テスト設計チュートリアル
U-30クラス向け 2020年度版

ソフトウェアテスト技術振興協会（ASTER）

このお話しのゴール

テストを開発するプロセスを
イメージできるようになる！

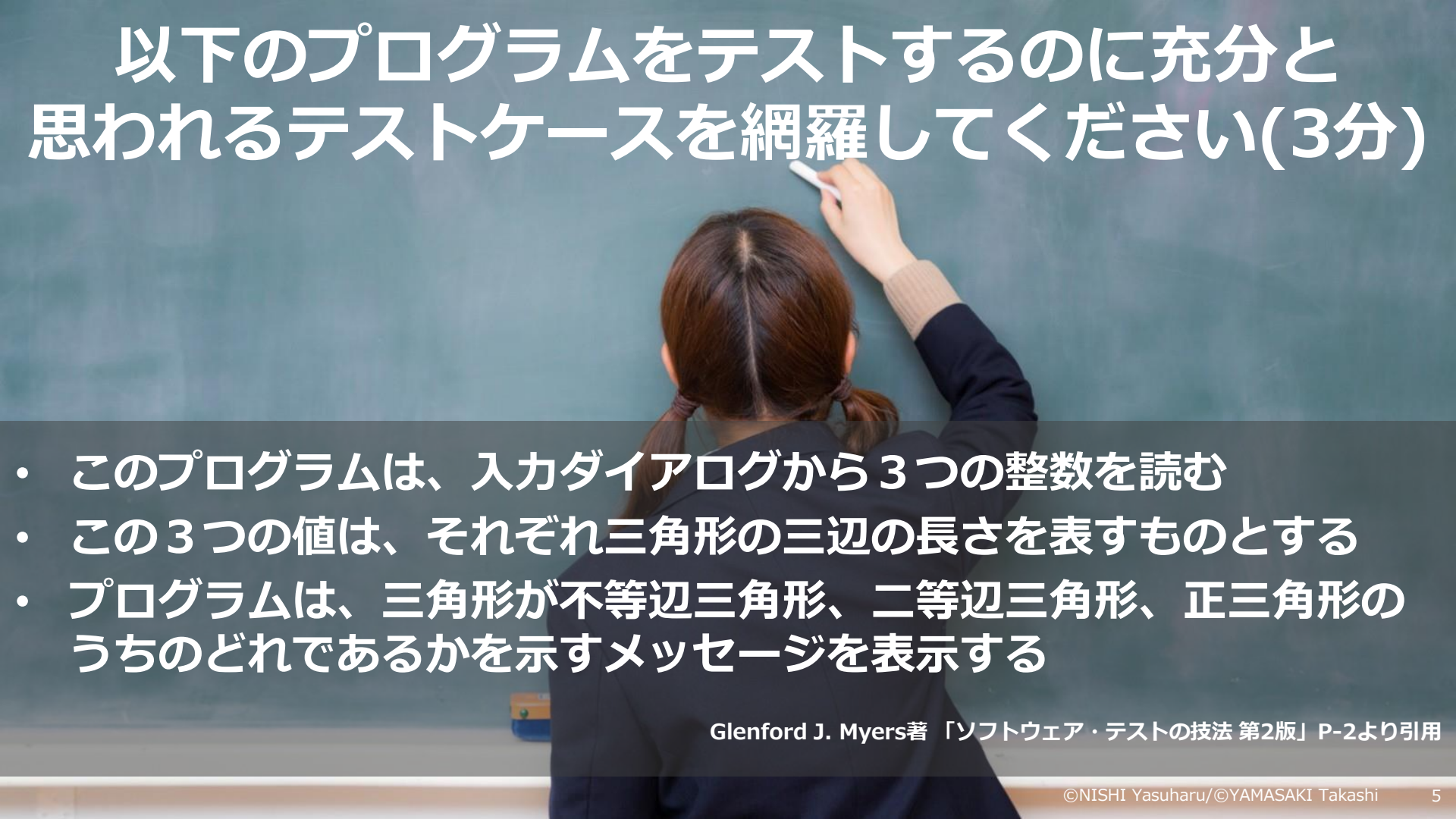
チュートリアルの流れ

1. テスト観点を整理して網羅的に挙げよう
2. テストフレームを考えよう
3. テストコンテナとその間の順序関係を考えよう
4. テスト観点からテスト値を網羅しよう
5. テスト開発を意識して進めてみよう

チュートリアルの流れ

1. テスト観点を整理して網羅的に挙げよう
2. テストフレームを考えよう
3. テストコンテナとその間の順序関係を考えよう
4. テスト観点からテスト値を網羅しよう
5. テスト開発を意識して進めてみよう

以下のプログラムをテストするのに充分と思われるテストケースを網羅してください(3分)



- このプログラムは、入力ダイアログから３つの整数を読む
- この３つの値は、それぞれ三角形の三辺の長さを表すものとする
- プログラムは、三角形が不等辺三角形、二等辺三角形、正三角形のうちのどれであるかを示すメッセージを表示する

Glenford J. Myers著 「ソフトウェア・テストの技法 第2版」P-2より引用

同じ机の人と共有してみましょう（3分）

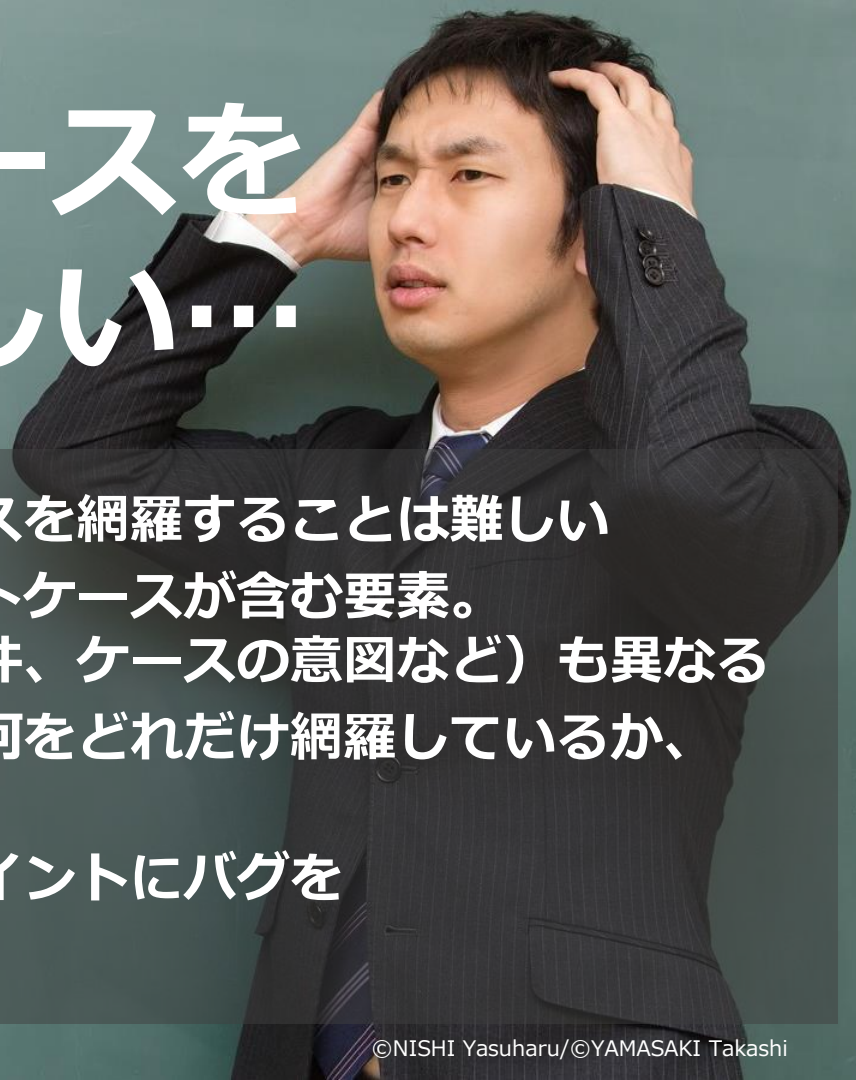
参考書による解答例（14点満点）

1. 有効な不等辺三角形
2. 有効な正三角形
3. 有効な二等辺三角形
4. 有効な二等辺三角形の際の三種類の辺の組合せ
5. 一つの辺がゼロ
6. 一つの辺が負の値
7. 二辺の和がもう一辺と等しい
8. 二辺の和がもう一辺と等しい際の三種類の辺の組合せ
9. 二辺の和がもう一辺より小さい
10. 二辺の和がもう一辺より小さい際の三種類の辺の組合せ
11. 三辺がゼロ
12. 整数でない辺
13. 辺の数が三つ以外
14. 各ケースに期待結果を示している

参考書の答え以外にも、こんなの思いつきませんでしたか？

- signed intの最大値を超える長さの辺を与えてみたいよね
- このプログラムをサーバの上に置いて100万人同時アクセスさせてみよう
- １ヶ月連続稼働させたらメモリリークで落ちるんじゃない？
- 子供が使ったら【判定】ボタンみたいなやつを連打しないかな？
- Androidのバージョンが違っても同じように動くかな
- 文字コード変えてみよう
- 常駐のウィルスチェッカが悪さしないかな
- 入力はしやすいかしら？
- Javascriptのコードを入れて次の画面で動作しちゃったら草www
- 計算の途中を見切って電源コード抜いてみるか

実際にテストケースを網羅するのは難しい…



- 仮に簡単な仕様であってもテストケースを網羅することは難しい
- 人によってテストケースの表現（テストケースが含む要素。例えば具体的な値、期待結果、前提条件、ケースの意図など）も異なる
- そもそも導出しているテストケースが何をどれだけ網羅しているか、テストケースから読み取るのは困難
- 網羅的なテストだけではなく、ピンポイントにバグを狙いだすようなケースもありえる

先ほどの例でテストすべきことを少しまとめてみる(構造化)

Before

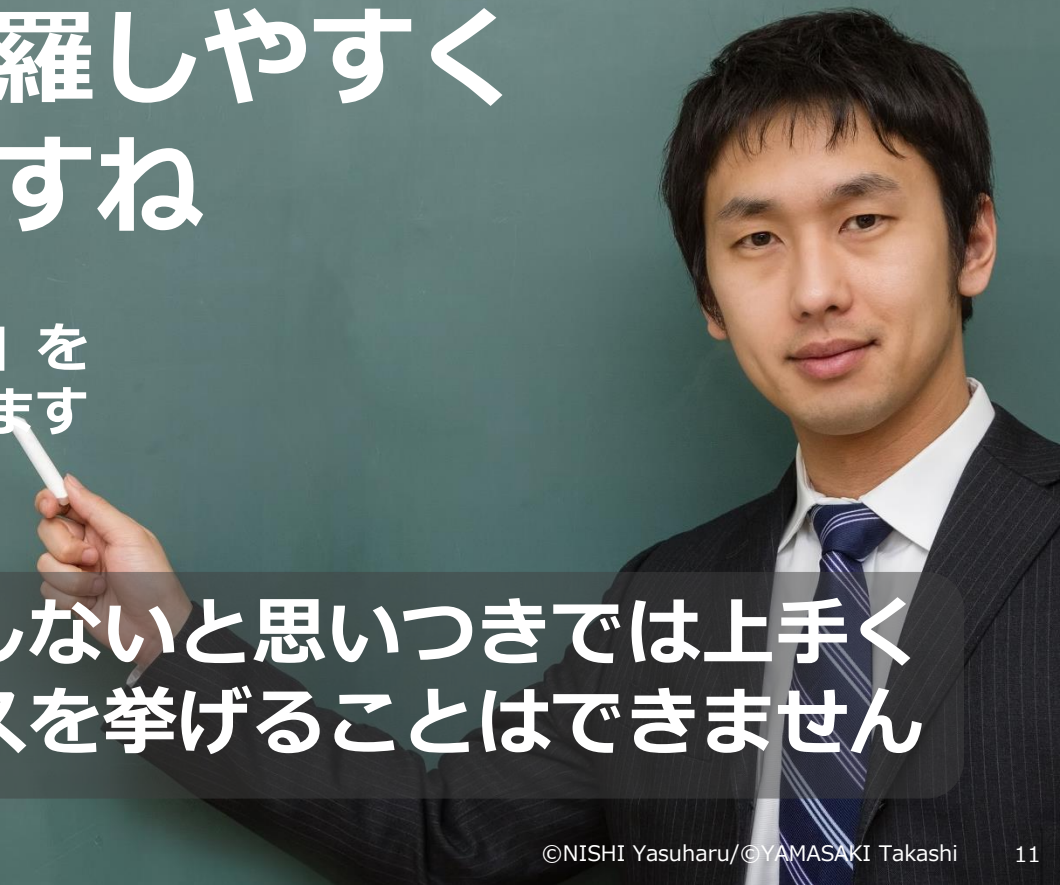
1. 有効な不等辺三角形
2. 有効な正三角形
3. 有効な二等辺三角形
4. 有効な二等辺三角形の際の三種類の辺の組合せ
5. 一つの辺がゼロ
6. 一つの辺が負の値
7. 二辺の和がもう一辺と等しい
8. 二辺の和がもう一辺と等しい際の三種類の辺の組合せ
9. 二辺の和がもう一辺より小さい
10. 二辺の和がもう一辺より小さい際の三種類の辺の組合せ
11. 三辺がゼロ
12. 整数でない辺
13. 辺の数が三つ以外
14. 各ケースに期待結果を示している

After

1. 三角形の形に関するテスト
    - 三角形が成立する場合のテスト
        - 有効な不等辺三角形
        - 有効な二等辺三角形
        - 有効な正三角形
    - 三角形が成立しない場合のテスト
        - 直線になってしまう場合
            - 二辺の和がもう一辺が同じ
        - 面を構成できない場合
            - 二辺の和がもう一辺より短い
            - 長さが0の辺がある
2. 辺の組合せに関するテスト
    - 三種類の辺の組合せ
3. 入力の仕様に関するテスト
    - 整数でない辺
    - 辺の数が三つ以外
4. 期待結果を示してあるかどうか

