2023年度
ソフトウェア工学
ソフトウェアテスト

2023年12月18日

渥美 紀寿 (情報環境機構)

京都大学

参考図書

- 鯵坂 恒夫著: ソフトウェア工学入門, サイエンス社

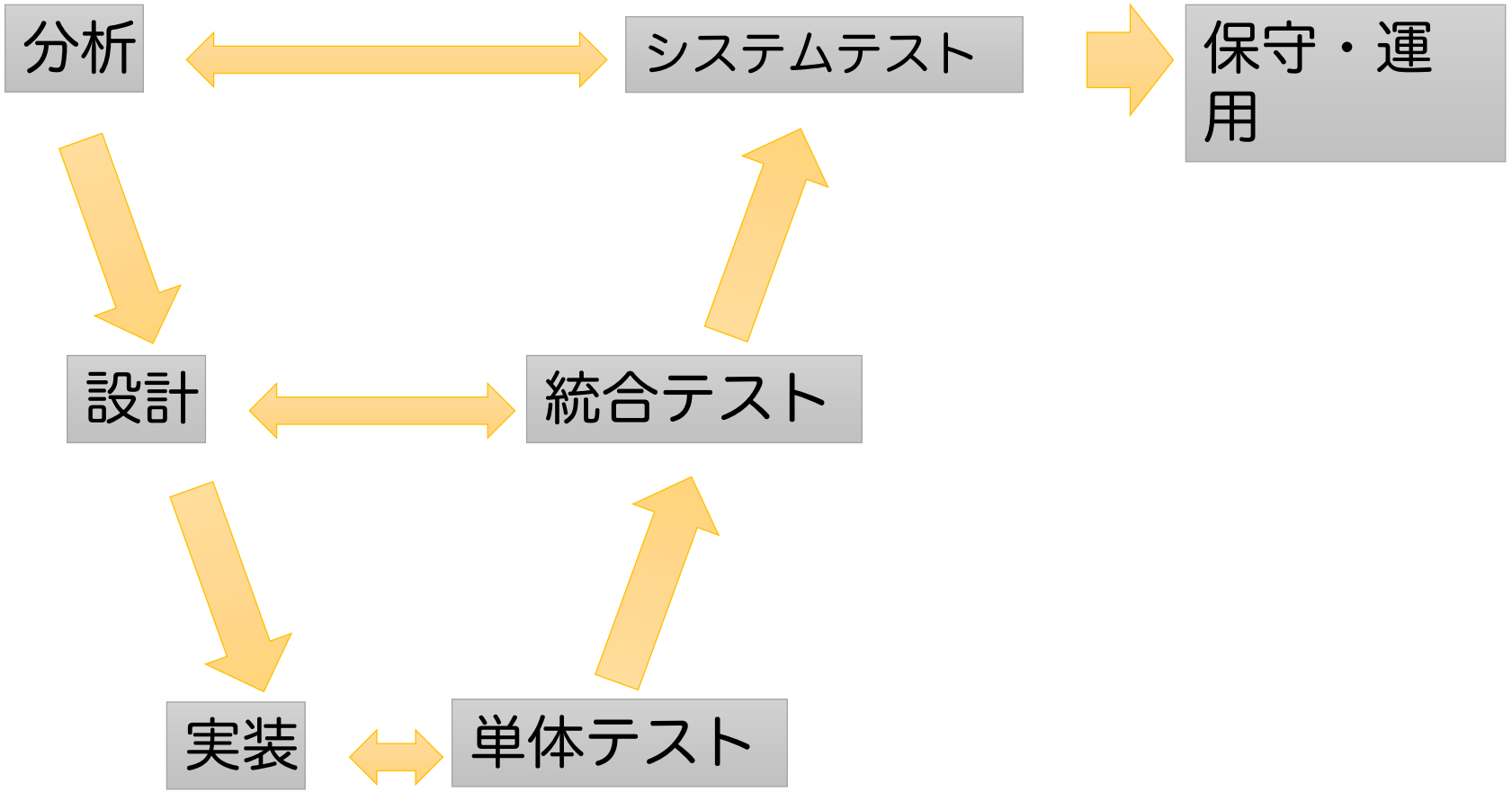
ソフトウェア検査の目的

- ソフトウェアの「正しさ」を確認すること
- 「正しさ」を妨げる要因があれば，その箇所と原因を指摘

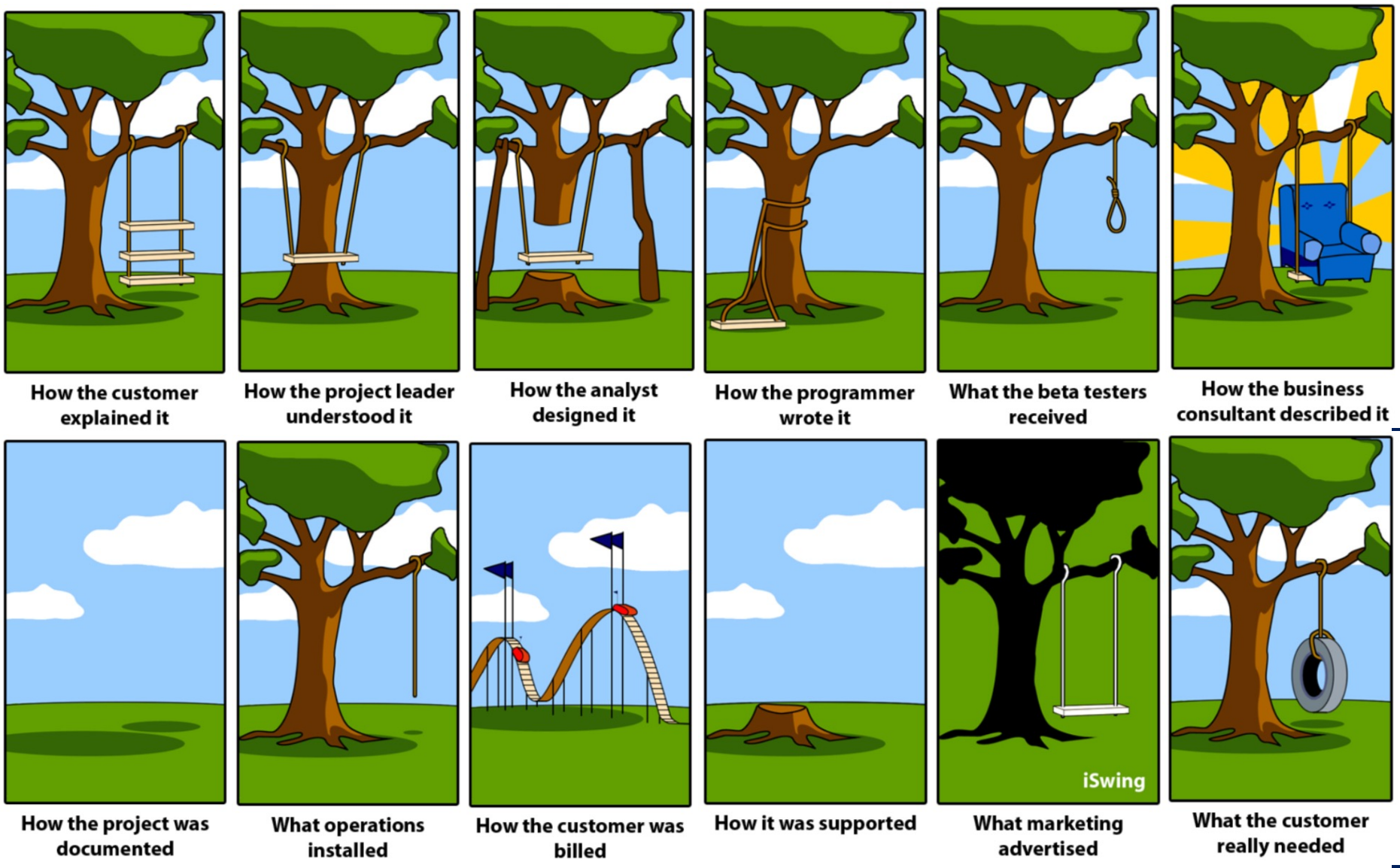
ソフトウェアの「正しさ」とは

- ソフトウェアが求められる機能を実行できる
  - 要求される目的・使命を果たすことができる
- ソフトウェアが求められる品質を満たす
  - 実行の性能や精度
  - ソフトウェアの信頼性

