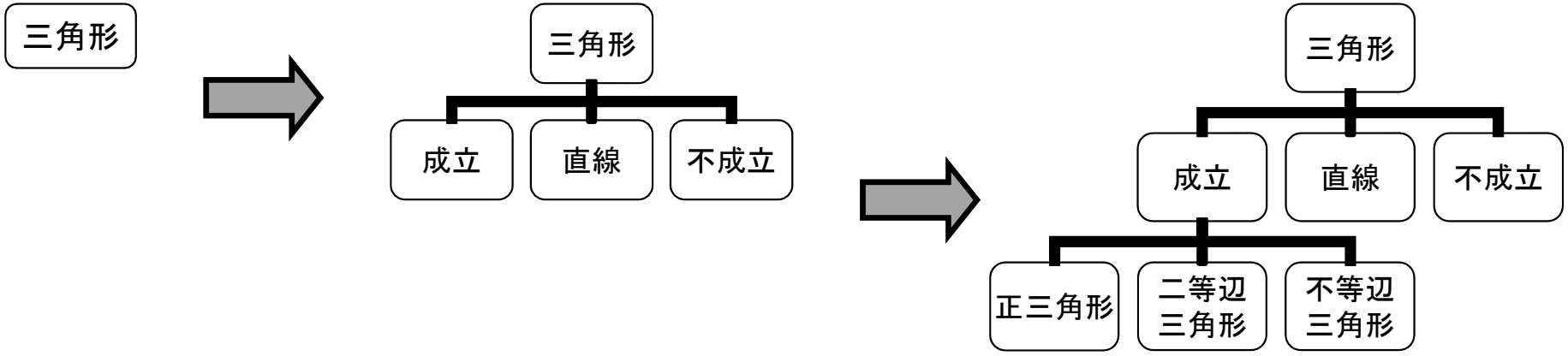
テスト要求モデリング：観点の列挙・詳細化・体系化

- まず大まかなテスト観点を列挙し、階層的に詳細化していく
  - 動作環境→プラットフォーム→OS→・・・
  - テスト観点を階層的に詳細化していく
    - » 最も詳細化されたテスト観点は、テスト設計で同値クラスになる
    - » 詳細化の作業は、近くからモノを見る（ズームイン）ようなものである
    - » テスト観点がMECEになっているかや、異なるステレオタイプの関係が混ざっていないかどうかに気をつける
  - 階層の最上位のテスト観点を「ビュー」と呼ぶ
    - » 最上位のテスト観点に代表されるため、階層全体をビューと呼ぶこともできる
    - » どのようなビューがどのような関連でモデリングされるか、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