“テストすべきこと”※を考えると少し網羅しやすくなった気がしますね

※本講では「テストすべきこと」を「テスト観点」と呼ぶこととします



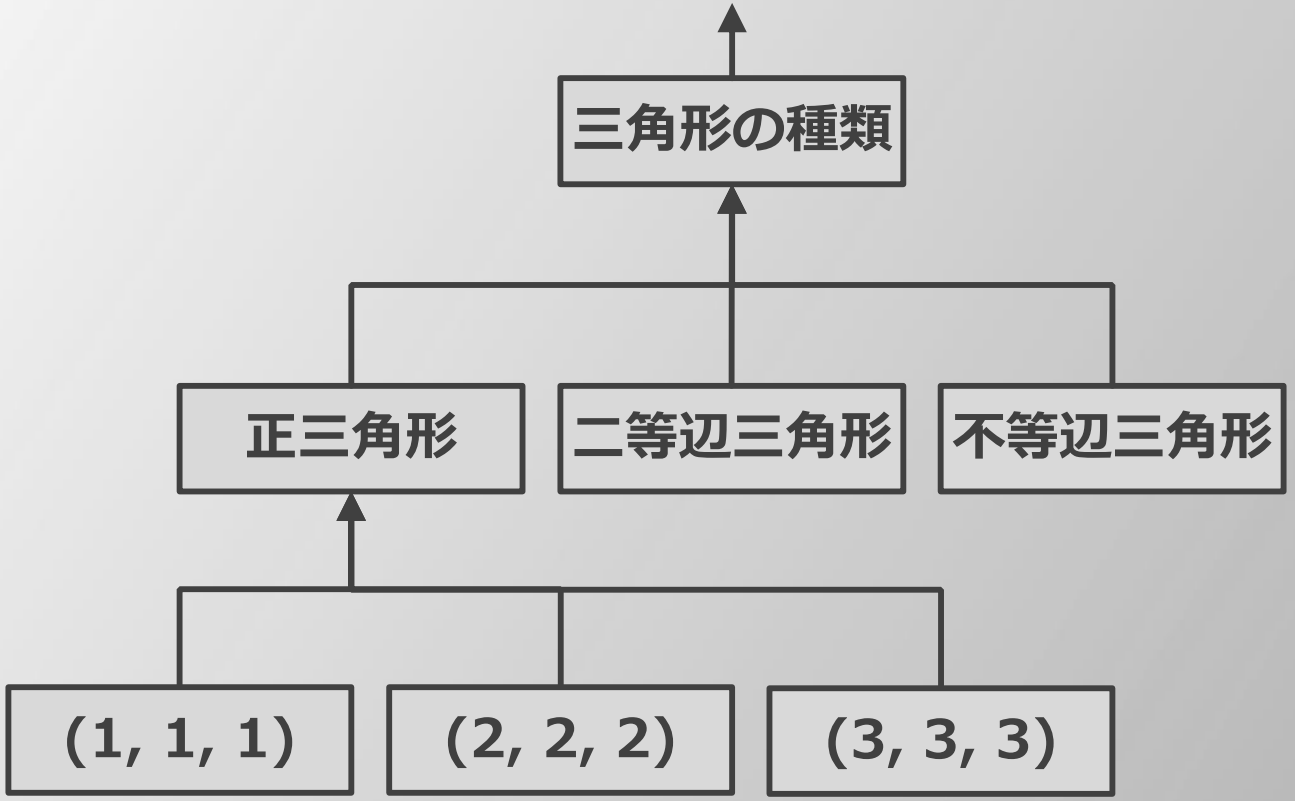
いくつかの点に注意しないと思いつきでは上手く整理してテストケースを挙げることはできません

“テストすべきこと”(テスト観点)の様に抽象化すると見通しが良くなる

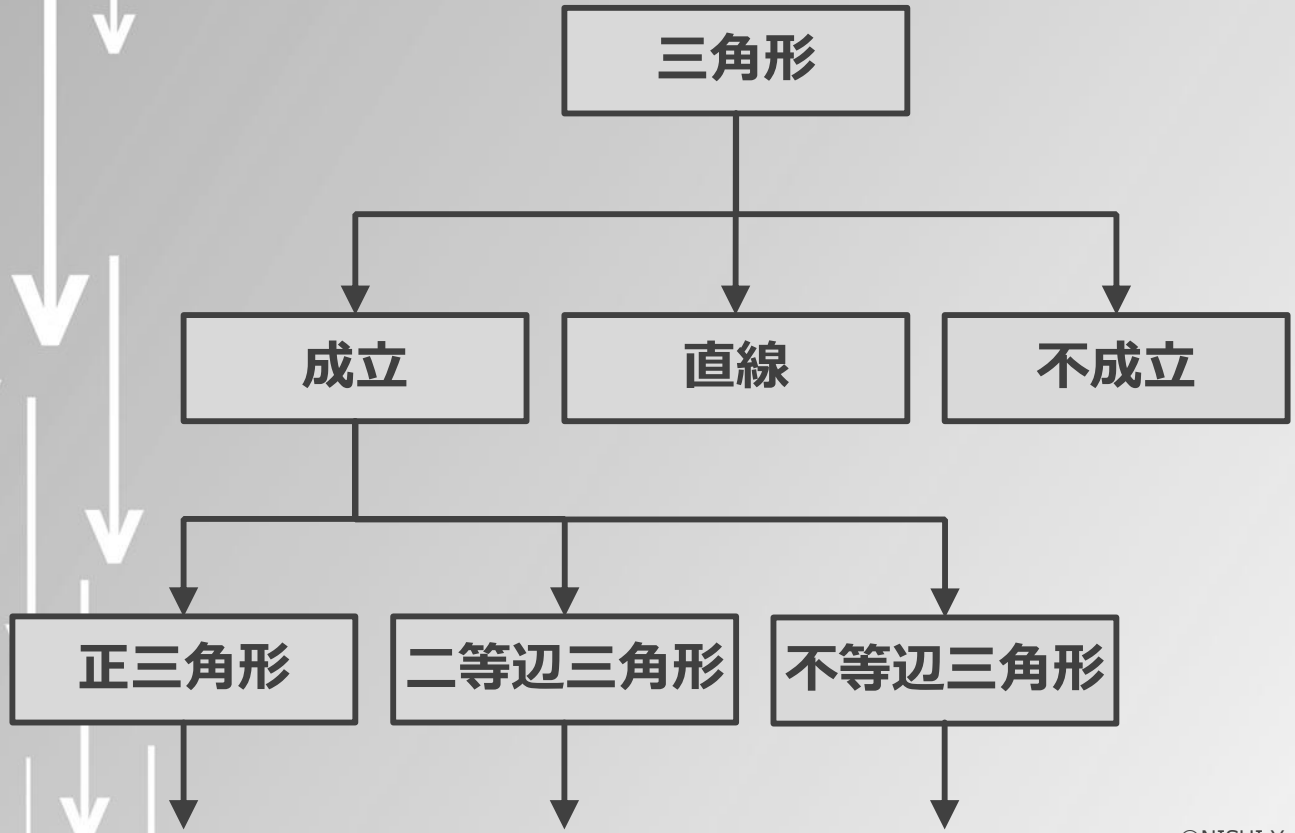
- 三角形の形
- 辺の組合せ
- 入力の仕様
- 負荷、使う人数、稼働させる時間
  - 連打、100万人、1ヶ月連続稼働
- 内部設計
  - signed int
- 動作環境、互換性
  - androidのバージョン、文字コード
- 期待結果
- セキュリティ
  - クロスサイトスクリプティング
- 操作性
  - 入力しやすいか
- 障害対応性
  - 電源コードを抜く
- 想定されるバグ
  - メモリーリーク
- 誰が使うか
  - 子供