ソフトウェア開発プロセス (V字型モデル)



顧客が本当に欲しかったもの



出展 http://www.projectcartoon.com/

ソフトウェア検査の理想

- 理想
  - ソフトウェアが仕様に従って正しく(欠陥ゼロ)製造されていることを保証
  - 顧客の要求を完全に満たしていることを保証
- 現実的には無理
  - ソフトウェアの誤りを完全に除去するのは困難
  - 顧客の要求を完全に仕様化するのは困難
  - コスト・規模の問題
  - 顧客の要求は常に変化

ソフトウェア検査の現実

- 欠陥ゼロを保証するものではない
  - 特定の目的に利用するのに適切な品質を保証
  - 特定の目的に利用できるか否かを判断する材料を提供
- 保証する性質はシステムの目的や顧客の期待度，市場環境により異なる
  - システムの目的・用途がクリティカルなら，より高品質が必要
  - 顧客の期待が低ければ低品質でも構わない
  - 市場の要求に追随することが第一義… など

検証と妥当性確認

- 検証 (verification)
  - ソフトウェアを正しく作っているか
  - 定められた仕様を満たしているか

- 妥当性確認 (validation)
  - 正しいソフトウェアを作っているか
  - 顧客の要求に適っているか

動作不良

- 故障 (failure)
  - 要求された機能を果たせないこと
- 欠陥 (defect)
  - failure を引き起こす原因
  - 障害 (教科書では「誤り」) (fault) と呼ぶことも
- 誤り (error)
  - 間違った結果を生み出す人間の行為

静的検査と動的検査

- 静的検査
  - 実行することなく，ソフトウェアを確認
  - 静的解析技術やレビュー，形式手法による証明
- 動的検査
  - プログラムを実行してソフトウェアを確認
  - テスト，動的解析

静的検査

- レビュー
  - 仕様 (要求・設計)やプログラムを読み直すことで検査
  - 顧客の要求(特に非機能要求)の充足を確認することは困難
- 静的解析
  - 型検査
  - 制御フロー解析，データフロー解析
  - コードクローン解析
- 形式手法
  - 仕様を形式言語で記述し，検査したい性質を形式的に表明
  - 自動で網羅的な検証が可能

レビューの目的

- レビューに参加するメンバによって，レビュー対象の欠陥を検出
- 経験則に基づく確認
- 相互理解や情報共有，合意形成
- 顧客への確認，上位管理者に対する承認

レビューの前提

- 対象となる文書の「正しさ」が与えられている
  - 正しさを検証うるためのチェックリストなど
- 対象となる文書について，構造や文法の正しいものが与えられている
  - 対象の中身の正しさを検証できる状態であること
- 開発の管理側(あるいは顧客側)が，レビューによる初期的なコスト増に合意している
- レビュー結果を対象文書の作成者に対する人事的な評価に使わない

レビューの種類

- 管理者レビュー
  - ソフトウェア製品やプロセスの体系だった評価
- 技術レビュー
  - 意図した利用に適しているか
  - 仕様・標準から逸脱していないか
- インスペクション
  - 仕様やプログラムをレビューするための構造化された手法
- ウォークスルー
  - 簡単なテストケースを与え，机上で模擬実行
- 監査
  - 仕様，標準，契約上の合意，その他の基準への準拠を査定

レビューの利点と欠点

- レビューの利点
  - 動作するプログラムがなくても作業できる
  - 開発プロセスを対象にできる
  - 一度に複数の欠陥を見つけることができる
  - ドメイン知識を効果的に活用できる
- レビューの欠点
  - 人手による作業のため大規模な適用が困難
  - 非機能要求の充足は検証できない

静的解析技術

- プログラムの特性を動作させずに解析
  - 解析の目的により視点は様々
  - 保守のためのプログラム理解も目的の一つ
- 型検査
  - コンパイラによる構文解析の延長
- 制御フロー・データフロー解析
  - プログラムの構造や意味を抽出
- コードクローン解析

制御フロー・データフロー解析

- プログラム要素間の依存関係の解析
- 制御フローグラフ
  - 制御の依存関係を図示
- データフローグラフ
  - データの依存関係を図示
- 依存関係の把握
  - 変更が影響する範囲の特定
  - 誤り箇所や原因の特定

プログラムスライシング

- 特定の文や変数に関係するプログラムの部分を抽出
- プログラム依存グラフを利用
  - 制御フローグラフ，データフローグラフを合成
- デバッグ支援を目的として考案
- テスト，プログラム理解，部品抽出など様々な用途で利用

コードクローン

- ソースコード中に存在する，全く同一あるいは類似したコード断片のこと
- コードクローンができる原因
  - 単なるコピーペースト
  - 常套句的なプログラム記述 (イディオム)
    - ファイルのオープン・クローズ
    - データベース接続・切断
  - 何らかの意図に基づく
    - 性能向上のためのループ展開

コードクローンの弊害

- ソフトウェアの保守を困難にする
  - プログラムを変更する際，それぞれのコードクローンについて変更の要不要を判断する必要がある
  - 作成後，時間の経過とともに見逃されがち
  - 一貫しない変更は誤りの原因
- 正しいソフトウェアを作成するという立場からも検出・除去することが望ましい

動的検査

- テスト
  - 特定の入力を与え，プログラムを実行
  - 期待する出力と比較
- 動的解析
  - プログラム実行時のトレースを収集し，トレースを解析
  - テストやデバッグに有用な情報を抽出

テストの限界と戦術

テスト方法
事前条件の設定 ⇒ 入力設定 ⇒
実行結果と期待される結果の比較

- 1つの実行 (テスケース) は 一例
- 有限のテストケース (テストスウィート) しか実行できない
- すべてをテストで確認することはできない

効果的なテストスウィート作成が重要

テストの心構え

- バグを全部見つけるのは無理
- 誤りは見つからないだろうという仮定のもとにテストを計画してはならない
- プログラム開発グループは自分たちのプログラムをテストしてはならない
- プログラムのある部分で誤りがまだ存在している確率は，すでにその部分で見つかったエラーの数に比例する
- どの部分に誤りが出やすいか，そこにどのようなテスト手法を適用すれば品質・ディペンダビリティの向上が効率良くできるかを知ることが重要

テストの役割

- 欠陥の発見
  - 故障に発展する可能性のある欠陥が存在することを見つける
- 欠陥箇所の特定
  - 発見された欠陥がプログラムのどの部分にあるかを見つける
- 結果としてディペンダビリティを向上
  - 欠陥が存在しないことを示すことはできない
    - どのようなテストケースを選択すべきか?
    - どこまでテストしたら完全か?

