先端エレクトロニクスDAQセミナー2020

～ ソフトウェア技術 ～

データ収集システム化技術（ソフトウェア工学）

広島工業大学

長坂 康史

nagasaka@cc.it-hiroshima.ac.jp

目的と達成目標

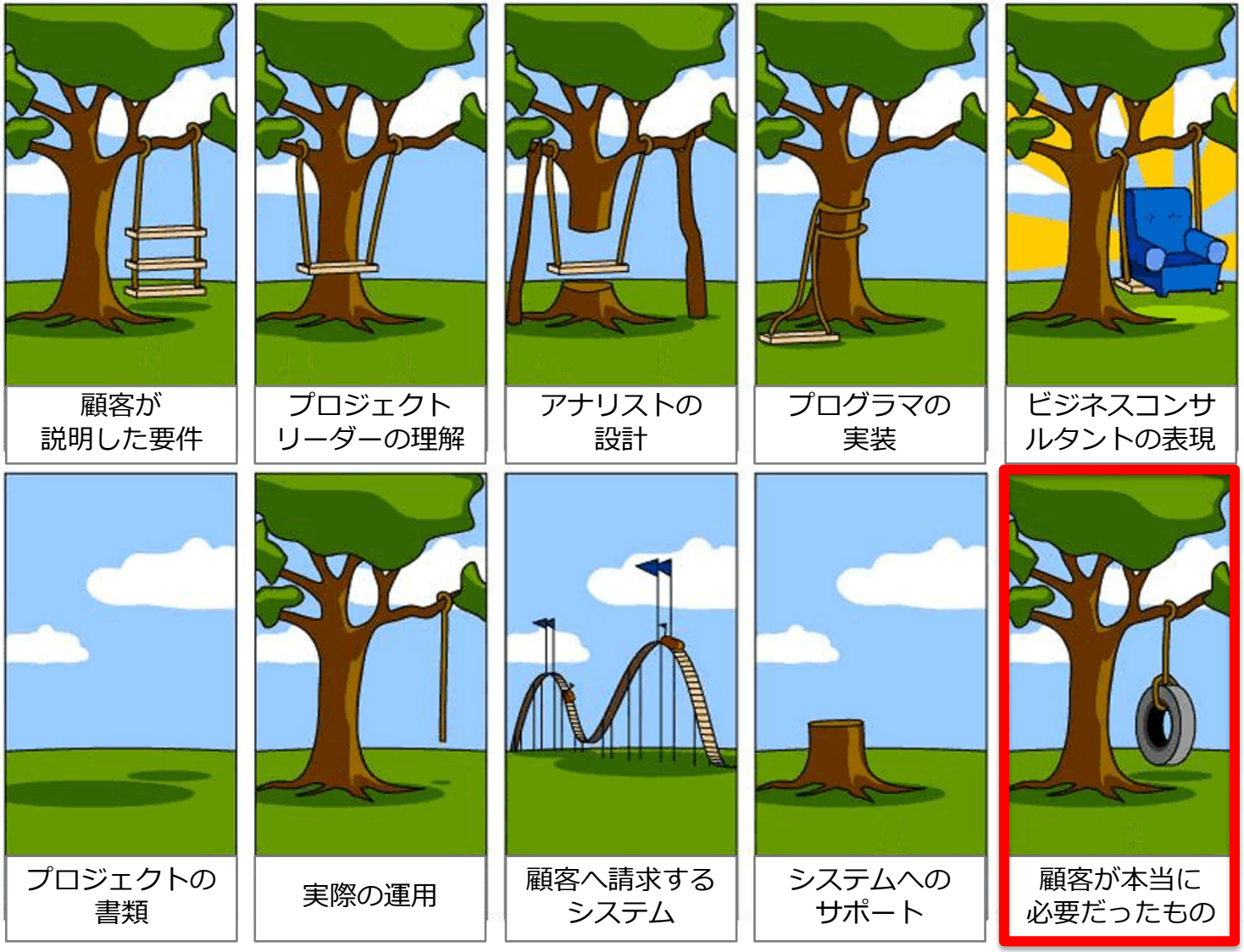
- 目的
  - データ収集システムを構築するためのシステム化技術、特に、システム構築に関するソフトウェア工学の知識とその周辺知識を修得する
- 達成目標
  - データ収集システムを構築するために必要なソフトウェア工学の知識を理解し、活用することができる

目次

- ソフトウェア工学
  - 概要
  - 開発プロセスモデル
  - ソフトウェアの設計と実装
  - ソフトウェアのテスト
  - テストケース設計技法
- システム設計言語
  - UML（統一モデリング言語）

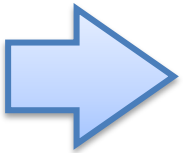
ソフトウェア工学
概要

ソフトウェア工学の必要性



ソフトウェア危機

- 1960年末～1970年代
  - ソフトウェアの危機（Software Crisis）
  - ソフトウェア開発が情報産業のボトルネック
    - ソフトウェア開発技術者の不足
      - プログラマやシステムエンジニア（SE）
    - ソフトウェア（プログラム）の品質の低下
      - ハードウェア　：大量生産で低価格化
      - ソフトウェア　：個別要求対応で高価格化
  - ソフトウェア開発の体系化（1970年代初め）

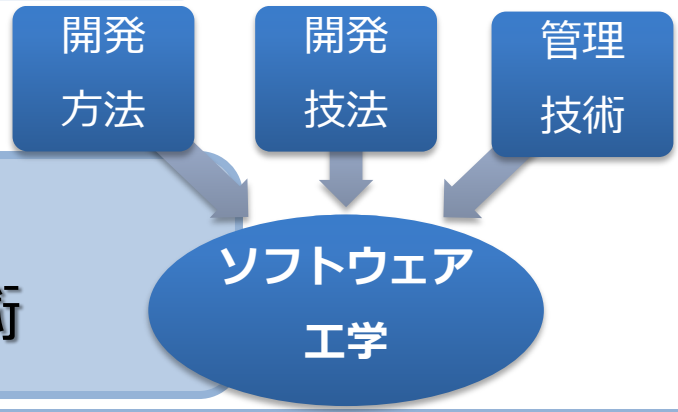


ソフトウェア工学
（Software Engineering）

ソフトウェア開発の課題と体系化

1. ソフトウェア開発
   - 要求に対応したソフトウェアの開発効率
   - 開発の時期と工期
2. ソフトウェア品質
   - ソフトウェアの信頼性
   - 社会的・経済的損失の回避
3. ソフトウェア管理
   - ソフトウェアの改修や仕様変更への対応
     - 安全性や効率性の向上

- ソフトウェア開発の体系化
  ①開発方法　②開発技法　③管理技術



ソフトウェア工学

- **ソフトウェア工学**
  - ソフトウェアの科学的概念と理論を抽出
  - 開発への工学的な開発方法論・技法の導入
    - ソフトウェア開発をより工学的に実施
- ソフトウェア危機の回避
  - 属人性を排した生産技術導入による生産性の向上
    - ソフトウェアのパッケージ化
    - 開発作業の標準化
    - ソフトウェア開発作業ツールによる開発の半自動化

- ソフトウェアの大規模開発・開発期間短縮の実現

ソフトウェア開発現場の課題

- 要求仕様決定の困難性
  - 要求分析の厳格化
- 再利用の困難性
  - 開発技法の活用および組織的取り組み
- プロジェクトにおけるトラブル発生の可能性
  - ソフトウェア開発プロセスに対応したプロジェクト管理と品質管理
- ソフトウェア開発規模・工数の見積りの誤り
  - 見積り技法や経験知の活用

ソフトウェア工学
開発プロセスモデル

開発プロセスモデル

- データ収集システム　⇒　情報システム
  - **ソフトウェア工学**の考え方を適用

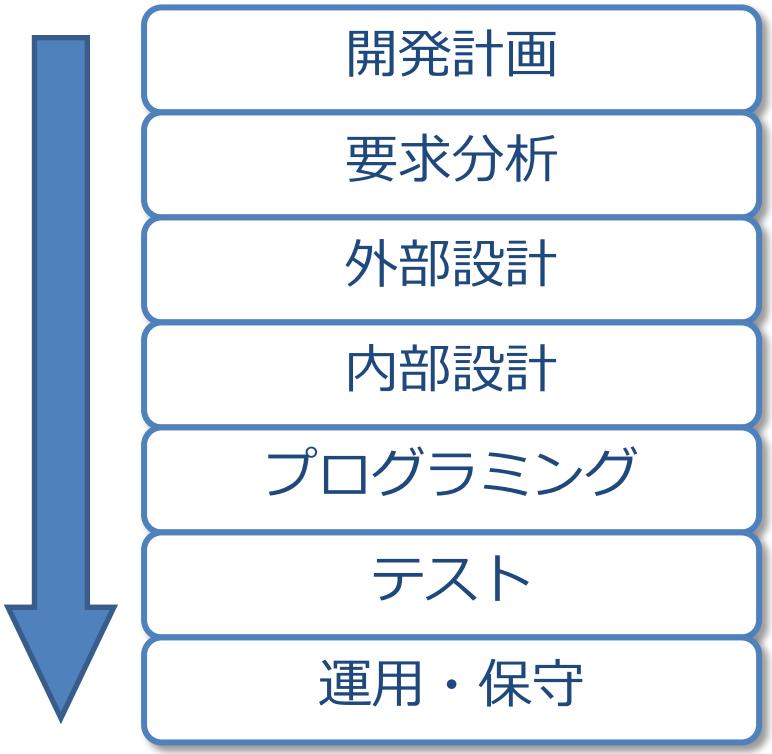
- 開発プロセス
  - システム開発プロジェクトを運営するための、プロジェクトの進め方に関する基本的な考え方をまとめたもの

- 開発プロセスモデル
  - システム開発のそれぞれの工程をプロセスと考えた開発プロセスモデル
    - ソフトウェアのライフサイクルのモデル
  - ウォータフォールモデル
  - プロトタイプモデル
  - スパイラル（反復型）モデル

