ソフトウェア工学

吉田 則裕

立命館大学 情報理工学部

2025年度 春学期

X: @NorihiroYoshida

考えて欲しいこと

- ソフトウェア開発は，どのようなステップで行われるのだろうか？
  - プログラミング
  - テスト
  - 顧客から何を作るか聞く
  - 設計？
  - 一度出来上がったあと修正？
  - ハードウェアならどういうステップになるか？
- 顧客や開発メンバと成果物についての認識を共有するためにはどうしたら良いのだろうか？
- どっちのソフトウェア開発が効率的だろうか？
  - 基本的な部分の開発を計画し，成果物ができてから少しずつ拡張
  - 計画を詳細に立てて，一度に開発

開発プロセスを学ぶ意義

- 企業で求められる人材の必要条件一つは

「そのプロジェクトで決められた開発プロセスに従って動ける人であること」

- 開発プロセスに従って動けるようになったら，開発プロセスを策定できるようになって欲しい
  - 標準的な開発プロセスを参考に，プロジェクトの事情にあわせて，開発プロセスを策定

- 開発プロセスを選べるようになったら，開発プロセスを改善できるようになってほしい.
  - プロジェクトの事情にあわせて，開発プロセスを改善

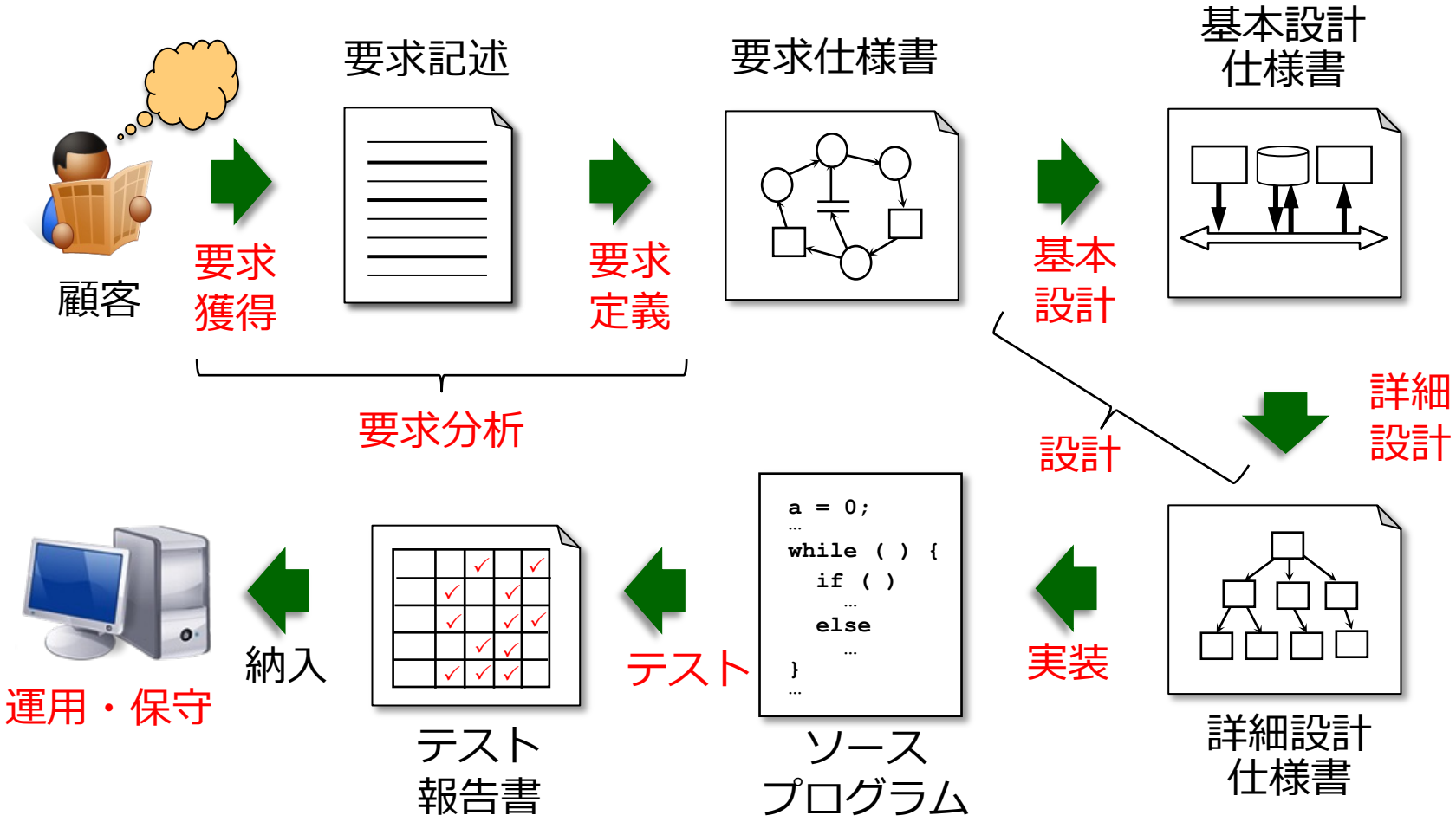
第2回: ソフトウェア開発モデル

- ソフトウェア開発工程(ソフトウェアプロセス)
  - 要求分析 ⇒ 基本設計 ⇒ 詳細設計
    ⇒ 実装 ⇒ テスト ⇒ 運用・保守

- ソフトウェアプロセスモデル
  - ウォーターフォールモデル
  - スパイラルモデル
  - 進化型プロトタイプモデル
    - イテラティブ開発とインクリメンタル開発
  - アジャイルプロセス