テストの計画と設計

実装後にしかテストの計画を立てられない?

- 要求をどのようにテストするか，分析時から検討すべき
- 分析から設計，設計から実装の各段階での漏れや矛盾点，あいまいな条件などをテスト方法を検討する中で検出可能
- 要求の種類によってテスト方法を選択
  - 機能要求
  - 非機能要求

テストの種類

- テスト対象の粒度
  - 単体テスト，結合テスト，システムテスト
- テスト環境
  - 開発環境テスト，実装テスト (実機テスト)
- テスト実施者
  - アルファテスト，ベータテスト
- 非機能テスト
  - パフォーマンステスト，ストレステスト，ユーザビリティテスト
- 回帰テスト

単体テスト

- モジュールごとのテスト
- モジュールを呼出すテストドライバを作成
- 入力を与え，実行結果と期待する出力と比較
- JUnit, CUnit, CppUnit など単体テストを支援するフレームワークが言語ごとに存在

結合テスト

- モジュール間の相互作用をテスト
  - モジュール間のインターフェイスやデータの受け渡しをチェック
  - 必要な下位モジュールがない場合，模倣するスタブを作成
  - 必要な上位モジュールがない場合，模倣するテストドライバを作成
- モジュール構成によりテスト方法が異なる
  - 同一ノード同一プロセス ⇒ モジュール呼出し
  - 同一ノード異なるプロセス ⇒ OS と連携
  - 異なるノード間 ⇒ ネットワーク管理機能と連携

システムテスト

- システム全体の振舞いのテスト
  - 発注者や利用者による受け入れ・検収もこのレベルで受け入れテストとも呼ばれる
- 非機能要求 (性能・効率・信頼性・セキュリティ・ユーザビリティなど) の確認が中心
- 実際の運用環境と同じ環境を準備

回帰テスト

- 変更の影響がある部分をテスト
  - 変更が要求通りに実現されているか
  - 意図しない変更がないか
- 変更の波及効果の分析
  - 変更の影響が及ぶ範囲の特定
  - 適切なモジュール構造の場合，調査が楽

ソフトウェアテスト技法

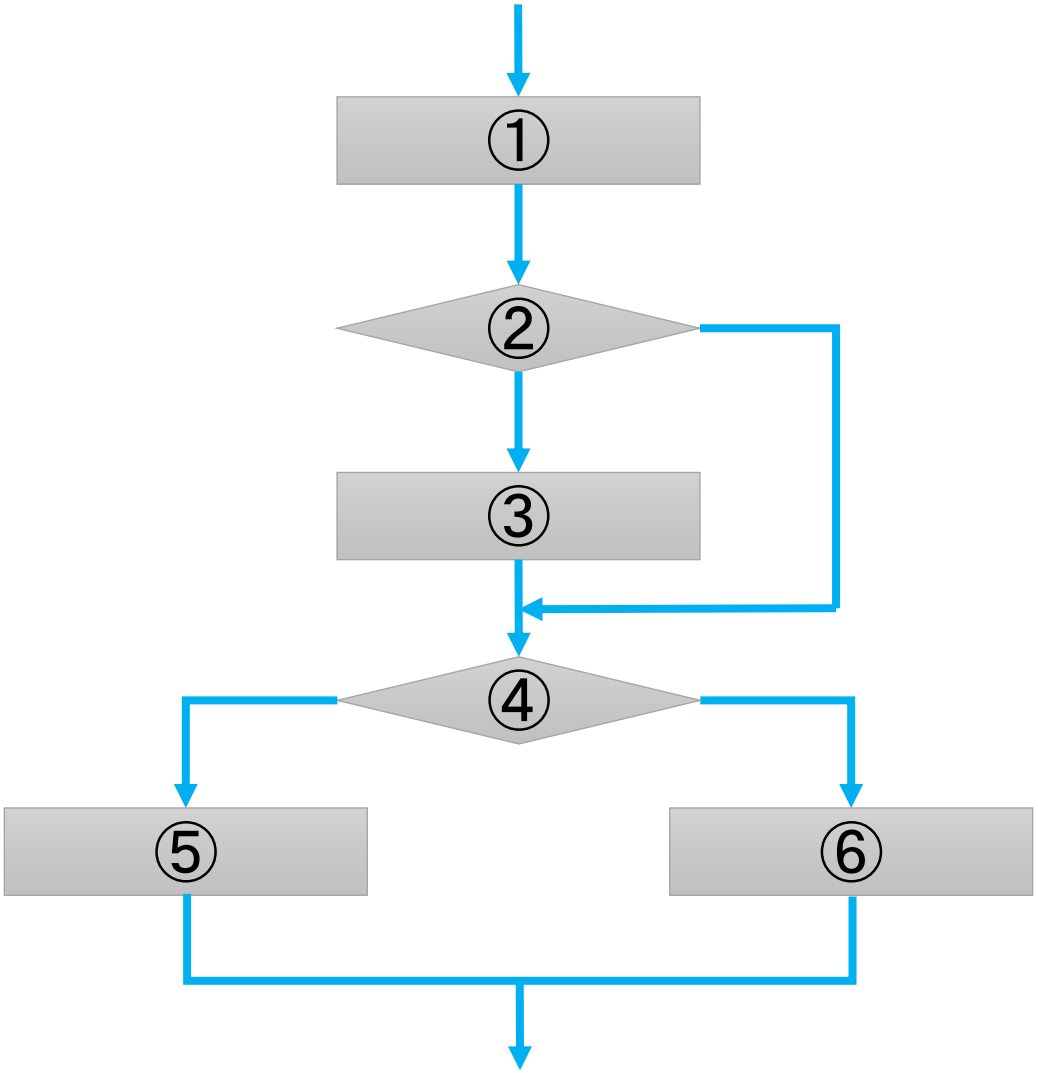
- ホワイトボックステスト
  - グラスボックステストとも呼ぶ
  - 内部仕様やプログラムコードに基づくテスト
  - 単体テストや結合テスト
- ブラックボックステスト
  - 外部仕様 (入出力の振舞い) に基づくテスト
  - システムテストや非機能テスト