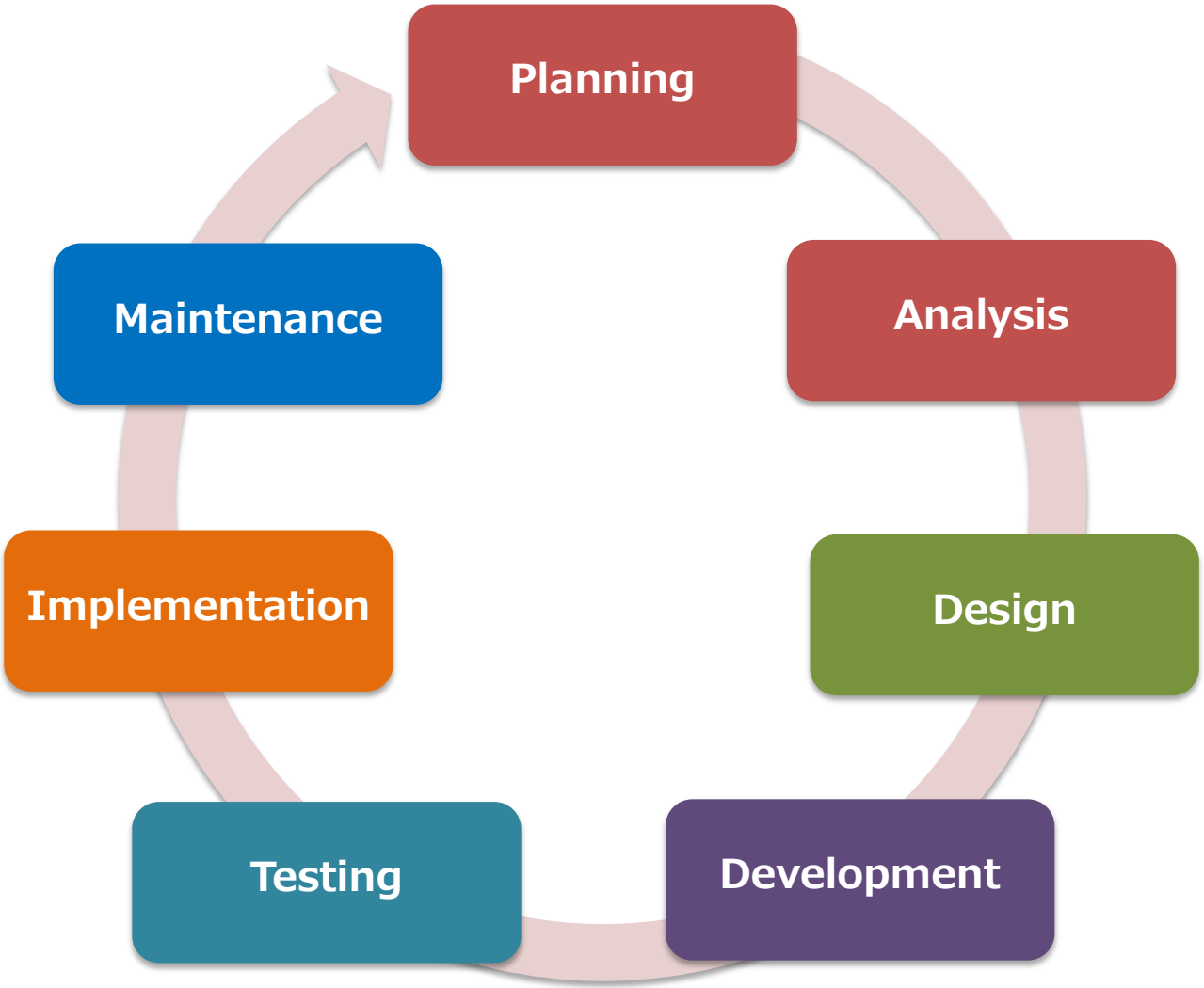
情報システムのライフサイクル

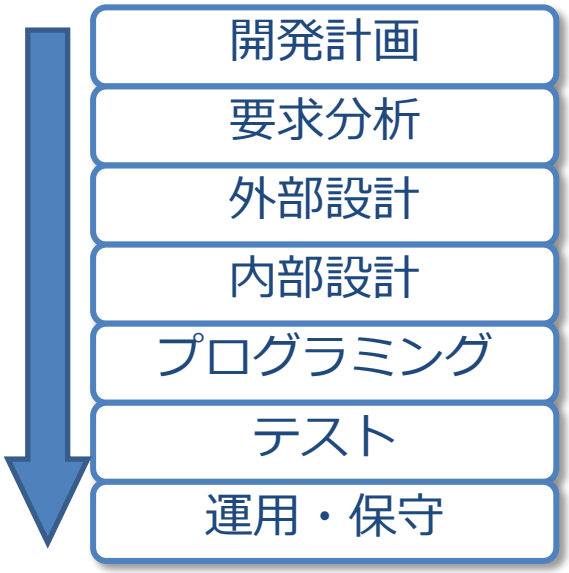
- ライフサイクル
  - 情報システム開発の計画、設計、運用、保守の工程

1. 開発計画
2. 要求分析
3. 外部設計
4. 内部設計
5. プログラミング
6. テスト
7. 運用・保守



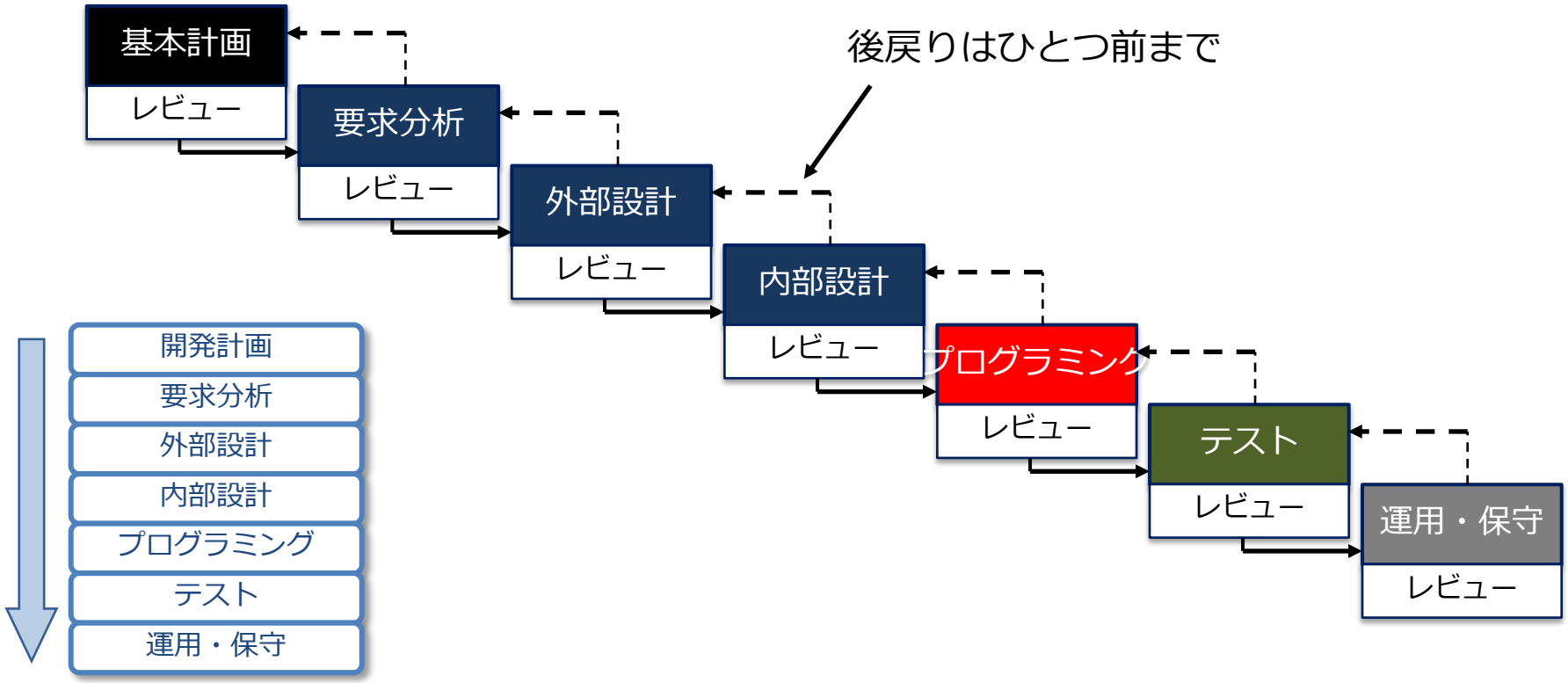
SDLC: Software Development Life Cycle





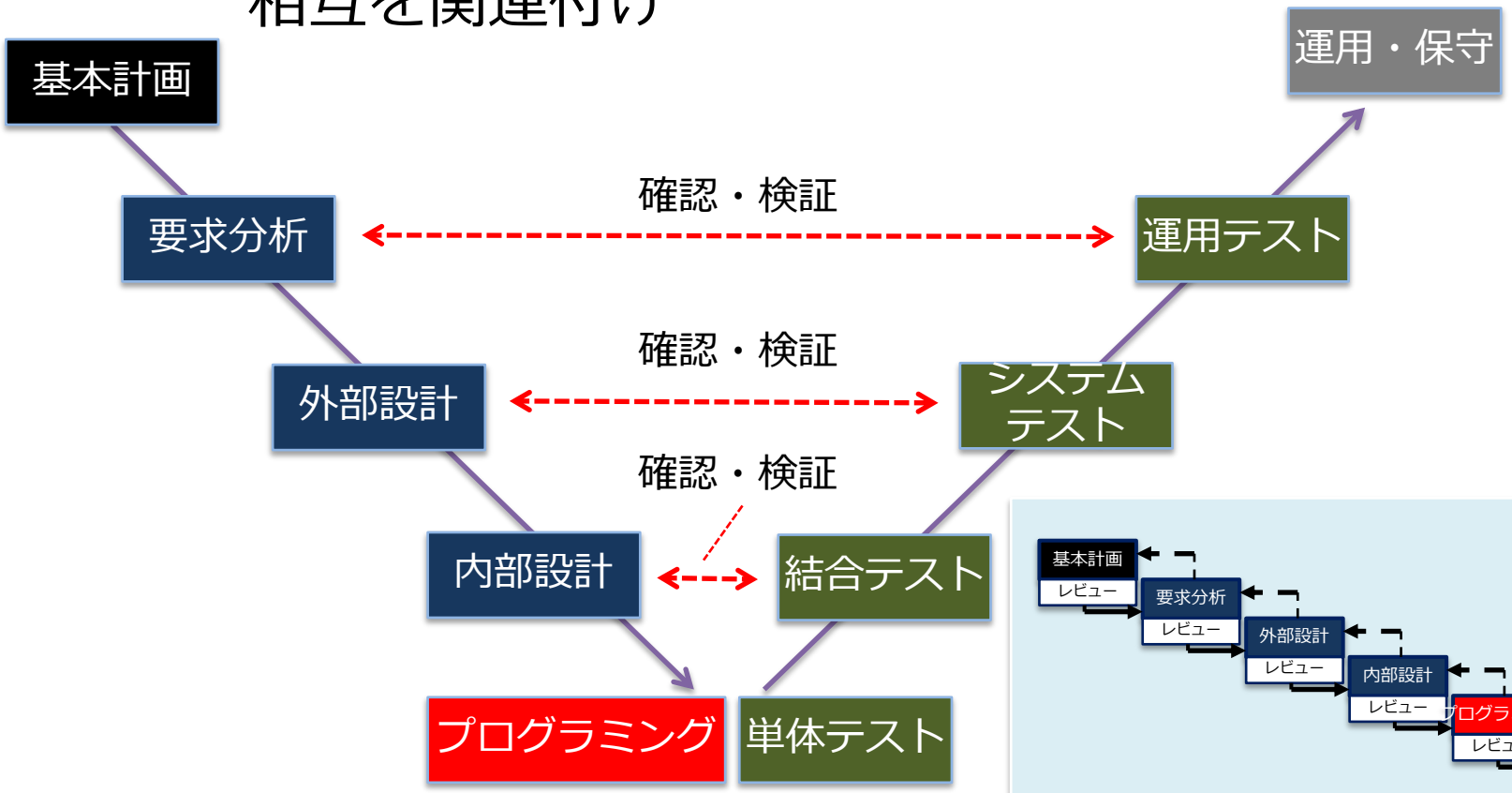
ウォータフォールモデル

- **Water Fall Model**
  - 基本計画から運用・保守までを滝の水が流れ落ちるように進めていく手法



ウォータフォールモデル（Vモデル）

- Vモデル（Vカーブ）
  - プログラミングと単体テストを折り返し点として相互を関連付け



ウォータフォールモデルの特徴

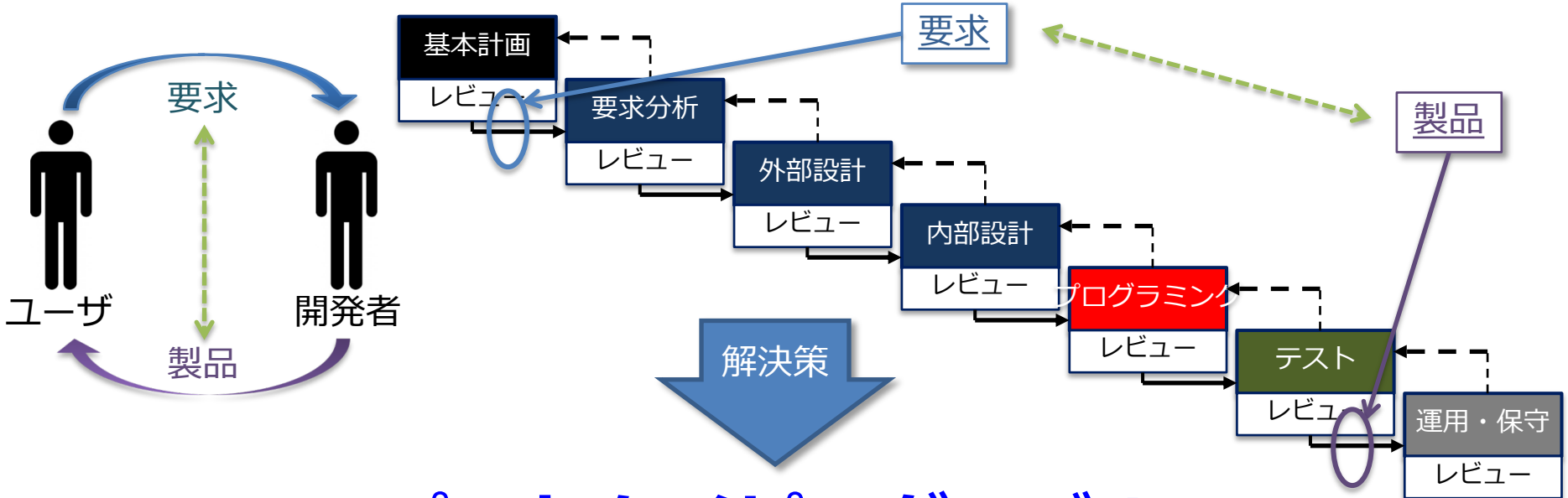
1. 逐次的な開発が可能
2. 各段階の成果を確認できる仕様書
   - 進捗管理の容易性