テスト観点で考えるとテストケースを網羅しやすくなる

ボトムアップ的に観点を整理していく方法

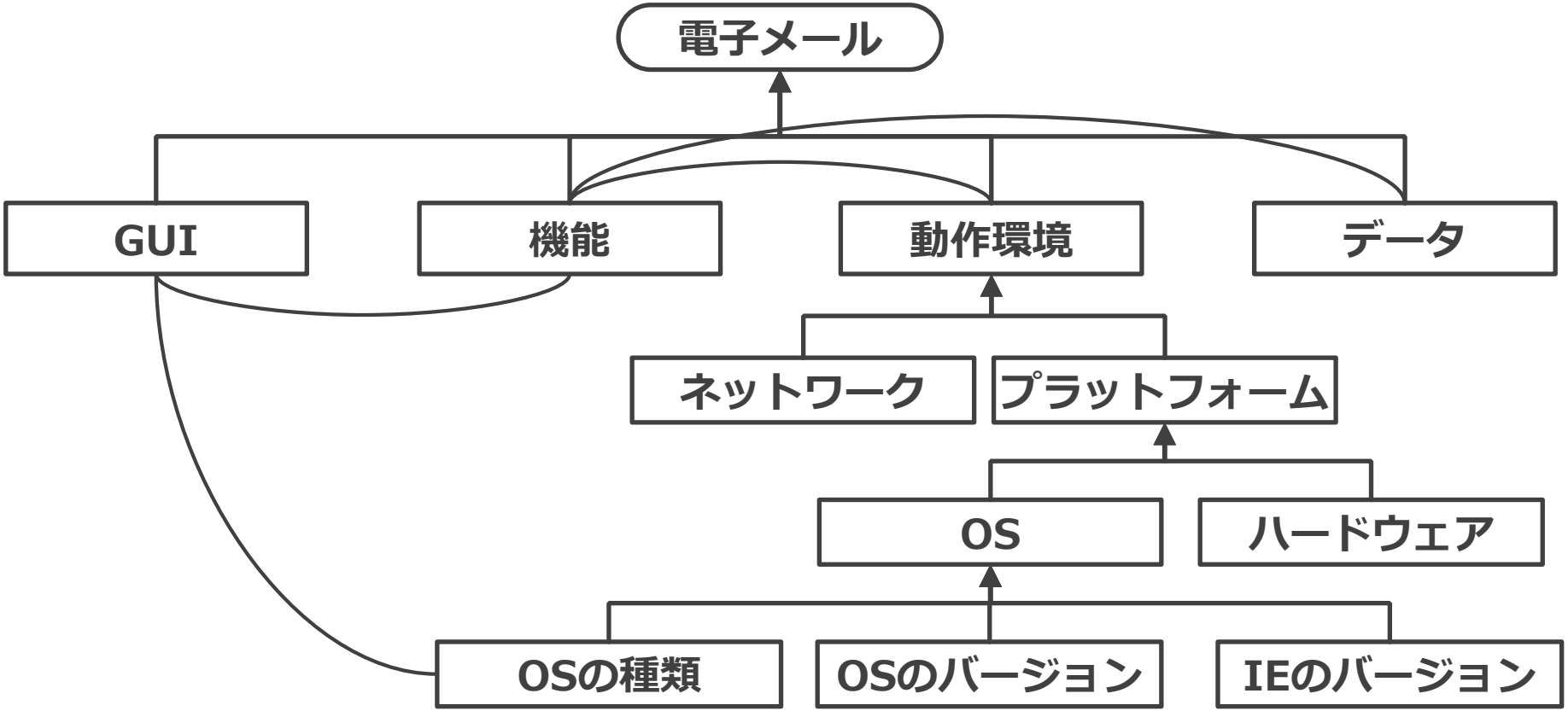


トップダウン的に観点を整理する方法



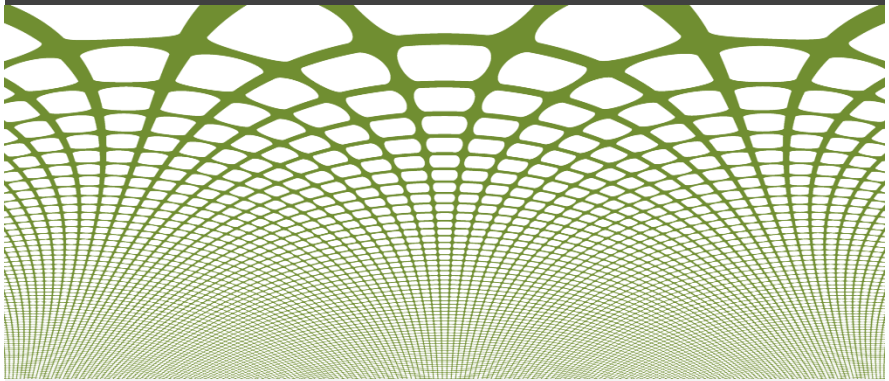
実際にはトップダウンと
ボトムアップを循環させながら
テスト観点を網羅していく

テスト観点を整理した例



基本的なテストの方針

網羅
「隅から隅までテストする」



- 漏れなくテストを行う
- 動作実績を積み上げて品質を保証する
  「保証型」のテスト

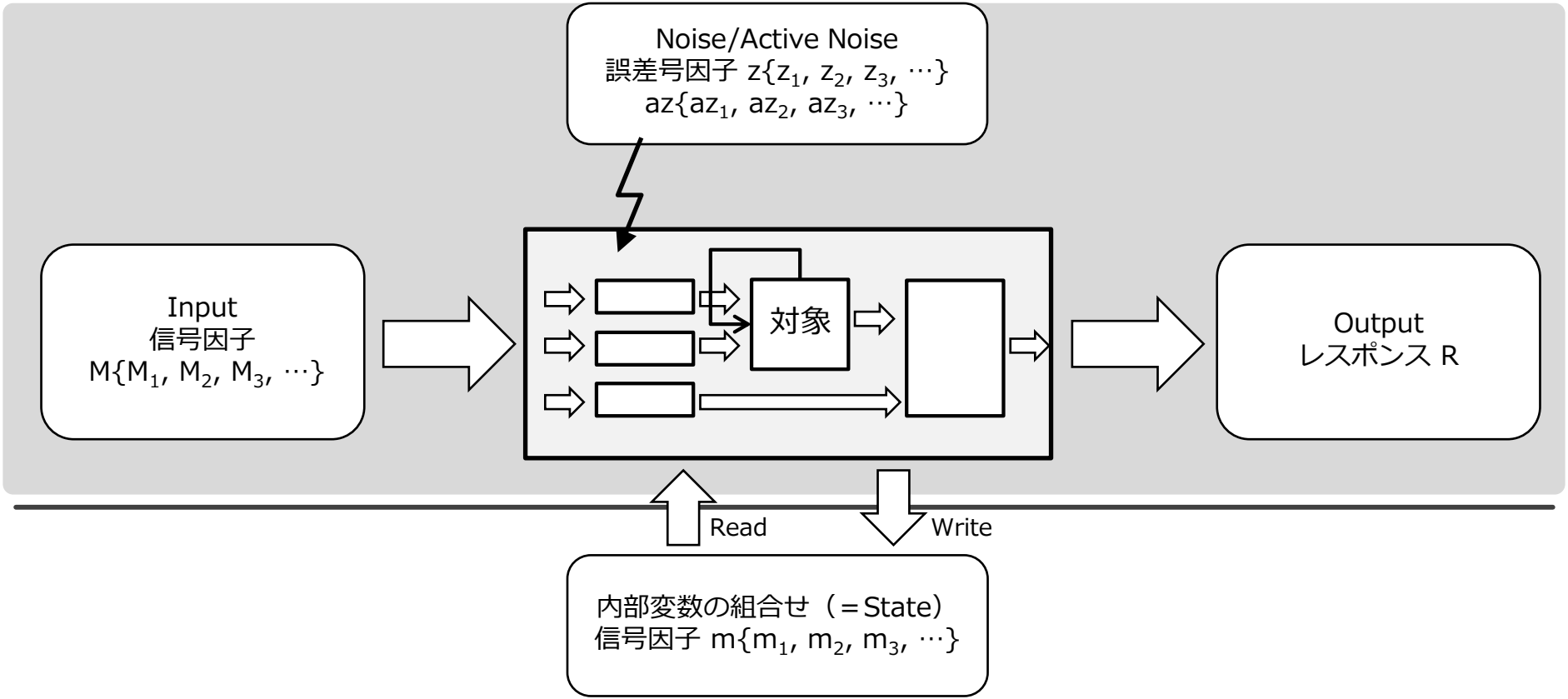
ピンポイント
「ヤバいところをテストする」

- バグが起きそうな所を狙ってテストする
- 少ない手間で沢山危険なバグを検出する
  「検出型」のテスト

テストは保証型と検出型の2種類を適切に組み合わせて行う

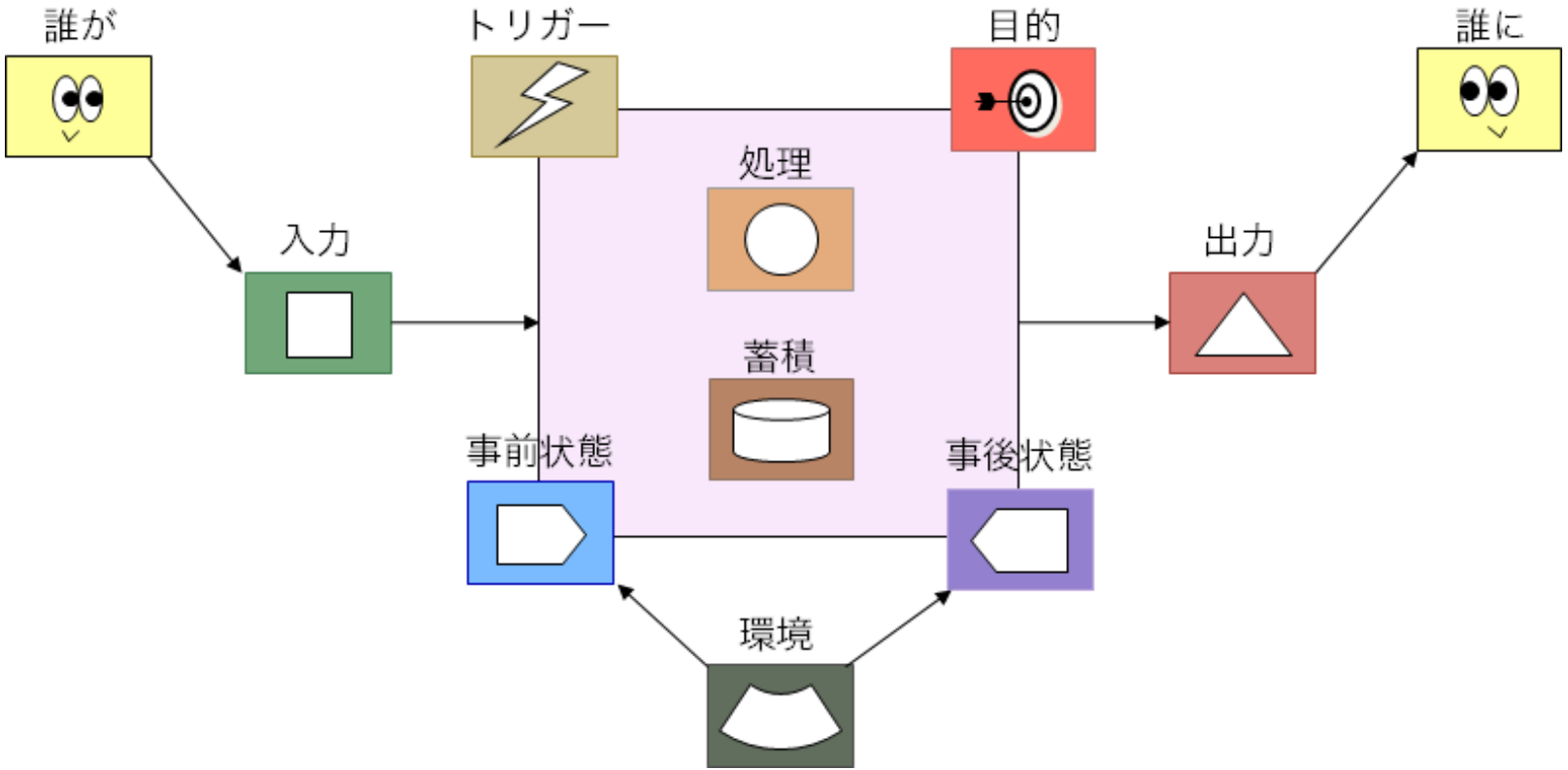
“テストすべきこと”を網羅的に挙げよう

ラルフチャート



「高信頼化ソフトウェアのための開発手法ハンドブック」より引用

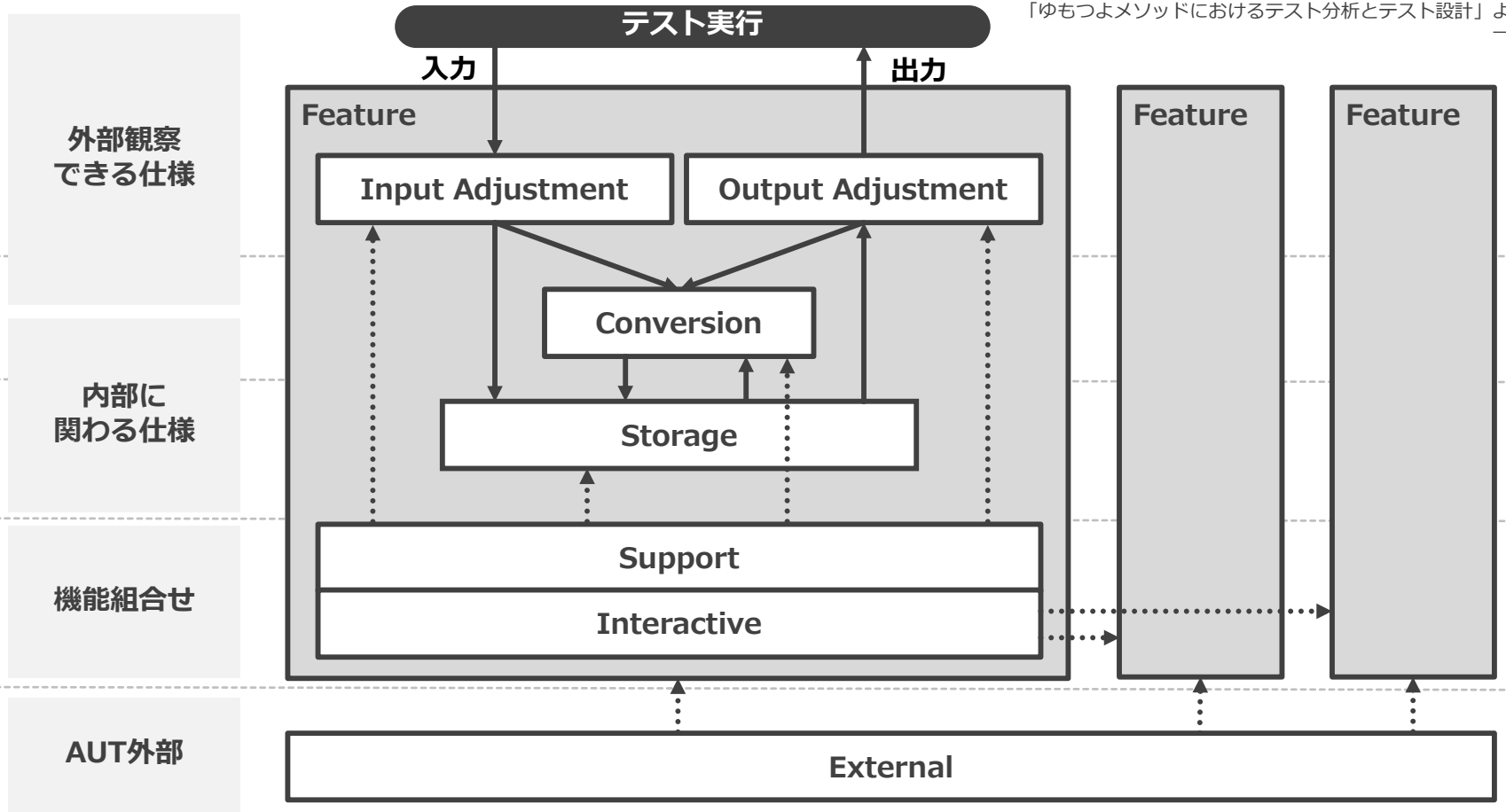
Tiramisで使用している基本構造構成要素



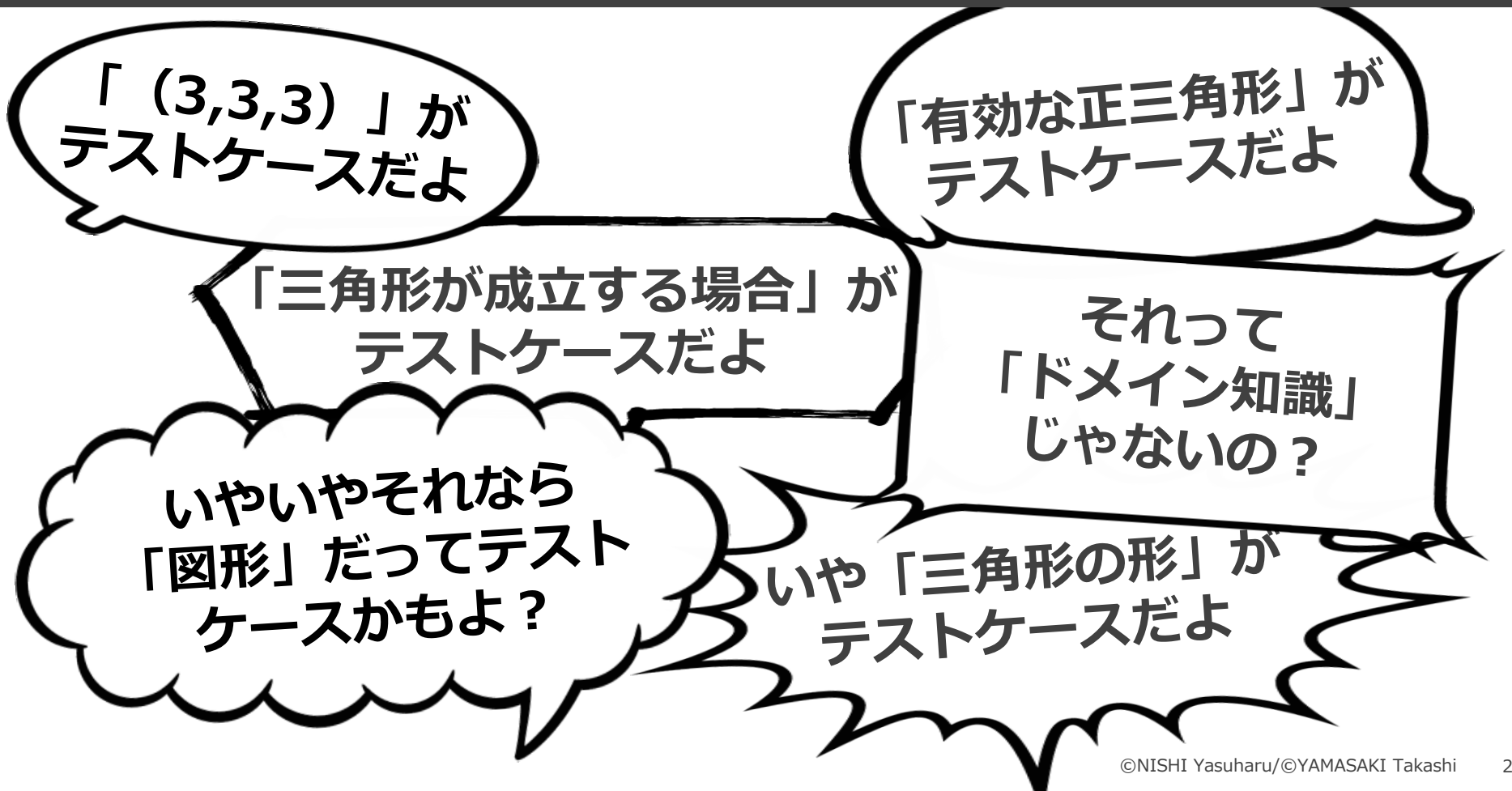
「https://twitter.com/mkoszk/status/509922586294112256 」より引用

