ソフトウェアテストの展望
SW機能テストから、システム挙動の評価へ

DebugEng デバッグ工学研究所
http://www.debugeng.com/
松尾谷 徹
(法政大兼任講師)

目 次

1. 課題は何か
2. 過去を振り返る
3. 禁則、有則、無則
4. 実装、設計との連携
5. システム挙動
6. まとめ

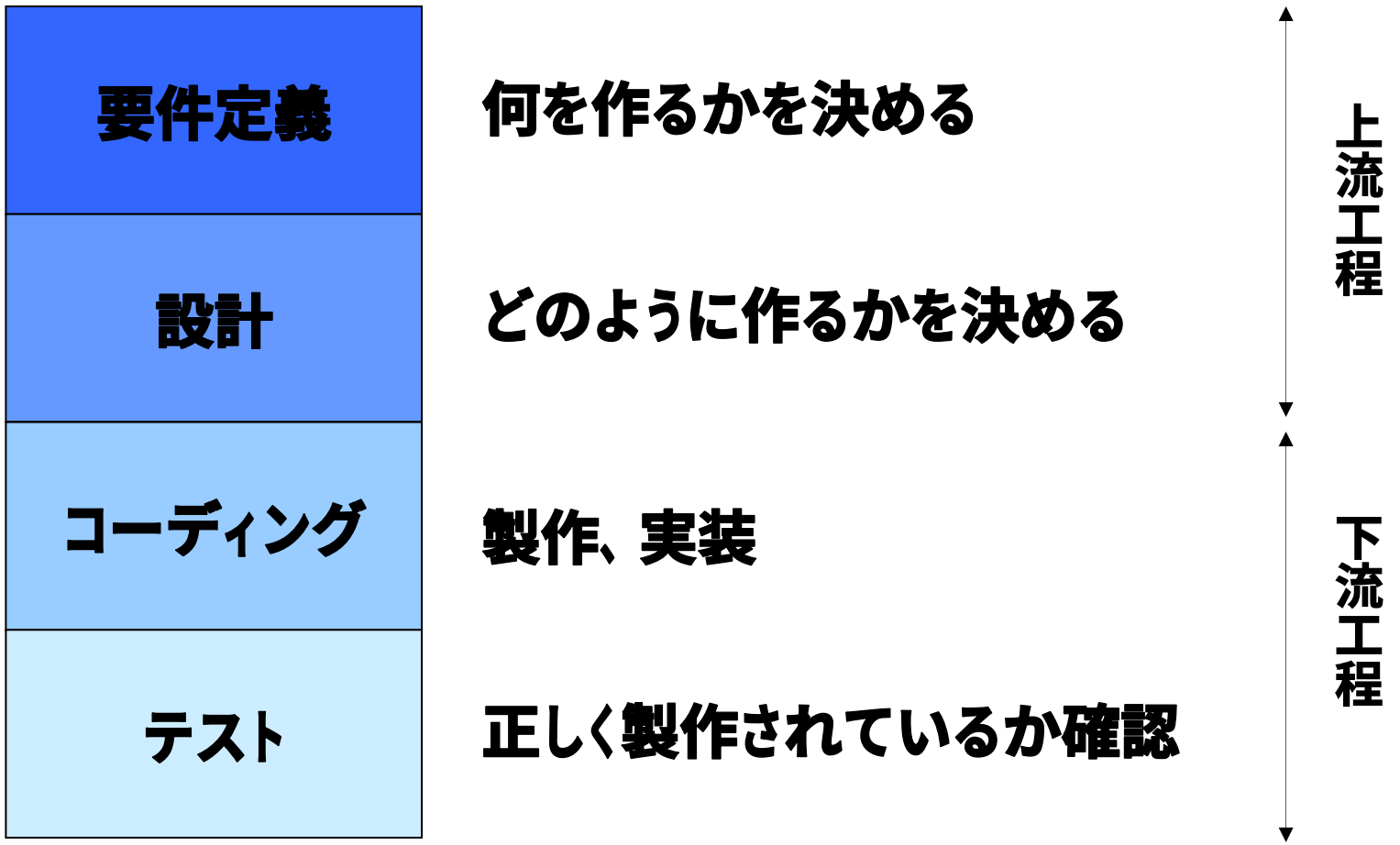
Debug Engineering

１．課題は何か

■テストを概観する

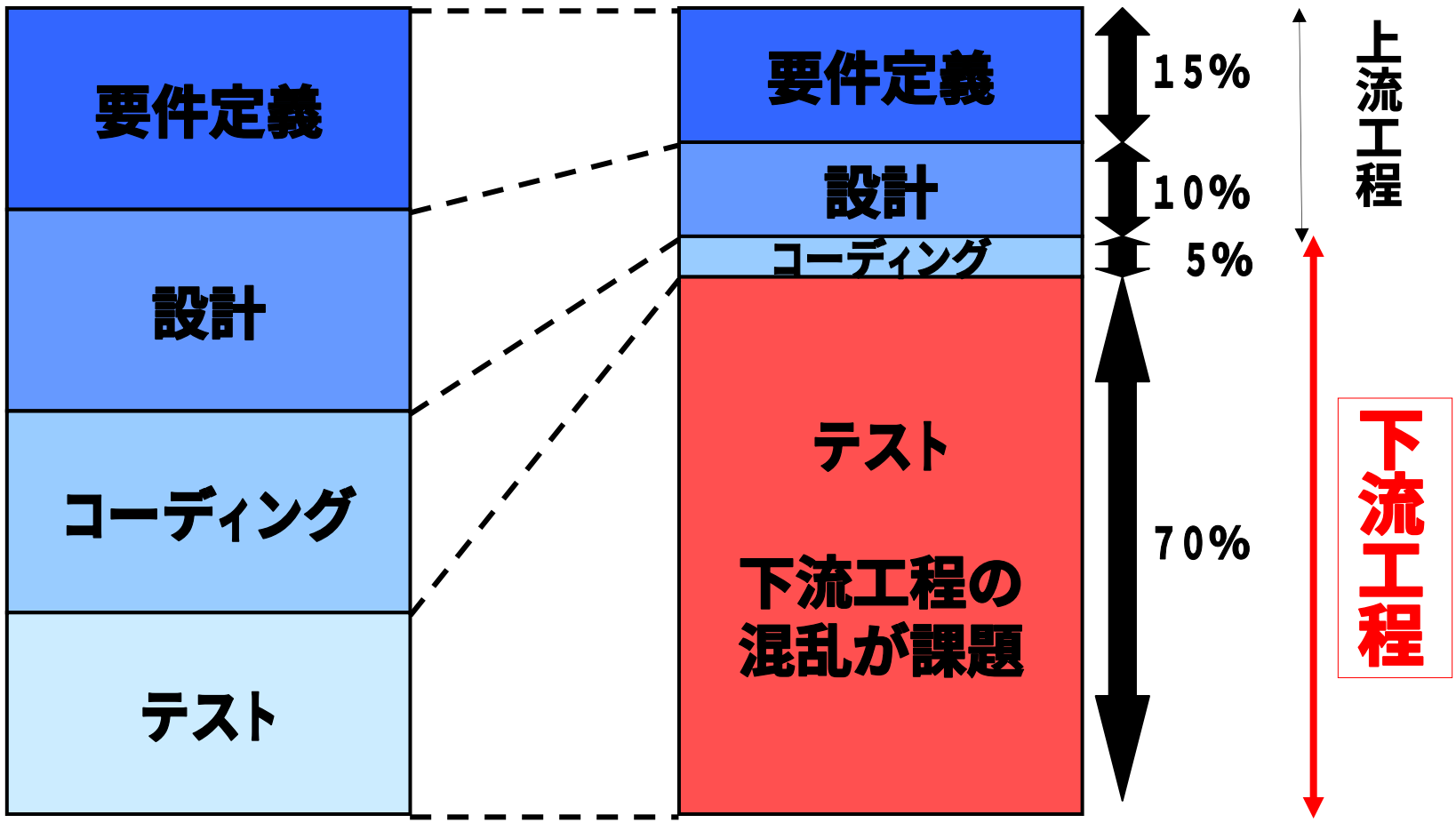
原点に戻って課題を整理する

■ソフトウェア開発ライフサイクル



何が課題なのか？

■SW開発における最大にコスト要因はテスト



工作機械の組込みソフトウェア開発

テストにコストが必要な訳と、対策（アプローチ）

- 何故、テストに膨大なコストが投入されるのか？

  実用に耐える品質でない

  テストと手直しのコスト（デバッグ）

- この課題に対するアプローチは、

1. 上流工程を改善し、実用に耐える品質にする。
   上流アプローチ
2. テスト自身を合理化し、実用に耐える品質にする。
   下流アプローチ
3. 両者の合理的な役割分担により、達成する。
   統合アプローチ

Debug Engineering

２．過去を振り返る

■何が解決して、何が未解決なのか

■下流工程の混乱

昔も、今も混乱は続いている・・・・現状認識

アプローチの結果

■*上流アプローチ*<設計重視>

- *成熟期:技術、管理など百花繚乱で多くの成果を出した*
- *しかし下流工程の混乱を解決できていないから*
- *何故解決できないのか？*
- *それは:　上流工程の品質向上は、テストのコストを下げないから*

■*下流アプローチ*<テスト回帰>

- *展開期:先進企業は合理的に品質を達成できている*
- *しかし、多くの企業は昔のまま(格差拡大期)*

■*統合アプローチ*<検証指向>

- *黎明期:さらなる品質のため今後の課題*

上流アプローチの限界

■上流工程の品質向上は、下流工程のコストをそれほど低減できなかった。