3. 作業分担が容易
   - 段階的細分化（モジュール化）
4. 大規模システム開発に対する高い適用効果
5. 充実した開発要員の教育システム

ウォータフォールモデルの問題点

1. 要求の検証までに必要とされる長い期間
   – 動作確認は原則、テスト終了後
2. 工程の一部分の遅れが全体の遅れに直結
   – 逐次的開発
3. 不具合修正時の上流工程に戻っての修正で起こる大きな影響

ウォータフォールモデルの問題点と解決策

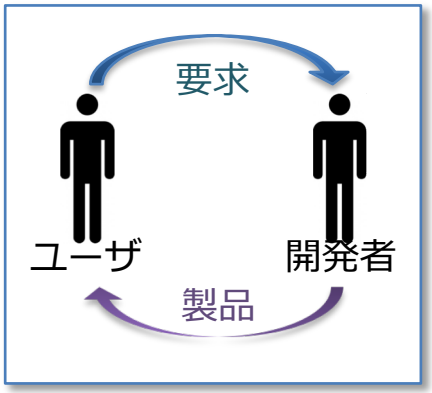
- ウォータフォールモデルの**問題点**
  1. 要求の検証までに必要とされる長い期間
  2. 工程の一部分の遅れが全体の遅れに直結
  3. 不具合修正時の上流工程に戻っての修正で起こる大きな影響

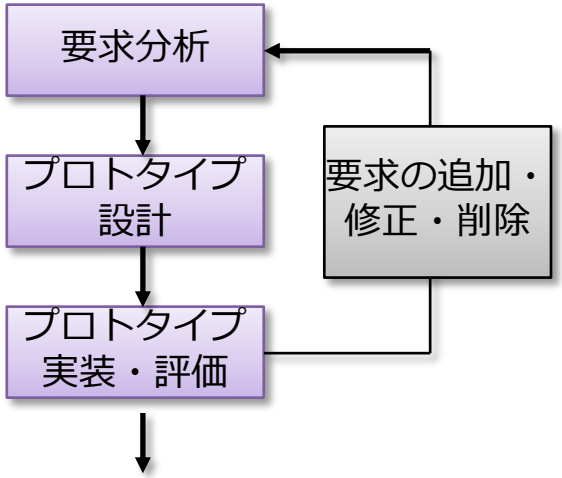


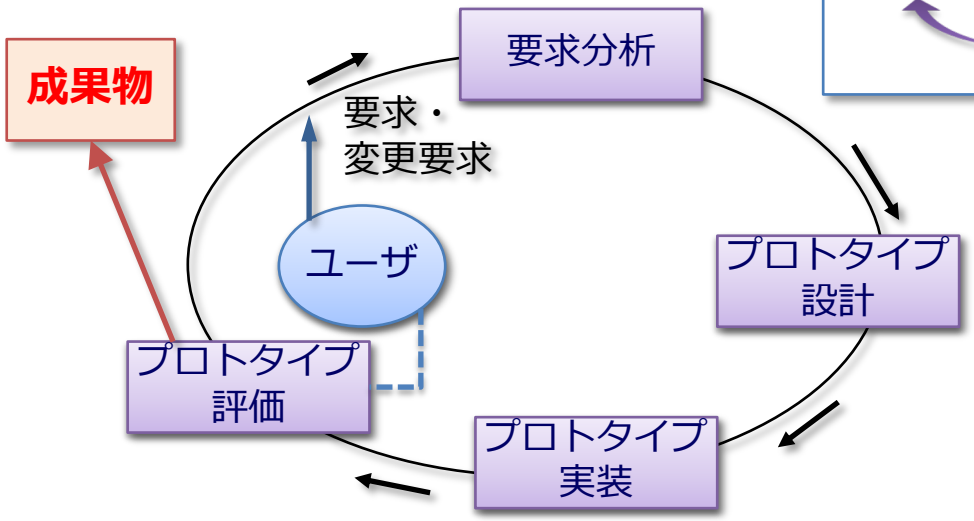
プロトタイピングモデル

プロトタイピングモデル

- **Prototyping Model**
  - プロトタイプ（試作品）の開発を早い段階で実施しチェックすることで、完成品とユーザ要件の間の差を小さくしようとするモデル