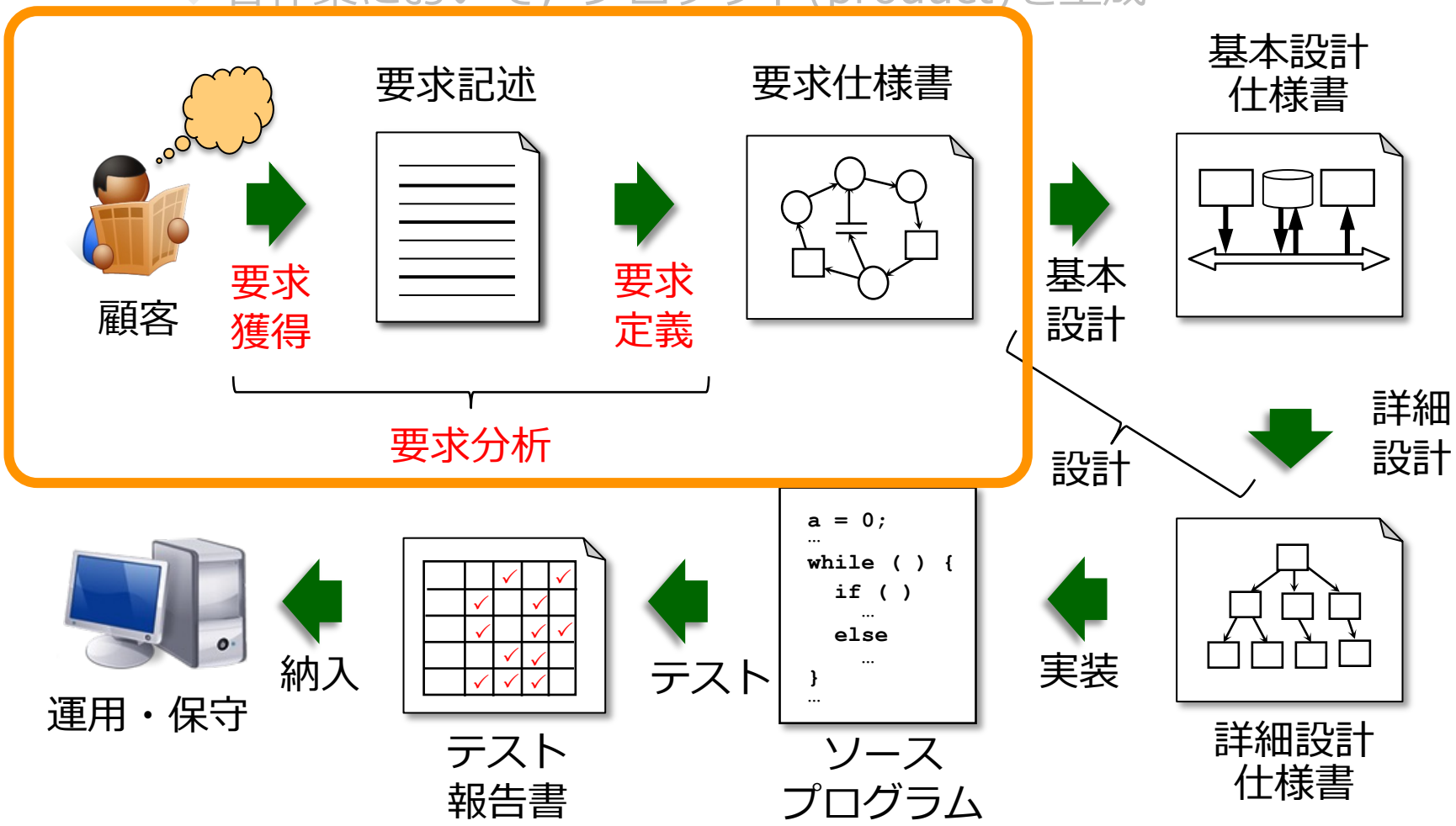
ソフトウェア開発工程とは

- ソフトウェアの開発をどのように進めるかを規定したもの
  - 開発における作業: 開発プロセス(process)
  - 各作業において，プロダクト(product)を生成



ソフトウェア開発工程とは

- ソフトウェアの開発をどのように進めるかを規定したもの
  - 開発における作業: 開発プロセス(process)
  - 各作業において，プロダクト(product)を生成

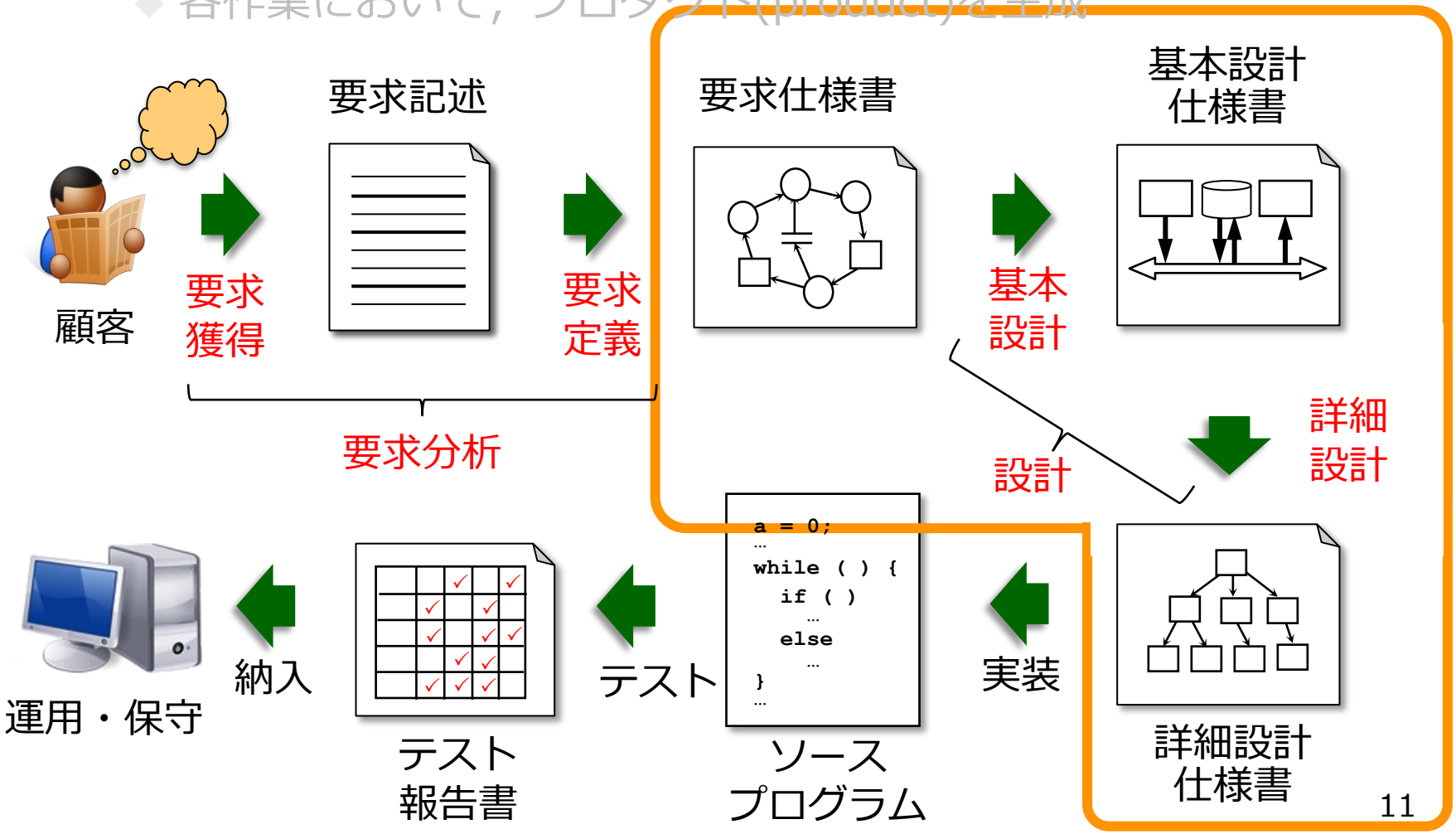


ソフトウェア開発プロセス(要求分析)

- 要求分析(requirements analysis)
  - どのようなソフトウェアを作るのかを明確にして，要求仕様を定める工程
- 要求獲得
  - 利用者が要求するシステムを整理し，自然言語で文書化
    - 要求記述
    - システム要求書(system requirements)
- 要求定義
  - どのようなシステムを作成するのかを決定
  - システム要求書を分析し，開発するシステムを形式的に文書化
    - 要求仕様書(requirements specification)
  - 分析者(analyst)が実施