論理的機能構造

「ゆもつよメソッドにおけるテスト分析とテスト設計」より引用
一部変更



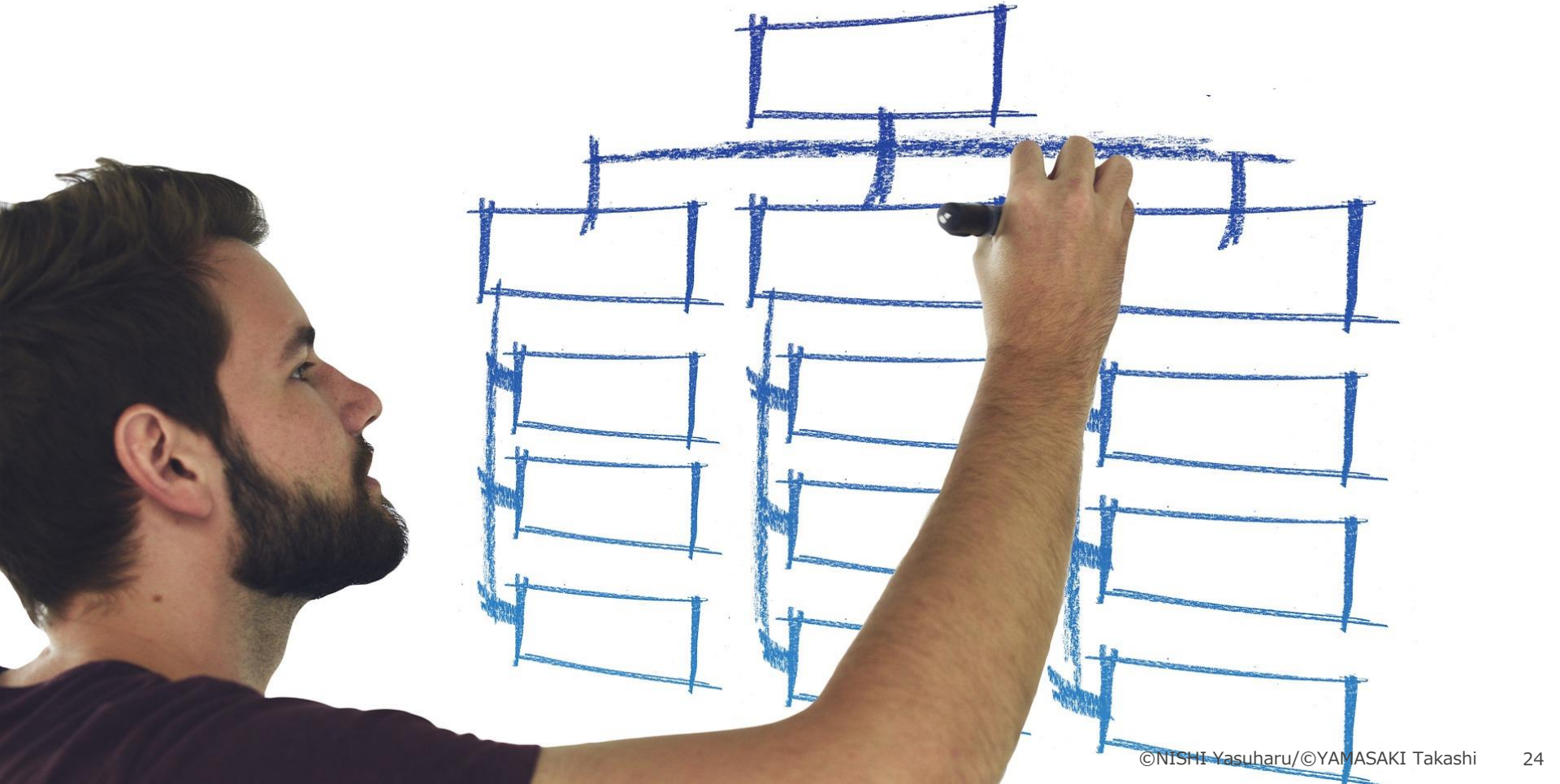
「テスト観点」といっても様々な抽象度がある？



本稿では以下の様に整理します

- 「3」のような具体的なものをテスト値と呼ぶ
- 「辺の長さ」のようにテスト観点の最も具体的なものをテストパラメータと呼ぶ
  - テストパラメータの特定の要素がテスト値となる
- 「(3,3,3)を入力すると正三角形と表示される」のようにテストケースはテスト値の集まりである
- 「有効な正三角形」から「ドメイン知識」まで、テストパラメータやテストパラメータを抽象化したものをテスト観点と呼ぶ
  - テストパラメータやテスト観点をテスト条件とも呼ぶ
- マインドマップやUMLツールなどを用いると描きやすいが、一覧表やマトリクスでも構わない

テスト観点には様々な抽象度があるので整理して網羅しましょう



テスト観点を挙げる時の注意点

仕様書に書いてある仕様や文・単語を書き写してもテスト観点は網羅できない

仕様書は通常不完全で、特に致命的なバグは仕様書に書かれていないテスト観点でしか見つからないことがある

これまでに挙げたテスト観点は、みなさんの組織の仕様書にすべて書いてありましたか？

テスト観点は多面的に挙げる必要がある

入力に関するテスト観点だけを挙げたり、期待結果やその観測に関するテスト観点だけを挙げたり、品質特性だけをテスト観点として採用すると、漏れが発生する

テスト観点を挙げるだけでバグが見つかることもある

開発者が考慮していなかったテスト観点はバグの温床であり、テストしなくてもバグだと分かることも多々ある

期待結果に関するテスト観点を挙げたり詳細化すると、仕様の抜けが分かることがある

※テストケースにも期待結果を必ず書くのを忘れないこと。「正しく動作すること」は期待結果ではない！

網羅した風のテスト観点の文書や納品物を作るのが本質ではなく
網羅しようと多面的にさまざまなことを考えつくすのが本質です

テスト観点の導出結果は実施者によって異なります

そもそも「全数テストは不可能※1」ですし「テストは条件次第※2 」なので
テストが一意的に収束することはなく、絶対的なテスト観点モデルもありません

下記についてはJSTQB Foundation Levelのシラバスを参照のこと
※1: テストの七原則の一つ。すべてをテストすることは現実的に不可能である(組合せも考慮するとすぐにテストが発散する）。そのため、テストはサンプリングになるため、何をテストしてなにをテストしないかが重要になる。
※2: テスト七原則の一つ。条件が異なればテストの方法や内容もことなる。コンテキストにあったテストを実施するためにも、テストを説明できることが重要。

http://www.freepik.com/free-vector/business-meeting-infographic_792711.htm

どんなテストをするのかを利害関係者とコンセンサスを得ることが重要であり、そのためにはテストを説明できる必要有り

ピンポイント型のテスト観点もきちんと挙げよう

起きて欲しくないこと

危険事象やハザードなど様々

例)顧客データ喪失、出火、顧客のお金に関わるデータの計算ミス

HAZOPや意地悪漢字などを仕様や設計にぶつけて想起

例） もし辺の入力がなかったら？（ガイドワード「ない」）

バグ

絶対に入って欲しくないバグ

例） バッファオーバーフロー、クロスサイトスクリプティング、SQLインジェクション

よく作り込んでしまうバグ

例）小数の丸め誤差、メモリ解放忘れ、例外に対する処理忘れ、初期化忘れ

後工程でバグを引き起こしそうな罠や弱点

構造が歪んでいるところや分かりにくいところ

全体が1対1に対応しているが実は一部のみ多対多になって入り組んでしまっているところ

プラットフォームや言語仕様に潜む罠や弱点

過去の技術的負債との互換性のために残っているOSのAPI

MISRA-Cのいくつかの項目（if文では=で変数に代入でき、値を持ってしまう、など）

どれだけ適切なピンポイント型のテスト観点を挙げられるかは、テストの質を示す重要な要素

ソフトウェアHAZOPのためのより詳しいガイドワード

強度	強い	弱い		範囲	長い	短い		タイミング	早く	遅く	
数量	多い	少ない			上限	下限		拡張	広く	狭く	
	増加	減少			広い	狭い			多く	少なく	
	余分	不足			最大限	最小限		頻度	多く	少なく	
	ある	ない	ゼロ	頻度	高い	低い			高く	低く	
種類	多種	小種			多い	少ない		程度	いつも	たまに	
規模	大きい	小さい		時期	早く	遅く		回数	多く	少なく	
距離	遠い	近い			長く	短く		接続	つなぐ	切り離し	切り替え
速度	速い	遅い			広く	狭く		方向	上へ	下へ	
	高速	低速			長く	短く		負荷	高い	低い	
	緩	急		順序	さきに	あとで			超過	不足	限界
設定	許容	禁止			同時に	並行に		位置	高く	低く	
	必須	任意			早く	遅く			遠く	近く	
金額	大きい	小さい			逆転	反復	挿入				

鈴木三紀夫さん、秋山浩一さんがご作成

ソフトウェアHAZOPのためのより詳しいガイドワード

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

鈴木三紀夫さん、秋山浩一さんがご作成

和風のガイドワード「意地悪漢字」

鈴木三紀夫さんがご作成

チュートリアルの流れ

1. テスト観点を整理して網羅的に挙げよう
2. テストフレームを考えよう
3. テストコンテナとその間の順序関係を考えよう
4. テスト観点からテスト値を網羅しよう
5. テスト開発を意識して進めてみよう

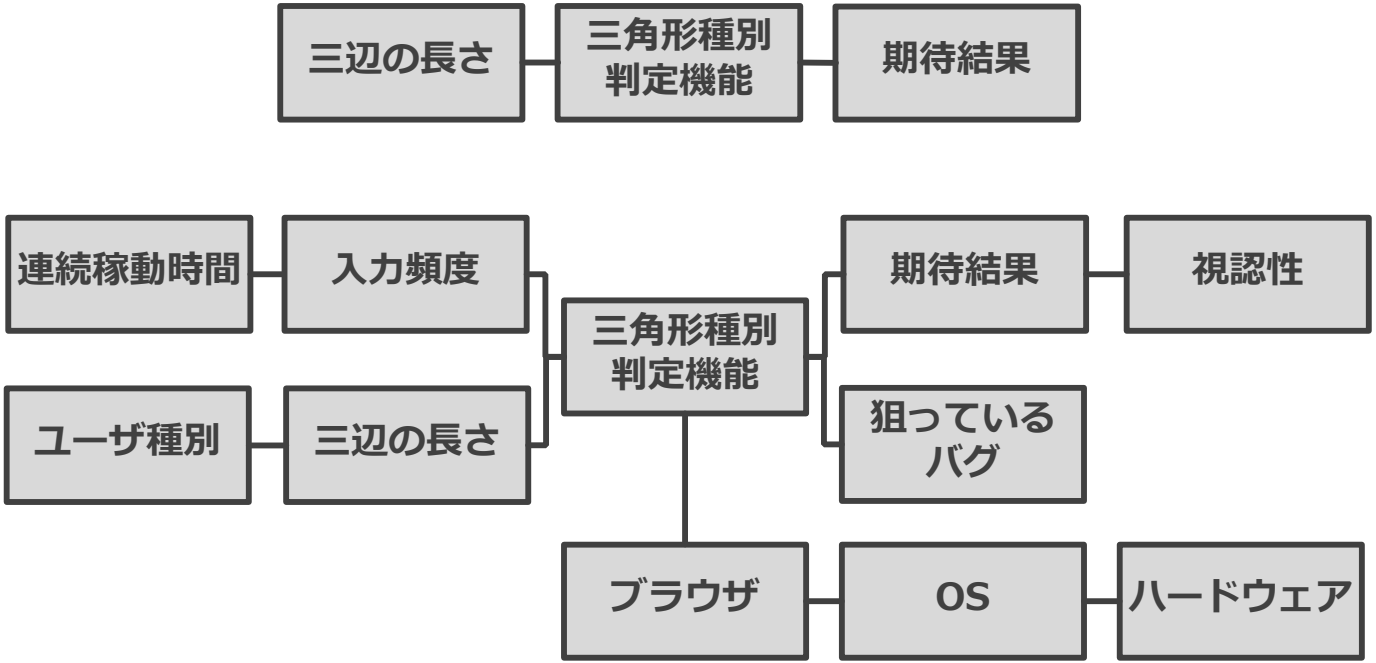
- ☑ （3,3,3）を入力して動作すること
- ☑ （3,3,3）を入力して「有効な正三角形」と表示されること
- ☑ iPhone6s/iOS9.3.2で（3,3,3）を入力して「有効な正三角形」と表示されること
- ☑ iPhone6s/iOS9.3.2で（3,3,3）を入力して「有効な正三角形」と見やすく表示されること
- ☑ iPhone6s/iOS9.3.2で（3,3,3）の入力を1ヶ月連続で行っても「有効な正三角形」と見やすく表示され、メモリリークが起きないこと
- ☑ iPhone6s/iOS9.3.2上のSafari（3,3,3）の入力を子供が1ヶ月連続で連打しても「有効な正三角形」と見やすく表示され、メモリリークが起きないこと

人によって簡単なテストケースもあれば複雑なものもある

テストケースの構造(テストフレーム)を考えよう

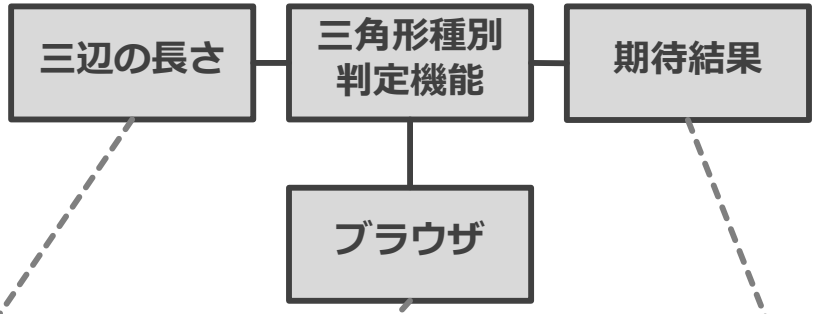
同じボタンでも、それぞれ色や形や模様が違うように、
テストケースも単純なものから複雑なものまで色々ありうる