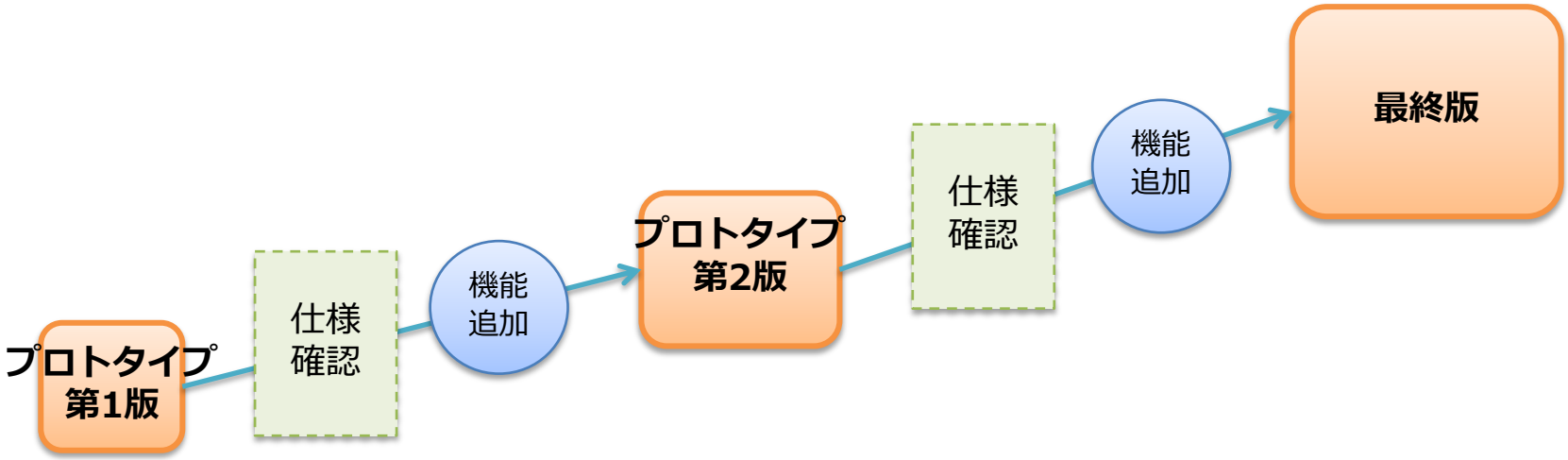
プロトタイプの種類

1. 破棄型

- 目的が達成したら破棄

2. 進化型

- それ自身を発展的に進化させ、最終システムになるように作り上げていく
  - 進化型モデル（Evolution Model）



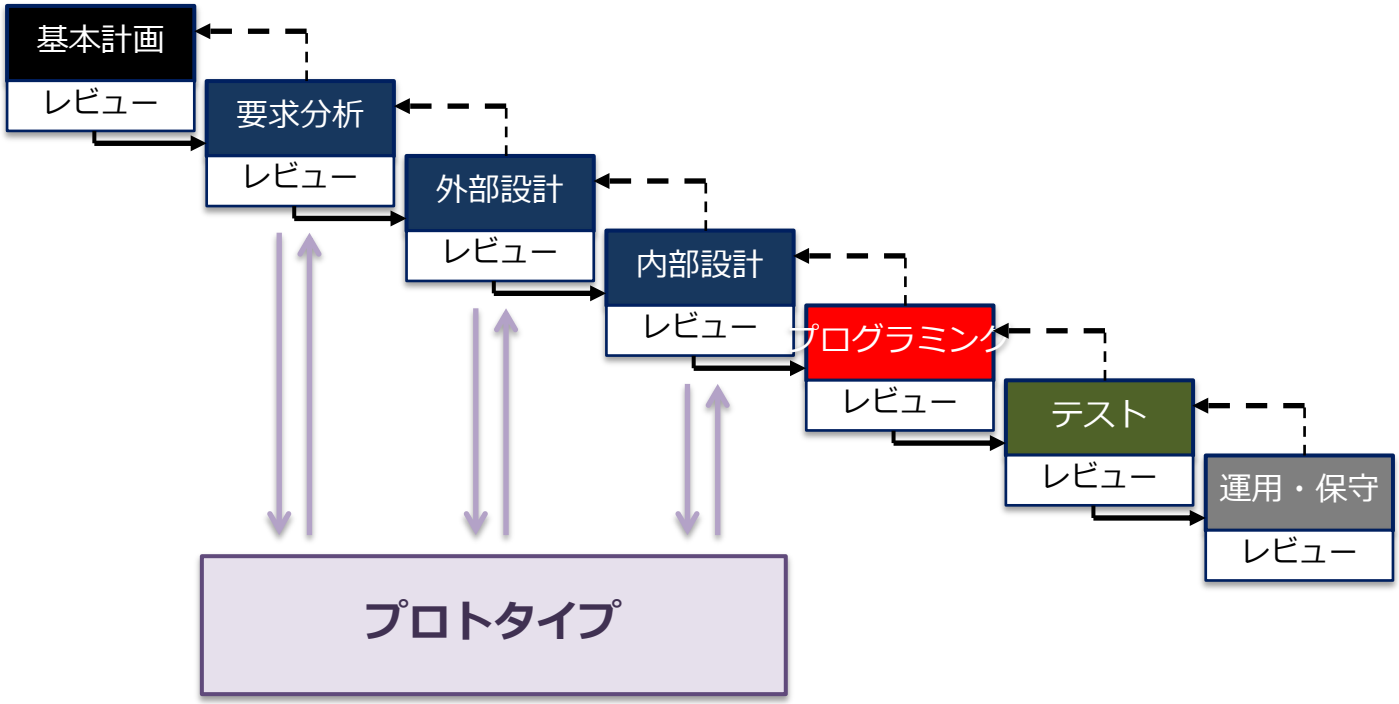
プロトタイピングモデルの特徴

- **利点**
  - プロトタイプ（試作品）の容易な開発・提示
  - ユーザーと開発者の相互理解の向上
  - 費用対効果の向上
  - システム開発速度の向上

- **欠点**
  - 全体像を見失う可能性
  - 開発者がプロトタイプ開発に固執する可能性
  - 開発の柔軟性欠如
  - 大規模アプリケーションには不向き

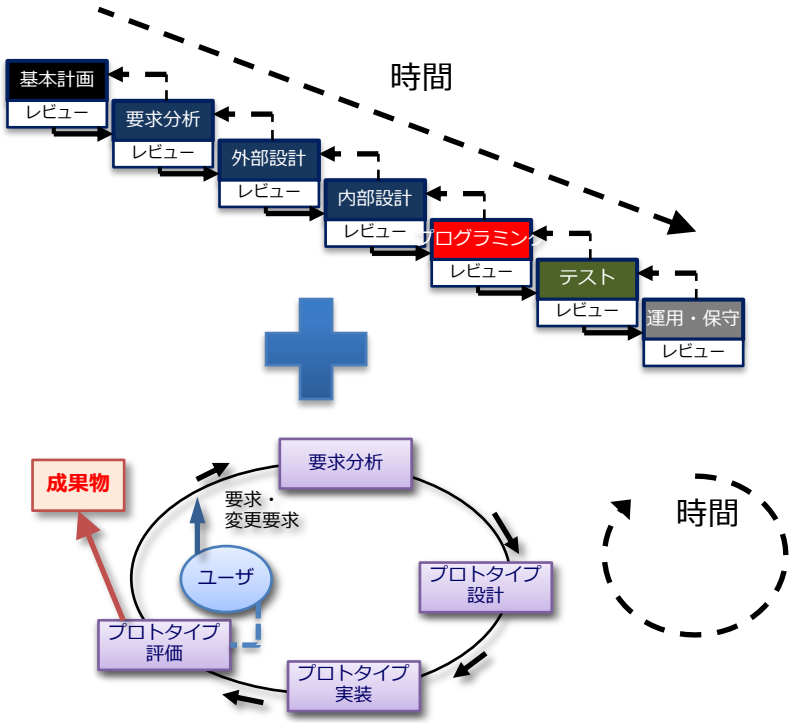
プロトタイピングモとウォータフォール

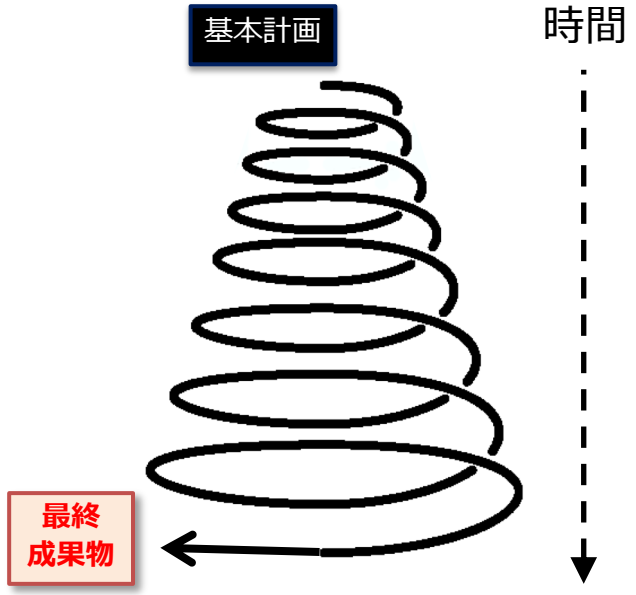
- 要求分析および設計段階で課題がある場合、プロトタイプを作成し、顧客とのギャップを少なくする



ウォータフォールとプロトタイピングとの融合

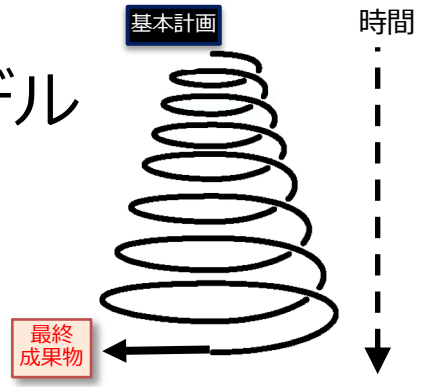
- ウォータフォールモデルの利点
  - 逐次的な開発
  - 進捗管理の容易性
  - 作業分担が容易
- プロトタイピングモデルの利点
  - プロトタイプによるユーザーと開発者の相互理解
  - 費用対効果の向上
  - システム開発速度の向上

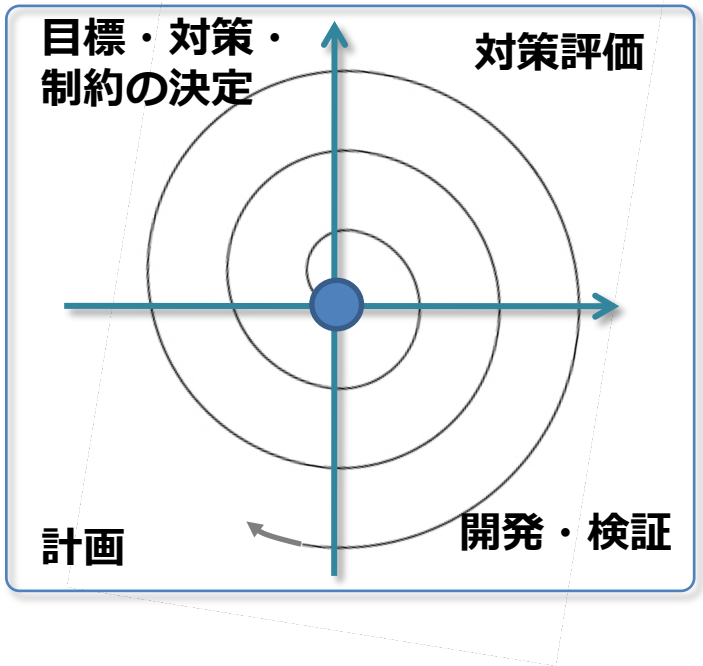




スパイラルモデル

- **Spiral Model （反復型モデル）**
  - ウォータフォールモデル＋プロトタイプモデル
  - 開発を４つの領域に分割
    - 計画
      - 解決すべきリスク・コスト・スケジュールを含む
    - 目標・対策・制約の決定
    - 対策評価
    - 開発・検証



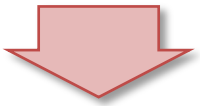


スパイラルモデルの特徴

- **利点**
  - ウォータフォールモデルとプロトタイプモデルの利点を合わせ持つ
  - システムの問題点の早期発見
  - 要求の追加・変更への柔軟な対応
  - 工程の明確な定義
- **欠点**
  - 全体像をとらえにくい
    - 全体を見ずに進めるため、誤った方向に進む可能性がある
  - プロジェクト管理が困難

スパイラル型開発プロセスの問題点

- プロジェクトの方向性を誤る
  - 全体像をとらえにくい
  - 部分的に進みすぎる
- プロジェクト管理が難しい
  - 成果物が予定通り完成しない
    - ユーザからのフィードバックなどによる



新しい開発プロセス

新しい開発プロセス

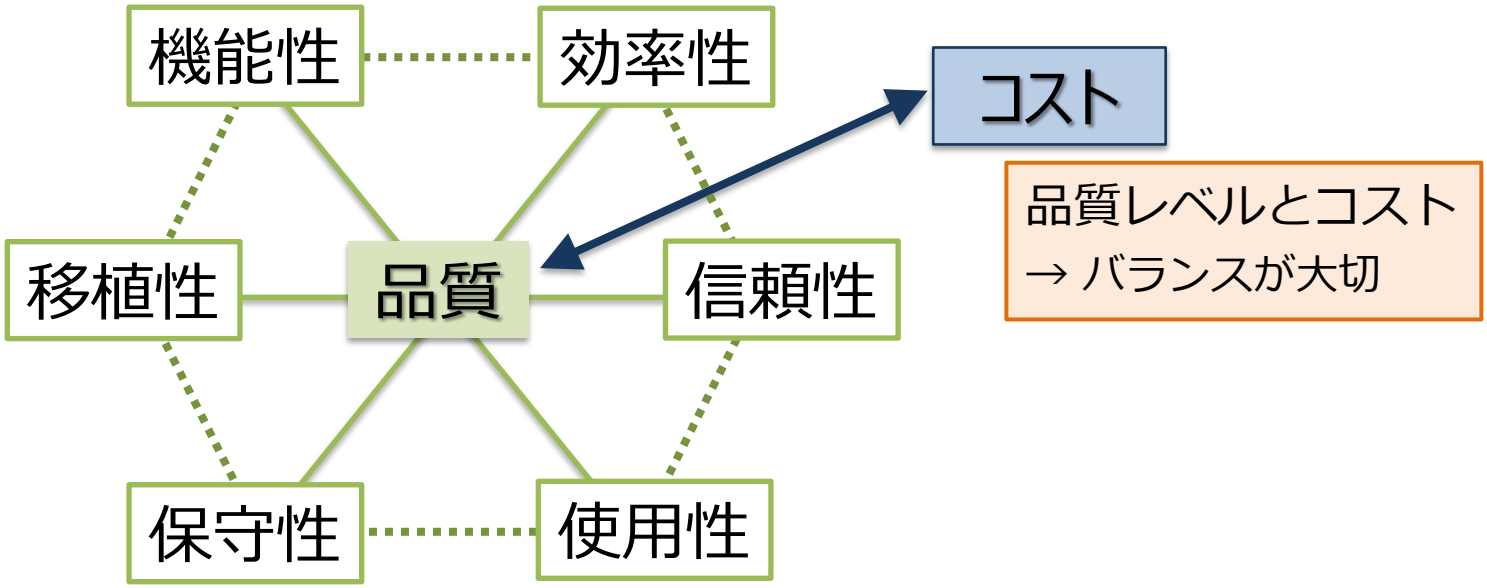
- 開発プロセス
  - 開発プロセスプロジェクトチームの
    - アクティビティの順序の提示
    - 成果物の規定
    - 個々、および、全体のタスクの指示
    - メトリクス（評価尺度）の提供