プログラムコードに基づく技法

制御フローに基づく網羅性の基準

- 命令網羅
- 分岐網羅
- 条件網羅
- 分岐/条件網羅
- 複合条件網羅
- パス網羅

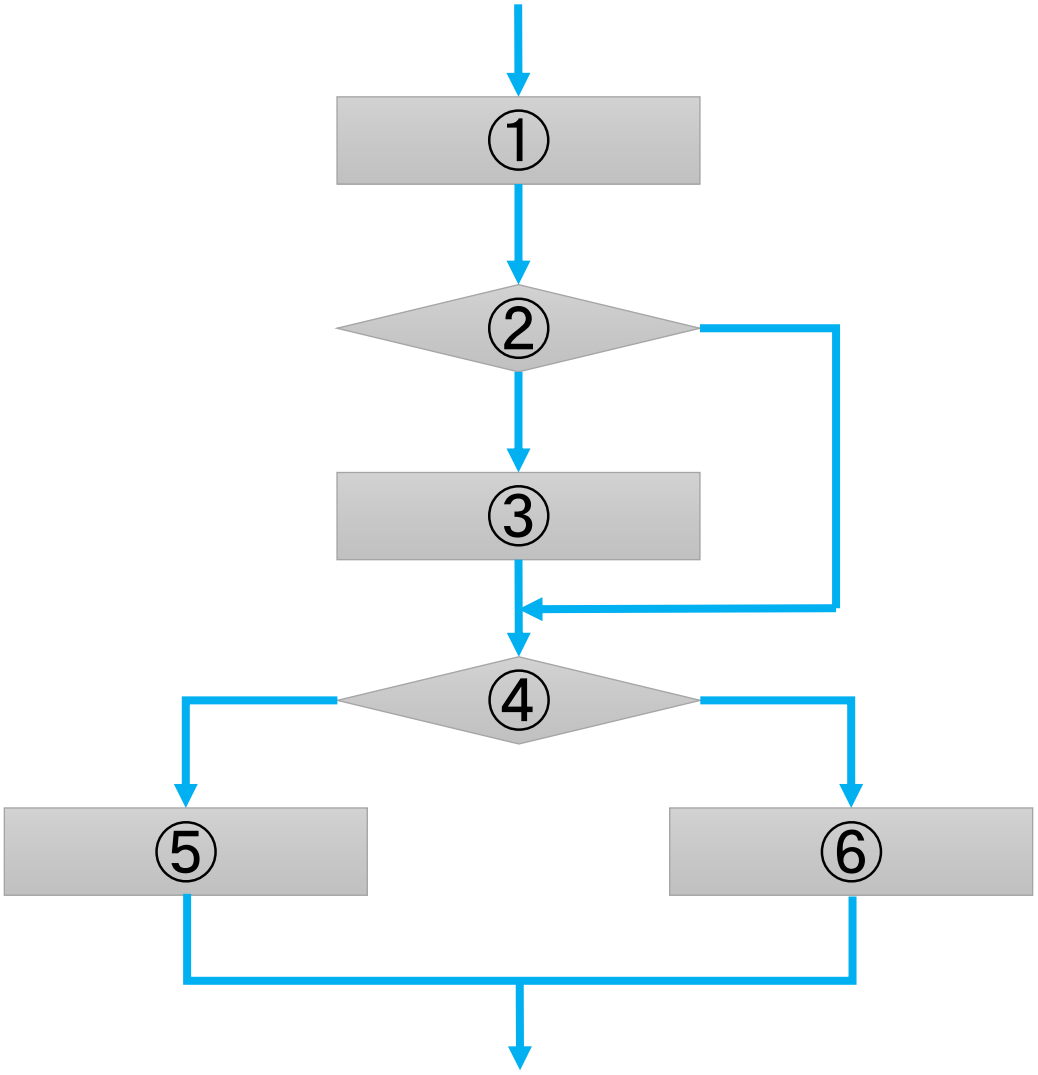
命令網羅



①⇒②⇒③⇒④⇒⑤

①⇒②⇒③⇒④⇒⑥

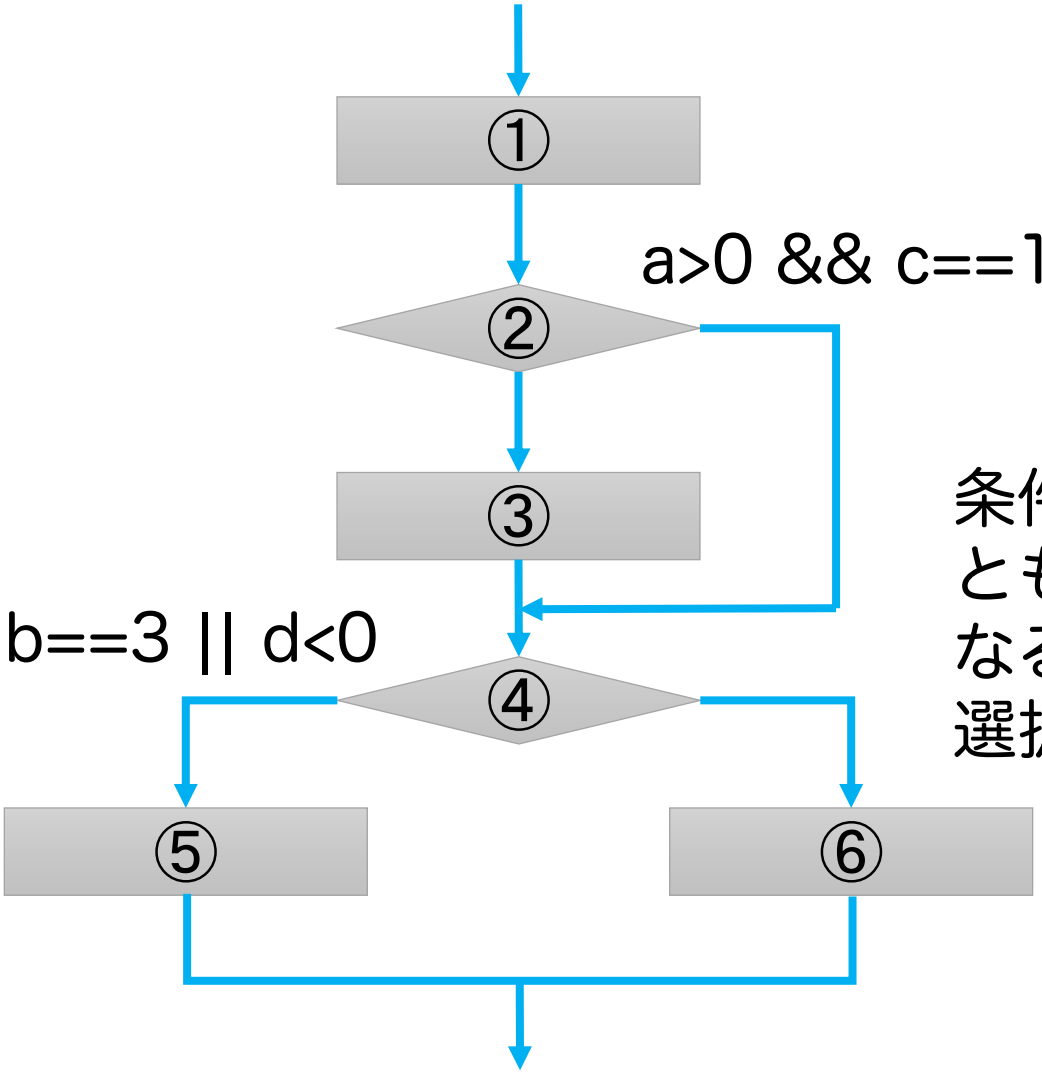
分岐網羅



①⇒②⇒③⇒④⇒⑤
①⇒②⇒④⇒⑥

①⇒②⇒③⇒④⇒⑥
①⇒②⇒④⇒⑤