テストフレームの単純な例と複雑な例



例えば同じ三角形の種別判定機能をテストするといっても
そのテストの関心事はテストフレームによって異なります

テストフレームに沿ったテストケースの構造を作成する

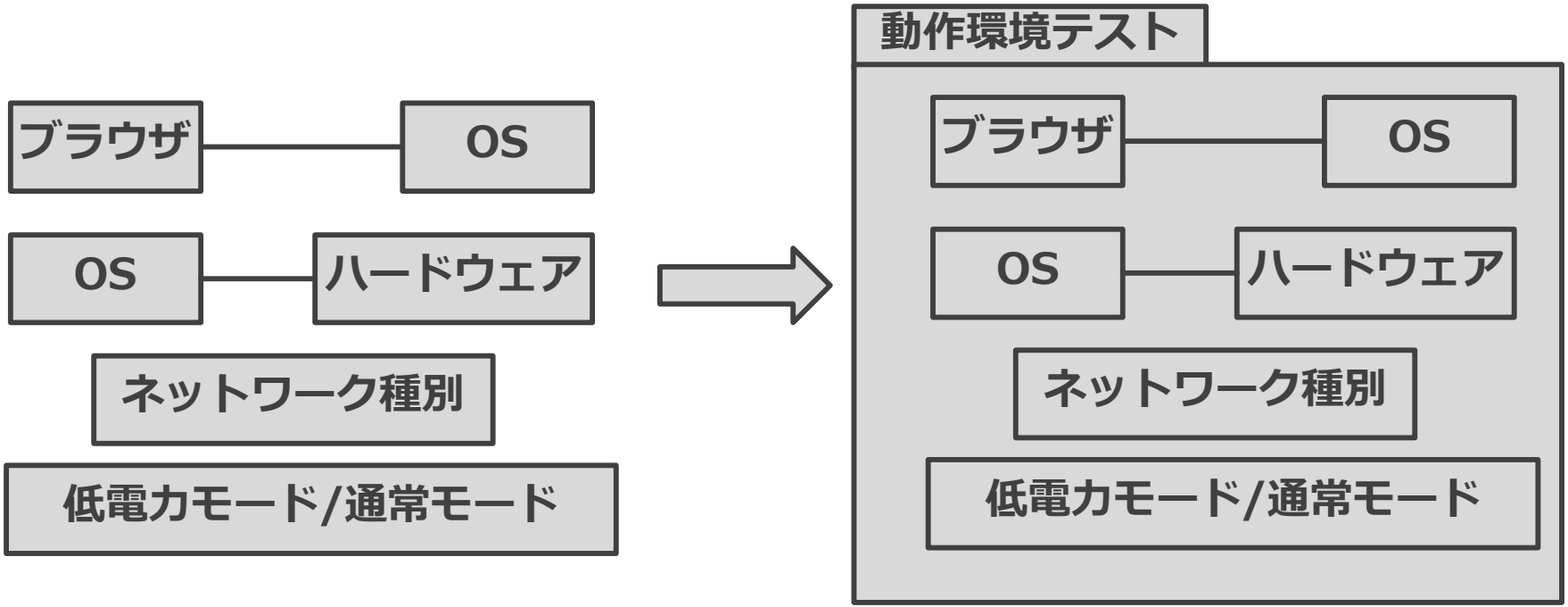


ID	三辺の長さ	ブラウザ	期待結果
1	(3, 4, 5)	Edge	「不等辺三角形」と表示される
2	(3, 3, 4)	Edge	「二等辺三角形」と表示される
3	(3, 3, 3)	Edge	「正三角形」と表示される
…	…	…	…

チュートリアルの流れ

1. テスト観点を整理して網羅的に挙げよう
2. テストフレームを考えよう
3. テストコンテナとその間の順序関係を考えよう
4. テスト観点からテスト値を網羅しよう
5. テスト開発を意識して進めてみよう

テストコンテナ（テスト観点やテストフレームをまとめたもの）



テスト観点やテストフレームが増えてきたら、「テストコンテナ」に
うまくまとめあげると見通しのよいテスト設計になる

テストコンテナにも様々な種類がある

テストレベル	テストタイプ	テストサイクル	その他
単体テスト 結合テスト システムテスト 受入テスト など	負荷テスト ストレステスト 構成テスト セキュリティテスト など	1回目の機能テスト 2回目の負荷テスト など	etc

テストコンテナは各組織で定められたものを使うことが多いため、あまり自分たちでは設計したことがないだろう

社内標準で決まっている、過去プロジェクトのテストコンテナを再利用している、などそのため、よく意味の分からないテストコンテナを適当に解釈して使っていることもある、意識せずに自分たちでテストコンテナを定義してしまっている場合もある

システムテストの前半、など

自分たちで無意識にテストコンテナを定義してしまっているくらいなら、きちんと設計した方がよい

本来は、テストごとにテストコンテナを設計する方がうまくまとまる

どんなテストコンテナを用いるかによってテスト設計の良し悪しが変わるので、なぜこれらのテスト観点やテストフレームをまとめてこのテストコンテナにしたか、はきちんと説明できないといけない（テストコンテナの責務の分担と呼ぶ）

既に存在したり一般に紹介されているテストコンテナを参考に、なぜそのようにまとめたのか、リバースエンジニアリングしてみることから初めてみよう

テストレベルの例

- 単体テスト/ユニットテスト
- 結合テスト/統合テスト
- システムテスト/総合テスト
- 受入テスト など

テストタイプの例

Ostrandの4つのビュー

- ユーザービュー
- 仕様ビュー
- 設計・実装ビュー
- バグビュー

テストタイプの例

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

テストタイプの例

ISO/IEC 9126の品質特性※

機能性/信頼性/使用性/
効率性/保守性/移植性

※現在はISO/IEC 25000（SQuaREシリーズ）に置き換えられている

よいテストコンテナってどんなものだろう？

結合度が低い

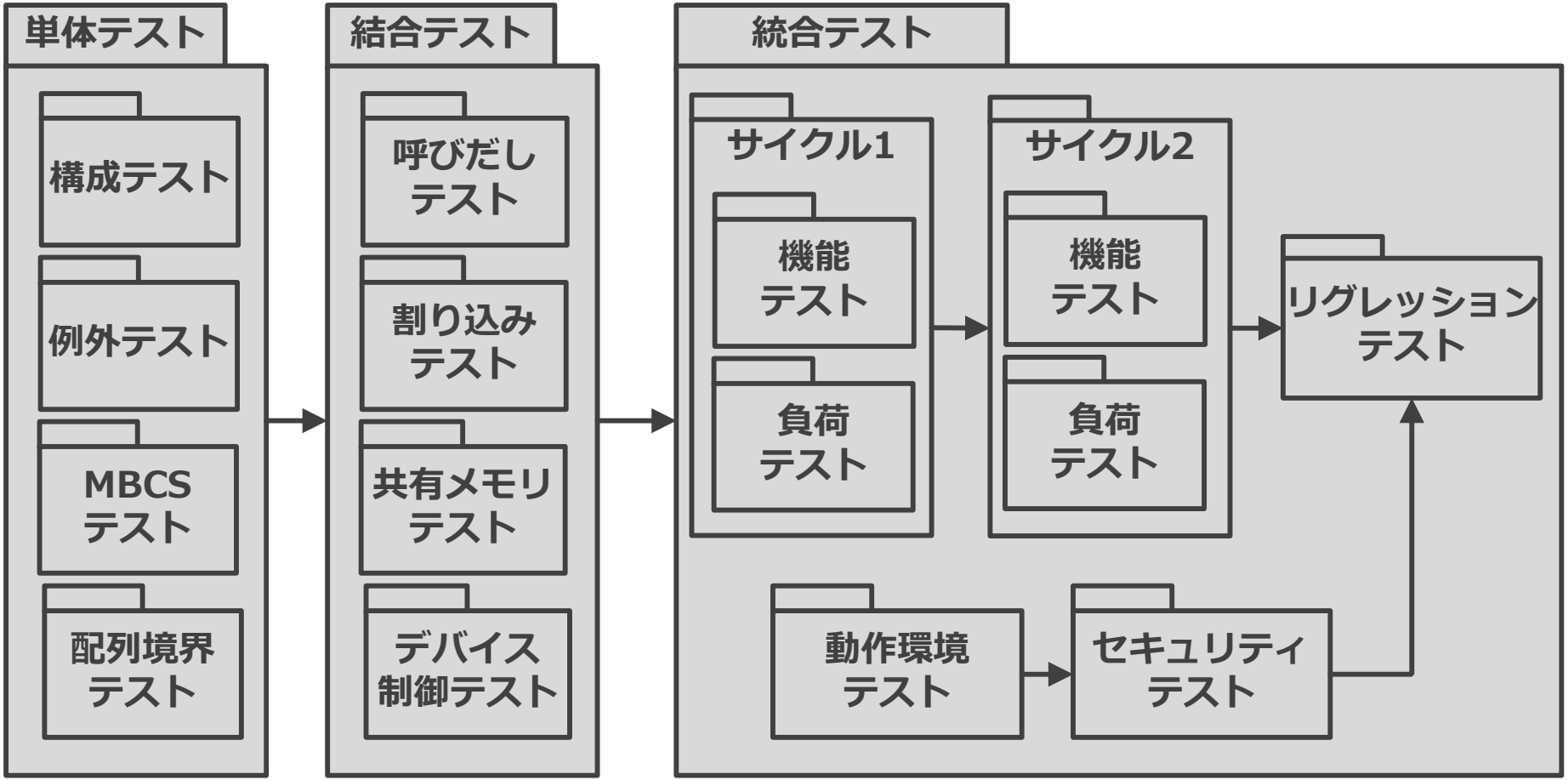
例えば、結合度の低いテストコンテナは、テストコンテナ間の依存性といった関連性が低く、他のテストコンテナの影響を受けにくい。一般的に結合度は低い方が良く、テストコンテナ間の関連性を最小化するとよい

凝集度が高い

例えば、凝集度の高いテストコンテナは、テストコンテナ内の関係性が高く、テストコンテナの意図が明確で分かり易く多義的ではない。一般的に凝集度は高い方が良く、テストコンテナ内の関連性を最大するとよい

低結合度、高凝集度を目指し、テストアーキテクチャ設計の品質特性(e.g. 保守性、自動化容易性、環境依存性)を高めていくとよい

テストコンテナの関連性や順序性をモデリングする



テストアーキテクチャを考える意味

テストコンテナ間の前後関係や並列関係を決めたものを「テストアーキテクチャ」と呼ぶ

- テストアーキテクチャを描くと、テストの全体像が俯瞰しやすくなる
  - 通常はテスト計画やテスト戦略に文章で記述されているため、俯瞰しにくくなっている
  - UMLツールなどを用いると描きやすいが、自然言語の文章や一覧表、マトリクスでも構わない
- 後工程や後プロジェクトでテストしやすいようにテストコンテナをまとめ、前後関係や並列関係を考える必要がある
  - テストコンテナをまとめている段階で、コンテナごとのテストの厚みやコンテナ間のバランスを考える必要がある
  - どんな責務のテストコンテナをどんな順序で並べるかによってテスト設計の良し悪しが変わるので、なぜこのテストコンテナをこの厚みでこの順序に並べたか、はきちんと説明できないといけない
- テストコンテナを適切に配置しないと、開発がやり直しになるような不具合が出荷間際に見つかったりする羽目になる
  - 最後の最後にまとめて性能テストを行うと、期待した性能よりもかなり遅かった場合にシステムアーキテクチャ設計からやり直さなくてはならない事態が発生することがある
  - 開発がやり直しになるような不具合が出てしまうような基本的な性能テストでなく、それに関わるテスト観点だけに責務を分担させた基本的な性能テストコンテナをテストの初期段階から反復的に拡張しながら行っていくようなテストアーキテクチャを設計していれば、早いうちにそのような不具合は防げていたかもしれない

チュートリアルの流れ

1. テスト観点を整理して網羅的に挙げよう
2. テストフレームを考えよう
3. テストコンテナとその間の順序関係を考えよう
4. テスト観点からテスト値を網羅しよう
5. テスト開発を意識して進めてみよう

テスト観点からテスト値を網羅しよう

① テスト観点がどのタイプの
テストモデルなのかを見極める

▼

② テスト観点を十分詳細化して
テストパラメータを導く

▼

③ 網羅基準（カバレッジ基準）を
定める

▼

④ 定めた網羅基準でテストパラメーターを
網羅するようにテスト値を設計する

テストモデルには4つある

範囲タイプ

一覧表タイプ

マトリクスタイプ

グラフタイプ

テスト値には2種類あるので注意する必要がある

テスト値が直接テスト手順の一部（テストデータ）として実施できるもの	
テストパラメータ：	辺の長さ
テスト値：	(1, 2, 3)など
テストデータ：	テスト値と同じ
※テストパラメータを網羅するために必要なカバレッジアイテムはデータのバリエーションであり、テスト値として導出するものは様々なデータ。テストデータとしてそのままテスト値を使うことができる。	

テスト値からさらにテストデータを導く必要があるもの	
テストパラメータ：	制御パス
テスト値：	パス1(a→b→d)など
テストデータ：	パス1を通るために必要なデータセットを指定する
※テストパラメータを網羅するために必要なカバレッジアイテムはパスのバリエーションであり、テスト値として導出するのは様々なパス(経路)。テストデータとしては、特定のパスを通るようなデータを作成する必要がある。	