- 新しい開発プロセス
  - RUP（Rational Unified Process）
  - XP（eXtreme Programming）
  - Agile

ソフトウェア工学
ソフトウェアの設計と実装

良い設計

- **<u>要求品質を満たす</u>ソフトウェアの開発・作成**
  - ソフトウェアごとに求められる品質レベルが異なる
    - ユーザ視点→製品品質
    - 開発者視点→コード品質



品質特性：ISO/IEC 25010

- **機能適合性（Functional Suitability）**
  - 機能完全性
  - 機能正確性
  - 機能適切性
- **性能効率性（Performance Efficiency）**
  - 時間効率性
  - 資源効率性
  - 容量満足性
- **互換性（Compatibility）**
  - 共存性
  - 相互運用性
- **使用性（Usability）**
  - 適切度認識性
  - 習得性
  - 運用操作性
  - ユーザエラー防止性
  - ユーザI/F快美性
  - アクセシビリティ
- **信頼性（Reliability）**
  - 成熟性
  - 可用性
  - 障害許容性
  - 回復性
- **セキュリティ（Security）**
  - 機密性
  - インテグリティ
  - 否認防止性
  - 責任追跡性
  - 真正性
- **保守性（Maintainability）**
  - モジュール性
  - 再利用性
  - 解析性
  - 修正性
  - 試験性
- **移植性（Portability）**
  - 適応性
  - 設置性
  - 置換性

システム設計におけるポイント

- ソフトウェア工学におけるシステム設計の課題
  - システムに求められる機能の高度化による複雑さ

- 複雑さ克服のための３つの指針
  - **a. 抽象化とモデルの利用**
    - システムの性質のみをクローズアップさせる
      - 性質を正しく表現できる適切なモデルの適用
      - 本質ではない事項をそぎ落とす
  - **b. 分割と階層化**
    - 構造化設計
      - システムの機能要素の階層化に主眼をおいた設計
  - **c. 独立性**
    - 高い独立性
      - 要素内：　密結合
      - 要素外：　疎結合・単純

ソフトウェア工学
ソフトウェアのテスト

テスト

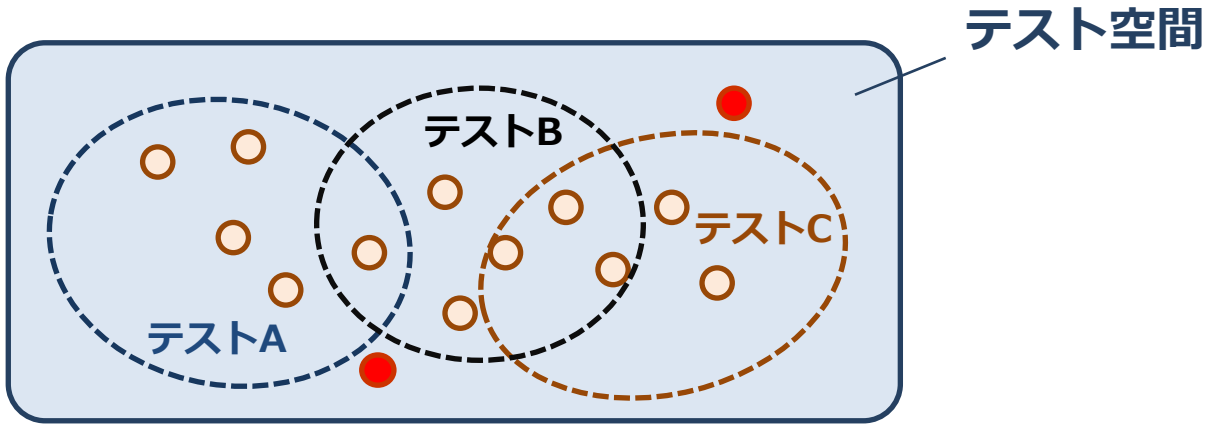
- **品質管理活動**における**一検査手段**
  - 目標の品質指標を達成しているかどうかを判定
  - プログラム内に残存するエラーを検知する行為

- テストの課題
  ① 限られた時間内で**最大効率化**
  ② **明確な品質指標**の設定

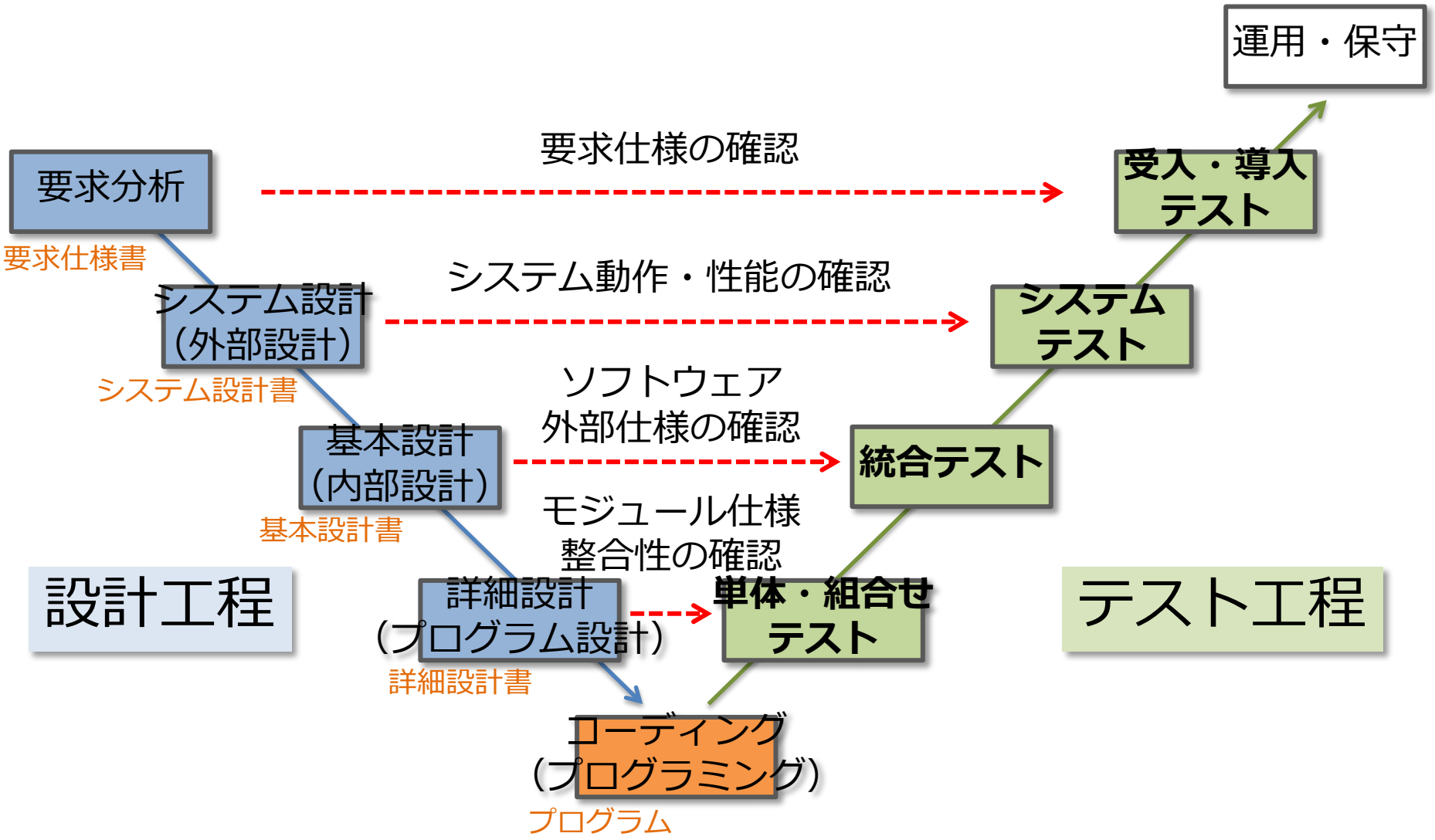
テスト空間

- テスト空間とテストの完全性
  - **テスト空間**
    - エラーがないことを保証できるテストケースの全集合
      - 大きすぎるため、すべてを網羅することは不可能
  - テスト空間全域にわたり試験を行うことは不可能

    ⇒　**完全性は確保できない**



テスト工程と設計工程との対比



ソフトウェア工学
テストケース設計技法

テストケース設計技法

- テストケース設計
  - **完全なテスト**を**限られた時間内**に行うことは難しい
  - **少ないテスト**で**多くのエラーを発見**できることが望ましい

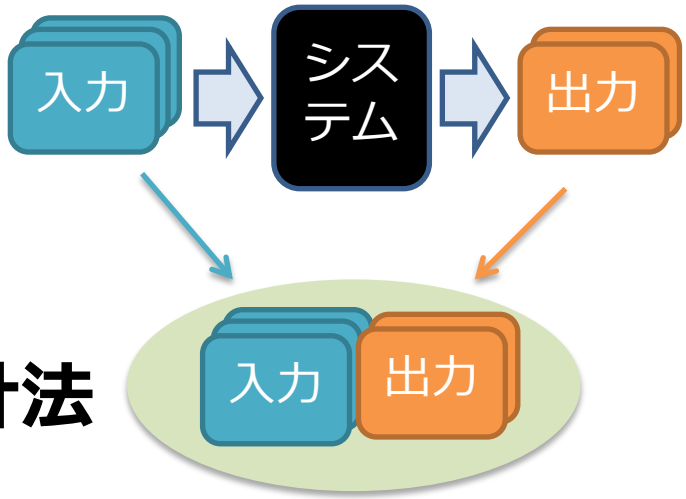
  ⇒ **テストケース**が**重要**
  - **入力の定義**と**予想される出力（結果）の定義**

- テストケース設計の種類
  1. **ブラックボックステスト**
  2. **ホワイトボックステスト**
  3. **ランダムテスト**
  4. **妥当性確認テスト**