ソフトウェア開発工程とは

- ソフトウェアの開発をどのように進めるかを規定したもの
  - 開発における作業: 開発プロセス(process)
  - 各作業において，プロダクト(product)を生成



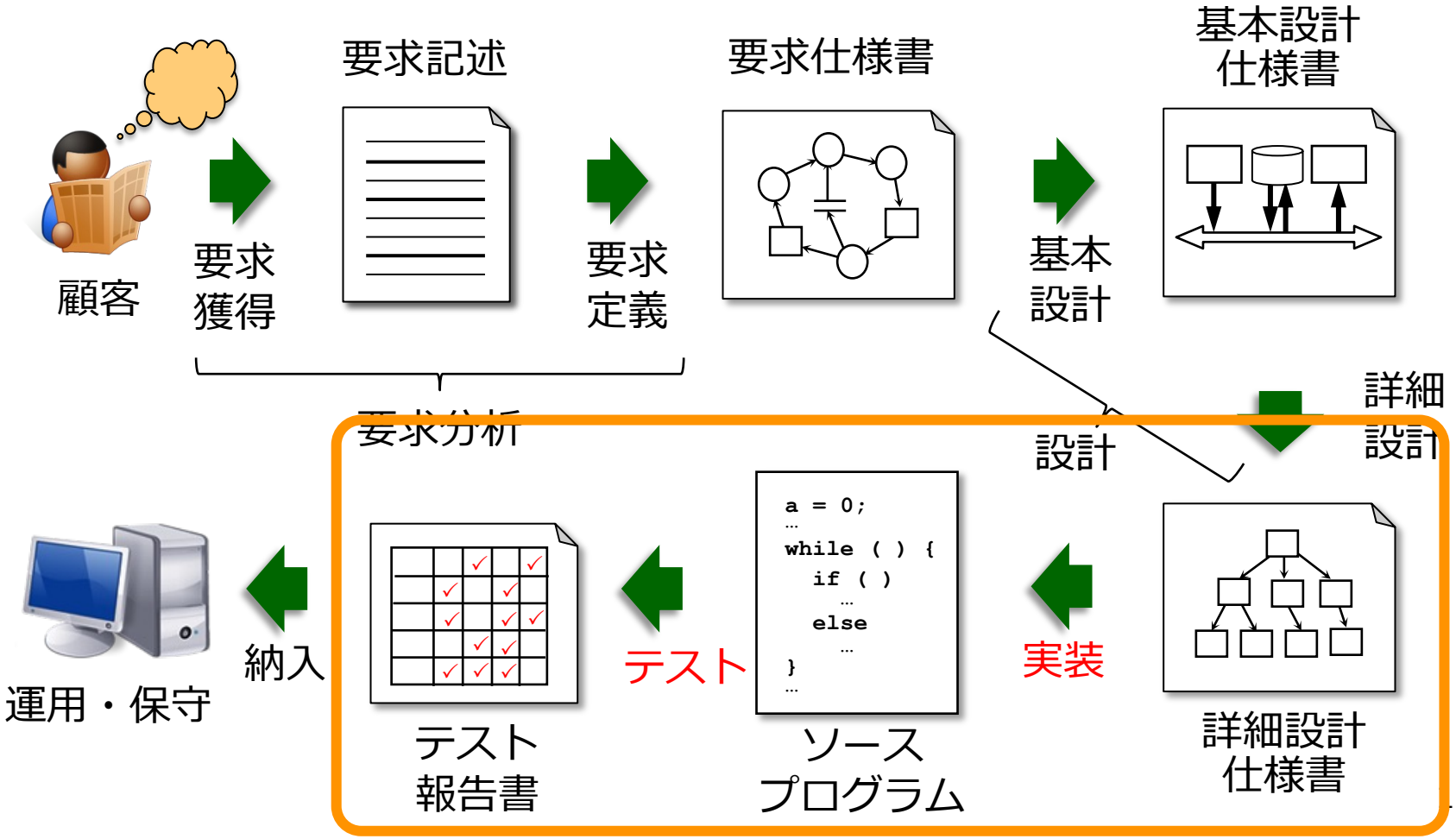
ソフトウェア開発プロセス(設計)

- 基本設計(basic design)
  - システム設計(system design)，外部設計(external design)ともいう
  - 外部から見たシステム全体をどのように作成するのかを決定
  - ソフトウェアアーキテクチャやユーザインタフェースを決定
    - 基本設計仕様書
  - 設計者(designer)が実施

- 詳細設計(detail design)
  - プログラム設計(program design)，内部設計(internal design)ともいう
  - 個々のモジュールのアルゴリズムやデータ構造を決定
    - 詳細設計仕様書
  - プログラマ(programmer)が実施

ソフトウェア開発工程とは

- ソフトウェアの開発をどのように進めるかを規定したもの
  - 開発における作業: 開発プロセス(process)
  - 各作業において，プロダクト(product)を生成



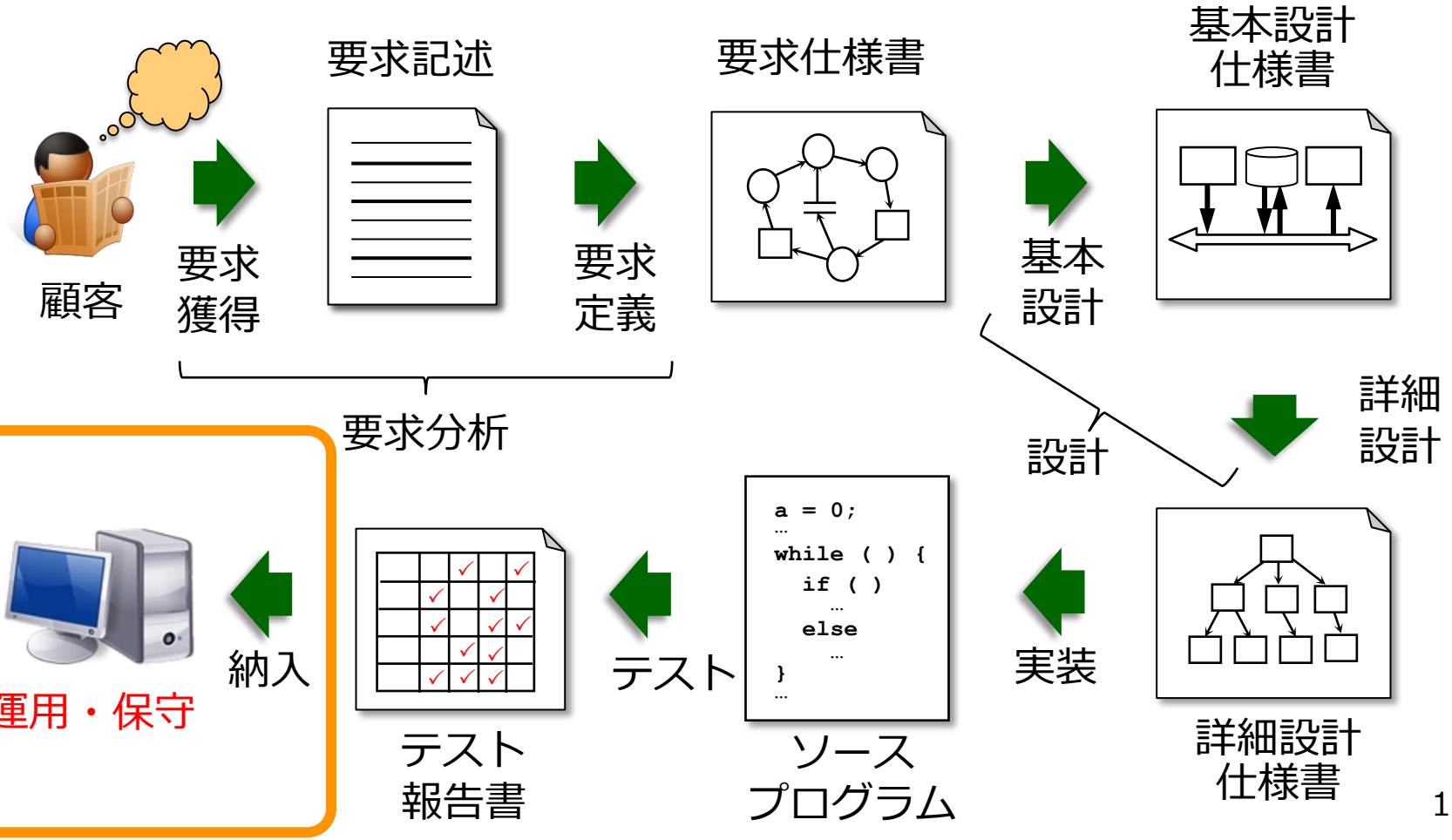
ソフトウェア開発プロセス(実装，テスト)

