「ソフトウェア工学」

(4) テスティング

東大理学部情報科学科講義

石川　冬樹（本務先：国立情報学研究所）
f-ishikawa@nii.ac.jp / @fyufyu
http://research.nii.ac.jp/~f-ishikawa/

大学共同利用機関法人 情報・システム研究機構
国立情報学研究所
National Institute of Informatics

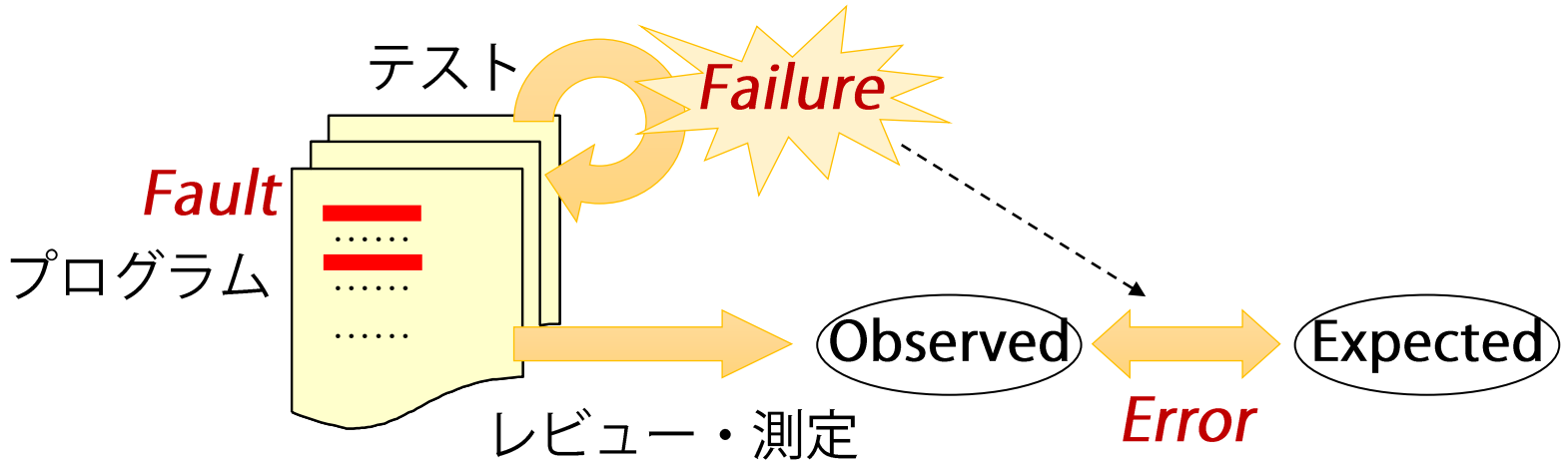
目次

- テスト概要
- 様々な観点からの分類
- ホワイトボックス・テスト
- ブラックボックス・テスト
- 組み合わせテスト

テスト

- テスト・テスティング（Testing）：

  ソフトウェアを調べ，不具合，バグまたは障害（fault）を発見するために故障（Failure）を発生させる
  - 実装されたプログラムのV&Vのために最も確かな方法
  - すべての不具合を確実に発見することはできない



用語

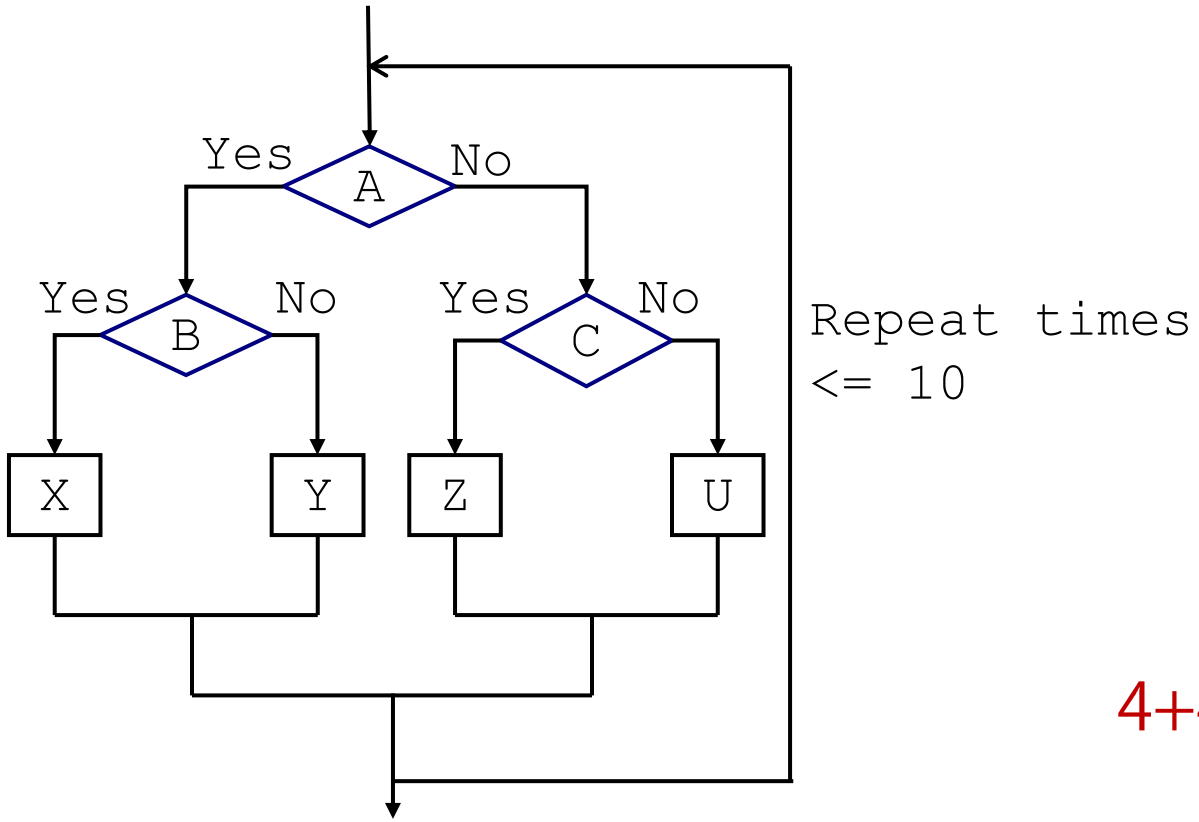
- Error（エラー）：理論的に正しい値や条件に対する，計算，観察，測定された値との差異
- Fault（障害，またはバグ，不具合）：プログラムに含まれる誤ったステップ，プロセスまたはデータ定義
- Failure（故障）：要求される機能が完了しない状態