下流アプローチ：整理が必要

■SWテストを体系化したものは？

ISTQB（International Software Testing Qualifications Board）のシラバスをJSTQC（Japan Software Testing Qualifications Board）が翻訳した。

http://www.swtest.jp/certification.html

よく出来たシラバスである。

■主に2つの視点からテストを体系化している

1. ライフサイクルから体系化
2. テスト設計技法から体系化

- *但し入門編*

Debug Engineering

3．禁則、有則、無則

■現在における、テストの実力

テストによって何を確認するのか？

Q1．　テストによって何を確認するのか？

A1.　仕様通りに作られていることを確認する

＜仕様ベースのテスト＞

Q2.　　仕様は何を定義しているのか？

A2.　機能を定義している

＜仕様ベースの機能テスト＞

このAnsは入門編です

仕様ベース･ブラックボックスのテスト技法

JSTQCのシラバスより

1. 同値分割法　　：入力を部分集合として扱う
2. 境界値分析　　：同値間の境界でテストデータを作る
3. デシジョンテーブルテスト　　：同値の組合わせテスト
4. 状態遷移テスト　　：順序論理に対するテスト
5. ユースケーステスト　　：入力の定義された順序性に対するテスト

1，2はテストの下準備

3，4，5がテスト技法

もう少し詳しく考える

組合わせテスト

- **組合わせ論理のテスト**

1. **全組み合わせ：マトリックステスト**
   - *単体テスト以外では非現実的*

2. **部分組合わせ：デシジョンテーブルテスト**
   - *全組合わせの、どの部分を抽出するか？*

抽出法が一番重要（しかし技法が普及していない）

仕様を考える：禁則、有則、無則

■禁則：組合わせることが出来ない入力セット

- 禁じられている則、例外的な則
- 完成度の高い仕様には書かれている（かも）
- 実験計画法、All-Pair法などで選択されないようにしている
- テストしなくて良いのか？　　禁則でも書が無いことは確認する必要がある