- 実装(implementation)
  - コーディング(coding)ともいう
  - プログラム設計仕様をプログラムに変換
  - 具体的なプログラミング言語(program language)による記述
    - ソースプログラム(source program)
  - プログラマが実施

- テスト(testing)
  - 仕様書どおりにプログラムが動作するかどうかを検査
    - テスト報告書(test report)
  - 試験者(tester)が実施
    - ソフトウェア作成と独立の組織を用意
  - デバック(debug)
    - エラー(error)の修正は設計者やプログラマが実施

ソフトウェア開発工程とは

- ソフトウェアの開発をどのように進めるかを規定したもの
  - 開発における作業: 開発プロセス(process)
  - 各作業において，プロダクト(product)を生成



ソフトウェア開発プロセス(運用・保守)

- 運用・保守(operation/maintenance)
  - 納入後のソフトウェアを維持・管理
  - 運用段階で検出された故障(残存エラー)の修正
  - 変更要求への対応
    - 新機能の追加
    - 既存機能の変更
    - 新しい環境への適合
  - 保守者(customer engineer)が実施

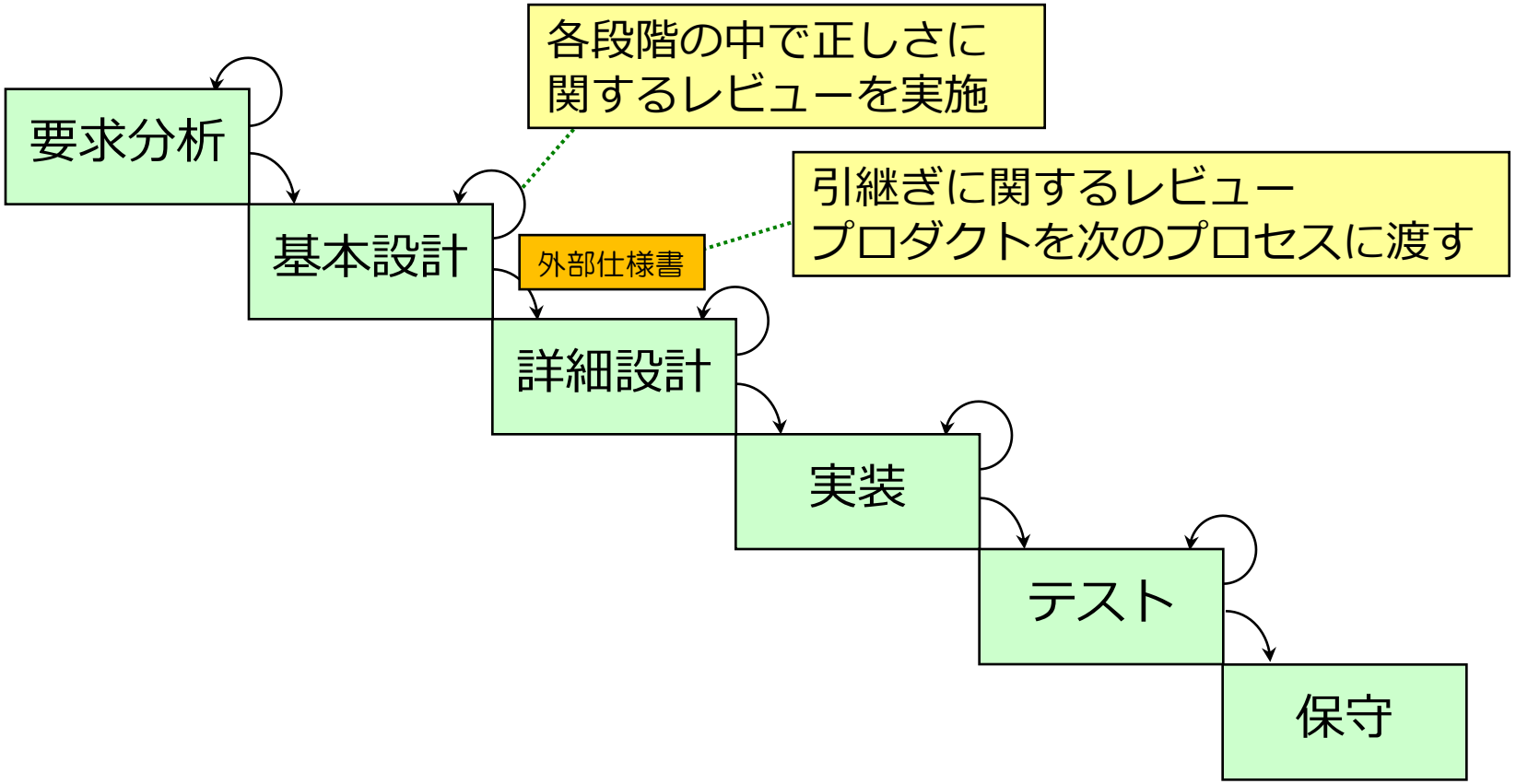
- 進化(evolution)
  - ソフトウェアの永続的な発展
  - ソフトウェア変化の積極的な取り込み
    - ソフトウェアは本質的に変化(修正，変更)し続ける
    - はじめから完全なソフトウェアを構築するのは困難

ソフトウェアプロセスモデル

- ソフトウェアプロセスモデル(process model)
  - ライフサイクル(life cycle)モデルともいう
- 推移
  - 1960年代: 流れ図(flowchart)を利用
    - 開発方法論なし
  - 1970年代: 構造的なソフトウェア開発
    - ウォータフォールモデル
  - 1980年代: ソフトウェアライフサイクル有害説
    - 新しいソフトウェア開発モデルの必要性
  - 進化型(成長型，発展型)プロセスモデルの登場
    - 使い捨て型プロトタイピング(throwaway prototyping)
    - スパイラルモデル(spiral model)
    - 進化型プロトタイピング(evolutionary prototyping)
    - アジャイルプロセスモデル(agile process model)