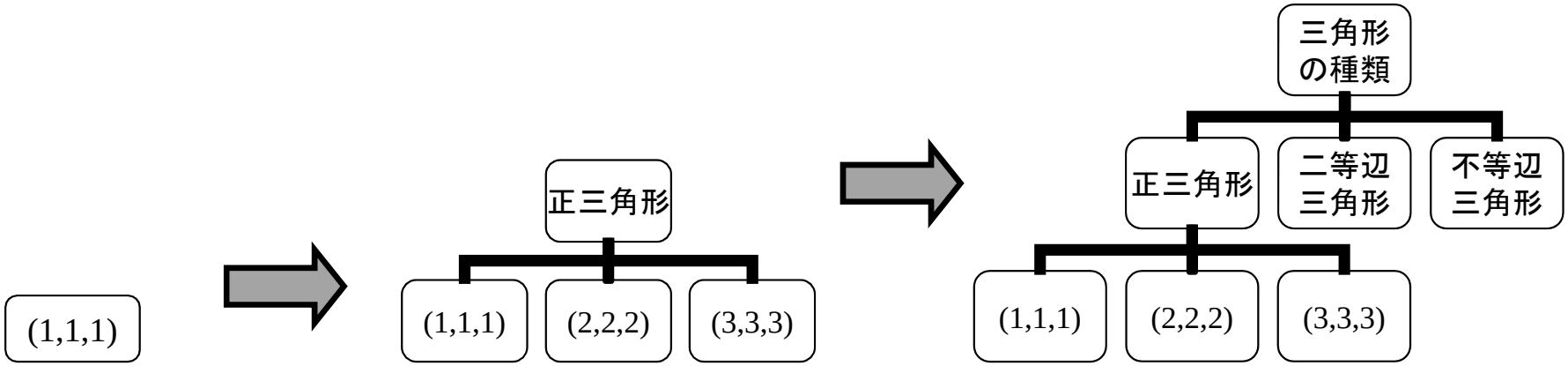
テスト要求モデル作成の2つの主なアプローチ

- 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



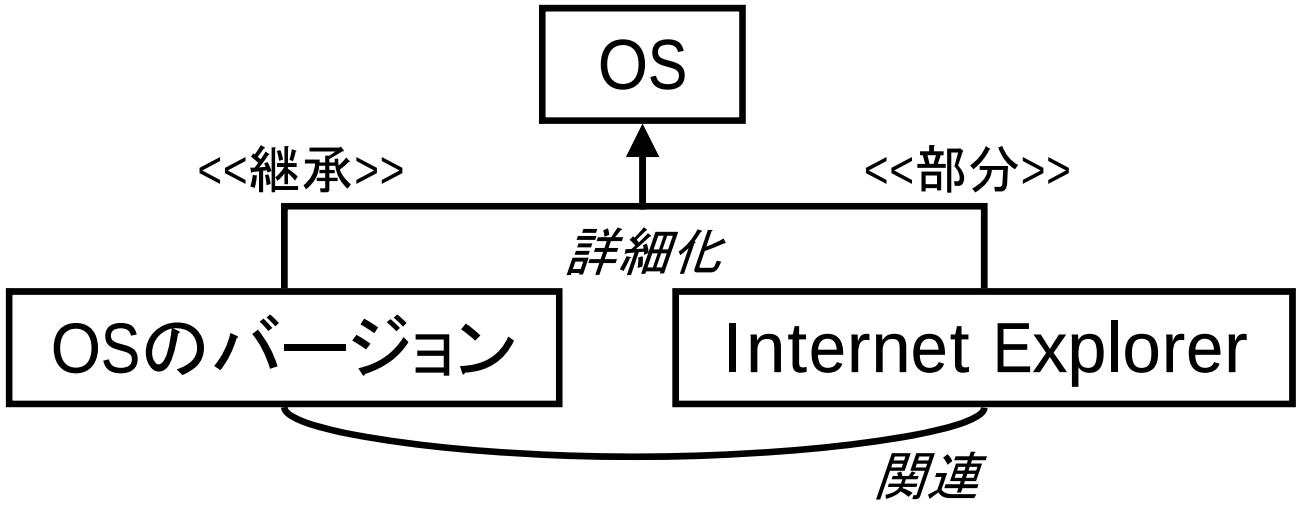
テスト要求モデル作成の2つの主なアプローチ

- 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



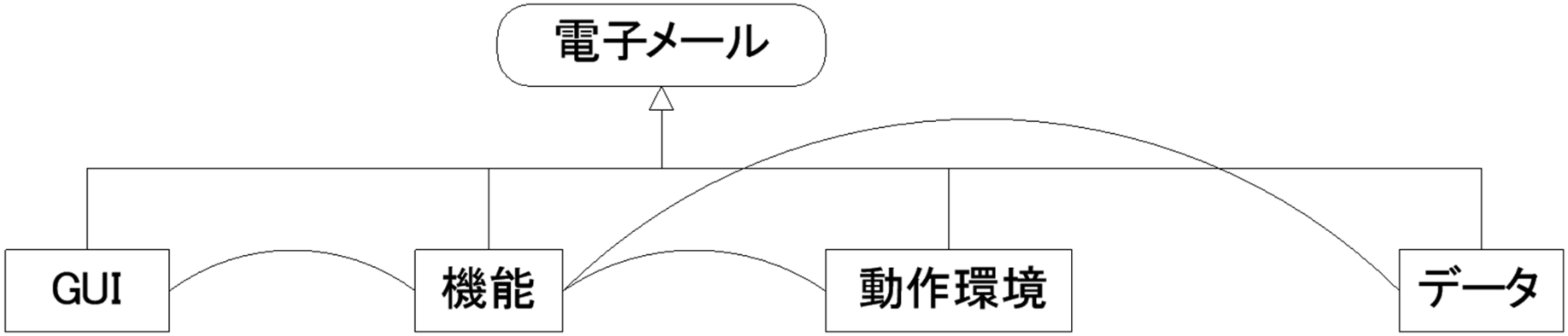
テスト要求モデリング：テスト観点間の関係

- テスト観点は2つの基本的な関係を持つ
  - 詳細化（Hierarchy relationships）
    - » テスト観点を階層的に詳細化してテスト条件を導き出し、直線の矢印で表す
    - » 継承、合成集約（部分）、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こともできる
  - 関連（Interaction relationships）
    - » 組み合わせてテストすることが必要なテスト観点を示し、曲線で表す
    - » 組み合わせるべき理由をステレオタイプで表すこともできる
- 関係の種類を“<<ステレオタイプ>>”を用いて表すと分かりやすい



テスト要求モデリング：関連の検討

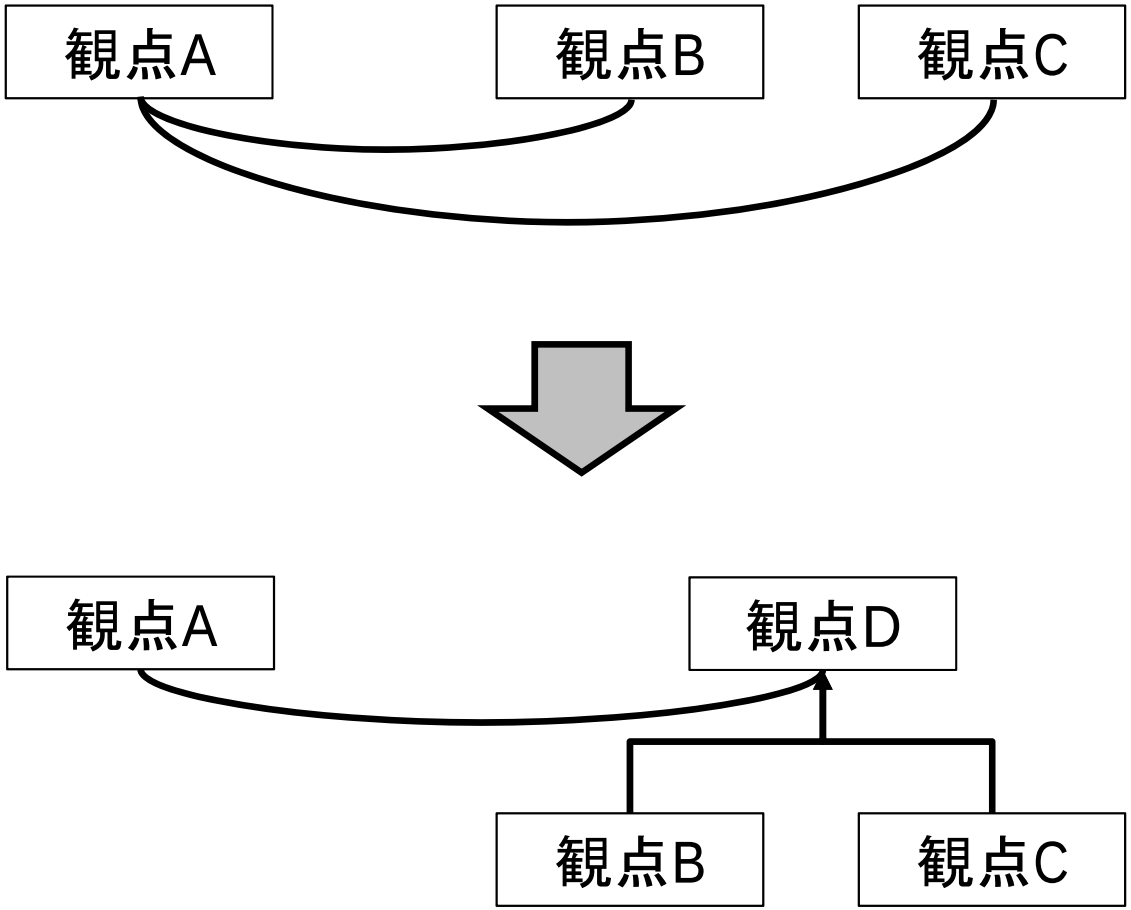
- 複数の観点を組み合わせるテストを「関連」で示す
  - 例）負荷テストでは、搭載メモリ量という観点と、
    投入データ量という観点を組み合わせてテスト設計を行う
    - » 搭載メモリ量と投入データ量のバランスによって、
      テスト対象のふるまいが変化するからである
  - 観点同士の依存関係を「関連」と呼ぶ
    - » テスト観点間の矢頭の無い曲線で記す
    - » 関連そのものを観点のように名前を付けて扱った方がよい場合もある
    - » 関連には条件－条件組み合わせ（<<combination>>）や
      条件－振る舞い組み合わせ（<<frame>>）がある



テスト要求モデリング：モデルのリファイン

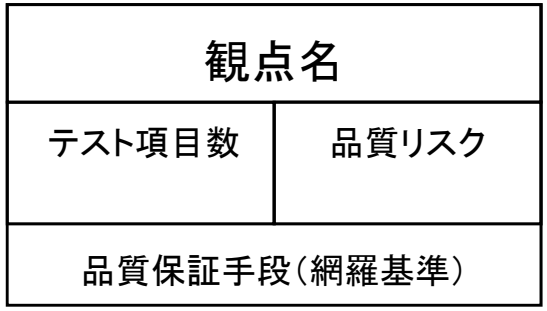
- 質の高いモデルにするために様々なリファインを行う
  - 詳細化
    - » 段階的かつMECEに詳細化されていない場合、
      テスト観点の追加・削除などを行ったり、
      詳細化のステレオタイプを明示して整理する（ステレオタイプ分析）
  - 整理
    - » 読む人によって意味の異なるテスト観点を特定し、名前を変更する
    - » テスト観点や関連の移動、分割、統合、名前の変更、パターンの適用、
      観点と関連との変換、観点と網羅基準との変換などを行う
    - » 本当にその関連が必要なのかどうかの精査を行う必要もある
  - 剪定
    - » テスト項目数とリスクとのトレードオフを大まかに行い、
      ズームイン・アウト、削除、網羅基準や組み合わせ基準の緩和などを行う
  - 確定
    - » 間引いたテスト項目の品質リスクを明らかにし、
      テスト設計全体の品質リスクを把握する