1. ブラックボックステスト

- 特徴
  - 可能な**入力の組み合せ**と**出力**を選択する方法
    - プログラムの**機能記述（仕様）に基づく**

- テストケース設計法の種類
  a. **同値分割法**
  b. **限界値分析法**
  c. **状態ベース仕様に基づく設計法**



a. 同値分割法（１）

- 特徴
  - 入力条件のみから、テスト入力空間を分割
    - 入力条件を網羅的にテスト可能
  - 入力空間を**同値クラス**で分割

  - **同値クラス**
    - プログラムにとって同じ扱いを受けるはずの値の範囲

a. 有効同値クラス

- プログラムにとって有効な値

b. 無効同値クラス

- プログラムにとって無効な値

a. 同値分割法（2）

- 指針
  1. **有効同値クラスと無効同値クラスの設定**
     - 入力条件ごとに設定
  2. **テストケースの設定**
     - できるだけ多くの有効同値クラスを利用
     - 無効同値クラスを一つだけカバー

b. 限界値分析法

- 特徴
  - 同値分析では発見できなかったプログラミング上の実装ミスを見出す
  - **条件の変わり目**と思われる値に注目
    - 単純な技法化は困難

- 指針
  1. 入力条件だけでなく、出力条件も考慮
  2. テストケースとして各条件の境界の値を選択

C. 状態ベース仕様に基づく設計法

- 特徴
  - **状態ベース仕様**からあるレベルの仕様範囲を含むことを目的としたテストケース設計技法
    - 状態ベース仕様：入力と内部状態から出力を記述する仕様
  - 仕様とプログラムとの一致性を保証
    - 機能的仕様の網羅度と対応
  - 仕様記述からテストケースを生成可能

- 指針
  1. **状態遷移表の作成**
     - ある内部状態にあるときに発生したあるイベントがどのような出力（次状態とアクションイベント）をもつべきかの表
  2. **なるべく多くのセルをたどるイベント列の選択・テストの実施**

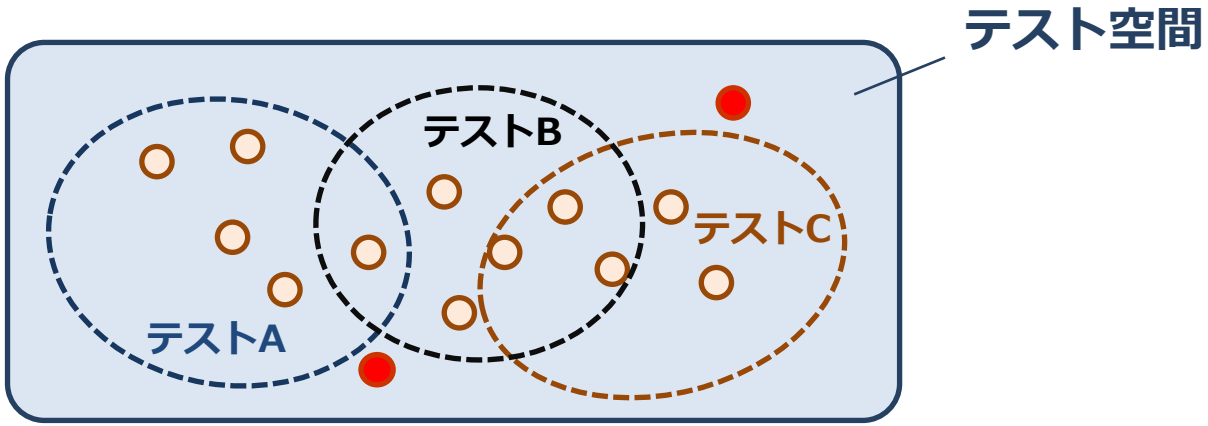
2. ホワイトボックステスト

- 特徴
  - プログラムの内部処理に注目し、プログラム構造を網羅するようにテストケースを見出す方法
  - 複雑な入力条件の組み合せで発生するエラー処理などをテスト可能

- 種類
  - **命令網羅テスト**（Statement Coverage Test）
    - 網羅度の基準：全命令が実行されているかどうか
  - **分岐網羅テスト**（Branch Coverage Test）
    - 網羅度の基準：全分岐が実行されているかどうか

3. ランダムテスト

- 特徴
  - 入力空間から入力データをランダムに抽出してテスト
  - コンピュータによるテストケース生成が可能
  - 網羅度は低い



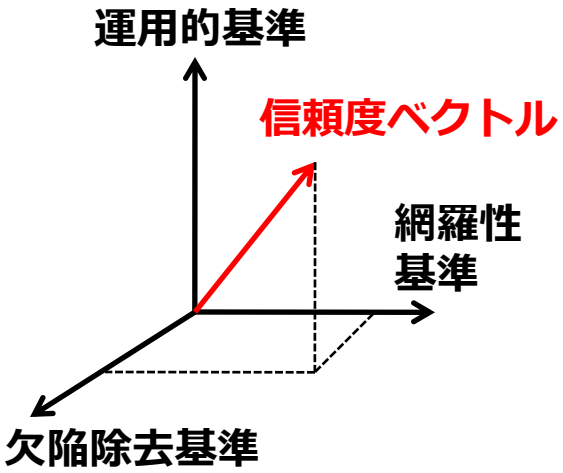
4. 妥当性確認テスト

- ユーザ要求の非機能的な側面に関するテスト

妥当性確認テストの種類		テスト対象
1	大容量テスト	想定データ量を超えた場合のシステムの振る舞い
2	ストレステスト	過負荷状態におけるシステムの振る舞い
3	有用度テスト	人間的要因
4	セキュリティテスト	データ保全性
5	性能テスト	応答時間・処理速度
6	記憶域テスト	プログラムで使用する記憶領域のサイズ
7	構成テスト	種々のシステム構成下での動作
8	互換性・変換テスト	既存システムとの互換性や変換手順の妥当性
9	設置テスト	システム設置手続きの妥当性
10	信頼性テスト	要求MTBF
11	回復テスト	障害からの回復機能の動作
12	サービス性テスト	保守機能の動作
13	文書テスト	文書の正確性と明瞭性
14	手続きテスト	オペレータ手続きの妥当性

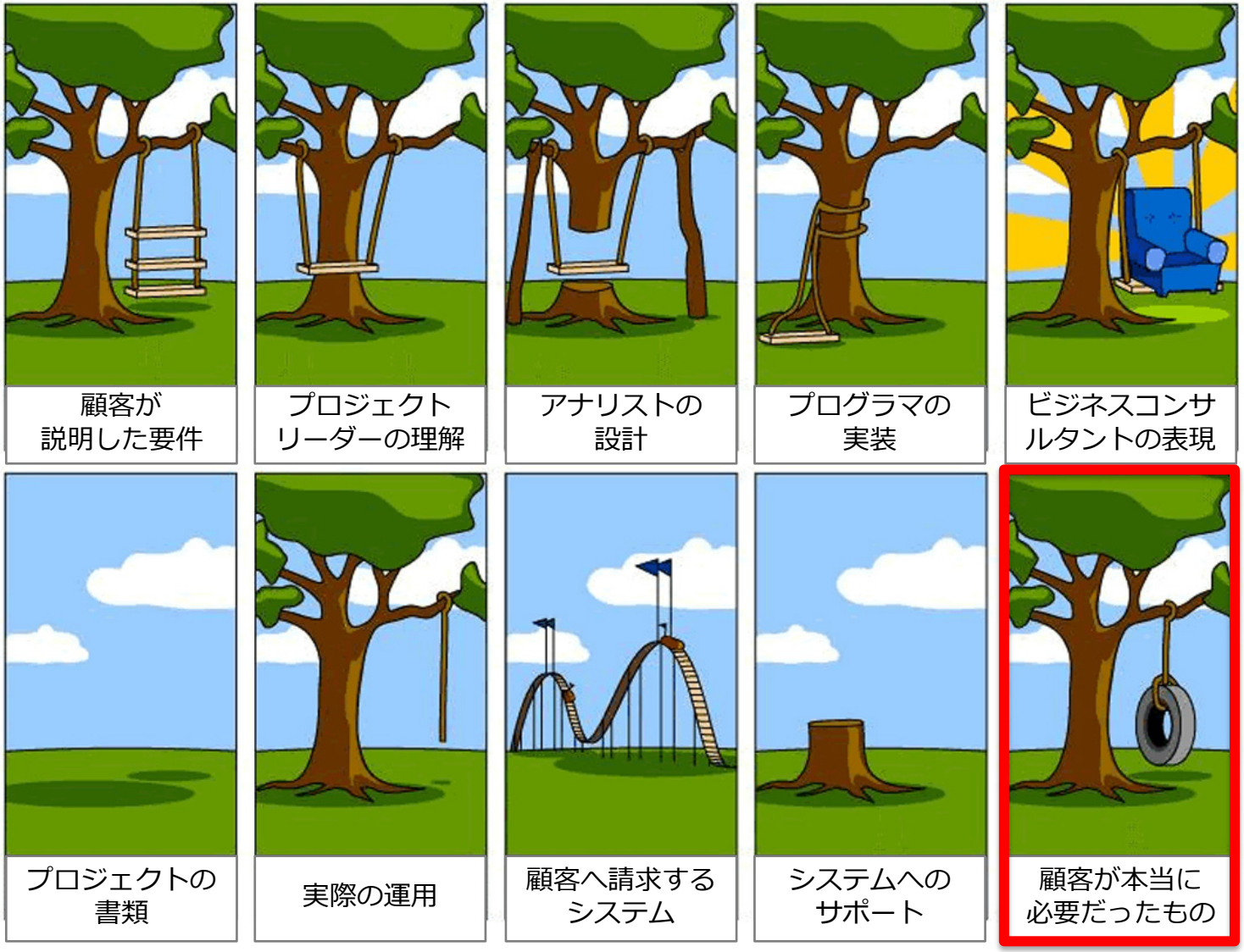
テストの評価

- **テストそのものの妥当性**
  - テストの品質を測る基準
    - テストをどの程度行えば、どのくらいの品質を保証できるか



- 基準
  1. **網羅性基準**
     - テストに使用したテストケースが、全テスト空間のうちのどのくらいを網羅しているかという基準
  2. **欠陥除去基準**
     - プログラムに内在するエラーの総数のうち、どのくらいのエラーが除去できたかという基準
  3. **運用的基準**
     - 運用環境下において、エラーが発生するまでの基準

ソフトウェア工学の必要性



ソフトウェア工学 参考書

- **ソフトウェア開発 (改訂2版)**
  - 著者：小泉 寿男, 辻 秀一, 吉田 幸二, 中島 毅
  - 出版社: オーム社; 改訂2版 (2015/12/25)
  - ISBN-10: 4274218414

- **ソフトウェア工学入門**
  - 著者：鯵坂 恒夫
  - 出版社: サイエンス社 (2008/3/1)
  - ISBN-10: 4781911935

データ収集システム化技術
UML

UMLとは

- **UML　(Unified Modeling Language)**
  - 統一モデリング言語
  - オブジェクト指向モデルを記述する言語
    - オブジェクト指向システムの分析・設計時に考えを具象化する道具
    - 技術者間のコミュニケーションの道具

- モデリング言語
  - モデルを作成するための表記法
  - 書式（モデル要素の描き方）
    ＋意味（モデル要素が表す内容）

UMLの特徴

- 高い表現力
  - 13種類のダイアグラムによる対象（課題）の可視化
- ダイアグラム間の強い相互関係
  - 対象（課題）の正しい表現および理解
- 世界標準言語
  - 自然言語でのコミュニケーションギャップの解消
- 実装言語への高い対応関係
  - オブジェクト指向言語でのシステム開発の容易性