テストパラメータやテストモデルを
特定せずに、闇雲にテストケースを
挙げるのは質の高いテストではない

テスト観点（テストパラメータ）に
対してどのような網羅基準をもった
テストケースのセットであるのかを
しっかりと説明できる必要がある

テストモデルには4つある

範囲タイプ

一覧表タイプ

ある連続した範囲を意味する
テスト観点を網羅する時に用いる

例) 辺の長さ（0以上65535以下の整数値）
範囲のことを「同値クラス」と呼び、同値クラスを導出技法を同値分割と呼ぶ

範囲タイプの網羅基準例

代表値網羅

テスト値は少数に収まるが漏れが発生する

例) 3

境界値網羅

範囲が開いている(上限/下限が不定)時は自分で定める必要があるがよほど良く検討しないと漏れが発生する

例)0, 65535, -1, 65536

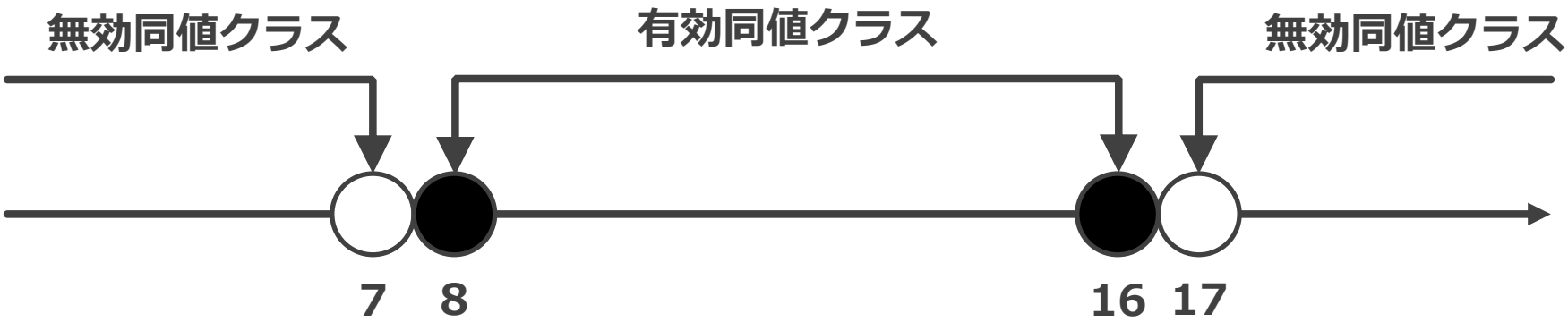
全網羅

漏れはないがテスト値が膨大になるため現実的ではない

例)0, 1, 2, 3, 4, … , 65534, 65535

同値分割・境界値分析

テストパラメータ：　「パスワードの文字列長」
テストモデル：　「範囲タイプ」
カバレッジ基準：　「境界値網羅」



- 境界値分析や境界値テスト、ドメインテストなど独立した技法として説明されることが多い
- 有効同値クラスの境界値だけではなく、無効同値クラスの境界値も忘れないこと
- 範囲が開いている（上限や下限が定まっていない）時は、自分で定める必要があるが、よほどよく検討しないと漏れが発生しやすくなる

テストモデルには4つある

範囲タイプ

一覧表タイプ

複数の要素からなる集合を意味する
テスト観点を網羅する時に用いる

例)ブラウザの種類

一覧表タイプの網羅基準例

代表値網羅

テスト値は少数に収まるが漏れが発生する

例) Chrome

境界値網羅

要素の何らかの属性が、ある連続した範囲の場合に、その属性の境界値を持つ要素を境界値要素とみなすことができなくもないが、よく検討しないと漏れが発生する

例)一番昔にリリースされたブラウザと一番最近にリリースされたブラウザ

全網羅

漏れはなく、通常はテスト値もそれほど膨大にならないが、組合せになったら無視できない程度にテスト値が多くなってしまう

例) Chrome, Firefox, Safari, Internet Explorer, Opera, etc

一覧表タイプの指標例

テストパラメータ：　「ブラウザの種類」
テストモデル：　「一覧表タイプ」
カバレッジ基準：　「代表値網羅（シェア80%以上）」

	ブラウザ	マーケットシェア	網羅率
1	Internet Explorer	43%	43%
2	Chrome	30%	73%
3	Firefox	13%	86%
4	Safari	3%	89%
5	Opera	1%	90%
6	その他	10%	100%

テストモデルには4つある

二つ以上のテスト観点の組合せを網羅するときに用いる

例)OSとブラウザの種類

範囲タイプ　　網羅率タイプ

マトリクスタイプ

グラフタイプ

マトリクスタイプの網羅基準例

代表値網羅

テスト値は少数に収まるが漏れが発生する
例)一番新しいOSと一番シェアの高いブラウザ

N-wise網羅

N+1以上の段数の組合せの網羅を意図的に無視することに
よって現実的な数でNまでの組合せを全網羅する
N-wise系の網羅手法と、直交配列表を用いた網羅手法がある

全網羅

漏れはないが掛け算によりテスト値が膨大になるため現実的ではない
例) OS(Win 10, 8.1, 8, 7, Vista, etc) × ブラウザ(IE, Edge, Firefox, Chrome)

※マトリクスタイプでは禁則(無効な組合せ)に注意する必要あり

ペアワイズテスト

テストパラメータ：「OS」「ｱｰｷﾃｸﾁｬ」「ブラウザ」の組合せ
テストモデル：　「マトリクスタイプ」
カバレッジ基準：　「 2-wise網羅（ペアワイズ）」

※N=2のとき
ペアワイズと呼ぶ

	OS	アーキテクチャ	ブラウザ
1	Windows 10	32bit	IE11
2	Windows 8.1	32bit	Chrome
3	Windows 7	32bit	Firefox
4	Windows 10	64bit	Edge
5	Windows 8.1	64bit	IE11
6	Windows 10	64bit	Chrome
7	Windows 7	64bit	IE10
8	Windows 8	32bit	Chrome
9	Windows 8.1	32bit	IE10
10	Windows 8	64bit	Firefox
11	Windows 7	64bit	Chrome
12	Windows 7	32bit	IE11
13	Windows 8.1	32bit	Firefox
14	Windows 8	32bit	IE10
15	Windows 8	64bit	IE11
16	Windows 10	32bit	Firefox
17	Windows 10	32bit	Edge

2因子網羅100% - 以下の組合せのすべてが少なくとも1回以上表中に出現

- **OSとアーキテクチャ**
- **OSとブラウザ**
- **アーキテクチャとブラウザ**

全網羅の場合38通りの組合せになるが、ペアワイズでは17通りの組合せになる

ペアワイズの網羅で十分かどうかを検討しないと漏れが発生する

テストモデルには4つある

丸と線で図を描けるような
テスト観点を網羅するときに用いる

例)プログラムのロジック、状態遷移図、シーケンス図、ユーザシナリオ、地下鉄の路線図など

マトリクスタイプ

グラフタイプ

※折線グラフや棒グラフのグラフではない
丸と線で図をグラフ、丸をノード、線を
リンク、丸と線による経路をパスと呼ぶ

グラフタイプの網羅基準例

代表パス網羅

テスト値は少数に収まるが漏れが発生する

ノード網羅（C0網羅）

グラフのノードを網羅するパスを生成(リンクの漏れ発生)

リンク網羅（C1網羅/0スイッチ網羅）

グラフのリンクを網羅するパスを生成(1スイッチ以上のパスの漏れ発生)

Nスイッチ網羅

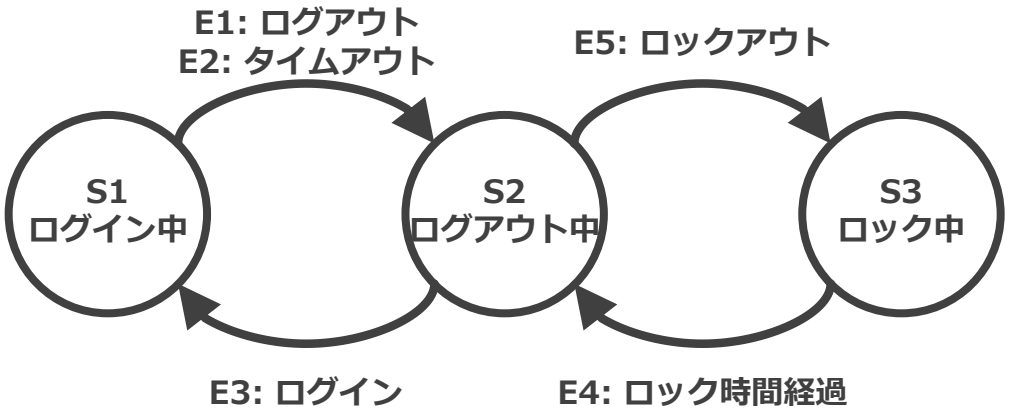
Nスイッチのパスを網羅（N+1スイッチ以上のパスの漏れ発生）

全パス網羅

漏れはないがテスト値が膨大になるため現実的ではない

状態遷移テスト

テストパラメータ:　「ログインステータスの遷移」
テストモデル：　「グラフタイプ」
カバレッジ基準：　「1スイッチカバレッジ」

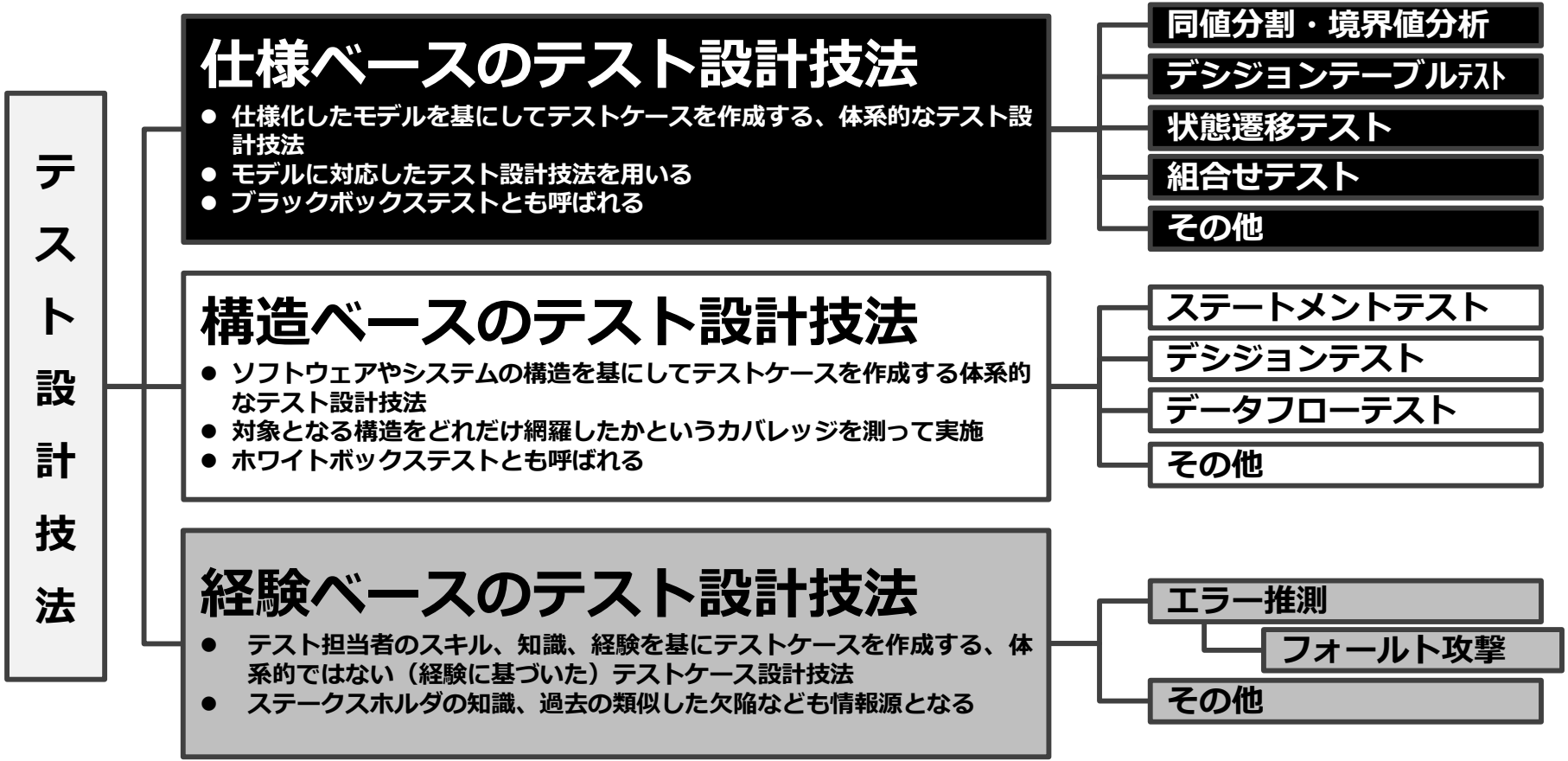


《状態遷移図》

遷移前/遷移後	ログイン中	ログアウト中	ロック中
ログイン中	E1E3+E2E3		E1E5+ E2E5
ログアウト中		E3E1+E３ E2＋E5E4	
ロック中	E4E3		E4E5

《状態遷移表》
1スイッチカバレッジの
パスは全部で９つある

様々なテスト設計技法を使ってテストパラメータを網羅する



チュートリアルの流れ

1. テスト観点を整理して網羅的に挙げよう
2. テストフレームを考えよう
3. テストコンテナとその間の順序関係を考えよう
4. テスト観点からテスト値を網羅しよう
5. テスト開発を意識して進めてみよう

チュートリアルの流れ

1. テスト観点を整理して網羅的に洗い出そう
2. テストフレームを考えよう
3. テストコンテナとその間の順序関係を考えよう
4. テスト観点からテスト値を網羅しよう
5. テスト開発を意識して進めてみよう