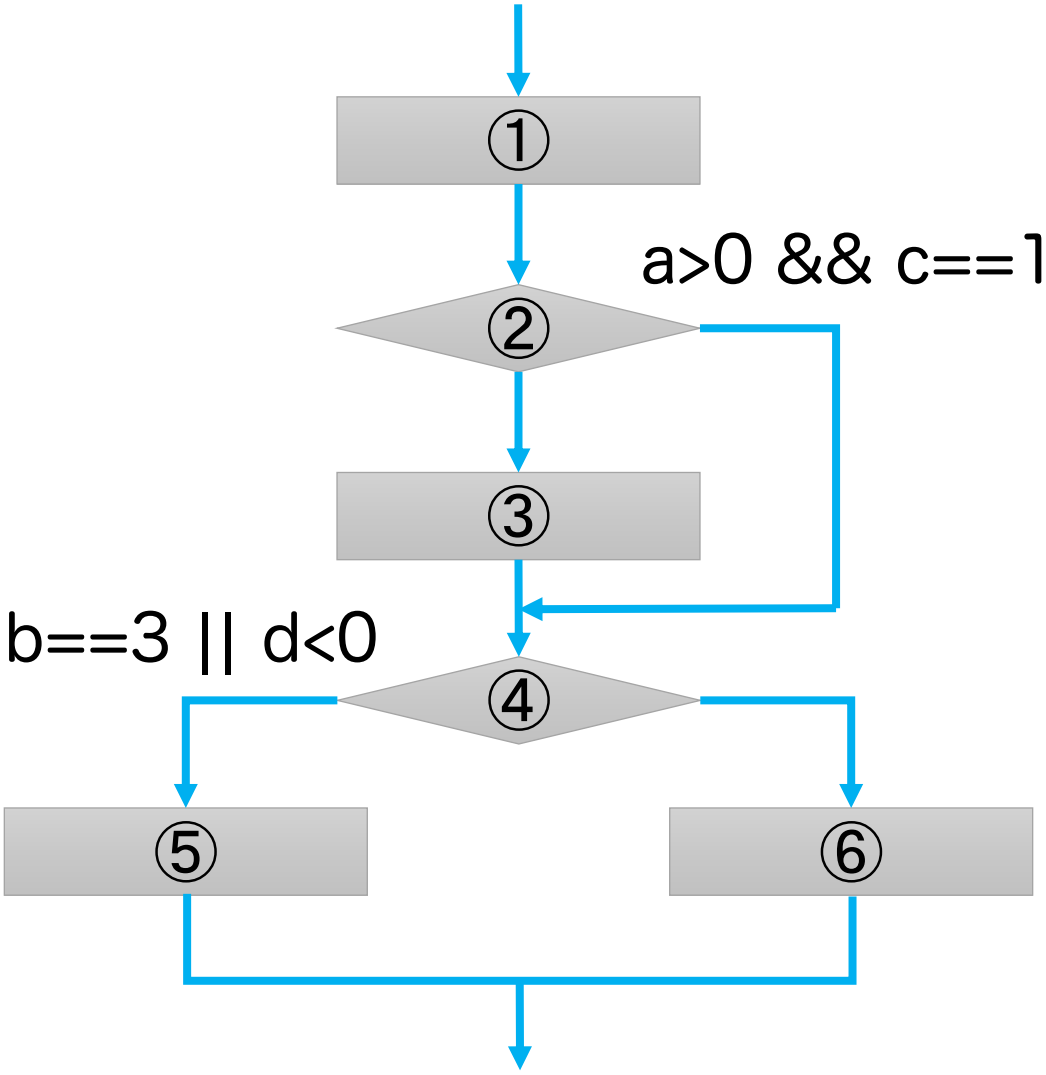
条件網羅



(a,c) = (1,0), (0,1)
(b,d) = (3,-1), (2,0)

条件節中の各条件が少なくとも1回ずつ true/falseになるようにテスト入力を選択

分岐/条件網羅



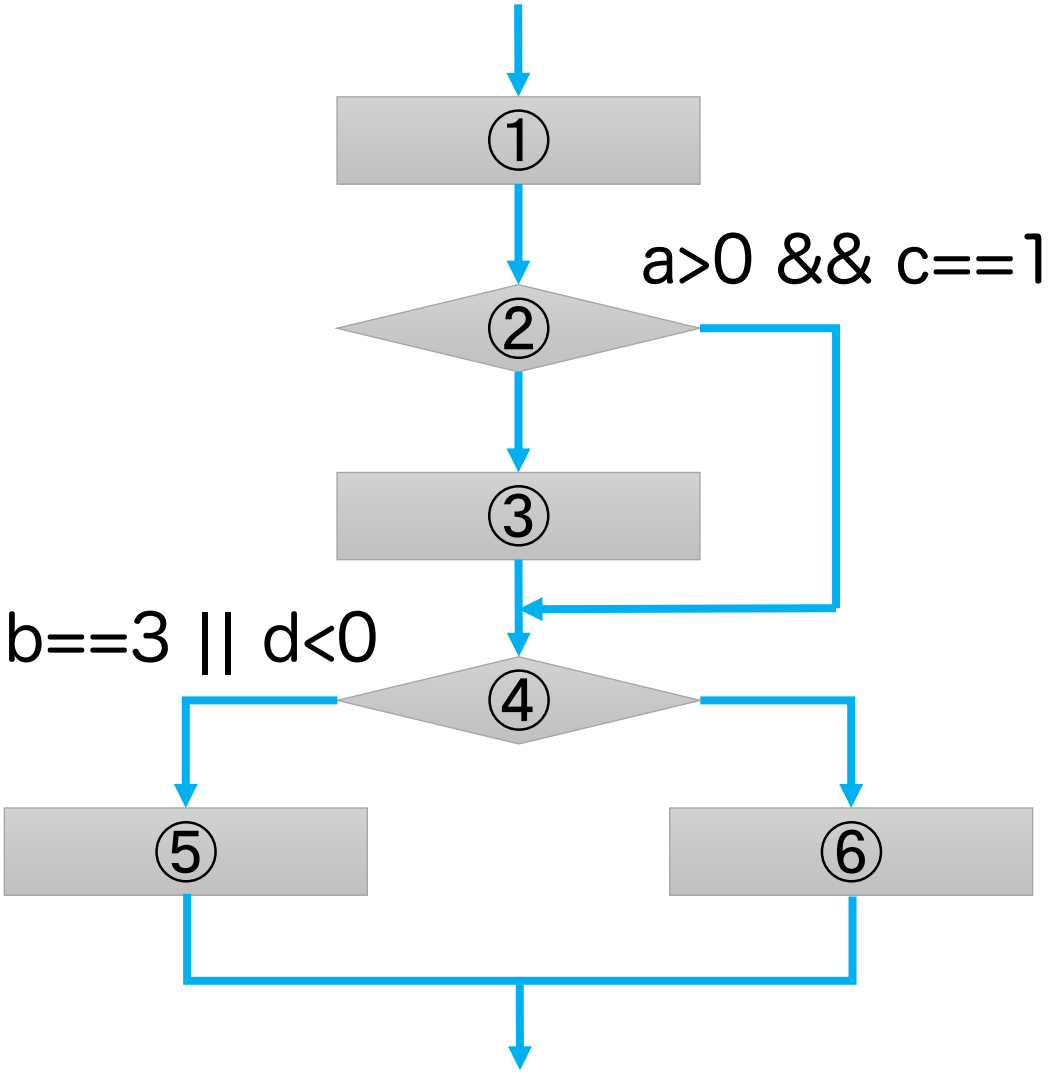
①⇒②⇒③⇒④⇒⑤
①⇒②⇒④⇒⑥

①⇒②⇒③⇒④⇒⑥
①⇒②⇒④⇒⑤

(a,c) = (1,1), (0,0)
(b,d) = (3,-1), (2,0)