UMLの歴史

- UML誕生前の表記法＋開発方法論
  - Booch法（G.Booch）
  - OMT（J.Rumbaugh）
  - OOSE（I.Jacobson）　など
- UMLの歴史
  - 1994　OMTの第二版（J.Rumbaugh + G.Booch）
  - 1995　UMLの初期版（J.R+G.B+I.Jacobson）
  - **1997　UML 1.0**
  - **1997　UML 1.1 OMGで承認　標準化**
    - OMG: Object Management Group

    :
  - 2003　UML 1.5
  - **2005　UML 2.0**

    :
  - 2010　UML 2.3

  :

UMLの仕組み

- **ビュー （View）**
  - システムをモデル化してできるもの
    - さまざまな側面から表現するため複数個存在
  - 複数のダイアグラムから構成
- **ダイアグラム（Diagram）**
  - ビューの内容を表すグラフ
    - UMLでは13のダイアグラムを利用してビューを表す
- **モデル要素（Model Element）**
  - ダイアグラムで利用される概念
    - UMLではクラス、オブジェクト、メッセージ、モデル要素間の関係など
- **その他**
  - コメントや情報の挿入などの一般的な機能
  - 拡張への仕組み

UML 2.x – 13 Diagrams

- 構造図 (Structural Diagrams)
  - 論理的な図
    - **クラス図 (Class Diagram)**
    - **オブジェクト図 (Object Diagram)**
    - **パッケージ図 (Package Diagram)**
  - 物理的な図
    - **コンポーネント図 (Component Diagram)**
    - **コンポジット構造図 (Composite Structure Diagram)**
    - **配置図 (Deployment Diagram)**
- 振る舞い図 (Behavioral Diagrams)
    - **ユースケース図 (Use Case Diagram)**
    - **ステートマシン図 (State Machine Diagram)**
    - **アクティビティ図 (Activity Diagram)**
  - 相互作用図 (Interaction Diagrams)
    - **シーケンス図 (Sequence Diagram)**
    - **コミュニケーション図 (Communication Diagram)**
    - **相互作用概要図 (Interaction Overview Diagram)**
    - **タイミング図 (Timing Diagram)**

構造図:論理的な図

- **クラス図 (Class Diagram)**
  - クラスに代表される要素の静的構造を記述
  - クラスおよびクラス間の関係を記述
- **オブジェクト図 (Object Diagram)**
  - クラスのインスタンスであるオブジェクトを記述
  - オブジェクトおよびオブジェクトの関係を記述
- **パッケージ図 (Package Diagram)**
  - モデル要素のグループ化を記述
  - 全体の概要を記述

構造図:物理的な図

- **コンポーネント図 (Component Diagram)**
  - ソフトウェア部品のコンポーネントを記述
- **コンポジット構造図 (Composite Structure Diagram)**
  - コンポーネントの内部構造や、クラスの全体と部分の関係を記述
- **配置図 (Deployment Diagram)**
  - システムの配置構造を記述
  - ハードウェアとソフトウェアの対応関係を記述

振る舞い図

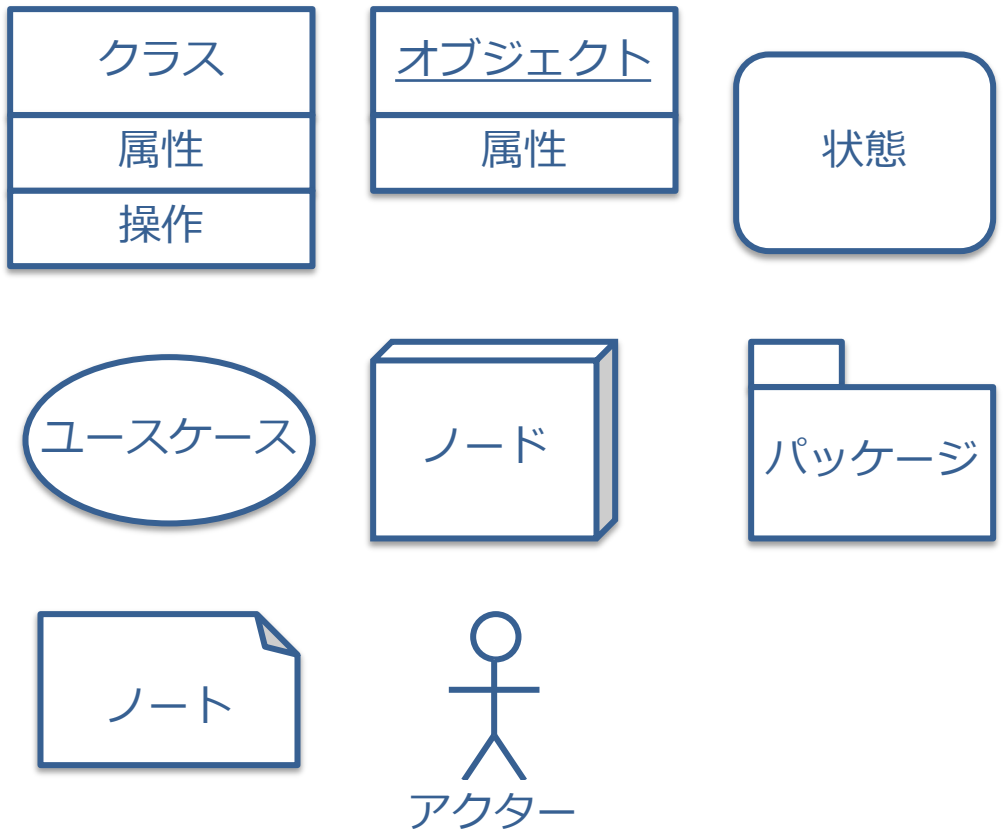
- **ユースケース図 (Use Case Diagram)**
  - 機能要求を表すユースケースによってシステム全体を記述
  - ユーザの役割（ロール）や外部システムとの接続を記述
- **ステートマシン図 (State Machine Diagram)**
  - 振る舞いステートマシン図で一つのクラスの状態の変化を記述
  - プロトコルステートマシン図でプロトコルを記述
- **アクティビティ図 (Activity Diagram)**
  - ビジネスプロセスの業務の手順やプログラムの処理などを記述

相互作用図

- **シーケンス図 (Sequence Diagram)**
  - オブジェクトの相互作用を時系列に沿って記述
  - システム間の相互関係を時系列に沿って記述
- **コミュニケーション図 (Communication Diagram)**
  - 相互関係を重視したオブジェクトの相互作用を記述
- **相互作用概要図 (Interaction Overview Diagram)**
  - 相互作用の概略を記述
- **タイミング図 (Timing Diagram)**
  - オブジェクトの状態を重視したオブジェクト間の相互作用を記述

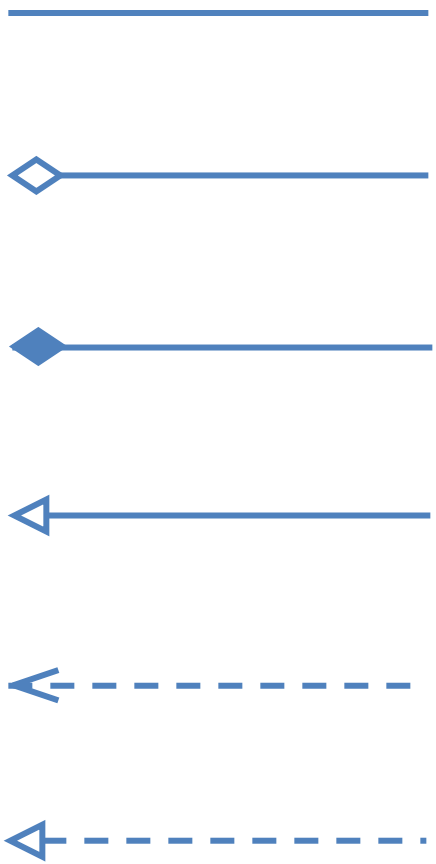
モデル要素：図形シンボル

- **図形シンボル**
  - クラス
  - オブジェクト
  - 状態
  - ユースケース
  - ノード
  - パッケージ
  - 注釈
  - アクター
  - ・・・



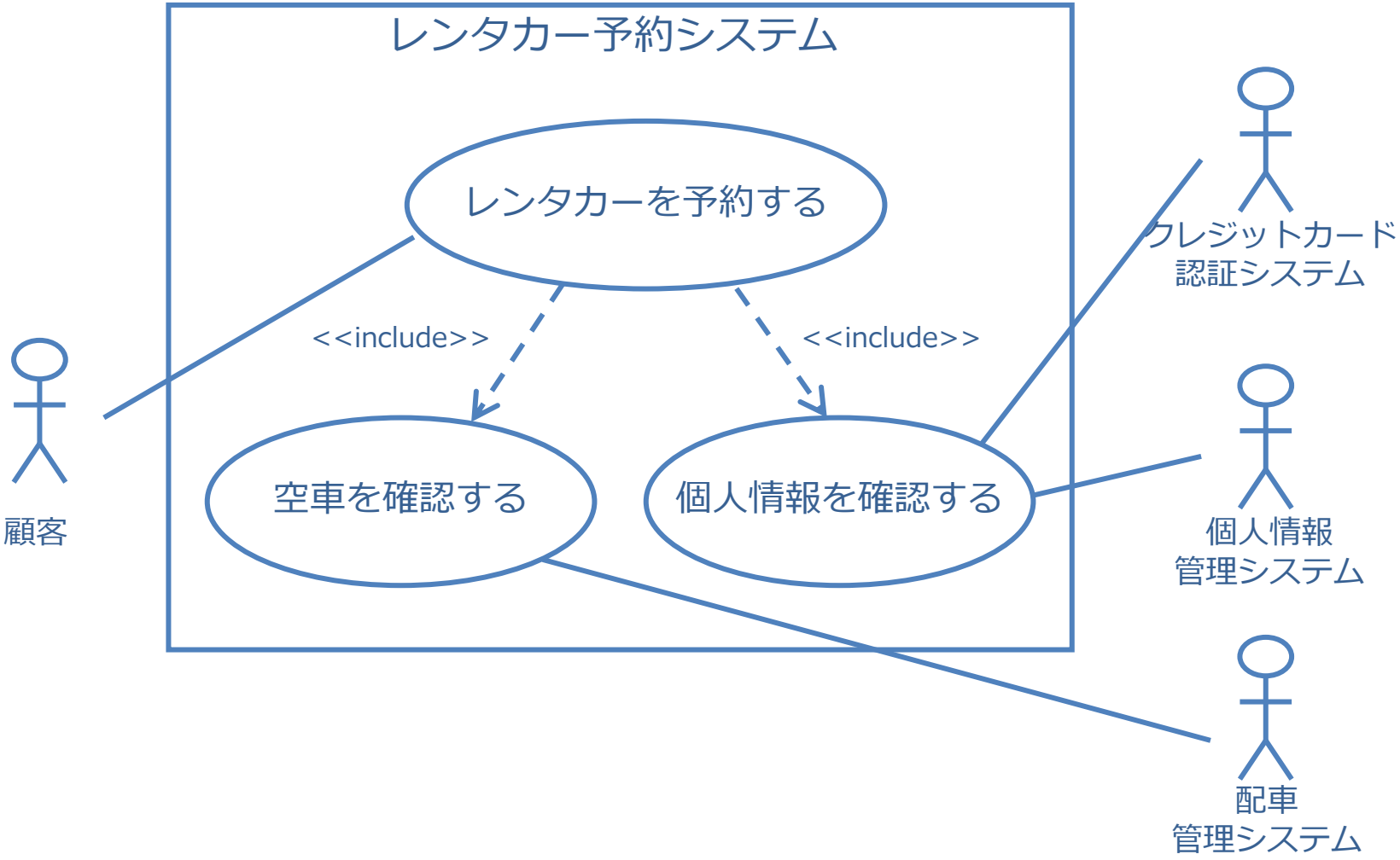
モデル要素：関係シンボル

- **関係シンボル**
  - 関連　（Association）
  - 集約　（Aggregation）
  - コンポジション　（Composition）
  - 汎化　（Generalization）
  - 依存　（Dependency）
  - 実現　（Realization）