ウォーターフォールモデル

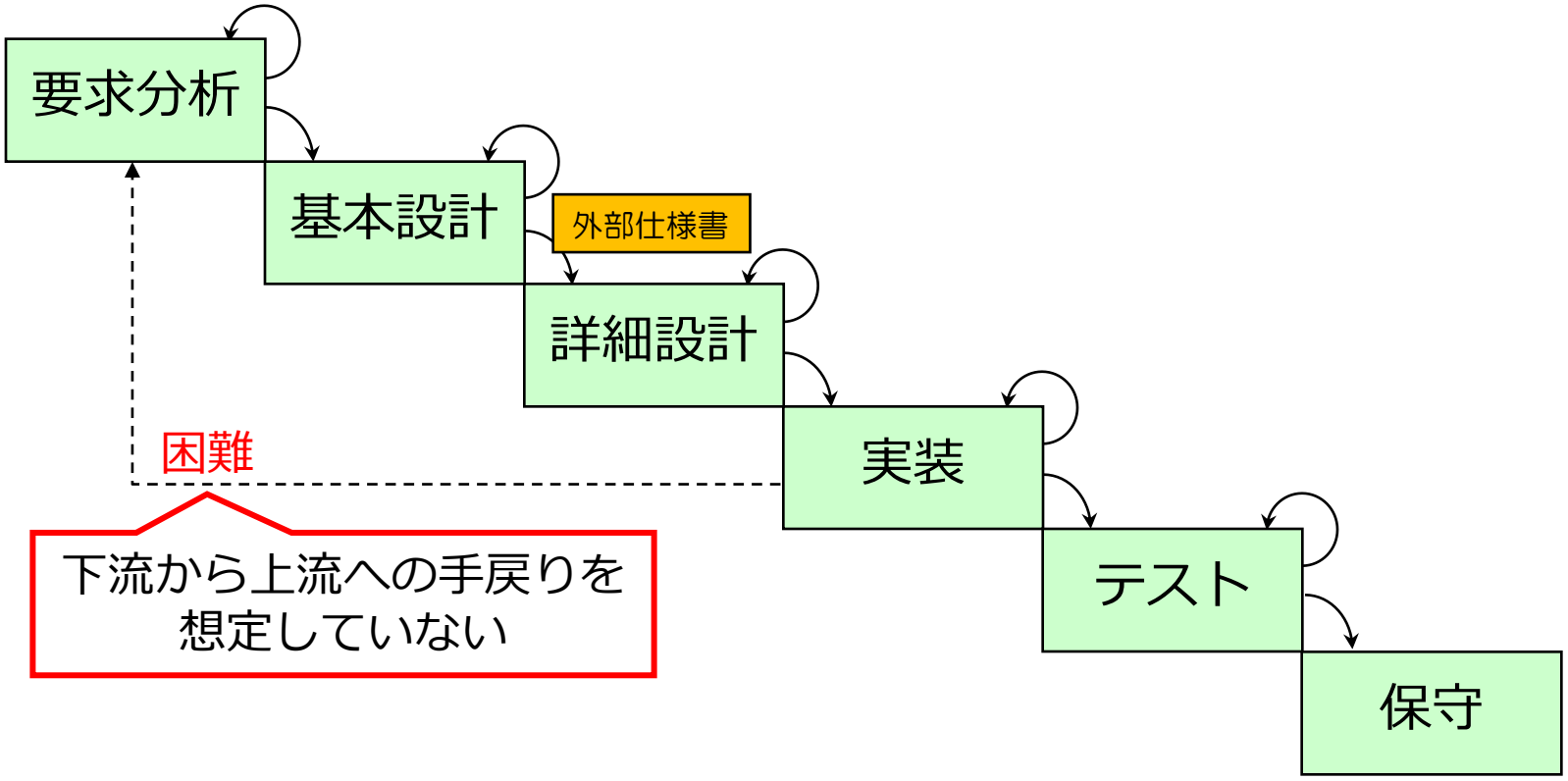
- トップダウンな開発プロセス: 滝(waterfall)



利点: 工数見積もりや進捗管理が容易(開発計画が立てやすい)

ウォーターフォールモデル

- トップダウンな開発プロセス: 滝(waterfall)



利点: 工数見積もりや進捗管理が容易(開発計画が立てやすい)