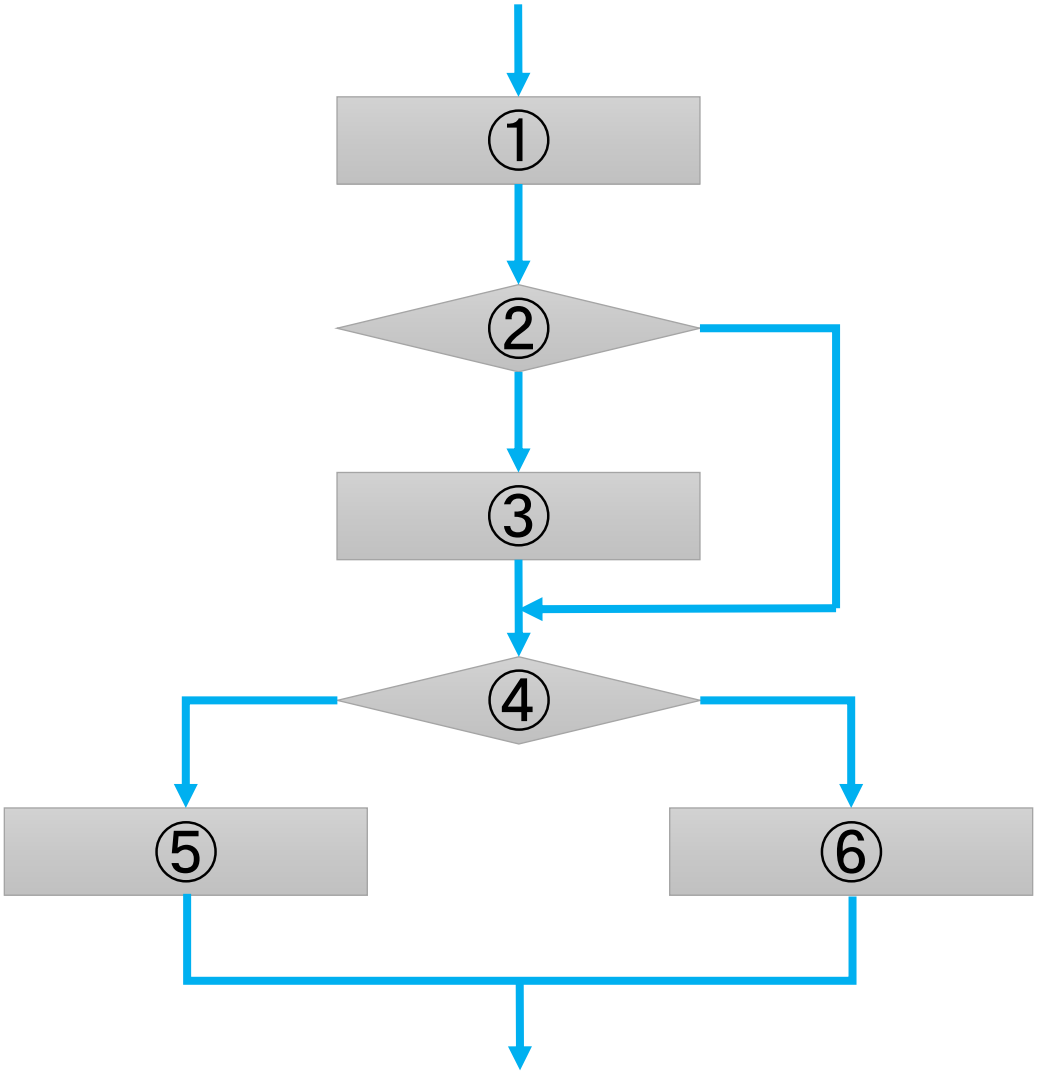
条件網羅を満たしかつ
分岐網羅を満たす

複合条件網羅



a > 0 の true/false
c==1 の true/false
の全組合わせ
b==3 の true/false
d < 0 の true/false
の全組合わせ

パス網羅



①⇒②⇒③⇒④⇒⑤

①⇒②⇒③⇒④⇒⑥

①⇒②⇒④⇒⑤

①⇒②⇒④⇒⑥

- 組み合わせ爆発
- 実行不可能なパス

データフローに基づく基準

- 変数の生成・定義・参照・破棄の様子を追う
- 特定の変数に着目した振舞いのテストで使用
- 特定の変数に関係する実行パスを選択

特定の変数に関係するコードを切り出した部分集合 ⇒ プログラムスライス
切り出すこと ⇒ プログラムスライシング

仕様に基づく技法

- 同値分割
  - プログラムの振舞いに対する影響が同じとみなされる値を同じ部分集合とし，その中の特定要素をテストデータとする
- 境界値分析
  - 同値分割の境界値をテストデータとする
  - 頑健性テスト (期待されるデータとかけ離れた値をテストデータとする)
- 決定表
  - 条件の組合わせに対する各アクションの可否の真理値表
- 状態遷移モデルに基づく技法
- プロトコルに基づく技法

同値クラス

プログラムの振舞いに対する影響が同じとみなされる値

例：

入力A: 1以上99以下の整数

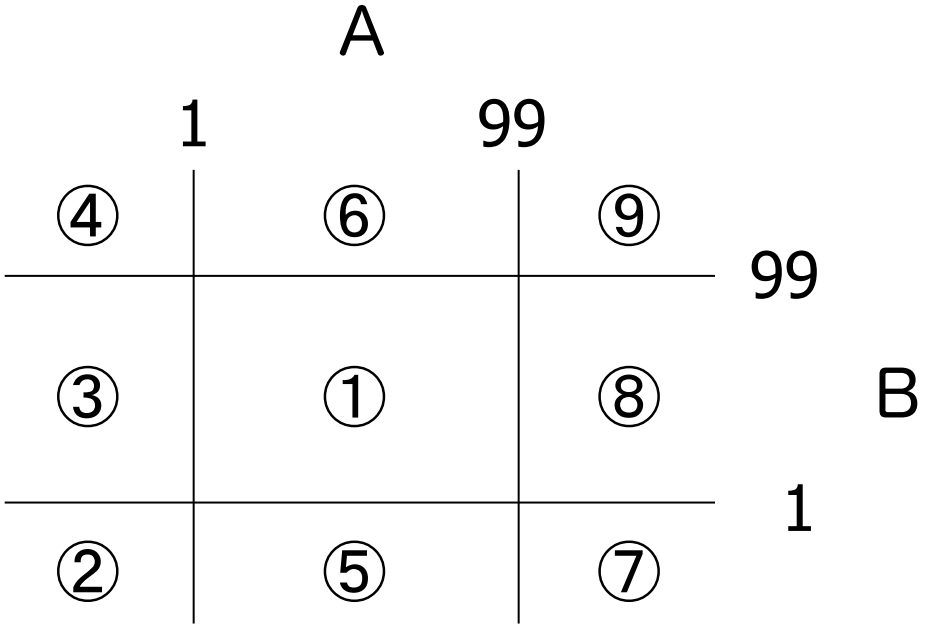
入力B: 1以上99以下の整数

出力C = 入力A x 入力B

無効同値〜0	有効同値1〜99	無効同値100〜

同値分割テスト

- 同値クラスに属する入力を組合わせてテストデータを作成
  - 有効同値のテストケース
    - ① (30, 40)
  - 無効同値のテストケース
    - ② (-5, -20)
    - ③ (-30, 40)
    - ④ (-20, 200)
    - ⑤ (20, -30)
    - ⑥ (30, 150)
    - ⑦ (140, -50)
    - ⑧ (220, 50)
    - ⑨ (305, 410)