■有則（造語）：定義された機能が動作する入力セット

- 求められた有用な則で仕様に有る則
- 例えば、「これこれ」の条件で、「これこれ」の処理（出力）が行われる
- 普通のテストは、有則を対象としている

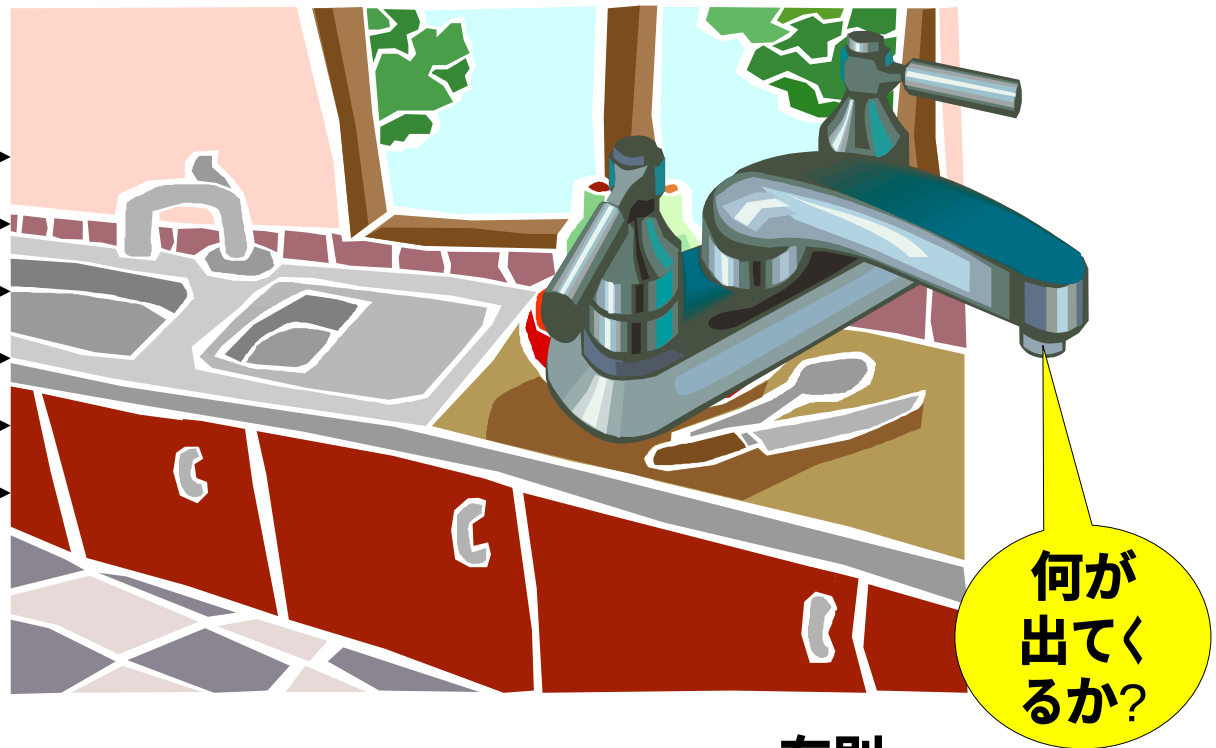
■無則（造語）：組合わせても何の影響もない入力セット

- 則が無い則
- 仕様に書かれることはまれ、定義された組合せの補集合
- 無則のテストは、悪影響が無いことをテストする
- 実験計画法、All-Pair法は、無則の一部をテストする技法

確認するのは何則？

■マンション購入時の瑕疵確認テストについて考える

水道
ガス
電気
電話
LAN
テレビ
喚気
・
・
・



有則
禁則
無則

ガスと電話?
お湯とテレビ?

テスト項目を整理すると

無則: テレビ、LAN、電話などの影響が無いこと

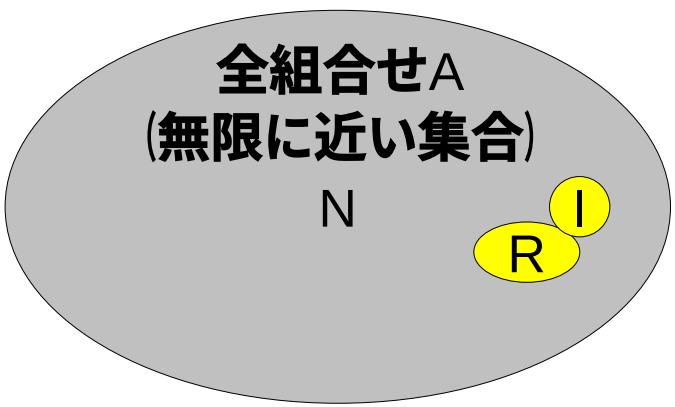
- テスト項目数が爆発するのでテスト不可能
- 故障統計から、2項目の組合せに重点を置く：実験計画法、All-Pair法