※ 使い分け・訳語は文脈・分野で異なる

- 一部部品が機能を満たさないこと（Fault）を「故障」と訳すことも
- Fault-Tolerant（耐故障）

問題：「よい」テストとは？

- 例： 起きうるすべての実行パスを検査するためには，
  何回（何種類）の実行が必要？



$4+4^2+4^3+\cdots 4^{10} \geqq 10^6$

問題：「よい」テストとは？

- 完璧でないなら，最小限の労力で，「ベストな成果」を
  - 例： 絶対値を求めることが期待されるプログラムに対して，
    二つのテストケースの集合（テストスイート）のうち，
    どちらが「不具合を見つける能力が」高そうか？

```
if (x>=1) return x  else return -x
```

テストスイート1

1. input x = 3, check the result is 3
2. input x = 5, check the result is 5

テストスイート2

1. input x = 3, check the result is 3
2. input x = -3, check the result is 3

Myersの三角形問題

- 対象プログラム
  - 入力ダイアログから，3つの整数を読む
  - これらは三角形の3辺の長さを表すとし，不等辺三角形，二等辺三角形，正三角形のいずれであるかを示すメッセージを表示する
- テストケースの集合を定義してみよ

Myersの三角形問題：評価指標(1)

1. 意味のある不等辺三角形を入力するテストケースを含めているか？
2. 意味のある正三角形を入力するテストケースを含めているか？
3. 意味のある二等辺三角形を入力するテストケースを含めているか？
4. 上記3において並び順の異なる，最低3つのテストケースを含めているか？

   例：(3, 3, 4) (3, 4, 3) (4, 3, 3)
5. 1つの辺が0となるテストケースを含めているか？
6. 1つの辺が負となるテストケースを含めているか？

Myersの三角形問題：評価指標 (2)

7. 正の3つの値を持ち，うち2つの和がもう1つと等しいようなテストケースを含めているか？
   例：(1, 2, 3)など
8. 上記7において，最も大きい値の場所が異なる，最低3つのテストケースを含めているか？
   例：(1, 2, 3), (1, 3, 2), (3, 1, 2)
9. 正の3つの値を持ち，うち2つの和がもう1つより小さいようなテストケースを含めているか？
10. 上記9において，最も大きい値の場所が異なる，最低3つのテストケースを含めているか？

Myersの三角形問題：評価指標 (3)

11. すべての辺が0であるテストケースを含めているか？
12. 整数でない値を持つテストケースを含めているか？
13. 整数の個数が間違っているようなテストケースを少なくとも1つ含めているか？
14. ここまでのテストケースにおいて，予想される出力を定義したか？

Myersの三角形問題：まとめ

- 最後のもの（期待出力の設定）以外は，
  それぞれ異なる種類の誤りに対応している
- 7.8/14点が専門プログラマの平均点

- どのようにして，こういったテストケースを
  系統的に導けるのか？（手法）

テストに関する他の話題 (1) 終了基準

- 完璧たり得ないため，完了基準を定めることが難しい
  - 「時間または金額を使い切ったら」には，本質的な意味がない
  - 「定めたテストケースで不具合が見つからなかったこと」では，不具合発見能力が低いテストをすればよいことになってしまう
- 統計・経験則に基づく判断
  - プログラム行数などに対し，見つかると期待される不具合の数の平均値などと比較
  - 一定時間内に見つけた不具合数の変化（収束しているか？）
  - …

テストに関する他の話題 (2) 心理的側面

- 心理的な側面を考慮した原則
  - 前もって予想される出力・結果を定義しておく
  - 自分自身で，自分のプログラムをテストしない
  - 誤った入力も考えてテストケースを書く
  - 意図されなかった動きをするかどうかも調べる
  - エラーはないだろうという想定はしない
  - エラーが多く見つかった部分には，エラーがまだ多く残っている
  - プログラムの書き手に対する批判をしない
  - 創造的な挑戦と見なす

目次

- テスト概要
- 様々な観点からの分類
- ホワイトボックス・テスト
- ブラックボックス・テスト
- 組み合わせテスト

分類観点 (1) 静的・動的

- 動的テスト（Dynamic Testing）
  - プログラム実行
- 静的テスト（Static Testing）
  - インスペクション（レビュー），ウォークスルーなどと呼ぶ
  - チェックリストに照らし合わせる，あるいは頭の中で
    テストケースに対するプログラム実行を再現し追ってみる