関連の整理のモデリングパターンの例：親観点統合



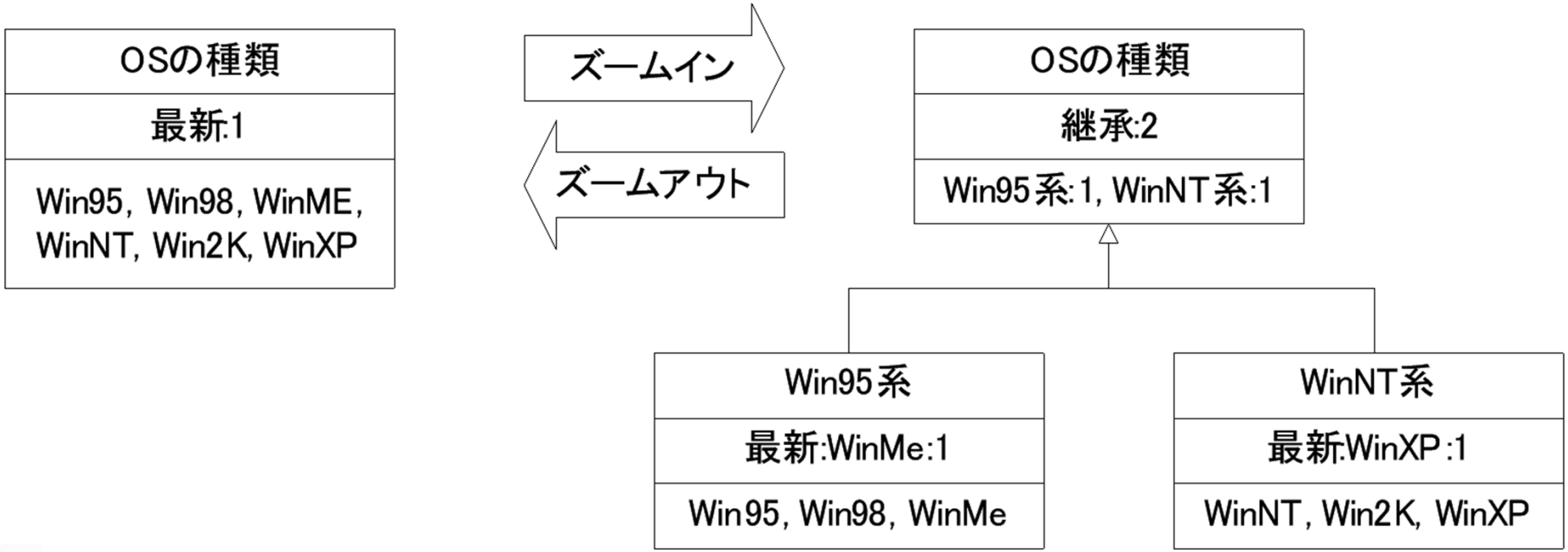
テスト観点と関連の剪定：項目数とリスクのトレードオフ

- テスト観点を詳細化しながら
  品質保証手段（網羅基準）と
  テスト項目数と品質リスクのバランスを取る
  - モデリングが進んできたら、品質保証手段（網羅基準）と
    テスト項目数と品質リスクを記述して概算する
    - » モデリングの進み具合に応じて、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、
    継承した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
  - テスト要求分析で行うことも、
    テストアーキテクチャ設計で
    行うこともある
    - » もちろん両方で行ってもよい



テスト観点と関連の剪定：ズームイン・ズームアウト

- 3つの方法でテスト項目数とリスクとのトレードオフ（剪定）を行いながら、計画されたテストエ数に合わせこんでいく
  - テスト観点内の網羅基準の緩和
  - ズームアウト
  - 関連の組み合わせ基準の緩和・削除

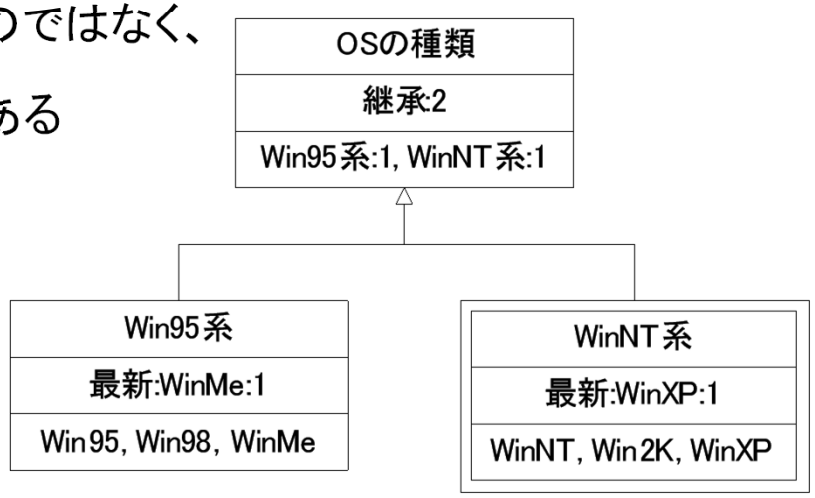


テスト観点と関連の剪定：ズームイン・ズームアウト

- 剪定の際には、テスト観点と同じ書式で関連のテスト項目数、品質リスク、品質保証手段（網羅基準）を記述する
  - 項目数やリスクを考える段階では、テスト観点のように
    各関連に品質リスクやテスト項目数を記述していく（関連ビューポイント）

「確定」によるテスト漏れリスクの軽減

- 剪定には、潜在するテスト漏れのリスクが付随してしまう
  - 同値分割がきちんと行われていない(ズームアウトされた)テスト観点
  - 網羅基準が緩いテスト観点や関連
  - 削除したり組み合わせ網羅基準を緩めた関連
- 潜在するテスト漏れのリスクを軽減する作業を「確定」と呼ぶ
  - 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場合「確定」する
    - » 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などは確定できる
    - » 品質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
    - » 子テスト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
  - 「確定」によってテスト設計全体の品質リスクを評価する
    - » 全てのテスト観点を確定させることが目的なのではなく、品質リスクの高いテスト観点を明示し他の工程で策を講じやすくすることが目的である