禁則: ガスの圧力が2倍になったら

- そんなことしてはいけない
- しかし、供給者は、圧力弁、温度異常など安全性に関してテストが必要です。
- 経験的には意地悪テストなど

有則: 水道、ガス、電気が定格入力状態で、お湯が出る

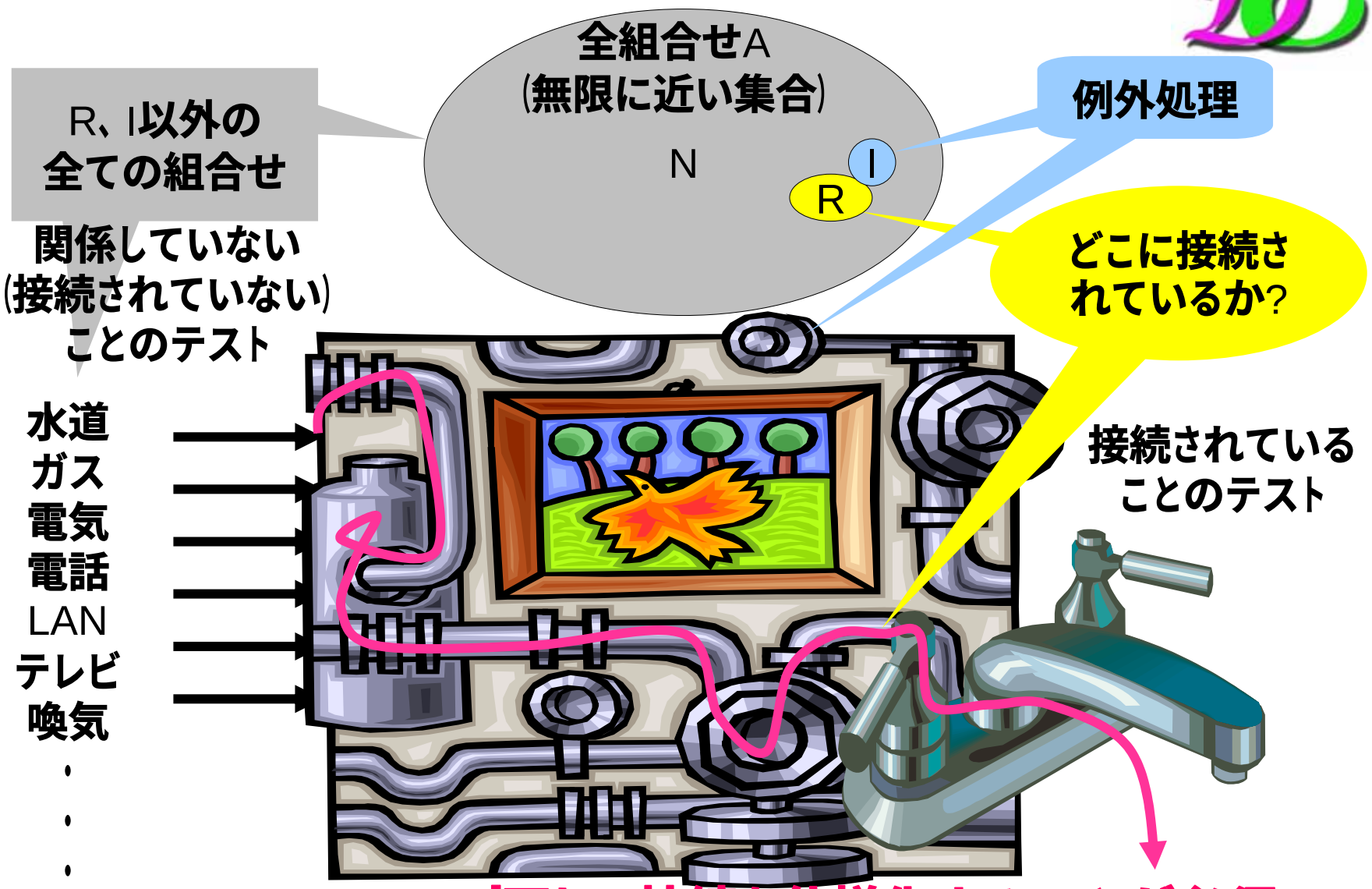
お湯の量は？温度は？（正解値が必要）



- 結合テスト以降の手法が問題
- 経験的な方法でデシジョンテーブル
- 原因結果グラフからデシジョンテーブル
- 原因流れ図からデシジョンテーブル

N: No effect　R: Requirement　I: Inhibition

具体的にはどういうことか？



「正しい接続を仕様化する」ことが必須

さらに順序論理の課題

■順序論理：入力の順序によって機能が異なる論理

- *例　締日と伝票修正（締日前ならデータ修正、後なら誤発注処理）*
- *　　通信プロコル、組込み系のシーケンス制御*

■順序論理のテスト

- *電子回路/ICのテストにおいて研究済み*
- *結論は、状態変数をテスト入出力として取り扱うこと*
- *これが実装されているテスト技法が見当たらない*

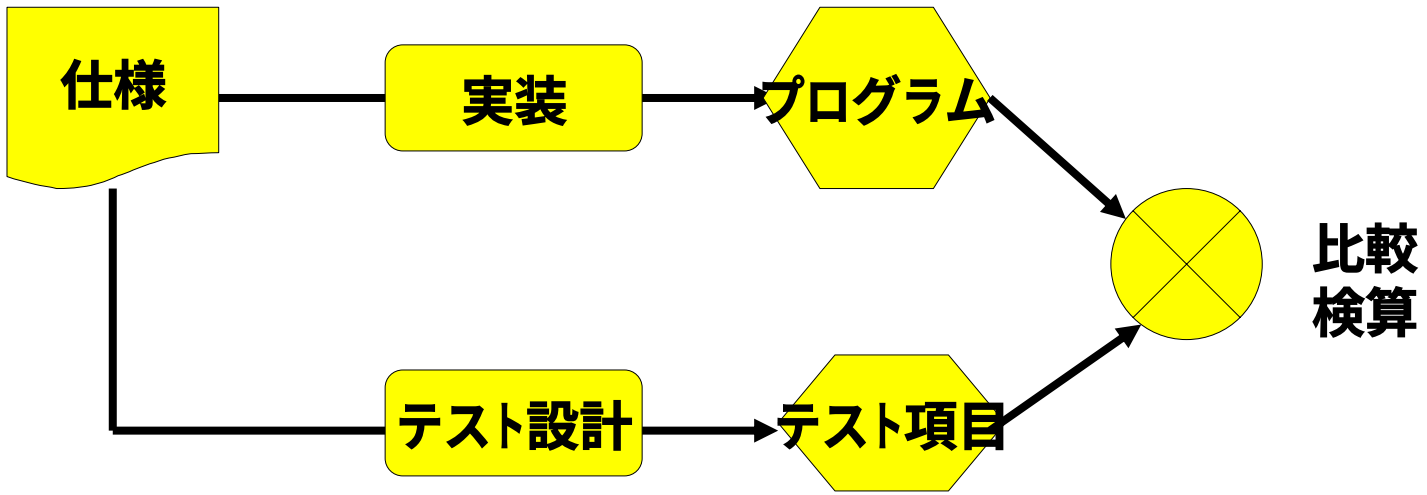
  CFD（Cause Flow　Diagram）は、これを実装している

■順序論理にも、禁則、無則、有則がある

- *現在テストできるのは、*
- *単体レベルなら、状態を同値と見なして全組合せ（状態マトリックス）*
- *それ以上になると、禁則と有則 でも困難*
  - テスト項目数は指数関数で爆発するのでモデル検証でも同様

テスト技法の実力　まとめ

■仕様書通りにつくられているか否かを確認する



■仕様書は、有則を定義しているが、禁則や無則の確認も必要である。(禁則、無則のバグが存在するから)

■単体テストは、全組合せ(状態遷移も含めて)が可能

■結合テストは、有則、禁則を抽出するテスト技法が必要

■無則のテストは組合せ論理の一部しかできない

仕様ベース･テスト技法の例

■技法との対応例

		有則	禁則	無則
仕様ベース・テスト技法	組合せ論理 単独	有効系 DT	無効系 DT	一部なら 実験計画
	順序論理 単独	状態遷移 マトリックス	状態遷移 マトリックス	小規模 なら モデル検証
	複合論理 (普通のSW)	拡張CFD の有効系DT	拡張CFD の無効系DT	ランダムテスト
挙動ベース		次の課題		