欠点: 仕様変更に弱い，誤り発見の遅れ，修正コストの増大

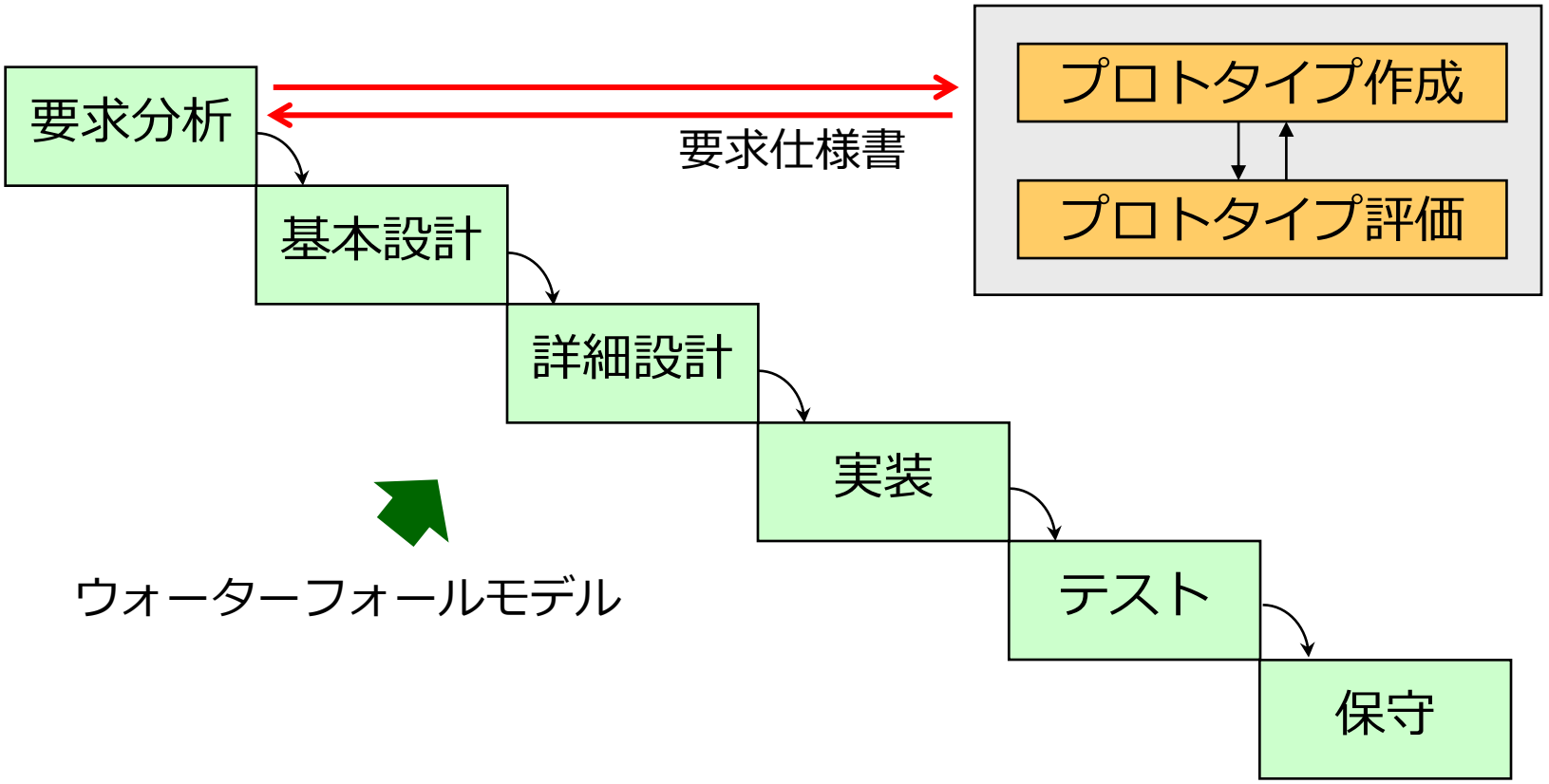
要求仕様が明確な大規模ソフトウェアの開発に向いている

ウォーターフォールが使用される場面

- 社会インフラを支える大規模なソフトウェアは，ウォーターフォールが多い
  - 金融系
  - 電力系
  - 鉄道系
- 組込みシステム向けソフトウェアのうち，高い信頼性が求められるものは，ウォーターフォールが多い

使い捨てプロトタイピング

- システム設計時にプロトタイプ(試供品: prototype)を構築
  - 問題点の早期発見と要求のあいまいさの解消
    - プロトタイプに基づき要求仕様書を作成
  - 大幅な手戻りを減少