テストアーキテクチャ設計

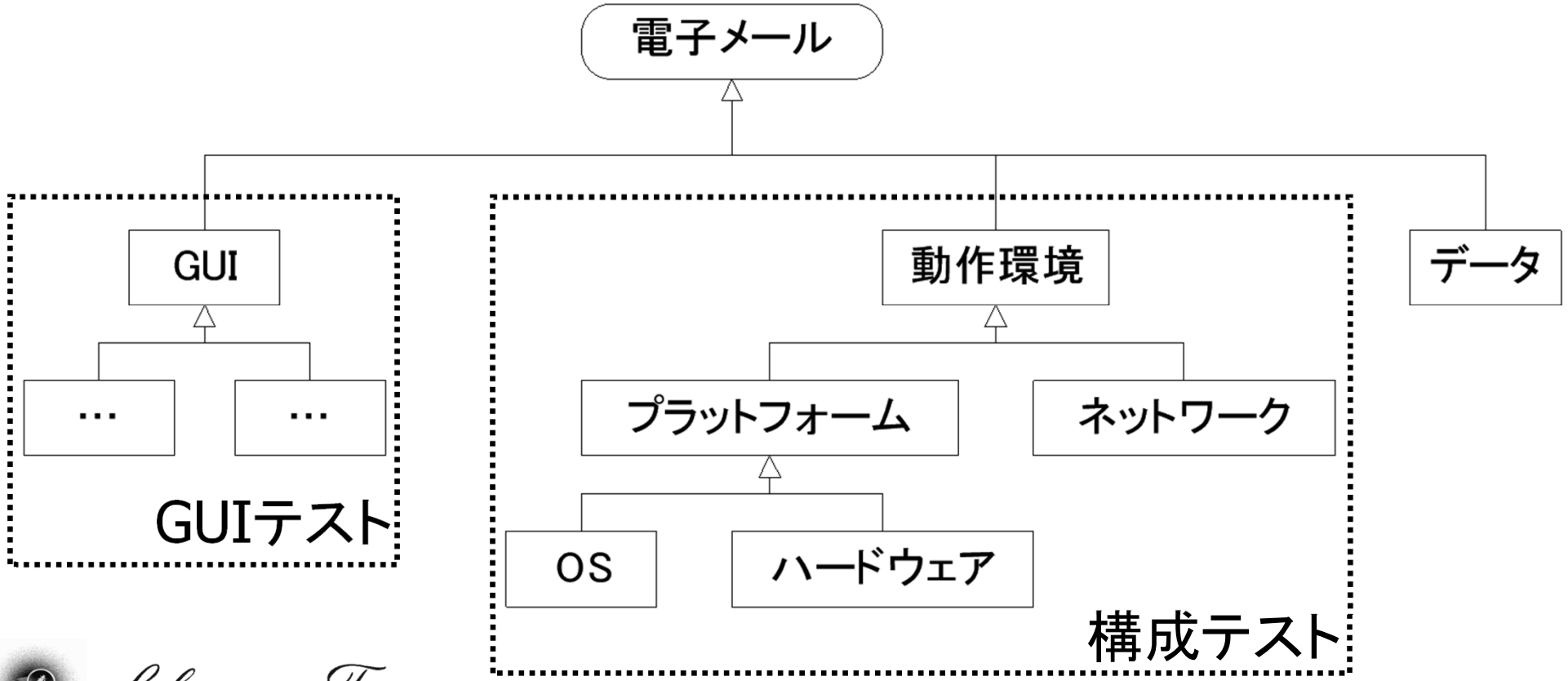
テストアーキテクチャ設計とは、
テストの全体像をテストタイプやテストレベルで
表すとともに、
テストタイプやテストレベルそのものを
（テスト観点を用いて）適切に設計することである

テストアーキテクチャ設計

- テスト要求モデリングで得られたモデルを
  テスト詳細設計・実装・実行しやすいようにリファインする
  - テストコンテナ分割: テスト観点群をまとめて、テストスイート全体を俯瞰する
    - » テストケース群全体のことをテストスイートと呼ぶ
    - » 複数のテスト観点をテストコンテナ(テストタイプやテストレベル、テストサイクル)に
      まとめる、すなわちテストスイート全体をテストコンテナに分割することで俯瞰性を高める
    - » 現場では経験的に様々なテストタイプやテストレベルに分割しているが、
      妥当な分割をしているかどうかは分からない
  - テストフレーム構築: テストコンテナをテストケースの構造に合わせる
    - » テスト観点が多すぎるとテスト条件などが増えてテスト詳細設計が難しくなる
    - » 各テストコンテナがテストケースを詳細設計するのに必要な種類の観点を含む、
      すなわちテストコンテナをテストフレーム(テストケースの構造)に合わせる必要がある
  - 剪定と確定も引き続き行う

テストコンテナ分割

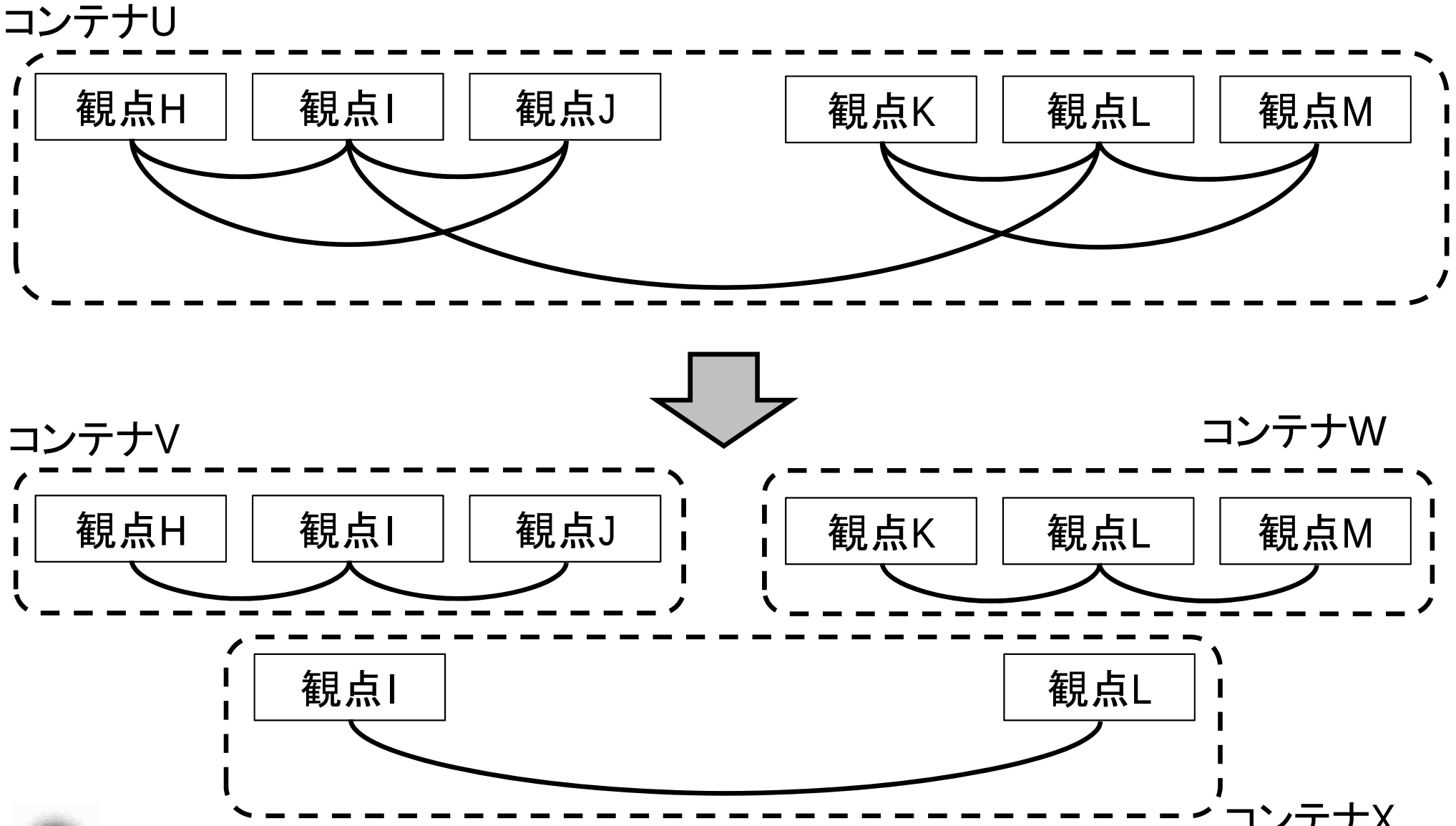
- テスト要求モデリングで得られたテスト観点モデルを
  より小さなテストコンテナに分割する
  - テスト観点をまとめたものがテストコンテナである
  - テストコンテナがテストタイプやテストレベルになる