Debug Engineering

4．実装、設計との連携

■統合アプローチの可能性

統合アプローチの違い

■いままでの上流アプローチ

- *品質を高めることを狙った。しかし、バグ・ゼロにはできない。*

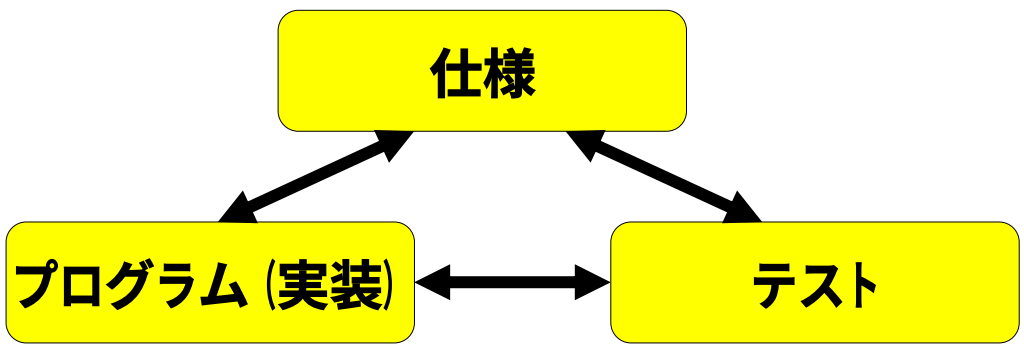
これから：テストを効果的に行うためのアプローチを考える

■いままでの下流アプローチ

- *極力、実装や設計とは独立に進めてきた（例えばブラックボックステスト）*
  しかし、テスト項目数の爆発が生じている。

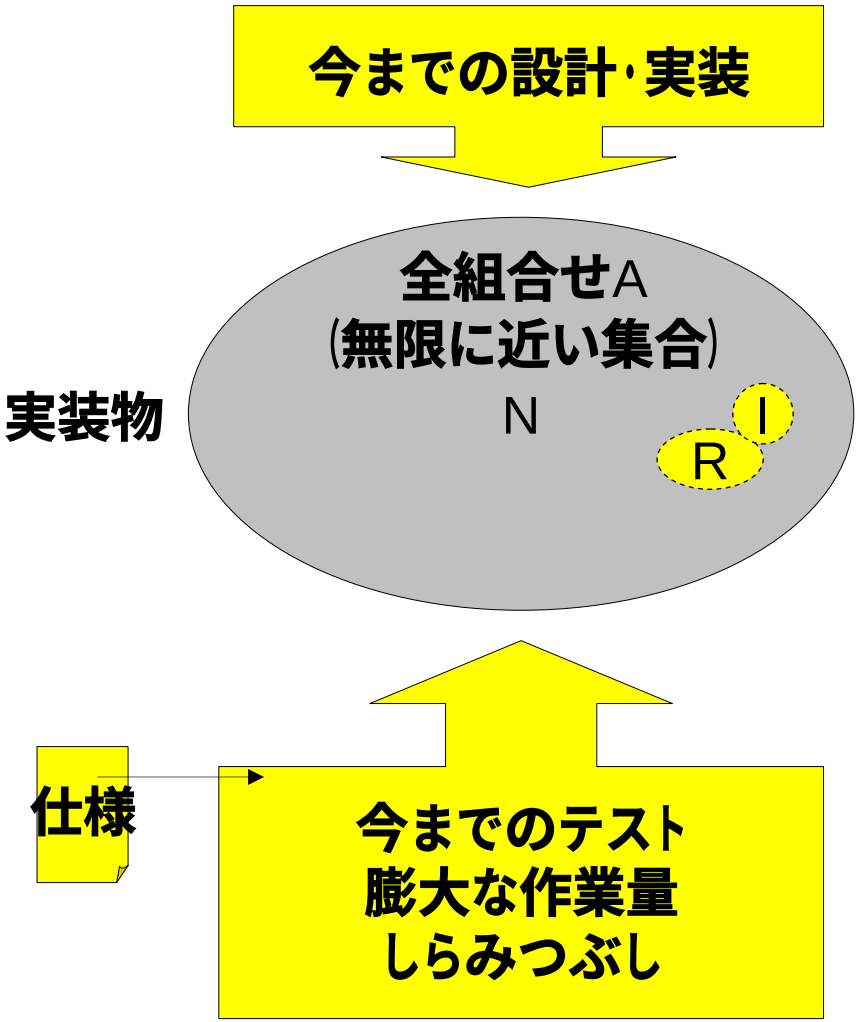
これから：テストで確認する範囲を狭くし、確実に行うアプローチを考える

■さらに、トライアングル検証

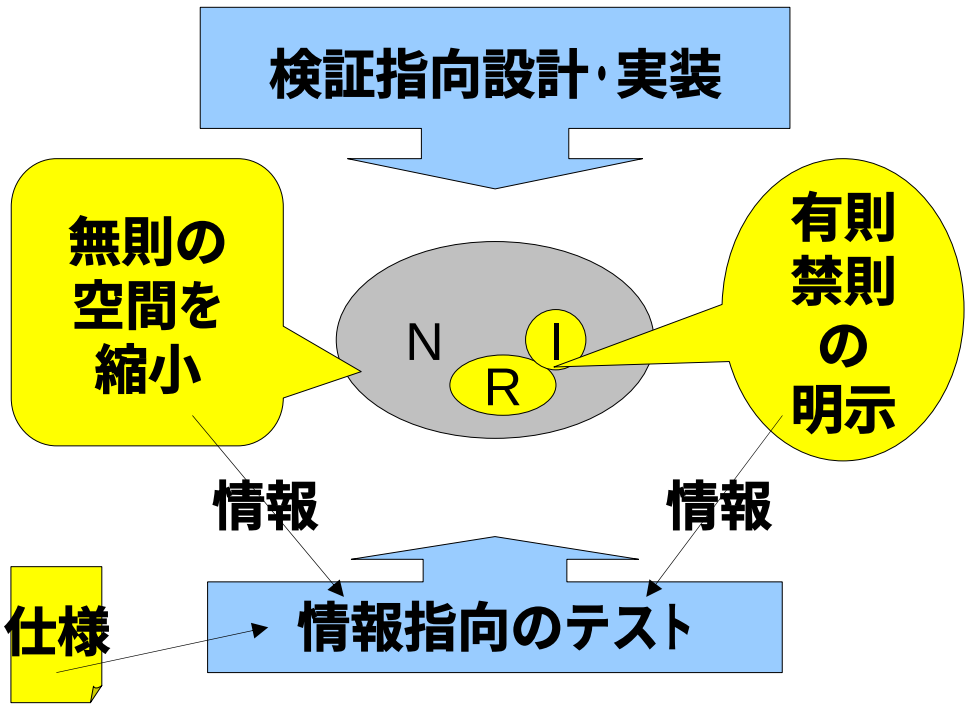


統合アプローチの概念図

作り易い設計、実装



検算を意図した設計、実装



探索に係わる情報量を増やし
合理的なテストを行う

今までにもあった

■ベテランの知恵

- *アサーション、アブノーマルリターン(処理打ち切り)、*
- *引数のアドレス渡しを禁止、初期値の徹底、・・・・*

■安全性が要求される分野

- *再帰アルゴリズムの禁止： 探索空間の縮小*
- *チップ自身を分離： 無則の範囲を狭くする(独立性を高める)*
- *MISRA-C (コーディングルール)*