無効同値クラスのテスト

- テストすべきかは設計方針に依存
- 契約による設計 (DbC: Design by Contract)
  - 入力に対する事前条件が満たされなければモジュールが動作しないよう設計
  - 再利用コンポーネントなどの設計・実装で利用
  - 無効同値クラスのテスト不要
- 防御的設計
  - 事前条件を満たさない入力でも受け付け，動作するよう設計
  - 無効同値クラスのテストが必要

同値分割テストの適用範囲と制約

- 単体テスト，結合テスト，システムテストのどのテスト粒度でも利用可能
- 入出力の大半が少数の同値クラスに分類できる場合に有効
- 入出力の同値クラスへの分類が仕様から行えることが前提
- テストケース数を大幅に減らすことができる
- 同値クラス内のデータをシステムが同じ方法で処理している (だろう) ことが前提

境界値分析

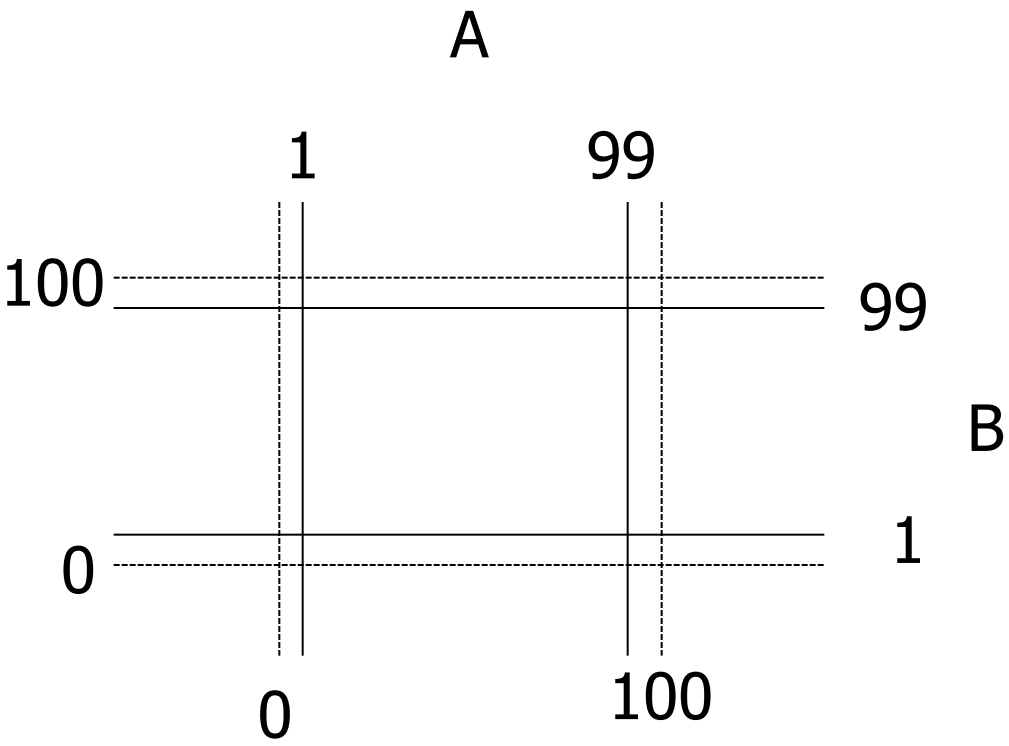
- 同値クラスの協会に注目
  - 境界上の値
  - 境界値のすぐ上
  - 境界値のすぐ下
- バグは境界値に多い
  という観測に基づく

例：

A，B ともに

0, 1, 99, 100 を

テストデータに加える



経験則によるテストデータ

- 良いデータ・悪いデータをテストケースにする
- 良いデータ
  - ユーザがよく使うデータ
  - プログラムが許す最小のデータ
  - プログラムが許す最大のデータ
  - ゼロ
- 悪いデータ
  - 非常に小さいデータ
  - 非常に大きいデータ
  - 長いデータ
  - 無効なデータ

境界値テストの適用範囲と制約

- 単体テスト，結合テスト，システムテストのどのテスト粒度でも利用可能
- 入出力の大半が少数の同値クラスに分類できる場合に有効
- 入出力の同値クラスへの分類が仕様から行えること，境界値を明確に識別できることが前提
- テストケース数を大幅に減らすことができる

決定表

- 条件とルールの組合わせ表
- それぞれのルールでの動作を表わす

	ルール1	ルール2	ルール3	…
条件				
条件1	Yes	Yes	No	…
条件2	Yes	No	Yes	…
…				
アクション				
アクション1	TRUE	TRUE	FALSE	…
アクション2	FALSE	TRUE	FALSE	…
…				

決定表テスト

- 決定表からテストケースの生成
  - 二値条件の場合，そのままテストケースに
  - 多値条件の場合，境界値分析の考え方を統合

	TC1	TC2	TC3	…
条件				
条件1	Yes	Yes	No	…
条件2	Yes	No	Yes	…
…				
期待される出力				
アクション1	TRUE	TRUE	FALSE	…
アクション2	FALSE	TRUE	FALSE	…
…				

ペア構成テスト

- 入力の種類が多く，すべての組合わせをテストできない場合
  - Webサイトの互換性テスト
    クライアント(ブラウザ)とそのOS,
    プラグインの種類，サーバとそのOS，…
- 入力のサブセットを選択する必要
  - できるだけ欠陥が発見できるサブセットの選択
  - すべての変数値のペアをテスト

すべてのペアが有効な理由

- 多くの故障は次のいずれか (経験則)
  - シングルモード
    - そのモジュールや機能が単体で正しく動かない
  - ダブルモード
    - 個々のモジュールや機能は単体で動く
    - 特定のモジュールや機能と組合わせで動かない
    - 他のモジュールや機能との組合わせでは動く
- ペア構成テスト
  - シングルモード・ダブルモードの故障をテストする最小のテストケースを導出

ペア構成テストの方法

- 直交表の利用
  - 任意の2列にすべての組合わせが出現
  - 例: 因子4つでそれぞれ3パターンの値 (水準)
- 入力変数を識別するのに必要な最小の直交表に入力値を割り当てる
  - これをテストデータとする
  - 期待値は別途必要

L9直交表

	因子1	因子2	因子3	因子4
TC1	1	1	1	1
TC2	1	2	2	2
TC3	1	3	3	3
TC4	2	1	2	3
TC5	2	2	3	1
TC6	2	3	1	2
TC7	3	1	3	2
TC8	3	2	1	3
TC9	3	3	2	1

ペア構成テストの例

- さまざまなクライアント・サーバの組合わせで正常に見られるかテストしたい
- 変数と取り得る値
  - クライアント (IE, Chrome, Safari)
  - クライアントOS (Windows, macOS, Linux)
  - サーバ (apache, IIS, nginx)
  - サーバOS (Windows, Linux, FreeBSD)
  - すべての組合わせ $3^4 = 81$通り
- 直交表
  - L9直交表で対応可能

ペア構成テストの例

- 直交表へのテスト問題の割り当て
  - 不足がある場合は何らかの有効な値を代入
- テスケース作成
  - それぞれの組合わせで期待される出力を決定
- テストケース数
  - 81通りから9通りに削減