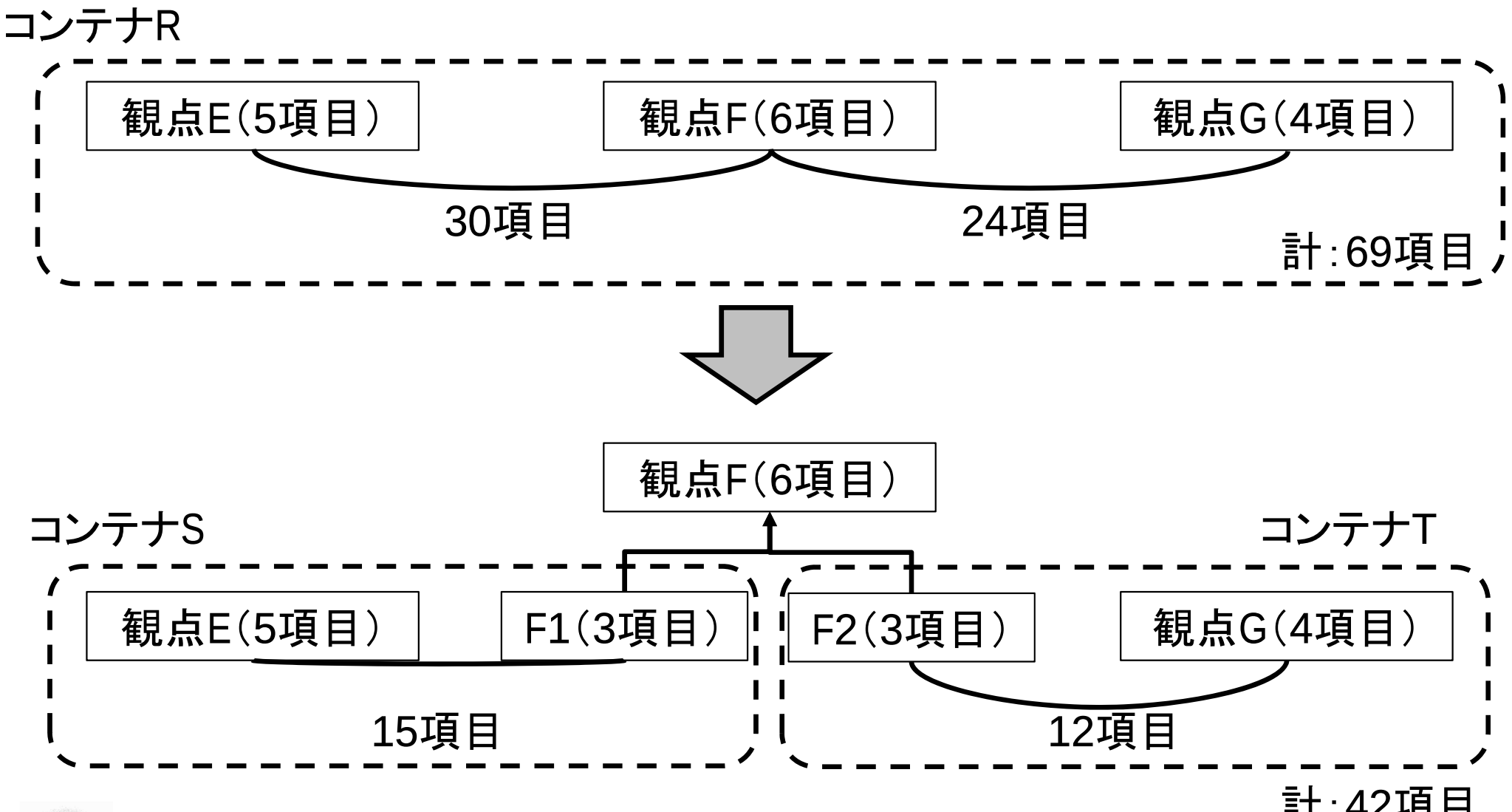
テストコンテナ分割

- テストコンテナ：テストタイプやテストレベルのように、複数のテスト観点やテストケースがまとまった塊のこと
  - テストタイプ：負荷テスト、構成テスト、性能テストなど
  - テストレベル：単体テスト、結合テスト、システムテストなど
- テストコンテナを用いてテストアーキテクチャを表す
  - 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タイプやテストレベルの一覧で表すのはよく行われる
    - » 分割しすぎると俯瞰性が低下し分かりにくくなる
  - 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
  - テストコンテナの意味合いを把握できるように分割する必要がある
    - » テストコンテナ間の関連が小さいように分割する（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）に求められる性質によって、異なるテストアーキテクチャを設計する必要がある
    - » 高い保守性が必要、自動化しやすい、など