■検証指向プログラミング (VOP)

- *実装段階においてCFDを記述し、テストとの情報交換を密にする*

■検証指向設計 (VOD)

- *データの制約(状態変数、制御変数、処理変数)と*
- *モジュール属性*
  - パッシブモジュール：判断なし、処理のみ
  - アクティブモジュール：処理なし、判断のみ

統合アプローチを進めるには

- **品質保証的なテストでは実現できない**
  - *設計、製造、検査の枠組みでは無理*
- **設計、実装、検証のプロセス構造設計**
  - *プロセスだけでは出来ない*
- **対象となるプロダクトの構造まで入って考える**
  - *分野によって異なった形態になると思われる*
  - *エンタープライズ系*
  - *組込み系：チップ焼き込み系、安全系、リアルタイム制御系、・・・*

仕様ベースのテスト課題は解決のめどがある

- **次の課題、システム挙動のテスト**

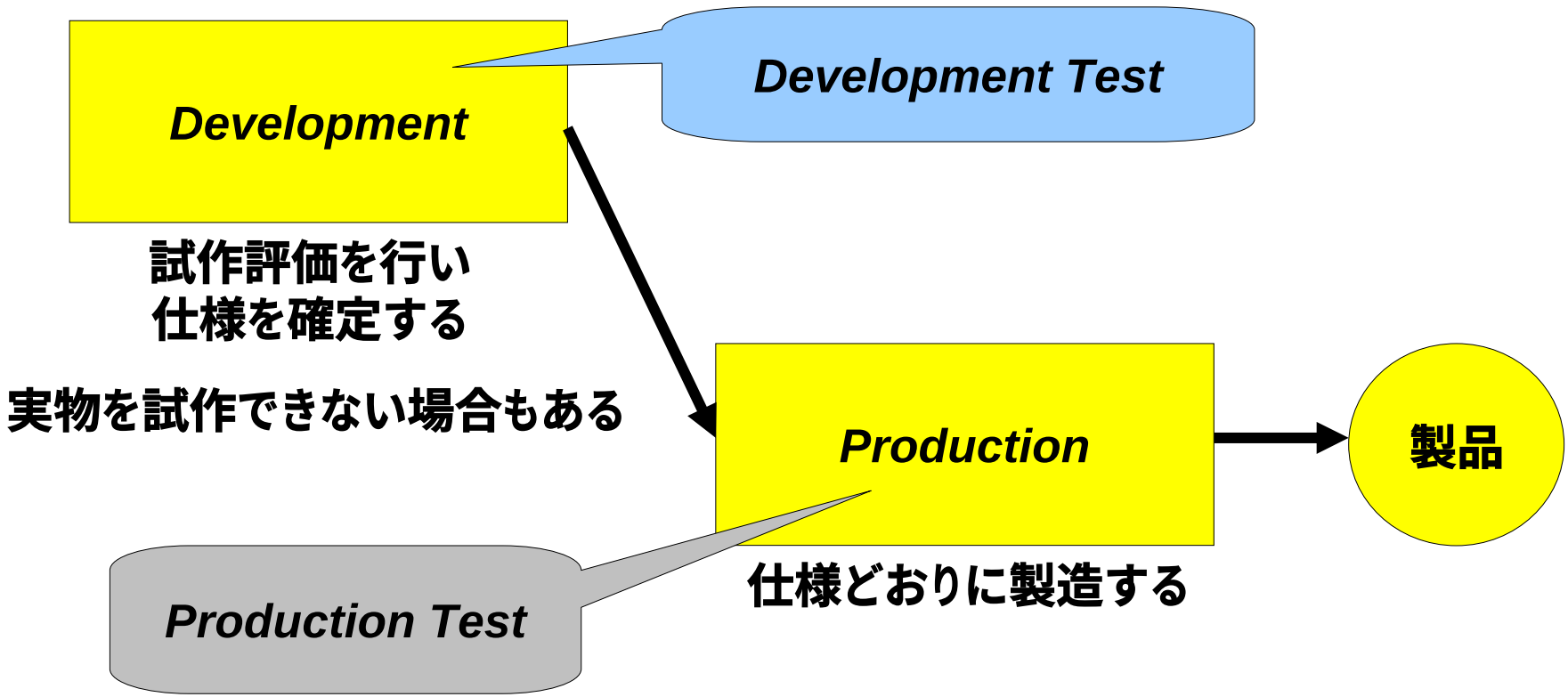
Debug Engineering

5．システム挙動

■シーケンスだけでなく、
環境やタイミングの影響

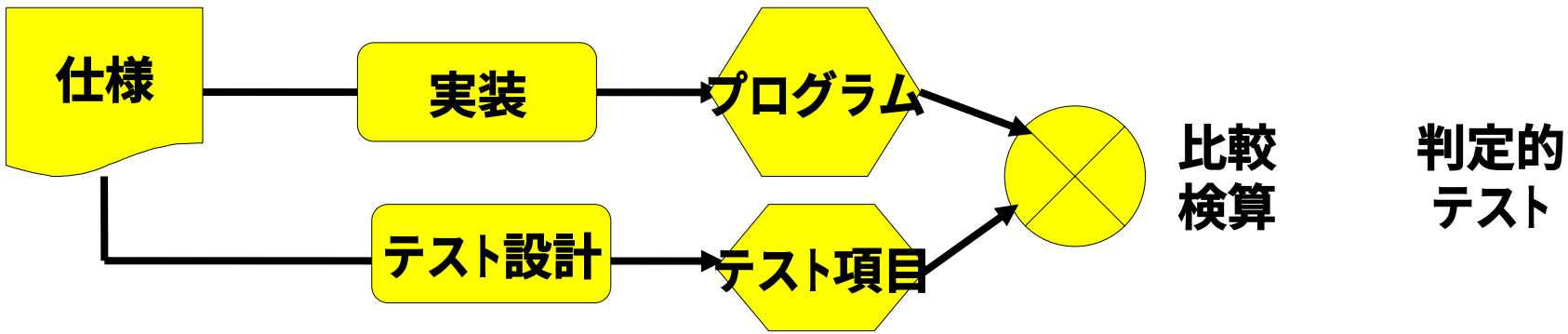
Production Test とDevelopment Test

- SWが組込まれた装置やシステムについて考える
- 製造業におけるライフサイクル



何が違うか

■仕様書通りであるか否かをテストする
　　これは、Production Test の特徴
現在のSWテストは、Production Test である。

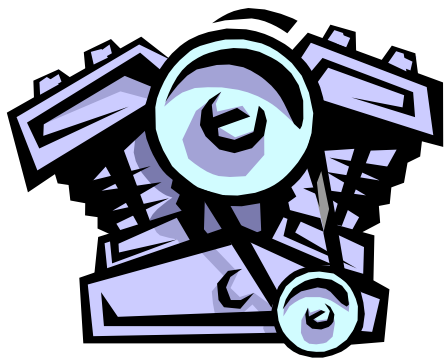


■*Development Test では何を確認するのか？*

- *内部仕様を確認するのではない*
- *例えば、車開発における、試験車のテストドライバーの役割*
- *いろいろなバリエーションで動作させ、変な挙動の兆候を見つける*
- *探索的なテストが必要*

Developmentにおけるステージとテスト

ユニット	サブシステム	システム
静的な特性 寸法、精度、素材	動的な特性 馬力、応答性、持久力	信頼性 (故障しない特性) 安全性 (故障時の特性)



探索的テスト

判定的テスト

「判定的テスト」と「探索的テスト」

■2つの異なったテストが考えられる

■「判定的テスト」

- *仕様書から、合否判定が可能なテスト項目を設計できる*
- *H/Wなら寸法、材質、機能動作、電圧、電流に相当*
- *品質保証や品質管理によるマネジメントが可能*