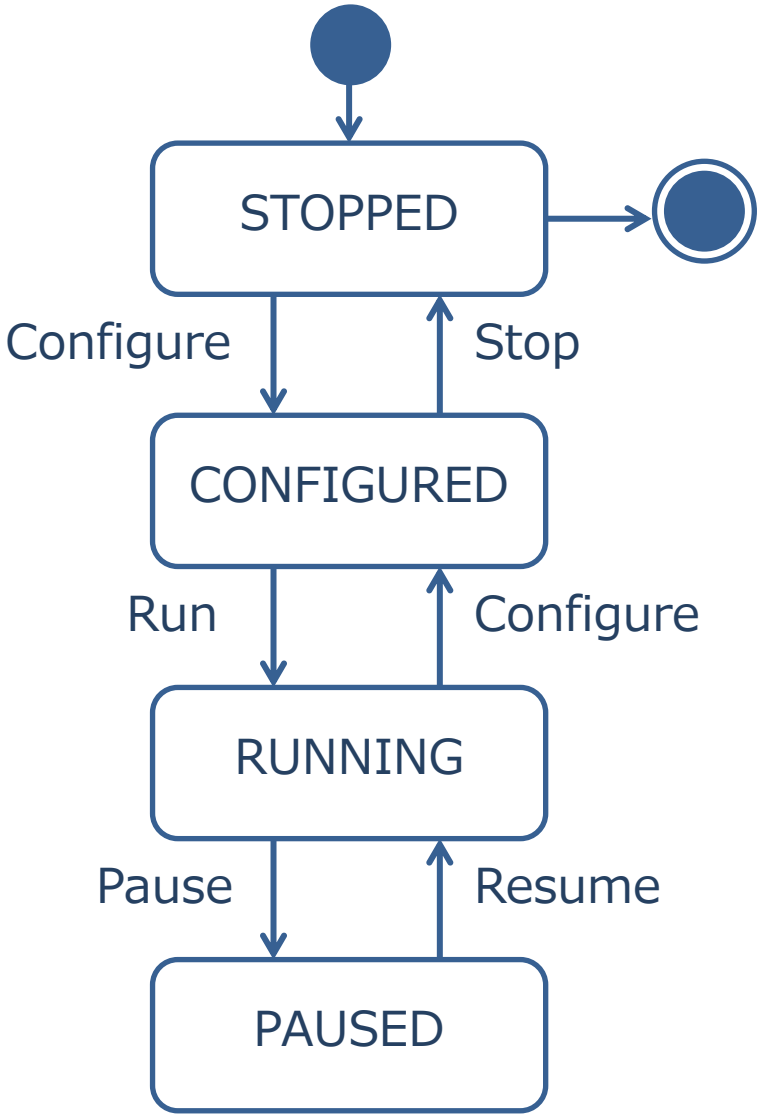
ユースケース図の例

- レンタカー予約システム

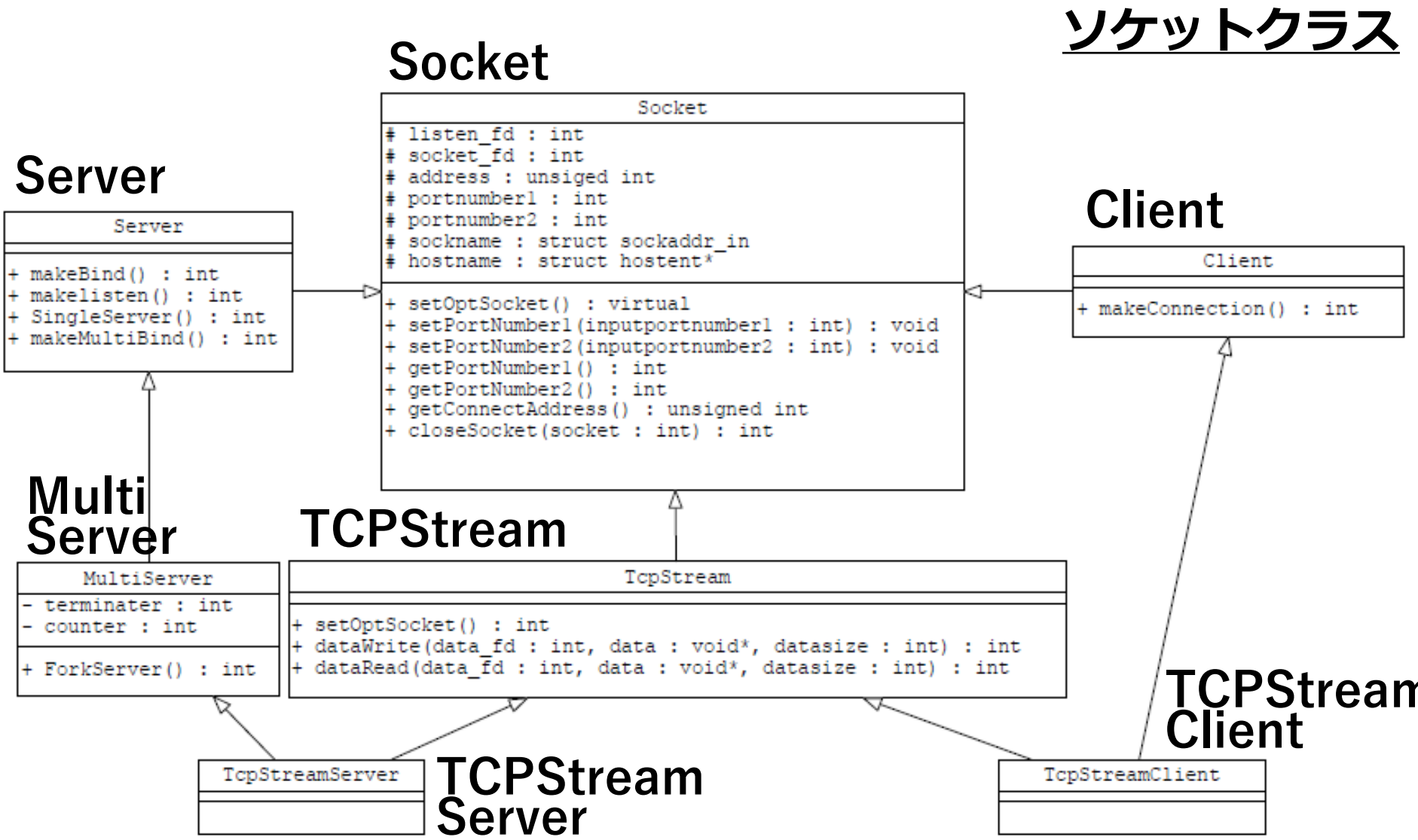


ステートマシン図の例

- 状態（ステート）
  - STOPPED
  - CONFIGURED
  - RUNNING
  - PAUSED
- イベント
  - Configure
  - Run
  - Pause
  - Resume
  - Configure
  - Stop



クラス図の例


ソケットクラス
Socket
Socket
listen_fd : int
socket_fd : int
address : unsiged int
portnumber1 : int
portnumber2 : int
sockname : struct sockaddr_in
hostname : struct hostent*
+ setOptSocket() : virtual
+ setPortNumber1(inputportnumber1 : int) : void
+ setPortNumber2(inputportnumber2 : int) : void
+ getPortNumber1() : int
+ getPortNumber2() : int
+ getConnectAddress() : unsigned int
+ closeSocket(socket : int) : int
Server
Server
+ makeBind() : int
+ makelisten() : int
+ SingleServer() : int
+ makeMultiBind() : int
Client
Client
+ makeConnection() : int
Multi Server
MultiServer
- terminater : int
- counter : int
+ ForkServer() : int
TCPStream
TcpStream
+ setOptSocket() : int
+ dataWrite(data_fd : int, data : void*, datasize : int) : int
+ dataRead(data_fd : int, data : void*, datasize : int) : int
TcpStreamServer
TCPStream Server
TcpStreamClient
TCPStream Client

UML モデリングツール

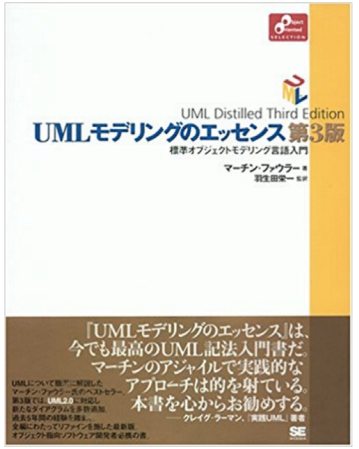
- **Modelio（無償）**
  - https://www.modelio.org/
- **StarUML（無償）**
  - http://staruml.io/
- **Papyrus（無償）**
  - http://www.eclipse.org/papyrus/

- **astah* community（有償）**
  - https://astah.change-vision.com/ja/product/astah-uml.html
- **Microsoft Visio（有償）**
  - https://products.office.com/ja-jp/visio/
- **Enterprise Architect（有償）**
  - http://www.sparxsystems.jp/products/EA/ea.htm

UML 参考書

- **ダイアグラム別UML徹底活用 第2版**
  - 著者：井上 樹
  - 出版社: 翔泳社; 第2版 (2011/2/25)
  - ISBN-10: 4798118443

- **UML モデリングのエッセンス 第3版**
  - 著者：マーチン・ファウラー
    羽生田 栄一（翻訳）
  - 出版社: 翔泳社; 第3版 (2005/6/16)
  - ISBN-10: 4798107956



まとめ

- DAQシステムはソフトウェアであることから、ソフトウェア工学の知識を活用
  - 品質の高いシステム構築をするために、一般的なソフトウェア開発の知識を活用
  - 力技でも動くシステムはできるが、その品質は高くないことが多い
- 情報システムを構築するために必要な技術を活用

- ソフトウェア工学の知識
  - 開発プロセスモデル
  - ソフトウェアの設計と実装
  - ソフトウェアのテスト
  - テストケース設計技法
- UML（統一モデリング言語）
  - オブジェクト指向システムの分析・設計時に考えを具現化する言語
    - ユースケース図・ステートマシン図・クラス図・シーケンス図