※「テスト」という語は，動的テストを指す場合が非常に多い

インスペクション（レビュー）

- 系統的に行う
  - 時間を決め，グループを構成して行う
    （議長，該当部分の担当プログラマ，若手，ベテランど）
  - 照らし合わせるルールを明確にするとよい
    （初期値未定義，配列インデックス範囲，異なる数値表現の混合演算，論理式解釈，ループ停止，パラメーターの単位，…）
  - 不具合の指摘は，プログラマに対する批判とならないようにする
    （プログラミング言語・環境が未成熟であり誰でも誤りを犯す）

分類観点 (2) ホワイトボックス・ブラックボックス

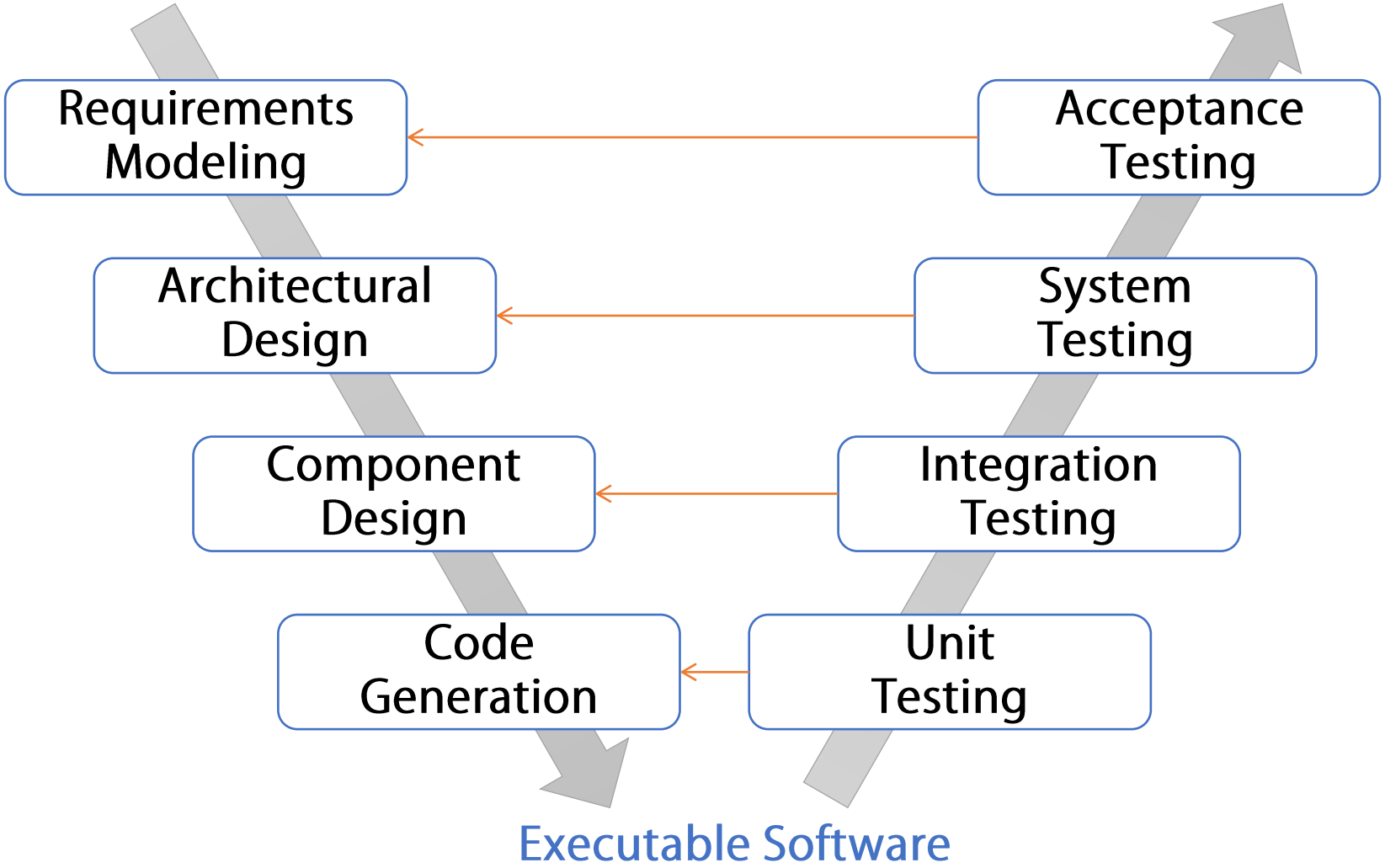
- ホワイトボックステスト
  - プログラムの内部構造を踏まえてテストを設計する
    （例： if-else文の分岐両方を通るようにする）

- ブラックボックステスト
  - プログラムの内部構造は意識せず，仕様に基づいてテストを設計する
    （例： ユースケースに基づいてテストを定める）

分類観点 (3) 工程

- Unit Testing（ユニットテスト・単体テスト）
  - メソッドなど小さい部品が対象
- Integration Testing（結合テスト・統合テスト）
  - 個々はテスト済みである部品の組み合わせが対象
- System Testing（システムテスト）
  - ネットワーク，ハードウェア，データベースなど他のシステム要素との組み合わせが対象
- Acceptance Testing（受け入れテスト）
  - 実際の利用状況としてユーザの満足，負荷，長期実行などが対象