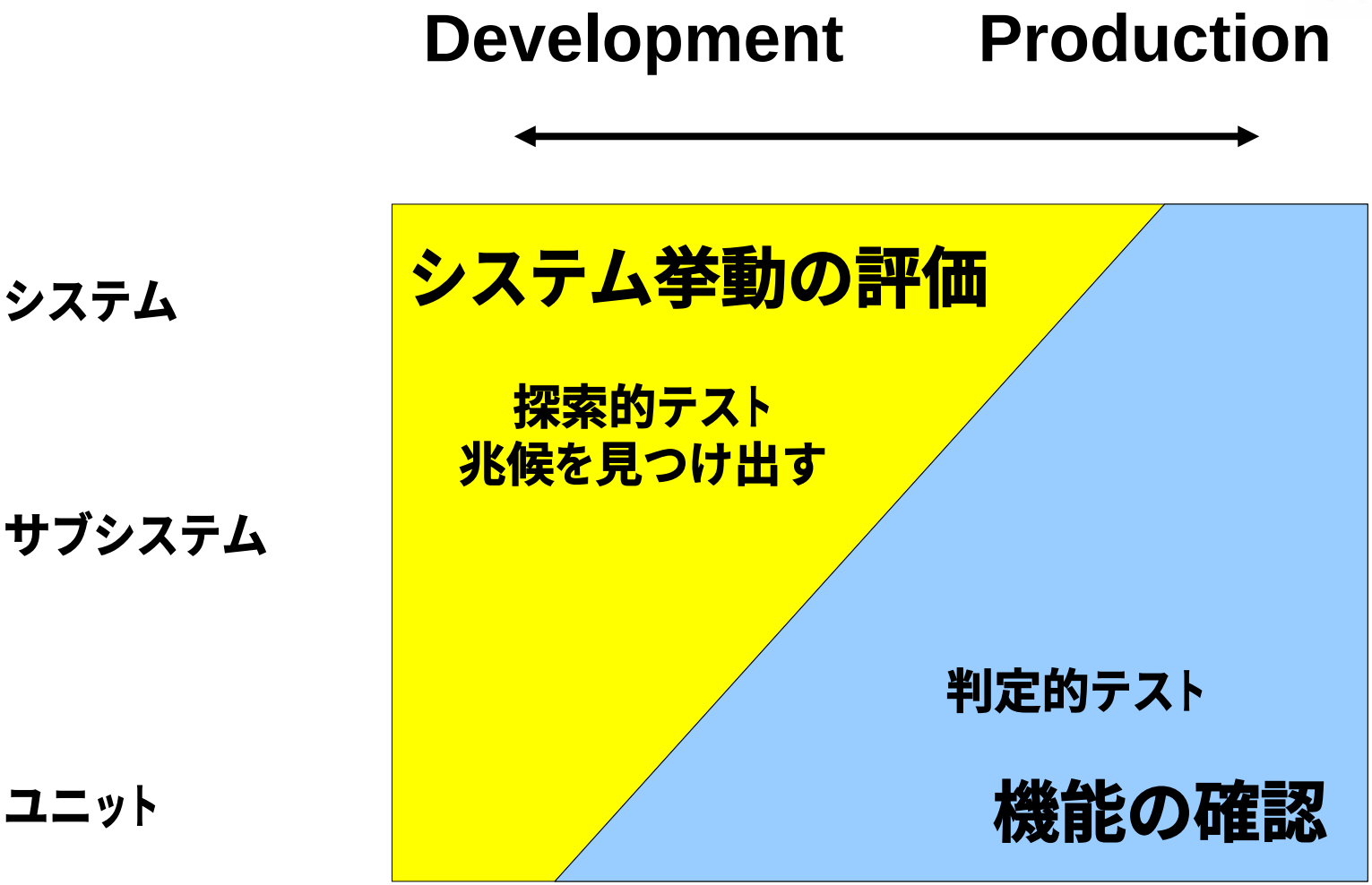
■「探索的テスト」

- *仕様書は要件や緒元であり、評価方法を別途考案する必要がある*
- *性能、応答時間、耐久性など*

  システム挙動を評価する（テストパイロット？）

- *ツールは使うが、評価しながら進めるため自動化は困難*
- *品質保証や品質管理によるマネジメントは無力*

両者の組合わせと「テストの形態」



エンタープライズ系は

■事務処理システム

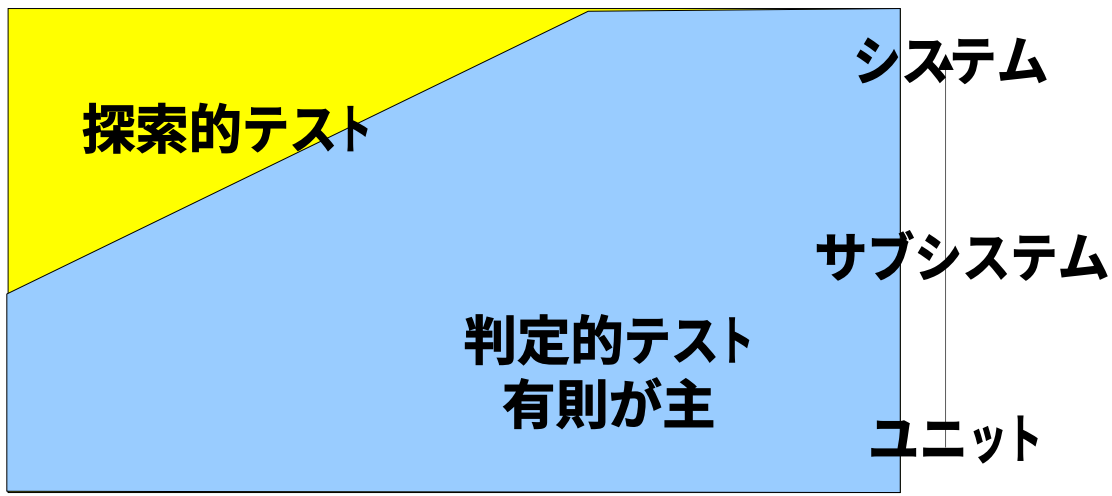
- *既存の事務処理をコンピュータを使って合理化する*
- *使用するOSやミドルソフトも既製品*
- *DBの定義や、NWの設定に不安はある（使う側のスキル）*

■仕様を確定できる

- *よって、テストは「判定的テスト」が多くなる*

■探索的テストは？

- *業務フローの失敗シナリオ*
- *待機系との切り替え*
- *DBの移行*
- *OSバージョンUP*
- *性能問題など*

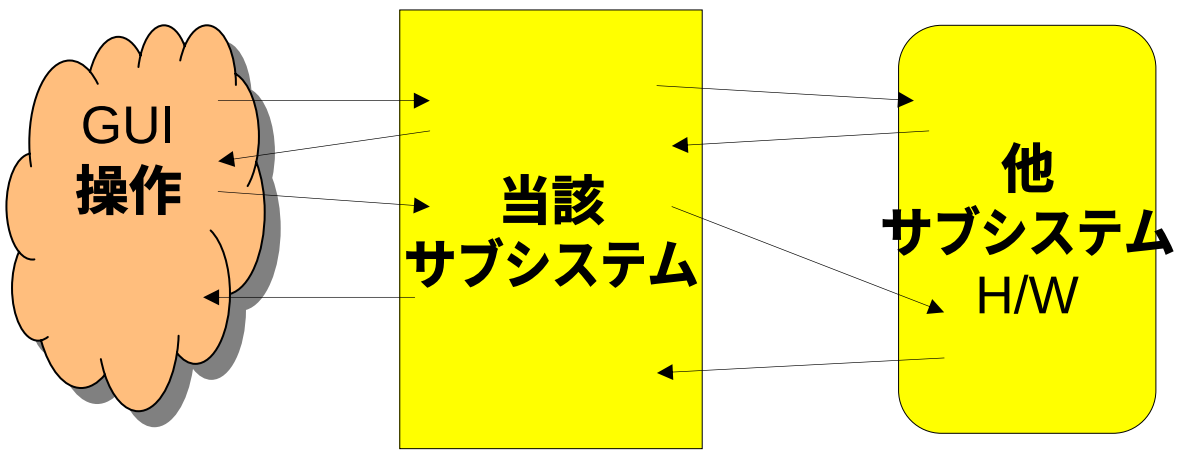


■新ビジネスモデルの場合

- *探索的テストが増加する*

「探索的テスト」の体系化 ①

- まだまだ、未完成の領域
- 多くは、順序論理を含む無則、禁則の中に潜む問題
- ユニットテストでは
  - *アルゴリズムの検証、*
  - *解が求まらない場合、オーバーフローが生じたら、・・・*
- サブシステムテストでは
  - *順序性のある制御（シーケンス）の例外*



シーケンスとは

- 例　通信プロトコル
  - *正常なシーケンス以外に、例外や割込みあり*
  - *特に例外とその回復処理のテスト*
- 例　コンパイラ