L9直交表

	クライアント	クライアントOS	サーバ	サーバOS
TC1	IE	Windows	apache	Windows
TC2	IE	macOS	IIS	Linux
TC3	IE	Linux	nginx	FreeBSD
TC4	Chrome	Windows	IIS	FreeBSD
TC5	Chrome	macOS	nginx	Windows
TC6	Chrome	Linux	apache	Linux
TC7	Safari	Windows	nginx	Linux
TC8	Safari	macOS	apache	FreeBSD
TC9	Safari	Linux	IIS	Windows

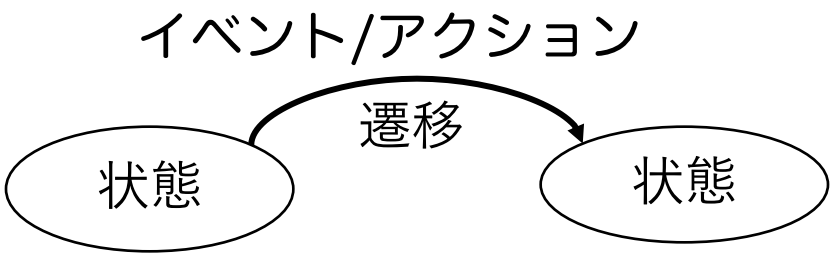
直交表のサイズの決定

- 最大水準因子の組み合わせから求める
  - 最大種類の水準を持つ因子と，その次に多い水準をもつ因子の組み合わせパターン数

- 自由度からの見積もり
  - 必要なパターン数 = 自由度の総和 + 1
  - 自由度：因子の水準数-1

状態遷移モデルに基づく技法

- システムの状態遷移仕様に基づく
  - 状態遷移図
  - 状態遷移表 (状態とイベント，アクションの組合わせ表)
- 状態遷移仕様
  - 状態
  - 遷移
  - イベント
  - アクション



現在の状態	イベント	アクション	次の状態
S1	E1	A1	S2
S2	E2	A2	S1
…	…	…	…

状態遷移テストの網羅性

- すべての状態を少なくとも1回訪れる
- すべてのイベントが少なくとも1回起動される
- すべてのパスが少なくとも1回実行される
- すべての遷移が少なくとも1回行われる

状態遷移テストの適用範囲と制約

- 単体テスト，結合テスト，システムテストのどのテスト粒度でも利用可能
- システム要求に状態遷移が仕様化されている場合に有効
- システム外部からのイベントにシステムが応答する必要のない場合には適用できない

オブジェクト指向プログラムのテスト

- コンストラクタ・デスクトラスタによるオブジェクトの生成・廃棄
- 継承によるオーバライドやスーパクラスの参照
- クラスフィールドやパブリックなデータフィールドの定義・参照関係

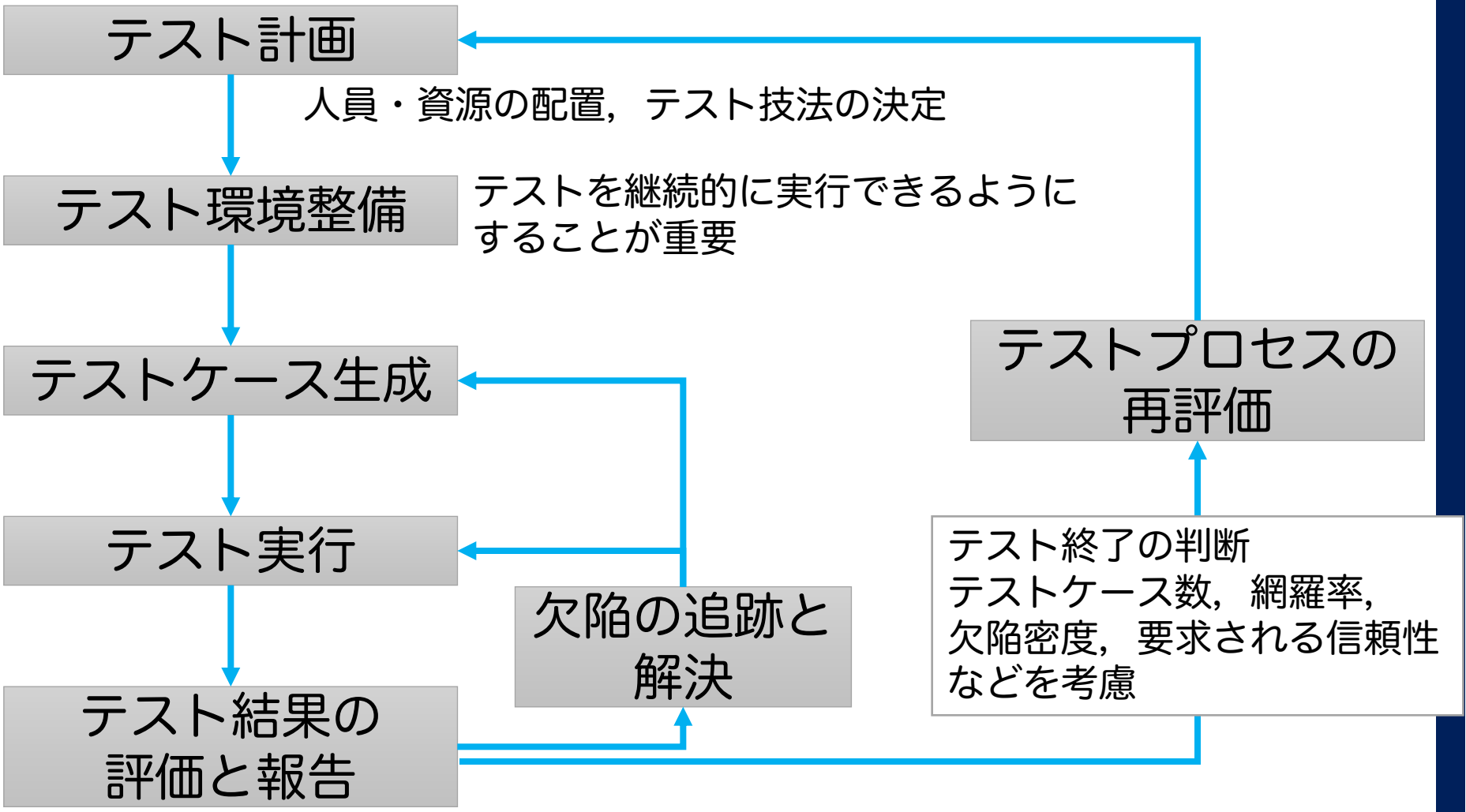
Webシステムのテスト

- ページとその遷移
- セッション
- 動的にページ (URLを含む) が生成される
- クライアントサイドスクリプトがサーバと独立して動作

その他のテスト

- GUIのテスト
  - 目視が重要な要素
- 並行処理，リアルタイム処理
  - 実行のタイミング
- 信頼性保証
  - 多数のランダムに生成されたデータの投入
  - 実際の運用に近いデータ生起確率に基づく統計的テスト

テストのプロセス


テスト計画
人員・資源の配置，テスト技法の決定
テスト環境整備
テストを継続的に実行できるように
することが重要
テストケース生成
テスト実行
テスト結果の
評価と報告
欠陥の追跡と
解決
テストプロセスの
再評価
テスト終了の判断
テストケース数，網羅率，
欠陥密度，要求される信頼性
などを考慮

まとめ

- ソフトウェアの正しさを確認するための手法
- 様々なテストの種類
- テストの網羅性基準
  - 制御フローに基づく基準
  - 命令網羅，分岐網羅，パス網羅
- テストデータの決め方
  - 同値分割，境界値分析など