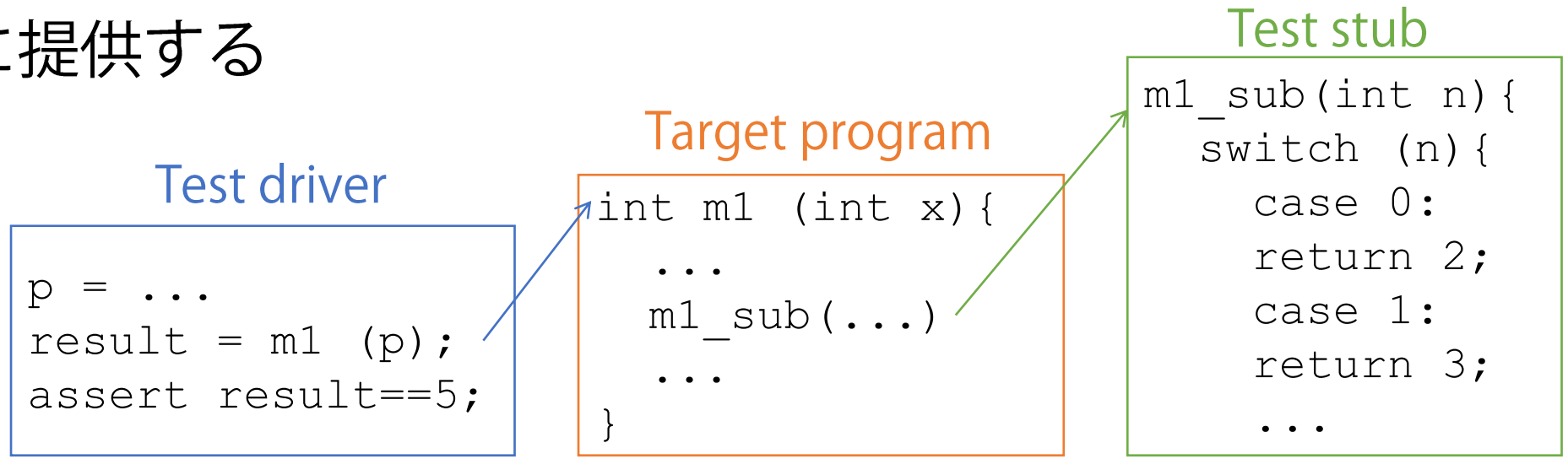
（再掲）V字プロセスモデル



[C. Bucanac, 1999]

基本概念：ドライバ・スタブ

- 不具合の追求を容易とするため，各テストケースでは対象部分（SUT: System Under Test）に集中したい
  - テストドライバ：対象プログラムを特定の入力で呼び出し，結果を観測する
  - テストスタブ：実装やテストが終わっていない部品の機能を擬似的に提供する



ユニットテスト

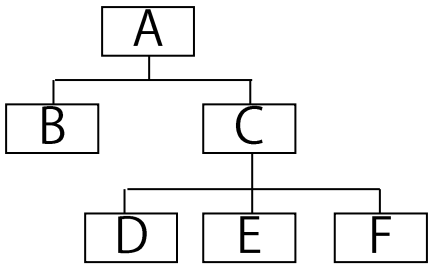
- ドライバ・スタブを用いて特定の小さな部品に集中する
- 専用のフレームワークを活用することが多い
  - Junit，CppUnit，PHPUnit，unittest (Python), …
  - アサーション（表明）を用いて個々のテストケースを分離し定義
  - それらを登録し，まとめて実行，記録する

```
public class TestCase1
    extends TestCase{
  public void testFun1(){
    ...
    assertEqual( x, 3 );
  }
}
```

```
  suite.addTest(new TestCase1());
  suite.addTest(new TestCase2());
  ...
  suite.run(result);
  ...
```

結合テスト

- 部品の組み合わせは段階的に行う
  - そうしないと不具合同定が難しい（ビッグバンテスト）
  - 増加テスト（Incremental Testing）
  - ボトムアップは上位目的レベルのテストが後回しになるので大きな変更が発生しやすいが，実装と並行しやすい



Top-down
1. A and B (with C stub)
2. A, B, and C (with D, E, F stubs)
3. A. B, C, and D (with E, F stubs)
4. …

Bottom-up
1. C and D (with E, F stubs)
2. C, D, and E (with F stub)
3. C, D, E, and F
4. …

システムテスト・受け入れテスト

- システムとしての仕様・設計に基づき様々なテストを行う
  - 高負荷（stress testing）
  - 性能（performance testing）
  - 大容量（volume testing）
  - 使いやすさ（usability testing）
  - セキュリティ，記憶域，構成，互換性，設置容易性，
    信頼性，回復，保守性，人手での手続き，ドキュメント，…
- 受け入れテストはユーザー側の承認・確認作業
  - 実際のユーザ，実際のデータ，…

回帰テスト

- 回帰テスト（Regression Testing）
  - 以前のバージョンより悪くなっていないかを調べる
  - 背景：プログラムのある箇所の修正により，
    別の機能に不具合が出ることは多々ある
  - Regression（回帰，退行）やdegrade（退化する）といった言葉で
    説明される（日本では「デグレ」と言われる）