  - *構文則に従ったシーケンス*
  - *複合エラー検出とオブジェクトの保証テスト*
- 例　業務フロー
  - *これも一種のシーケンス*
  - *伝票入力誤り、誤発注など例外時のテスト*
- 先ず、シーケンスが定義されていること (有則)
- シーケンスが乱れたり、例外発生時のテスト (禁則)
  シーケンス無則のテストは課題

「探索的テスト」の体系化 ②

■システムテストでは、HWやSWの故障時の挙動

禁則、有則、無則＋変則

■変則（造語）：仕様が守られないときの則

- *故障や異常入力*
- *イベントの衝突など定義されないタイミング*

■安全性：故障時でも人に危害を加えない特性

- *故障に対するテストが厄介なのは、タイミング*
- *シーケンスは、同期したタイミングだが、*
- *故障は非同期のタイミング（何時生起しても大丈夫か）*

■設計も異なる

- *故障時にも危険に至らない機構を設計することが必須*
- *＜故障しない＝信頼性＞とは別*
- *故障しても、発見し、緩和する、縮退する、代替するなどの機構*
- *その機構が動作することを確認する*

Debug Engineering

6．まとめ

■次の世代へ

デバッグ人生35年を振り返る

- デバッグ(SW)がおもしろかったのは、
  挙動として現れるから。
- 20年若くて、今、産業界に入ったら、
  たぶんテストに興味を持たなかった。

何故か？

判定的テストが主流になっているから。

- 判定的なテストは、***統合アプローチにより***、やがて解決される。
  (導入する/しないは企業の判断)
- これから、大事なことは、
  テストパイロットのような探索的テストへ

でも、いきなり墜落するのはイヤ

判定的テストや風洞実験(テスト環境)をしっかりしてから・・・