コンテナ分割のモデリングパターンの例：クラスタ分割

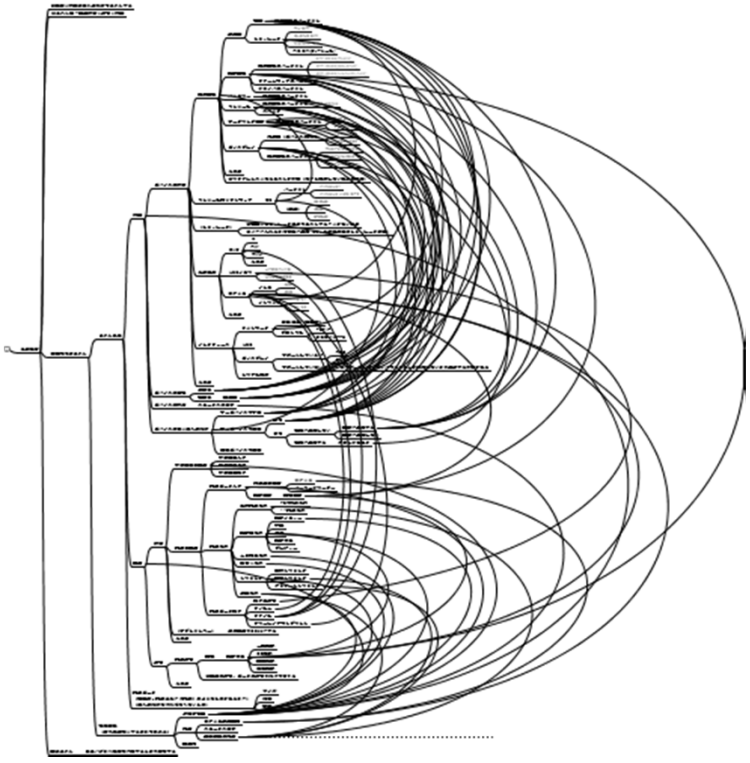


コンテナ分割のモデリングパターンの例：子観点分割

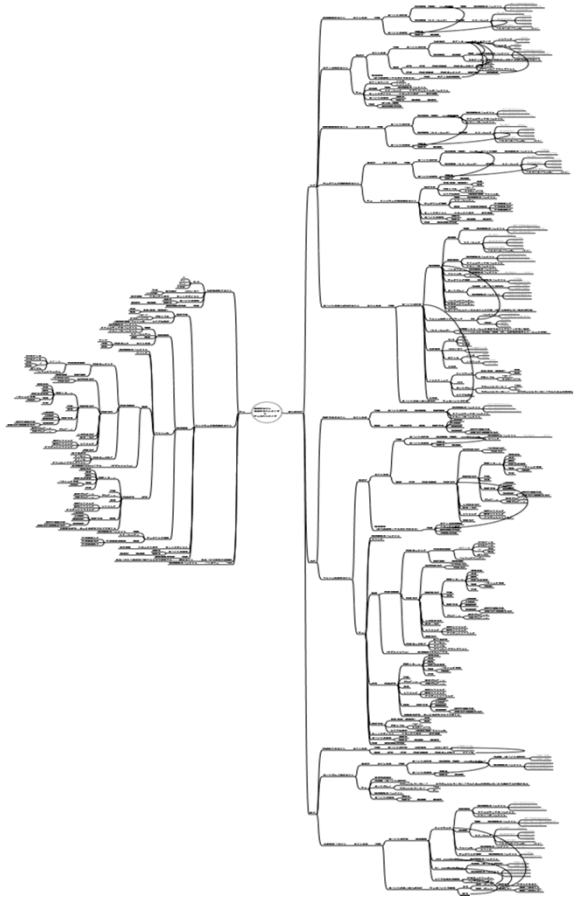


テストアーキテクチャ設計のリファインの例

- いくつかのモデリング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
  - 左図のテストアーキテクチャをリファインして右図のテストアーキテクチャにした
  - 右図のテストアーキテクチャは
    テスト詳細設計しやすくなっていると感じられる