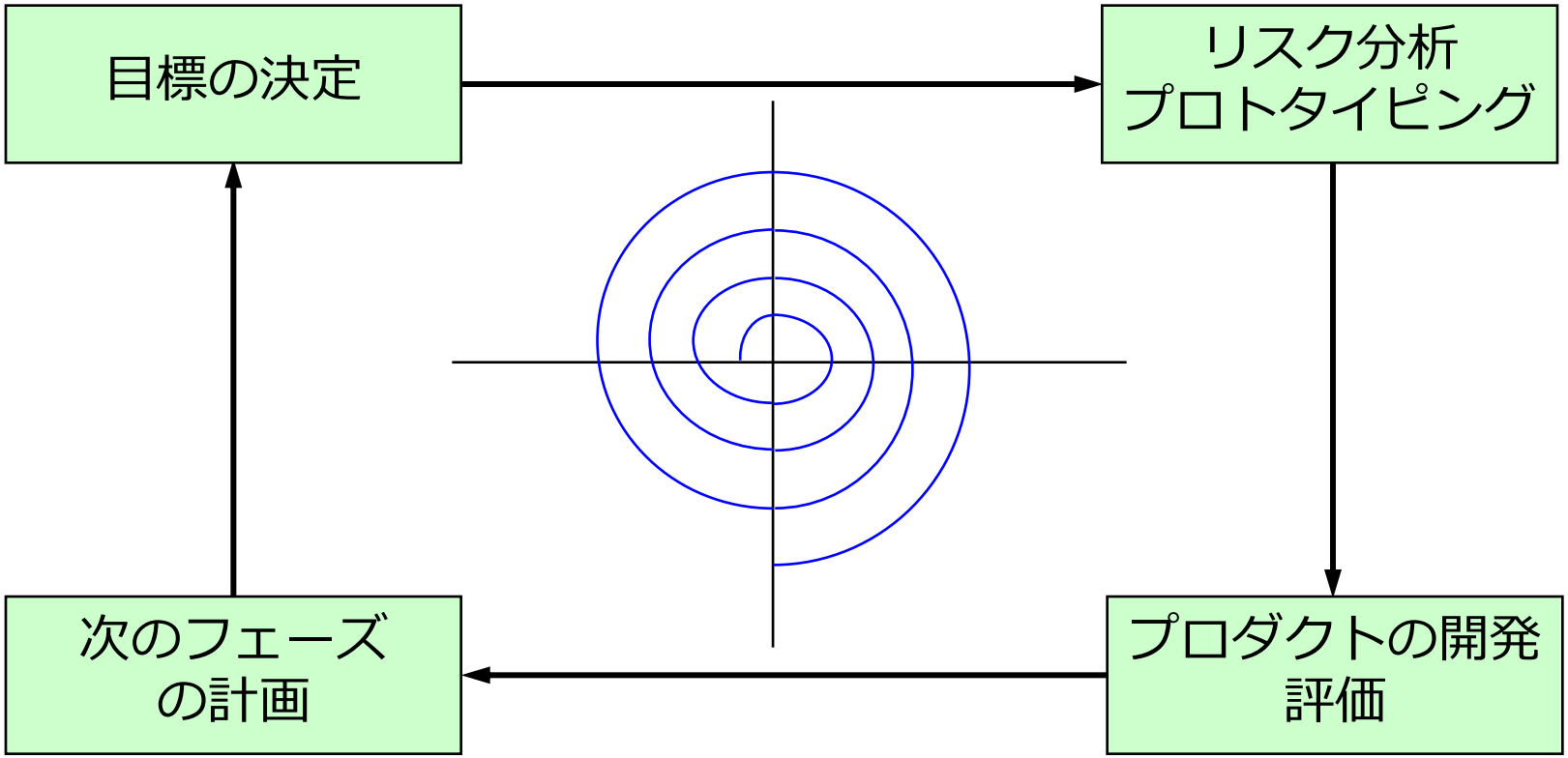
使い捨てプロトタイピングが使用される場面

GUIを伴うシステムはプロトタイプを開発することが多い

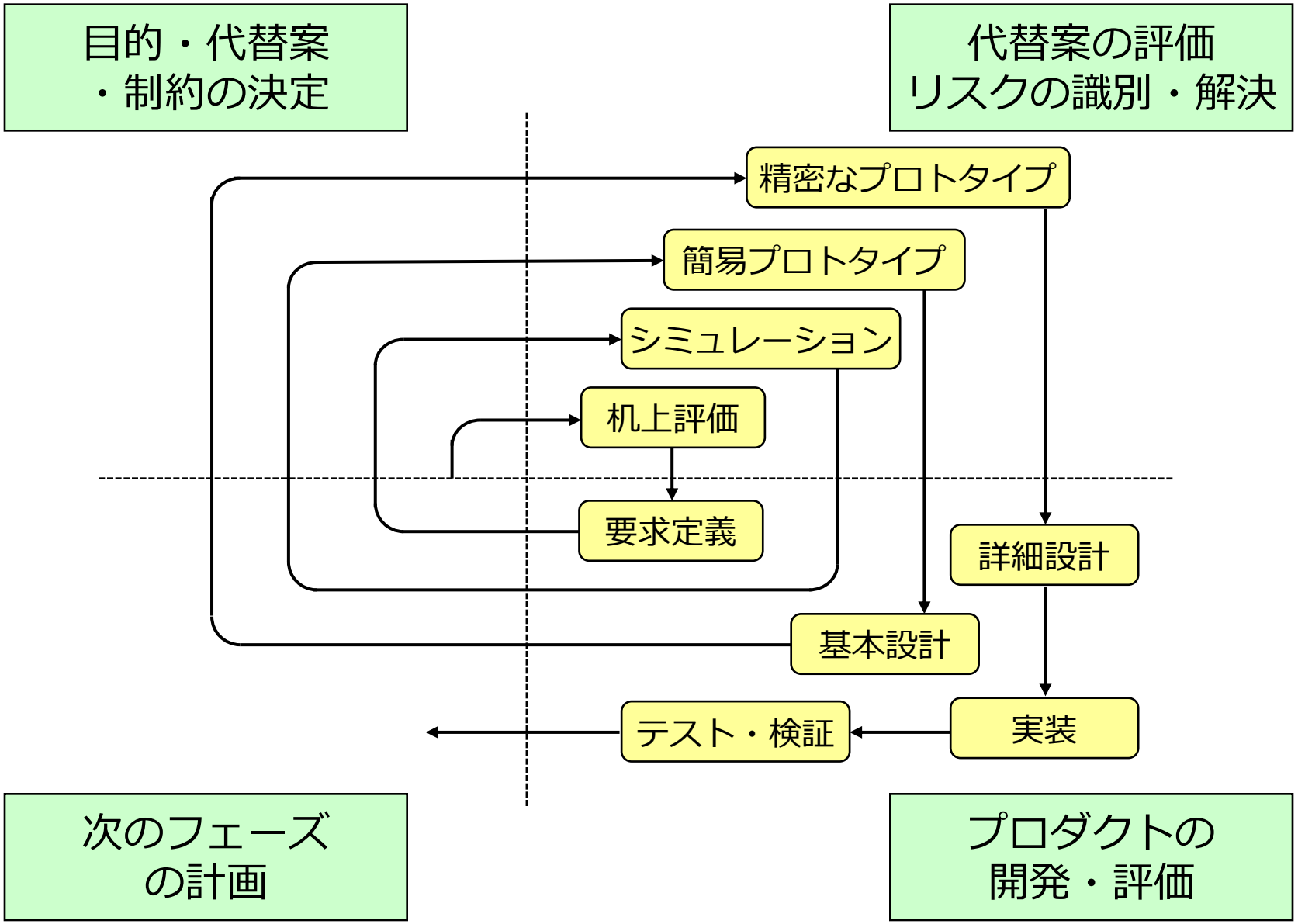
- 顧客に一度使ってもらった方が誤解を生みにくい

スパイラルモデル

- プロトタイプなどで利用者からのフィードバックに対応しながら，徐々にシステムを作成
  - リスク(開発の失敗)を分析することで，リスクを軽減

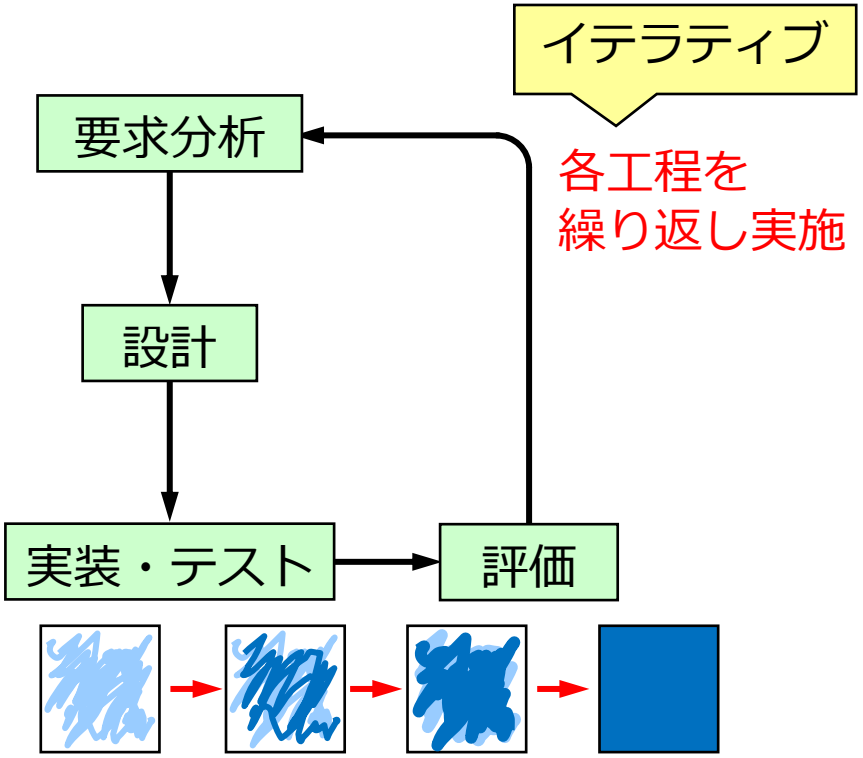


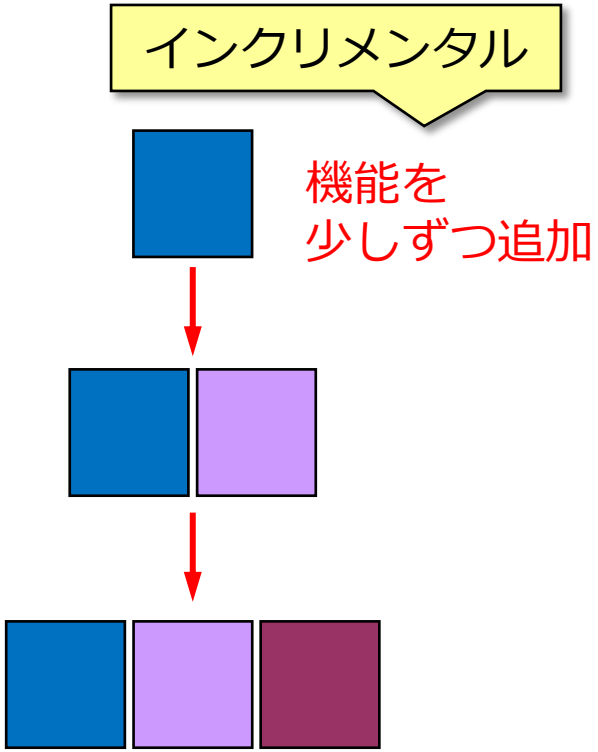
スパイラルモデル (ウォーターフォールモデルへの適用)