目次

- テスト概要
- 様々な観点からの分類
- ホワイトボックス・テスト
- ブラックボックス・テスト
- 組み合わせテスト

カバレッジ

- カバレッジ（被覆率，Coverage）：

どれだけ多くの「構成要素」が検査されたか？

- 例： if (P and Q) then … else …
  - 分岐の選択肢（then, else）

    → (P, Q) = (true, true), (false, true) により 2/2 網羅

  - P, Q 各々がとりうる値（true, false）

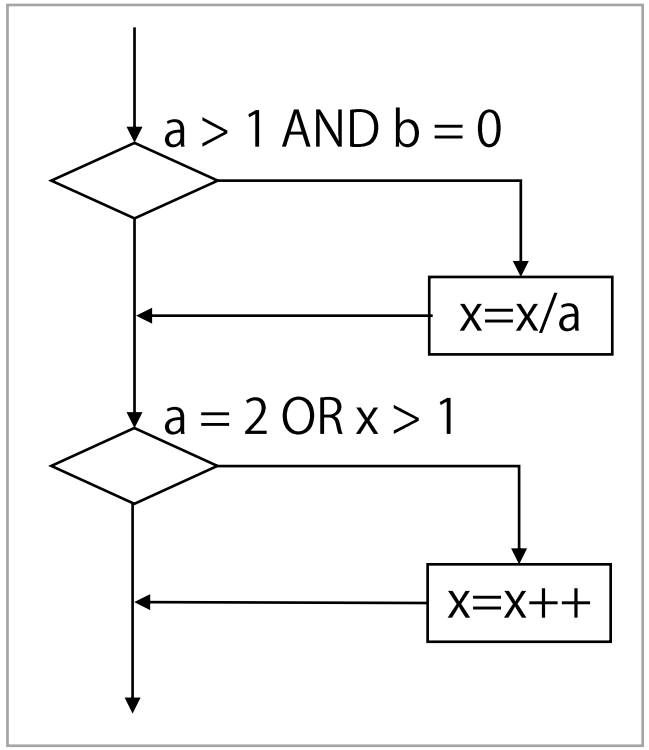
    → (P, Q) = (true, true), (false, false) により 4/4 網羅

命令網羅

■命令網羅（Statement Coverage）・C0

- ■各命令が少なくとも1回実行されるか？
- ■例： 右のプログラムに対して

  (a, b, x) = (2, 0, 3)

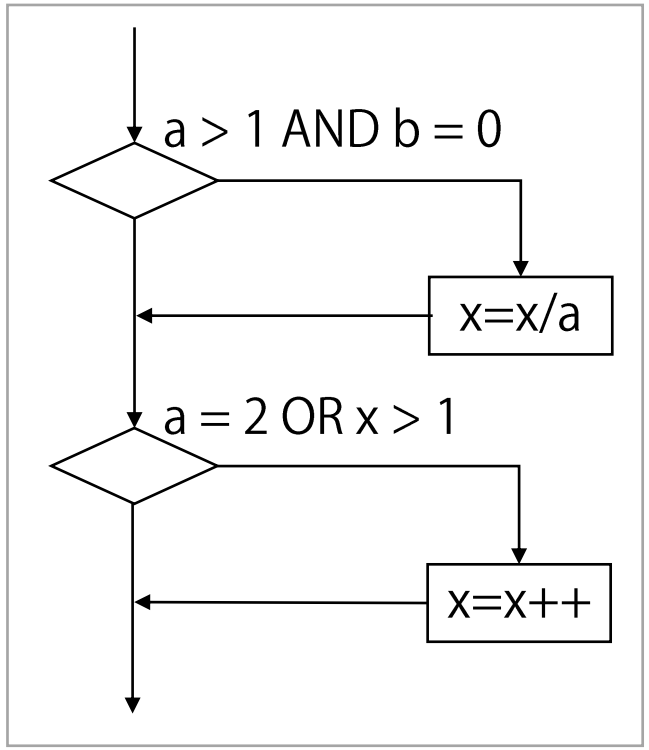


分岐網羅

■分岐網羅・判定条件網羅（Branch Coverage / Decision Coverage）・C1

- ■各分岐判定の取り得る結果が少なくとも1回実行されるか？（if文, switch文など）
- ■例： 右のプログラムに対して

  (a, b, x) = (3, 0, 3), (2, 1, 1)



命令網羅と分岐網羅

- 通常，分岐網羅は命令網羅より強いレベルととらえ，命令網羅を含むとすることが多い
- ただし，分岐網羅を満たすテストケースの集合が，命令網羅も満たさない場合もある
  - 到達しない命令がある場合（ある種の不具合）
  - 対象プログラム実行の入り口が複数ある場合
    （それらも網羅する必要がある）

条件網羅

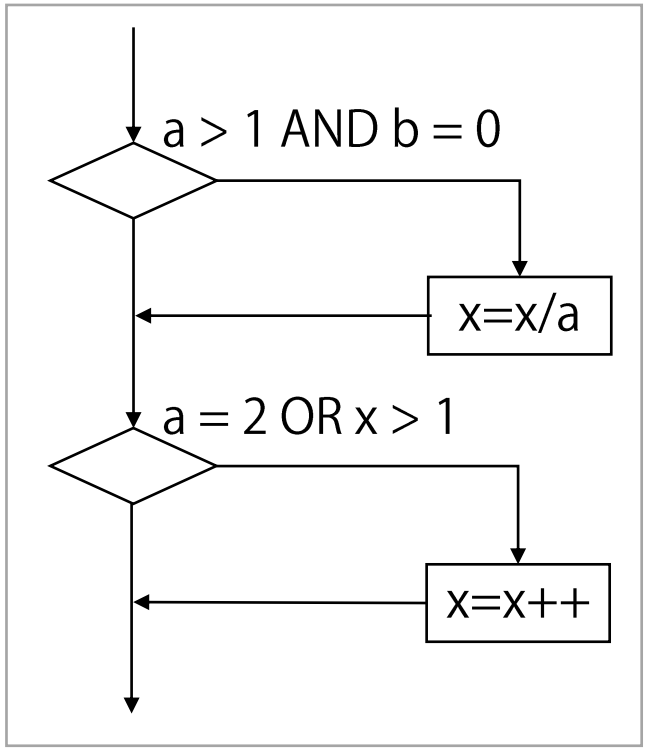
■条件網羅（Condition Coverage）・C2

- ■各分岐判定の個別条件が取り得る結果が少なくとも1回実行されるか？
- ■例： 右のプログラムに対して

  (a, b, x) = (1, 0, 3), (2, 1, 1)