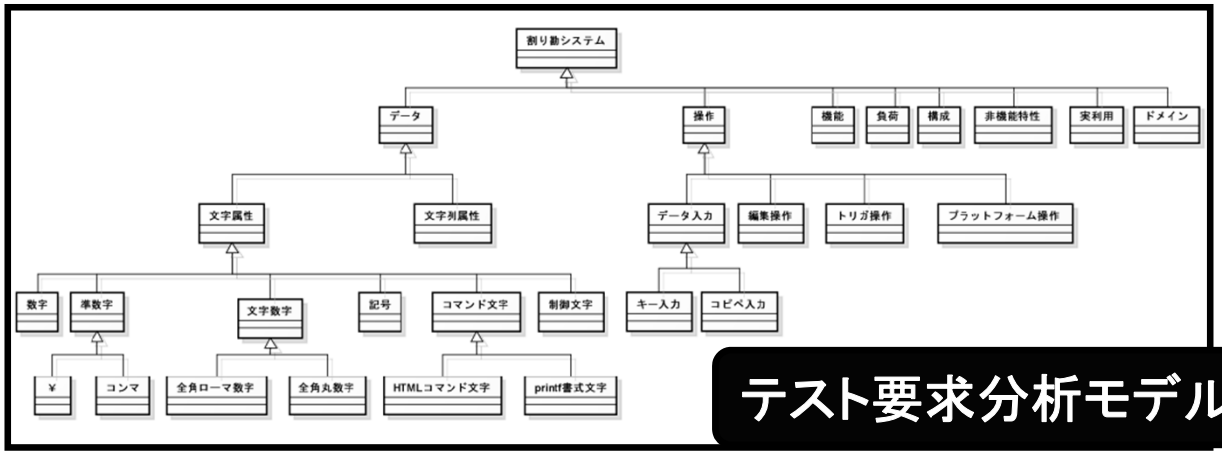
リファイン



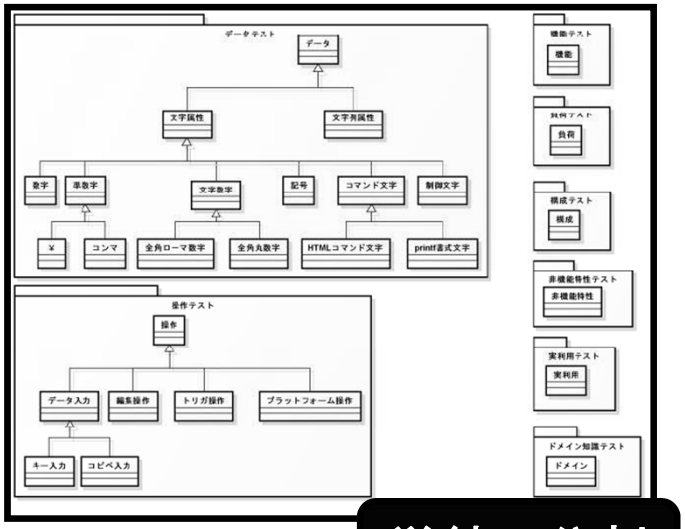
テストスイートの品質特性

- 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

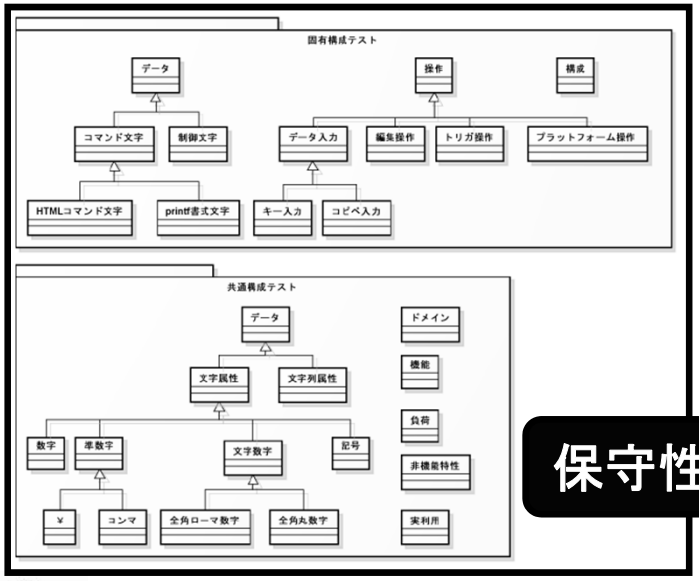
テストスイートに求められる品質特性によって異なるテストアーキテクチャの例



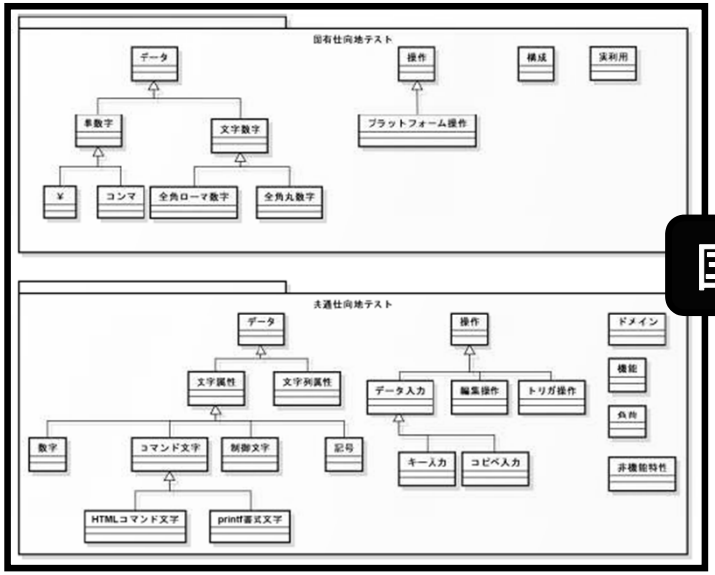
テスト要求分析モデル



単純に分割



保守性重視



国際化重視

テストフレーム構築

- テスト観点を十分に詳細化していても、テスト観点によってはそのままテスト詳細設計に移行できない場合がある
  - 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？
- テスト詳細設計が容易になるよう、テストコンテナにテストケース構造（テストフレーム）を反映させる
  - テストフレームとは、テストケース構造に合わせたテスト観点の組み合わせである
  - テストフレームの例
    - » テスト条件＋テスト対象
    - » テスト条件　＋振る舞い
    - » テスト条件＋テスト対象＋振る舞い
    - » テスト条件　＋狙っているバグ
    - » テスト条件＋テスト対象＋振る舞い＋狙っているバグ
  - テストコンテナ内のテスト観点に、テスト条件系観点、テスト対象系観点、振る舞い系観点、バグ系観点などが含まれているようにする
    - » 1つのテストコンテナに複数のフレームが含まれることもある
    - » テスト詳細設計やテスト実装で自明に分かる場合はテストフレームを組まなくてもよい

テスト詳細設計・テスト実装

- 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、テスト詳細設計を行っていく
  - 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テストケースを生成していく
    - » 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
    - » この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
- テストケースが生成できたら、テスト実装を行っていく
  - 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
  - 複数のテストケースを1つのテストスクリプトに集約して効率化を行う
    - » 同じ事前条件やテスト条件、テストトリガのテストケースを集約していく

Reverse VSTePによるテストの改善

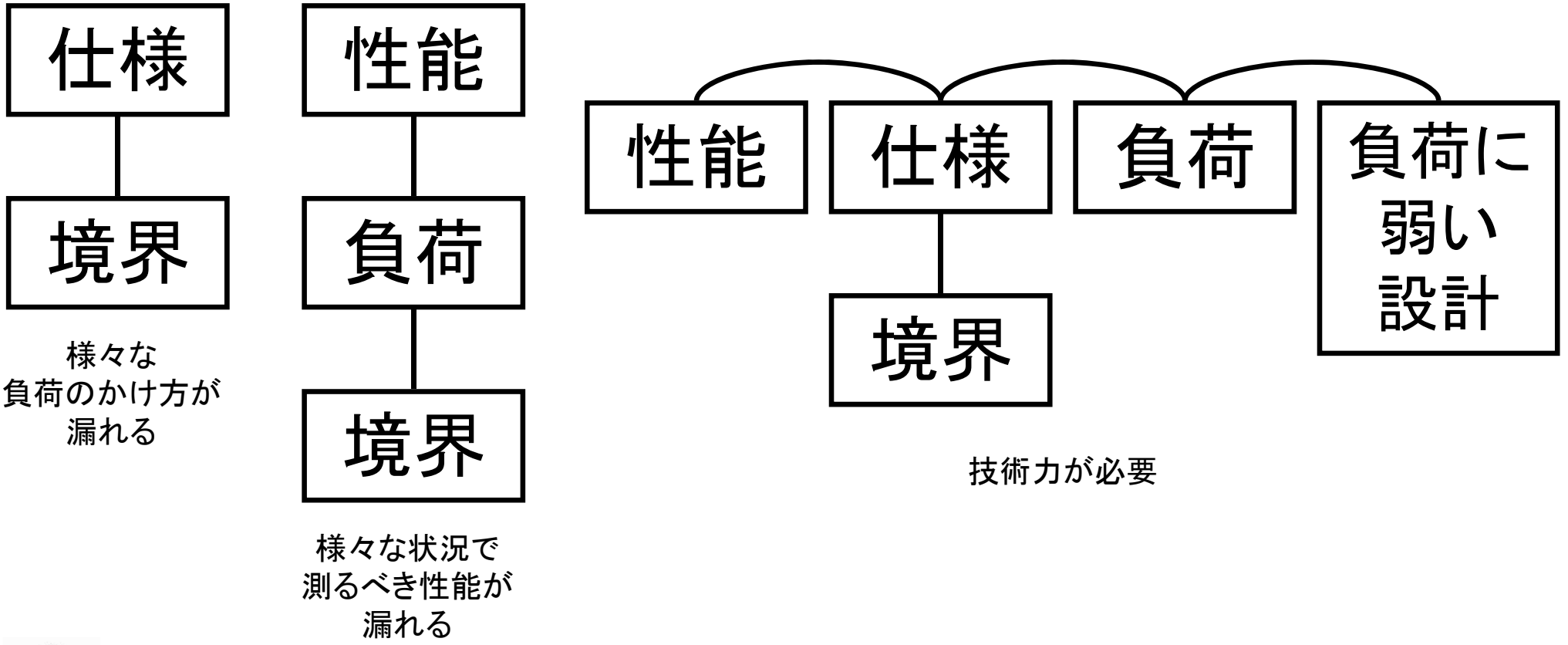
- Reverse VSTeP: テスト設計をリバースモデリングする手法
  - 既存の(十分設計されていない)テストケース群からテスト観点を読み取って、VSTePのテスト観点モデルをリバースモデリングして改善する手法群
- テスト観点モデルのリバースモデリングによる改善
  - 実際のテストケースを分析して、どういうテスト観点や関連でテストしようとしているかを列挙・整理し改善する
  - 実際に見逃したバグや検出されたバグ、テストケース外で検出されたバグを分析して、どういうテスト観点や関連が足りなかったかを列挙・整理し改善する
    - » テスト観点を網羅したつもりでも、解釈が曖昧だったり不適切な場合や、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