進化型プロトタイピング

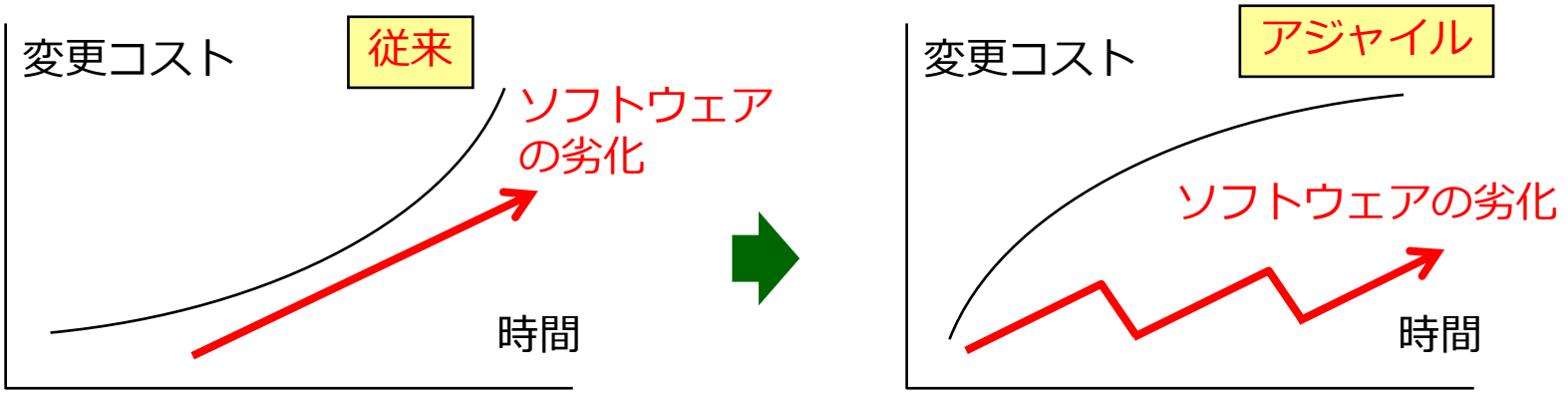
- プロトタイプを徐々に変更し，そのまま完成品として利用
  - 機能が明確かつ不確定要素が少ない部分を優先して開発
  - 以下の2つに分類される.
    - イテラティブ(iterative)開発
    - インクリメンタル(incremental)開発





アジャイルプロセスモデル

- 変化に迅速に対応するため，開発対象を多数の小さな機能に分割して，各機能を短期間で開発する手法の総称
  - 軽量(lightweight)プロセスともいう
    - XP(extreme programming), Scrumなど
- 1つの反復(iteration)で1機能を実現
  - 究極の反復型開発(進化型プロトタイピング)
- 実行可能なソフトウェアを随時提供
  - CI (Continuous Integration)
- ソフトウェアの劣化を抑えることで，変更コストを下げる



アジャイルプロセスモデルのポイント

- 狙い：プロダクトをコントロール可能にしておきたい
  - 大きな手戻りを避けたい
  - 簡単に修正できる状態を維持したい
  - 開発者全員がソフトウェアを状態を把握できるようにする.
- 小さい単位で開発する
  - 最初から大きいものを作ろうとしない.
- できるだけ早く実行可能な状態にし，それを維持する．機能追加は少しずつする.
  - 実行できなくなるような，大きな修正をしない.
  - 開発全員が常にソフトウェアの状態を把握できるようにする.

いきなり大きいものを作ろうとすると・・・

- 大きな計画が必要になる.
- 計画の修正も大変になる.
- 顧客や開発メンバと成果物に対する認識を共有しづらくなる.
  → 顧客に成果物を提出したら「思ってたものと違う」と言われるが，もうやりなおすことができないいったことが起きやすくなる.

  → 開発メンバーと分担して開発したが，いざ組み合わせようとするとうまくいかないといったことが起きやすくなる.

実行可能な状態にしておかないと・・・

大きな機能追加を行うおうとして，実行できない状態にしてしまう

→ 開発メンバが，ソフトウェアの現状を理解できなくなる.

→ 実行できる状態に戻す決断をしたときに，かなり以前の状態まで戻すことになり，大きな手戻りになる.

アジャイルが採用される場面

- グローバル開発
  - OSS
  - Google，Microsoft
- アプリ開発
  - スマフォゲームなど

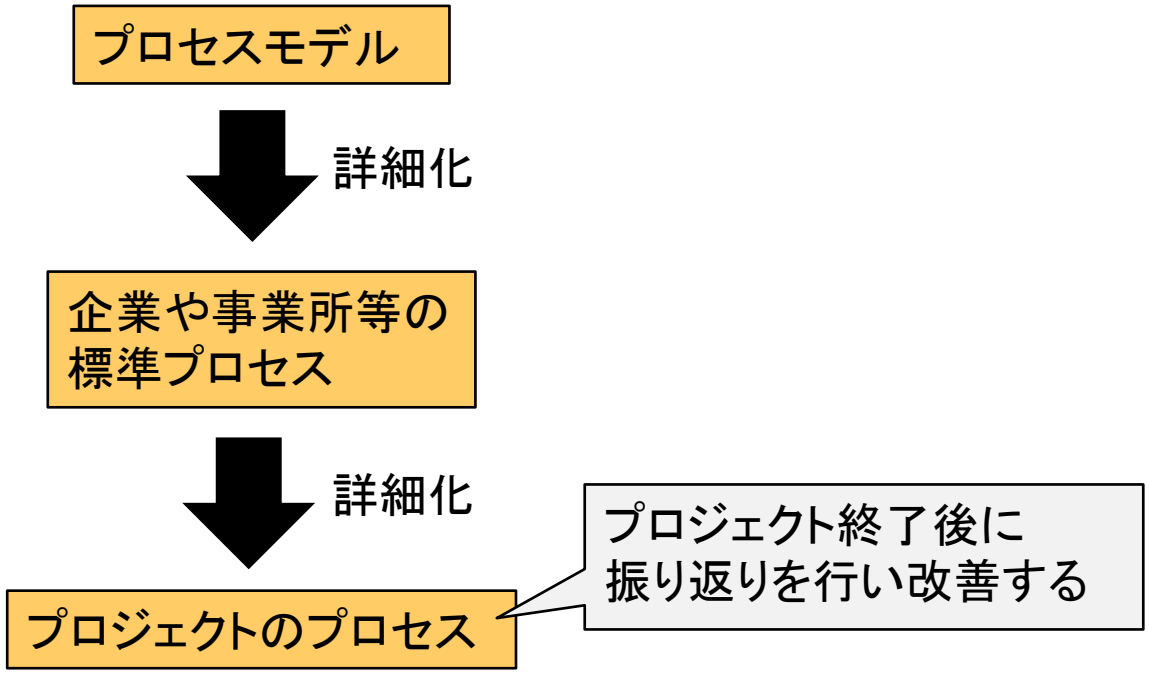
全体が小規模なソフトウェアシステムは，アジャイルが多い

納期があまり厳しくないものは，アジャイルが多い

プロジェクトごとにプロセスは異なる

- 今日説明した内容は，実際のプロセスではなく，プロセス「モデル」

- 実際のプロジェクトでは，プロセスモデルを元に，さらに詳細化したプロセスが採用されている．



まとめ

- ソフトウェア開発工程(ソフトウェアプロセス)
  - 要求分析 ⇒ 基本設計 ⇒ 詳細設計
    ⇒ 実装 ⇒ テスト ⇒ 運用・保守

- ソフトウェアプロセスモデル
  - ウォーターフォールモデル
  - スパイラルモデル
  - 進化型プロトタイプモデル
    - イテラティブ開発とインクリメンタル開発
  - アジャイルプロセス