テスト要求分析

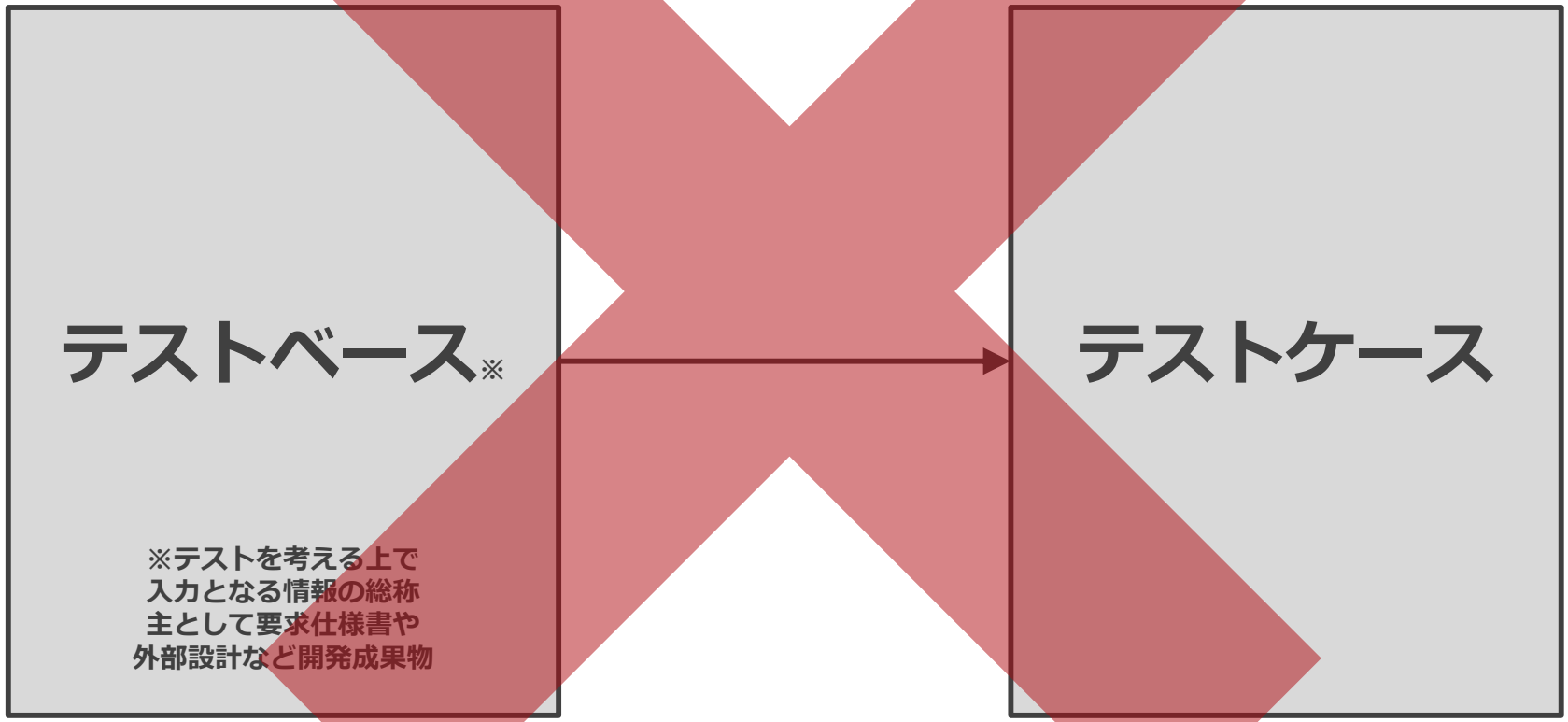
テストアーキテクチャ設計
(テストフレームモデリング)

テストアーキテクチャ設計
(テストコンテナモデリング)