テスト観点をリバースすると技術力も分かってしまう

- 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、
  技術力が分かってしまう

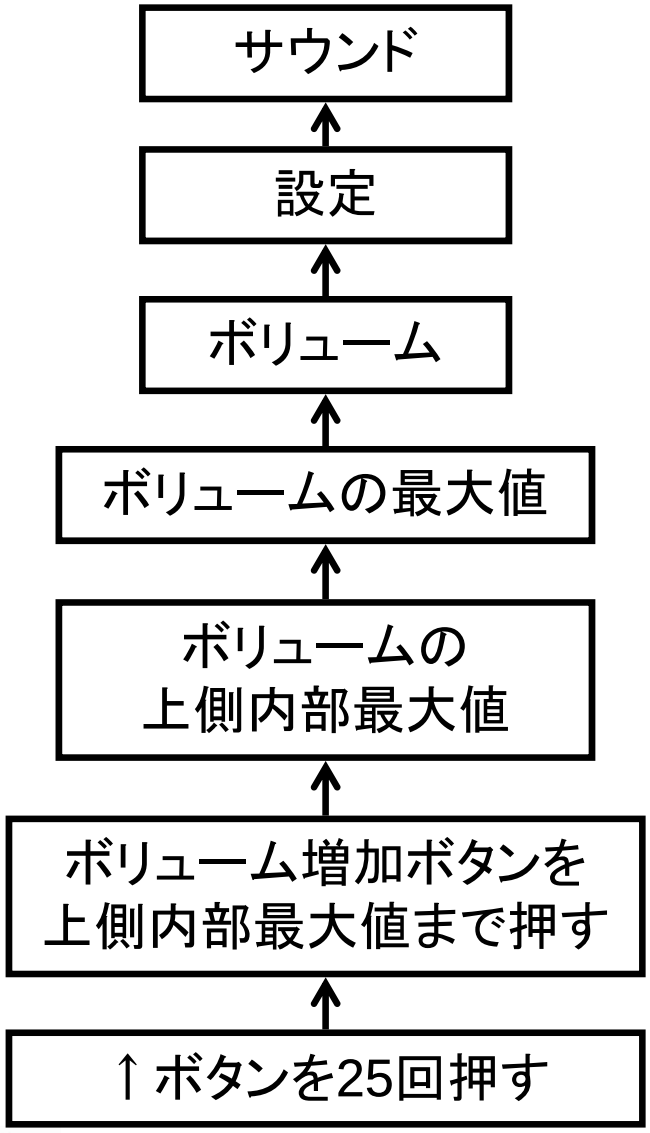


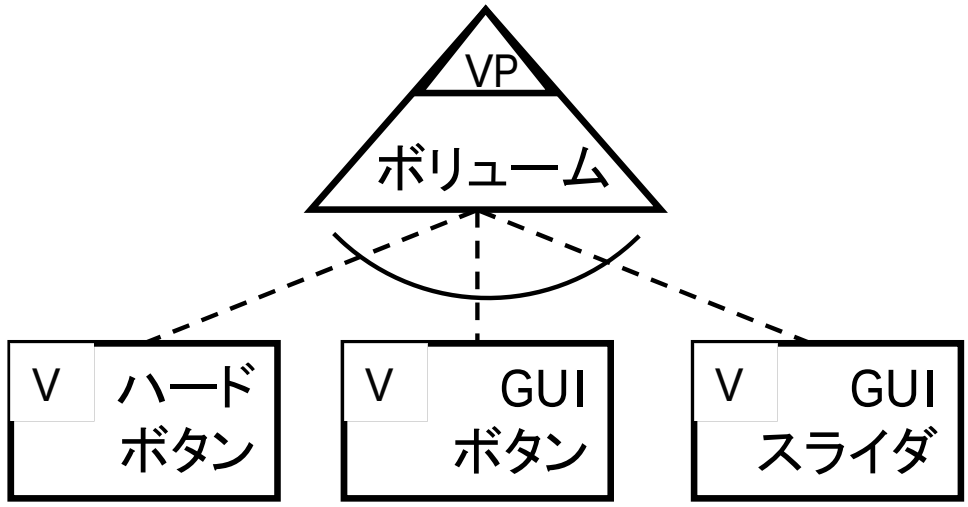
モデリングしないとテストケースを効率的に再利用できない

大項目	中項目	小項目	テスト ケース
サウンド	設定	ボリューム	↑ボタンを 25回押す

- 「ボリュームアップボタンを25回押す」というテストケースを効率的に再利用できるだろうか？
  - 次の製品では以下の仕様変更が起こる可能性がある
    - » ボリューム調整のUIの変更
    - » ボリューム調整の手段の増加
    - » ボリューム範囲の変更
    - » デフォルトのボリューム値の変更
  - 「25回」の意味が記述されていないので、意味不明のままテストケースを再利用する羽目になる
    - » 仕様変更に追従できず、テストケースを再設計せざるを得ない
    - » 仕様変更に追従できず、新たに増えた仕様のテストが漏れてしまう
    - » テストケースが本来の狙いを果たさなくなるため、バグを検出できない
    - » 意味不明のテストケースが増えていき、保守（削除）できなくなってしまう

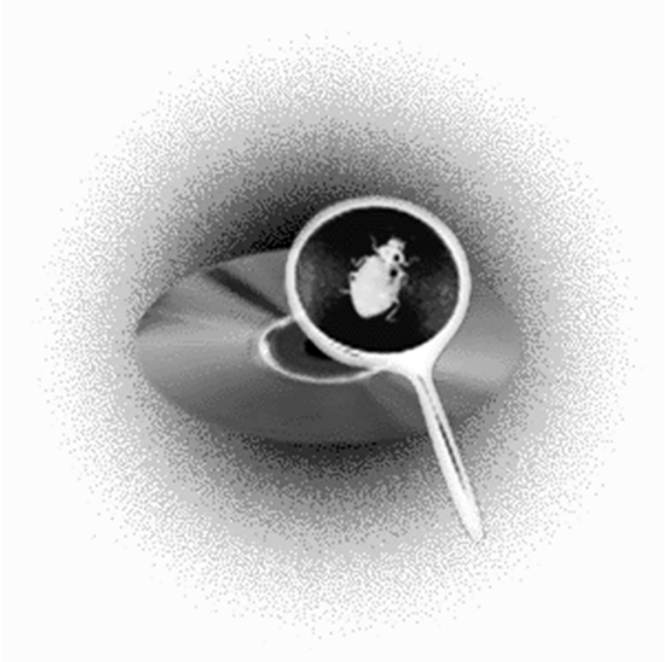
モデリングによってテストのプロダクトラインを目指す





- テストモデルに「テスト設計意図」や「テスト設計理由」を入れ込むことでテストケースを再利用しやすくなる
- 上流のモデルにテストのモデルを対応させておくと、
そもそも再利用しやすいテストを
設計しておくことができる
  - 上流からテストをモデリングすると、
テスト容易性を考慮するようになり
分析や設計がスッキリしていく

まずは使ってみて下さいな



電気通信大学　　西　康晴
Yasuharu.Nishi@uec.ac.jp