  a = 1, b = 0, x = 3

  a = 2, b = 1, x = 1

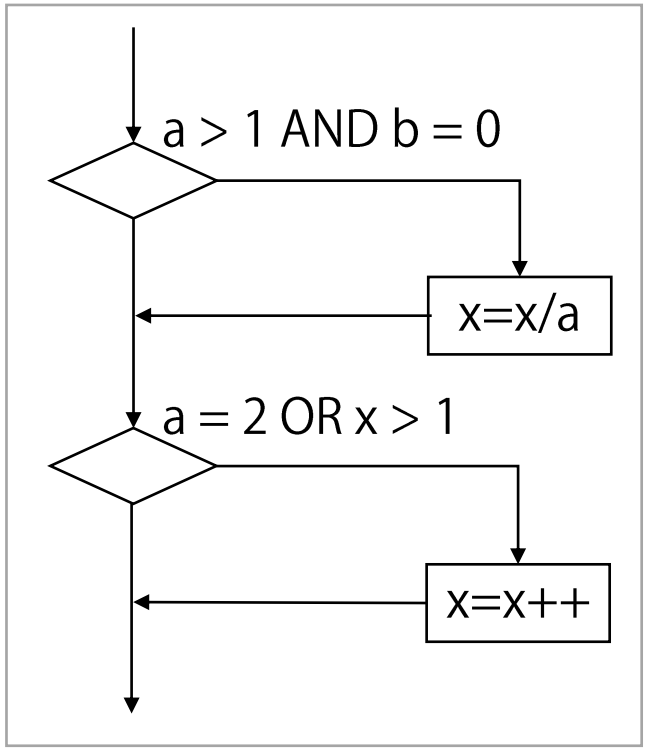


条件網羅と分岐網羅

- 条件網羅を満たしても分岐網羅を満たさない場合はある
  - 例： if P AND Q に対して

    P = true, Q = false

    P = false, Q = true



複数条件網羅

■複数条件網羅（Multiple-condition Coverage）

- ■各分岐判定の個別条件が取り得る結果の組み合わせすべてが少なくとも1回実行されるか？
- ■例： 右のプログラムに対して

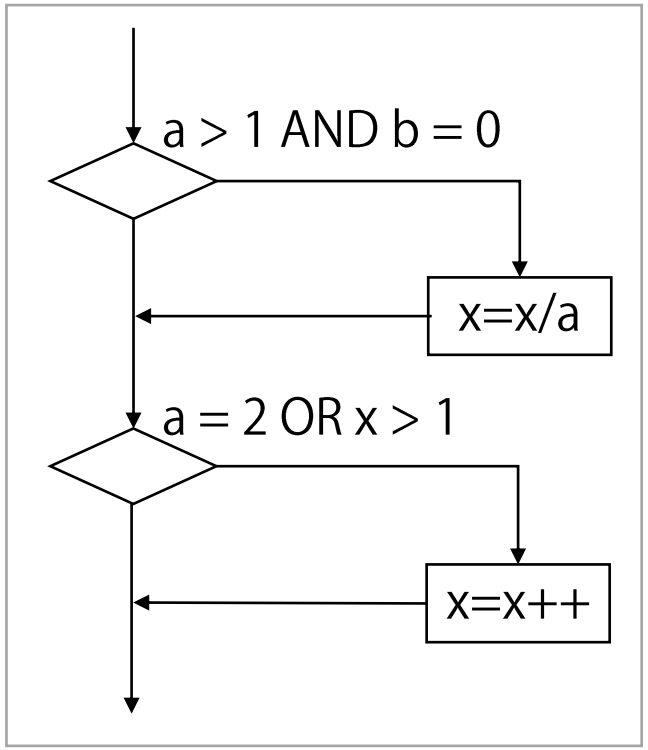
(a, b, x) =

(2, 0, 4) → (T-T, T-T)

(2, 1, 1) → (T-F, T-F)

(1, 0, 2) → (F-T, F-T)

(1, 1, 1) → (F-F, F-F)



MC/DC

- MC/DC（Modified Condition/Decision Coverage）
  - 各分岐において取り得る結果を少なくとも1回実行
  - 各分岐の個別条件が取り得る結果を少なくとも1回実行
  - 各分岐判定の個別条件が，単独で分岐判定の結果を左右
  - ➡ 実質使われない条件は「カバーした」と見なさない

例： if (P and Q) then … else …
→ (P, Q) = (true, true), (false, false) により 4/4 条件網羅？
→ 後者では，P=falseの時点で分岐が決まりQの評価はスキップ
→ Q=falseのケースをカバーしたと見なさない
→ (P, Q) = (true, true), (true, false), (false, true) により網羅

カバレッジに関する実態

- 網羅は「可能であれば」である
  - ともに真にはなり得ない条件の組み合わせなどもある
  - 全網羅（100%）を達成するテストケース集合の設計は難しい
  - ➪「90%達成すればよし」などと閾値を決めることも多い
- 判定条件網羅（C1）を最低限の必須とすることが多い
- 高信頼性が求められる航空業界ではMC/DCが必須

目次

- テスト概要
- 様々な観点からの分類
- ホワイトボックス・テスト
- ブラックボックス・テスト
- 組み合わせテスト

同値分割

- 同値分割（Equivalence Partitioning）：

特定の振る舞いに繋がる入力値を同値クラスとしてまとめる

- 無効な場合も含め，各同値クラスに対応するテストケースを（少なくとも）1つずつ含める
- 例：定形郵便物の配達料金計算

重さ	料金
<= 25g	80円
<= 50g	90円

必要なテストケース

-5g, 10g, 35g, 80g

同値分割

- 同値クラスの見分け方
  - 入力において，値の範囲がある・値の個数の範囲がある
    - 有効クラス，小さすぎる場合の無効クラス，大きすぎる場合の無効クラス
  - 入力において，値の種類が列挙されている
    - それぞれの種類に対応する有効クラス，どれにも属さない場合の無効クラス
  - 入力が満たすべき条件がある
    - 有効クラス，満たさない場合の無効クラス