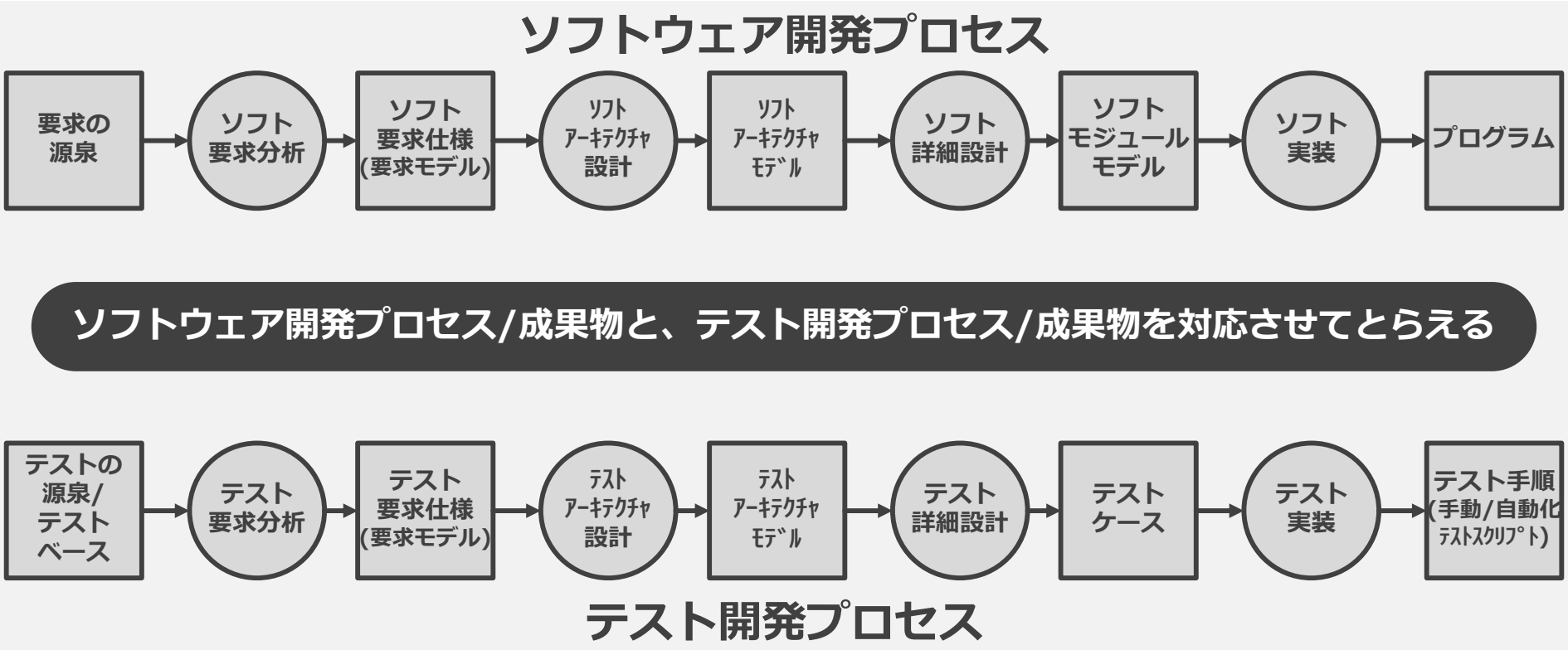
テスト詳細設計

テストを開発する段階的なプロセスが必要

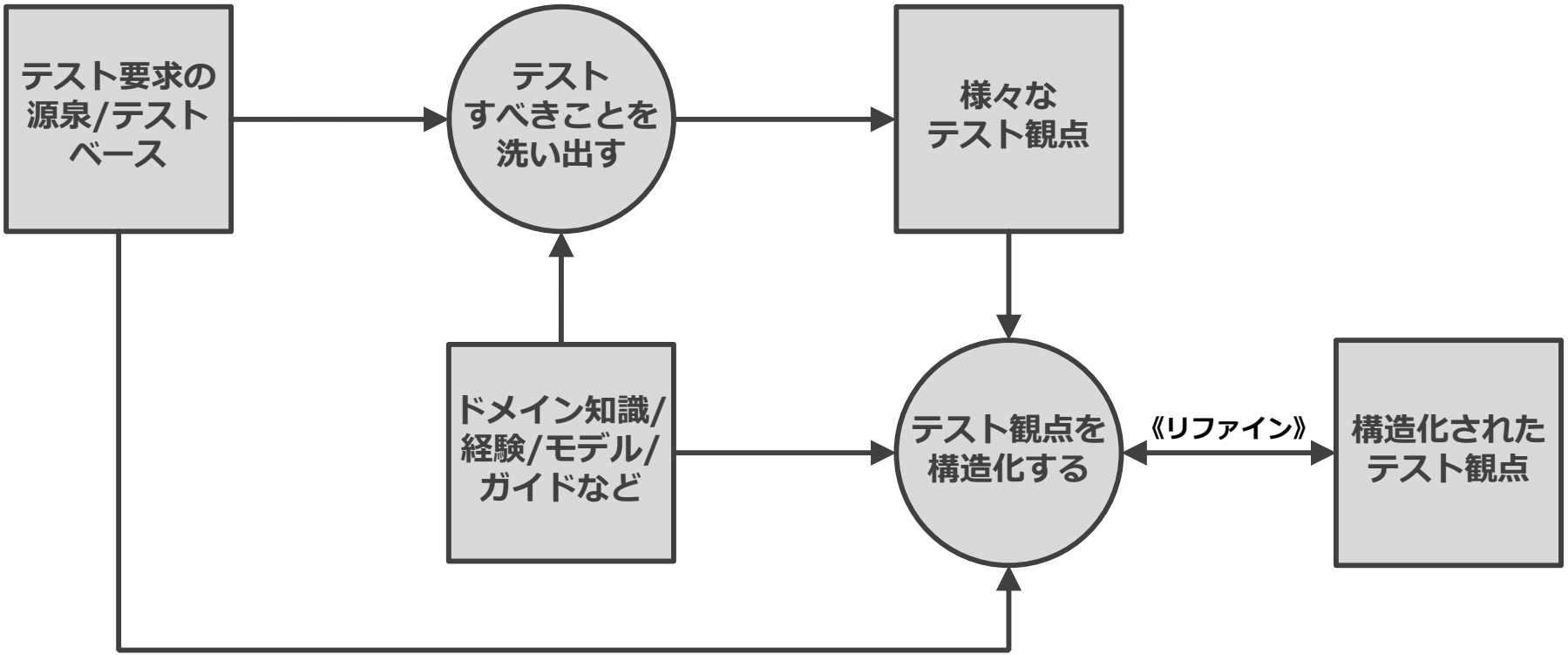
テストベースからいきなりテストケースを作るのは無理筋



テストケースを開発成果物と捉えて段階的に詳細化していく必要がある

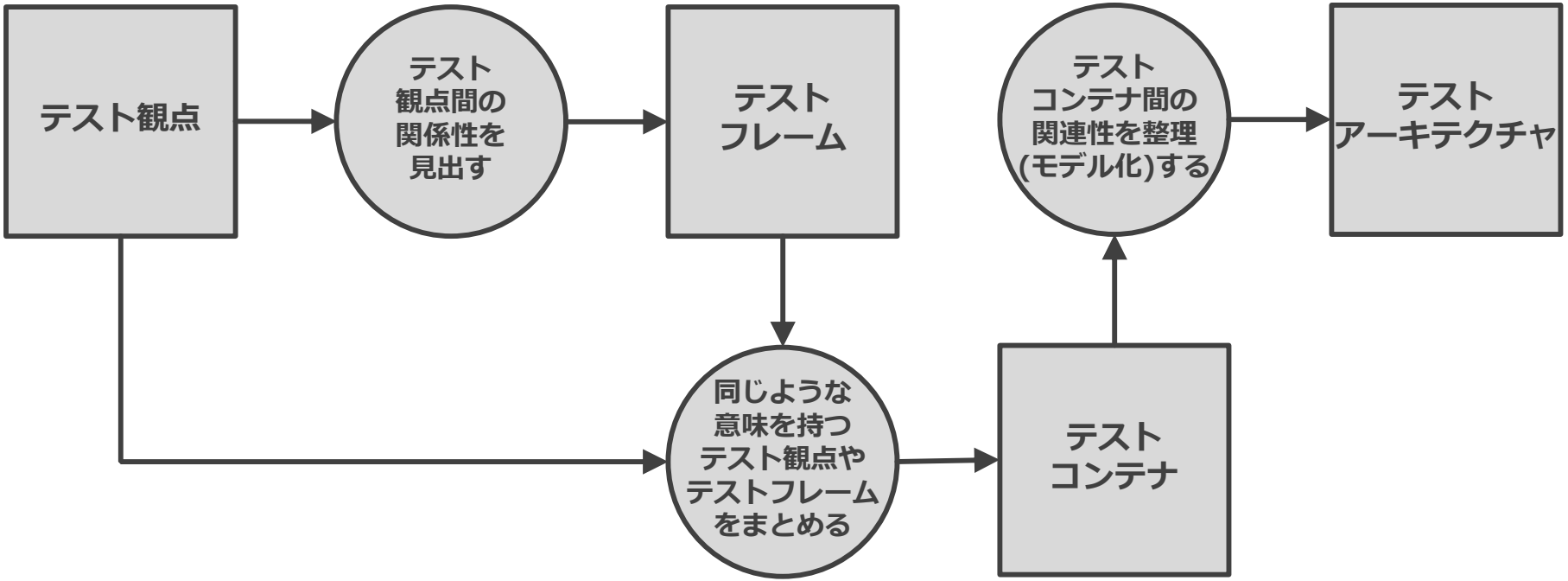


テスト要求分析の一例



※テスト要求分析を定義したものではなく、具体的にイメージするためにテスト要求分析をプロセス設計した一例

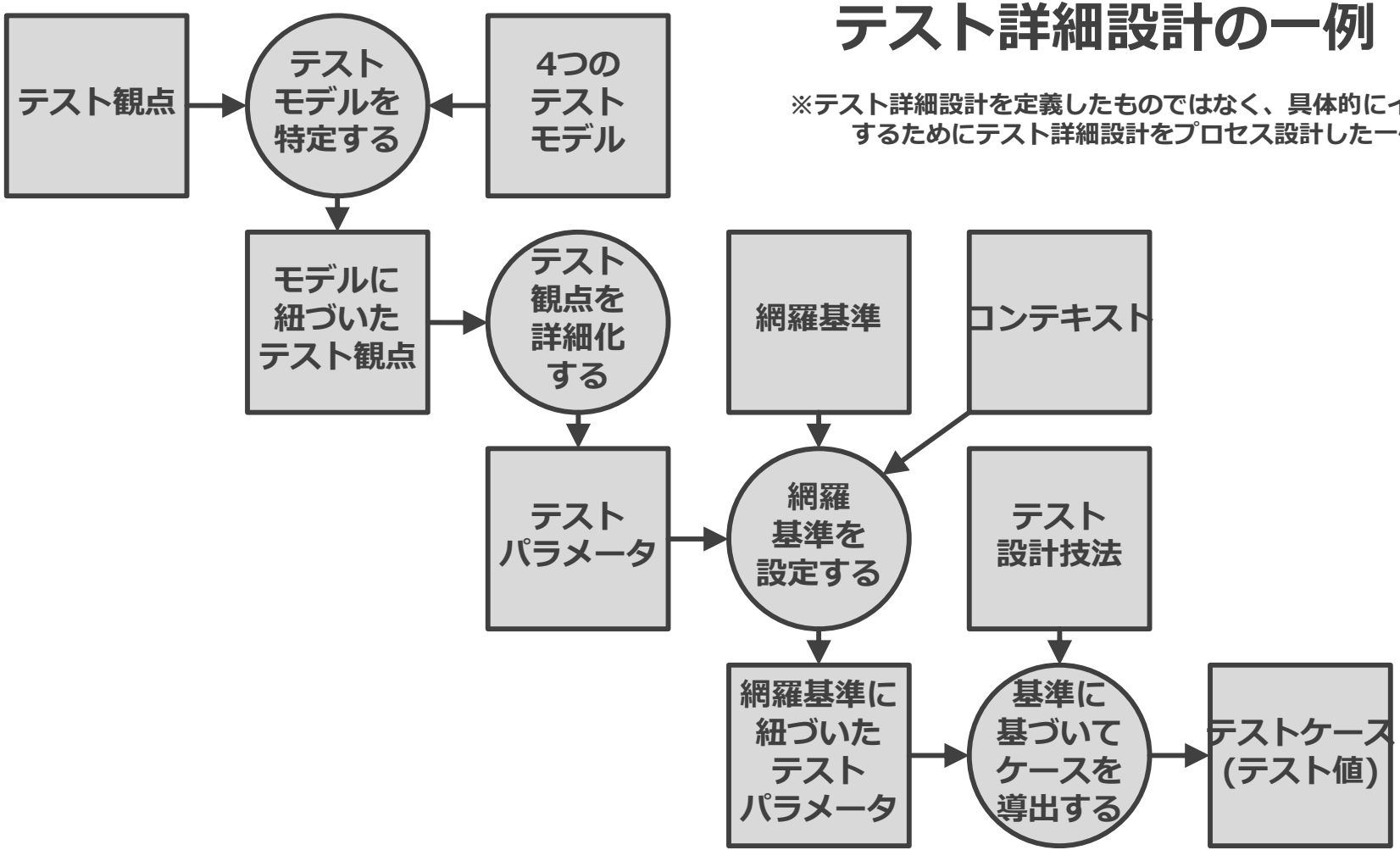
テストアーキテクチャ設計の一例



※テストアーキテクチャ設計を定義したものではなく、具体的にイメージするためにテストアーキテクチャ設計をプロセス設計した一例

テスト詳細設計の一例

※テスト詳細設計を定義したものではなく、具体的にイメージするためにテスト詳細設計をプロセス設計した一例



用語集

用語	意味
テスト観点	テストすべきこと。テストパラメータや、テストパラメーターを抽象化したもの
テストフレーム	テストケースの構造。テストケースを構成する複数のテスト観点
テストコンテナ	テスト観点やテストフレームをまとめたもの
テストアーキテクチャ	テストコンテナ間の前後関係や並列関係を決めたもの
テストパラメータ	テスト観点の最も具体的なもの
テストモデル	テストパラメータの属性（範囲タイプ、一覧表タイプ、マトリクスタイプ、グラフタイプの4つ）
テスト値	テストパラメータが取る具体的な値（数値や経路などテスト値の集合でテストパラメータを網羅する）
テストデータ	テスト値をテスト手順として実装する際に実際のデータとして定義されるインスタンス
テスト条件	テスト観点やテストフレームの別称

「テスト開発プロセス」を意識して日々の仕事に活かしてみよう

大規模化・複雑化する一方で高品質・短納期も求められるソフトウェア開発のために、柔軟で高速かつ精度の高いソフトウェアテストを開発することが社会的な急務になってきている

- 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、いかに体系的にスピーディに開発するか、が課題である

そのため「テスト開発プロセス」を構築し、見通しがよく保守性の高いテスト設計を行う必要がある

- 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
  - 柔軟かつ高速に、質の高いテストケースを開発するために必要な作業や、テスト観点、テスト値、テストフレーム、テストコンテナといった中間成果物を明らかにする

このチュートリアルとテスト設計コンテストは、そのための第一歩

- 高度な内容も含まれているが、テスト技術は奥が深いということを感じ入ってもらい、自分の腕を上げる機会が準備されているのだとワクワクして欲しい
- チュートリアルとコンテストでの経験や気づきを自分の日々の仕事に活かしてほしい
  - チュートリアルを聞くだけでは腕は上がらないし、コンテストに出るだけでは一過性のイベントになってしまう
  - 悩みながら自分の日々の仕事に活かしてこそ、腕が上がったと言えるし身についたと言える

https://www.pakutaso.com/20160111020post-6640.html

質疑応答

何なりと質問してくださいね

付録1

テスト設計コンテストU-30について

テスト設計コンテストとは

- 目的
  - ソフトウェアテストを分析設計から行うことを周知し、ソフトウェアテストエンジニアに対する教育の機会を提供する。
  - コンテストという形式をとることにより、ソフトウェアテストが創造的な作業であり楽しいということを経験してもらい、若年層及び初級テストエンジニアからベテランテストエンジニアまでテストへの興味を高める。
  - ソフトウェアテスト業界における技術開発を競技を通じ、促進する。
- 形式
  - 全国の地域予選を勝ち抜いたチームが決勝で腕をさらに競う
  - バグ出しコンテストではない。テスト設計成果物の良さを競うコンテスト
  - 各チームは共通のテストベースに対するテスト設計を行い、プレゼンテーションを行う

テスト設計コンテストの今まで

2011
【大賞】めいしゅ館（東海）
【湯本賞】奥村 健二（東海）
【にし賞】堀米 賢（東京）

2012
【最優秀賞】TETTAN（東京）
【審査員特別賞】あまがさきてすとくらぶ（関西）

2013
【優勝】TETTAN（東京） 2連覇
【準優勝】Yuki Da RMA（北海道）

2014
【優勝】ＴＦＣ　ＫＡ・ＲＩ・ＹＡ（東海）
【準優勝】MKE98（審査委員推薦枠：東海）

2015
【優勝】しなてす（東京）
【準優勝】TEVASAKIplus （東京）

2016
【優勝】SASADAN Go （書類選考）
【準優勝】しなてす （東京）

2017
OPEN
【優勝】STUDIO IBURI （北海道）
【準優勝】わんだーズ（東京）
U-30
【優勝】でこパン462
【準優勝】SHINNOSUKE

U-30クラス新設

2018
OPEN
【優勝】てすにゃんV3
U-30
【優勝】TBD

テスト設計コンテストの審査ポイント

- テスト設計コンテストの審査基準の多くは、普段の業務におけるテスト設計のレビューにも有効である
  - コンテスト用に特化したテスト設計では勝てないように工夫されている(はず)
  - 審査基準は自組織のテスト設計のレビューのチェックリストやガイドラインの素案になりうる
    - そもそも自組織でテスト設計をレビューしていますか？
    - テスト設計レビューのチェックリストやガイドラインをきちんと作成し、成長させていますか？
    - 自組織のチェックリストやガイドラインによく引っかかるテスト設計の問題点や、それらでは検出できない問題点を把握していますか？
  - 自組織の技術レベルに応じて、審査基準の解釈のレベルも変わりうる
    - 自組織のテストアーキテクチャ設計の技術レベルに応じて「全体像を把握」できているかどうかを判断する基準は当然異なる
    - 同じ審査基準でも、判断基準の妥当性を常に見直さないといけない

テスト設計コンテストの審査ポイント

- テストは説明責任の塊であるので、説明責任の果たされていない成果物は点数が低くなりやすいし、実務においても「ふーん、それで？」的になりやすい
  - ダメなテストは単なる作業報告になっている
    - これをやりました、あれをやりました、こう書きました…
    - これでは単なる作業報告や日報である
  - よいテストは、説明責任がきちんと果たされている
    - 説明すべきことが必要十分に説明されている
    - 一貫した、統一の取れた説明がなされている
    - 粒度の揃った説明がなされている
    - 全体像が把握しやすい
    - 意図や目的、根拠、理由が明示されている
    - トレードオフや設計選択の根拠や選択肢が説明されている
    - 構成要素とその構造が明示されている
    - なぜその構造が適切なのかが理解できるようになっている
    - 意図や目的、根拠、理由がきちんと果たされたことが一目で分かるようになっている
    - レビューしやすい配慮がなされている

審査基準を読み解くにあたってのアドバイス

様々な審査基準をフラットに考えると項目が多くて手に余りやすいのでU-30向けに意識したほうがよい審査基準を参考として示します

優先度	説明
High	テスト設計において基礎・基本となる要素であり、U-30としてまず初めに意識して検討していただきたい項目
Medium	より良いテスト設計のために必要となる要素であり、基礎・基本を押さえたうえで意識して取組んでいただきたい項目
Low	より納得感を与え、流用・展開可能な高度なテスト設計になる要素であり、OPENクラスも見据えて意識していただきたい項目

テスト要求分析・テストアーキテクチャ設計 （40点）		優先度
1.1	テスト要求分析・テストアーキテクチャ設計レベルでテストすべきこと（テスト観点）が考慮されているか	High
1.2	テスト観点がすべて盛り込まれているか	High
1.3	テスト観点が適切な粒度で表現されているか	Medium
1.4	テストレベルやテストタイプ、テストサイクルなどを用いるなどして、テストの全体像が把握しやすいか	Medium
1.5	テスト対象やテスト目的など、テスト観点間の関係が分かりやすいか	Low
1.6	テストの厚みやバランスが考慮されているか	Medium
1.7	テストスコープが明示されているか、明示されたスコープは妥当性があるか	High
1.8	仕様書の問題を指摘できているか	High

テスト詳細設計・実装（30点）		優先度
2.1	特定のテスト観点において、テストケースが十分に記述されているか	High
2.2	選択したテスト観点が適切か	Medium
2.3	テストケースの構造が適切に表現されているか	Medium
2.4	テスト詳細設計・実装のための適切なテストの技法や記法を用いているか	High
2.5	網羅性やピンポイント性は適切に考慮されているか	Medium
2.6	リスクを考慮した優先順位付けが行われているか	Medium

工程間一貫性（10点）		優先度
3.1	テスト要求分析・テストアーキテクチャ設計とテスト詳細設計・実装の間のトレーサビリティが取れているか	Medium
3.2	テスト要求分析・テストアーキテクチャ設計とテスト詳細設計・実装のバランスが良いか	High
文書（20点）		**優先度**
4.1	文書の情報量は適切か	Medium
4.2	文書の構造は適切か	Medium
4.3	文書の論理性は高いか	High
4.4	文書は分かりやすいか（正確な情報が速やかに伝わるか）	Medium
4.5	文書の説得力は高いか	Low
4.6	文書体系が示されているか	High
4.7	文書の保守性を考慮しているか	Medium

総合（20点）		優先度
5.1	技術レベルが高いか	Low
5.2	独自性は高いか	Low
5.3	新規性は高いか	Low
5.4	発想は豊かか	Medium
プレゼンテーション技術（15点）※		優先度
6.1	内容、表現手法、その他特別点	Low

※書類選考予選ではプレゼンテーション審査は行いません。

最新情報はWebサイトにて随時発表します！

http://aster.or.jp/business/contest/rulebooku30.html

付録2：

2018年度までの傾向

- テスト開発プロセスに関する理解の不足
  - テスト要求分析、テストアーキテクチャ設計、テスト詳細設計がまだまだ腹落ちしておらず、テスト開発プロセスが未成熟
- 成果物であるドキュメントの質が低い
  - 納品物の体をなしていないものも多く、第三者が読めるものになってない
  - ドキュメントの体系が示されておらず、トレーサビリティもとれていない
  - 用語の未定義、または一般の意味とは異なる用語の使い方などが散見される
- プロセスや成果物に対する意図が読み取れない
  - 何故そうしたのか、それによって作られた成果物がなにを意味するのか不明
  - 最悪の場合は、本人も説明できないことも…
- テスト設計技法をしっかり使えていない
  - そもそもテスト設計技法があまり使われていない
  - 使われている場合でも技法として正しく使われていない

写真・画像素材
PAKTASO
REEIMAGES
Designed by Freepik.com

参考資料

ソフトウェア・テストの技法 第2版 Glenford J. Myers
ソフトウェアテスト標準用語集(日本語版)Version2.2.J03
テスト観点に基づくテスト開発方法論VSTePの概要 西康晴氏
事例とツールで学ぶHAYST法 秋山浩一氏
基本構造構成要素 鈴木三紀夫氏
ゆもつよメソッドにおけるテスト分析とテスト設計 湯本剛氏
高信頼化ソフトウェアのための開発手法ガイドブック
テスト設計コンテストOPENクラス チュートリアル資料 2017年度版
テスト設計コンテストU-30クラス チュートリアル資料 2017年度版
テスト技術者資格制度Foundation Levelシラバス日本語版Version 2011.J02

テスト設計チュートリアル U-30クラス版 資料 2020 Ver1.0.0

2019年11月22日発行

編集・発行　テスト設計コンテスト実行委員会
特定非営利活動法人 ソフトウェアテスト技術振興協会(ASTER)

連絡先 特定非営利活動法人 ソフトウェアテスト技術振興協会(ASTER) 事務局
〒１０５－００１４東京都港区芝2-29-10　ユニゾ芝2丁目ビル 7F
電話 ０３－５４４４－７６０１ FAX ０３－５４４４－８０９５
E-MAIL aster-tdc-query@qualab.jp
URL http://aster.or.jp/