同値分割

- 例：コンパイラによる配列宣言の解釈処理
  - 配列の個数　　1，1以上，なし
  - 配列名の長さ　　制限内，0，制限超
  - 配列名　　文字入り，数字入り，他入り
  - 配列名の頭文字　　文字，それ以外
  - 配列の次元数　　制限内，0，制限超
  - 各次元の要素数　　制限内，負，制限超
    指定あり，指定なし
    定数指定，整数変数，その他
  - ・・・

境界値分析

■境界値分析（Boundary Value Analysis）

- ■入力および出力の同値クラスの端となる値を用いる
- ■例：定形郵便物の配達料金計算

重さ	料金
<= 25g	80円
<= 50g	90円

必要なテストケース

0g, 1g, 25g, 26g, 50g, 51g

境界値分析

- 例：試験結果のソートと出力
  - 質問数の値（0，1，上限，上限超）やその有無
  - 学生数（0，1，上限，上限超）
  - 合計点によるソートの結果（全員成績が同じ，一部は同じ，全員違う）
  - 偏差値によるソートの結果
    （1人0点で他が100点で標準偏差最大，全員同点で標準偏差0）
  - 結果の改ページがギリギリ起きない・起きる
  - 結果のページ数が0，1，上限，上限超
  - ・・・

目次

- テスト概要
- 様々な観点からの分類
- ホワイトボックス・テスト
- ブラックボックス・テスト
- 組み合わせテスト

組み合わせテスト

- 複数要因に対する特定の値の組み合わせにおいて，
  不具合が顕在化する可能性がある
  - 動的Webサイト： OSの種類・バージョン，
    ブラウザの種類・バージョン，ブラウザプラグインのバージョン
  - 出張手続き： 申請者の役職，行き先区分，利用する予算
  - 履修手続き： 講義の対象（学部，大学院），学生の区分（同学科，同大学同学部他学科，同大学他学部，他大学），講義の区分（通常，集中講義）
  - …

組み合わせテスト

一般化して・・・

- 影響を与える項目（因子・factor）がn個，各項目が取り得る値がa個（水準・level）あると，組み合わせは a^n 個
  - 因子4個・それぞれの水準が3個でも81個の組合せ
  - 因子10個・それぞれの水準3個だと59049個の組合せ
- どうすれば少ないテストケースの個数で，効果的なテストが実現できるか？

ペア構成

- 1つのアイディア：
  2つの因子の組み合わせは，すべて調べるようにする
  - それで必ず十分であるという理屈があるわけではない
  - 過去の事例の調査などでは，リリース後に発覚した欠陥の大半が，この方針でテストを行なっていれば検出できたはずとされる

※「シングルモード」と「ダブルモード」の不具合が大半であるということ：機能やモジュール1個に問題があるか，それら2個の組み合わせに問題があるか

ペア構成

- 前述の方針をどう実現するか？
  - 例：因子A，B，Cがあり，それぞれ2値0, 1をとりうる
  - AとB，BとC，CとAというどの2つの項目においても，
    以下の4ペアをすべて含むようにしたい
    (0, 0), (0, 1), (1, 0), (1, 1)
  - 例えば以下の4テストケースで実現可能

	A	B	C
組1	0	0	0
組2	0	1	1
組3	1	0	1
組4	1	1	0

ペア構成を満たすテストスイートの作成

- 方法1：「前スライドのような表」を作って集めておき，あてはめて使えばよい
  例：http://neilsloane.com/oadir/
- 直交表（Orthogonal Arrays）
  - どの2つの列（あるいは一般化し n個の列）においても，それらの列には値のすべての組み合わせが同じ回数含まれている
- 方法2：アルゴリズムを用意し都度作成する
  - 全探索は非現実的なので，テストケース数を小さくするヒューリスティック，メタヒューリスティックになる

直交表

- 直交表のパラメータ
  - 列の数： 因子の数 c
  - 因子がとりうる値の数 n
  - 行の数： テストケースの数 x
    （これはパラメータというより結果）

標準表記
$L_x(n^c)$

0	0	0
0	1	1
1	0	1
1	1	0

$L_4(2^3)$

0	0	0	0
0	1	1	1
0	2	2	2
1	0	1	2
1	1	2	0
1	2	0	1
2	0	2	1
2	1	0	2
2	2	1	0

$L_9(3^4)$

直交表

0	0	0	0	0	0	0
1	1	1	1	1	1	0
2	2	2	2	2	2	0
0	0	1	2	1	2	0
1	1	2	0	2	0	0
2	2	0	1	0	1	0
0	1	0	2	2	1	1
1	2	1	0	0	2	1
2	0	2	1	1	0	1
0	2	2	0	1	1	1
1	0	0	1	2	2	1
2	1	1	2	0	0	1
0	1	2	1	0	2	2
1	2	0	2	1	0	2
2	0	1	0	2	1	2
0	2	1	1	2	0	2
1	0	2	2	0	1	2
2	1	0	0	1	2	2

大きな例
$L_{18}(3^7)$

0	0	0	0	0
1	1	1	1	0
0	0	1	1	1
1	1	0	0	1
0	1	0	1	2
1	0	1	0	2
0	1	1	0	3
1	0	0	1	3

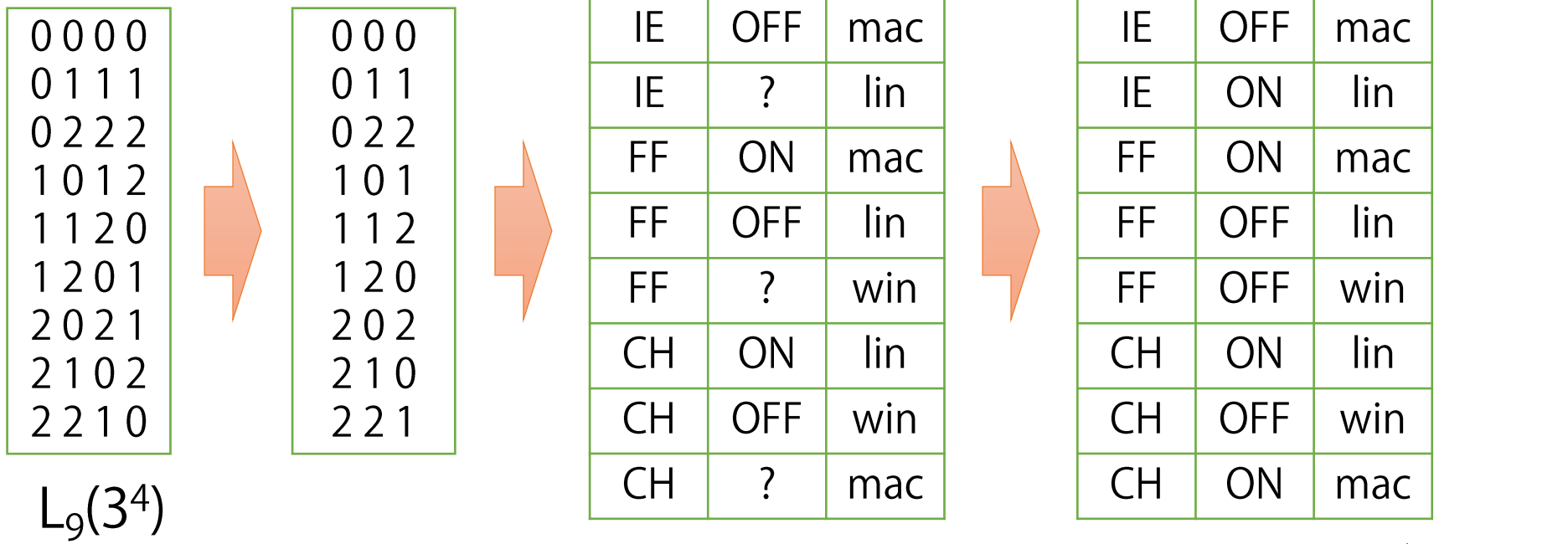
列によって水準数が異なる例
$L_8(2^4 4^1)$

0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(0, 0, 0) や (0, 0, 1) など3値の組を網羅する場合

直交表

- 個別のケースに当てはめる例
  - 例： A: {IE,FF,CH}, B: {ON, OFF}, C: {win,mac,lin}



$L_9(3^4)$

1列不要

実際の値に変更

水準値の個数が余分なところは適当に埋める

直交表

- 直交表の操作において，行はむやみには消せない
  - 行を消すと「ペア」がカバーできなくなる可能性が高い
  - このため，実際には起こらない組み合わせ（禁則と呼ばれる）を含めないようにするなど，例外を扱うのが難しい
- 全ペアが同じ回数現れる性質を持つ
  - 「全ペアを使う」だけならもっとテストケース数を少なくできる
  - 対称的な「ばらけ」により，3列以上の組み合わせも多くカバーできている

All-Pair法

- All-Pair法
  - アルゴリズムにより全ペアが現れる組み合わせを見つける
  - 例： A: {0, 1}, B: {0, 1}, C: {0, 1, 2}

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0
0	0	2
1	1	2

直交表と比べると
0 1 2
1 0 2
がない

All-Pair法

- All-Pair法：全ペアが現れる組み合わせを見つける
  - 対称性を求める直交表よりもテストケース数が少なくなる
    - 例：100因子，2水準だと直交表では101ケース，All-Pairでは10ケース
  - 起こらない組み合わせ（禁則）など例外を指定できるようにアルゴリズムを設計することが多い
  - 生成アルゴリズムはいろいろ
    - 列を1つずつ増やしながら探索
    - 遺伝アルゴリズムなどのメタヒューリスティックスによる探索
    - 既知の表を変更
    - …

論理関係を扱う手法

- 組み合わせテストは，要素間の依存関係を仮定しない
  - 「無則」と呼ぶこともある
  - 「この組み合わせでおかしなことは起きないはず」とせずに検査
- 論理関係が明確にある場合には，それを整理しテスト設計
  - 例：デシジョンテーブル
  - そのほか，
    原因・結果グラフなど

	ルール1	ルール2	ルール3	ルール4
条件				
既婚？	Y	Y	N	N
学生？	Y	N	Y	N
アクション				
割引率	60	26	50	0

今回の参考文献

- ソフトウェア・テストの技法 第2版
  - G. J. Myers，長尾・松尾訳，近代科学社，2006
- ソフトウェアのテスト技法
  - 宗訳，日経BP社，2005

まとめ

- テスト
  - 正しさの定義（期待される結果）に対する検証，
    本来の要求に対する妥当性確認を行う
  - 完璧にあらゆる不具合を見つけることはできず，
    費用対効果を追求する
  - 工程・目的に応じて様々なアプローチがある
  - 潜在的な不具合を効率よく顕在化させるような，
    テストケースの設計を目指す