ソフトウェア工学 第2回
― ソフトウェア開発モデル ―

大連理工大学・立命館大学 国際情報ソフトウェア学部

大森 隆行

講義内容

- ソフトウェア開発モデル
  - 概説、ソフトウェア開発プロセス
  - ソフトウェアプロセスモデル

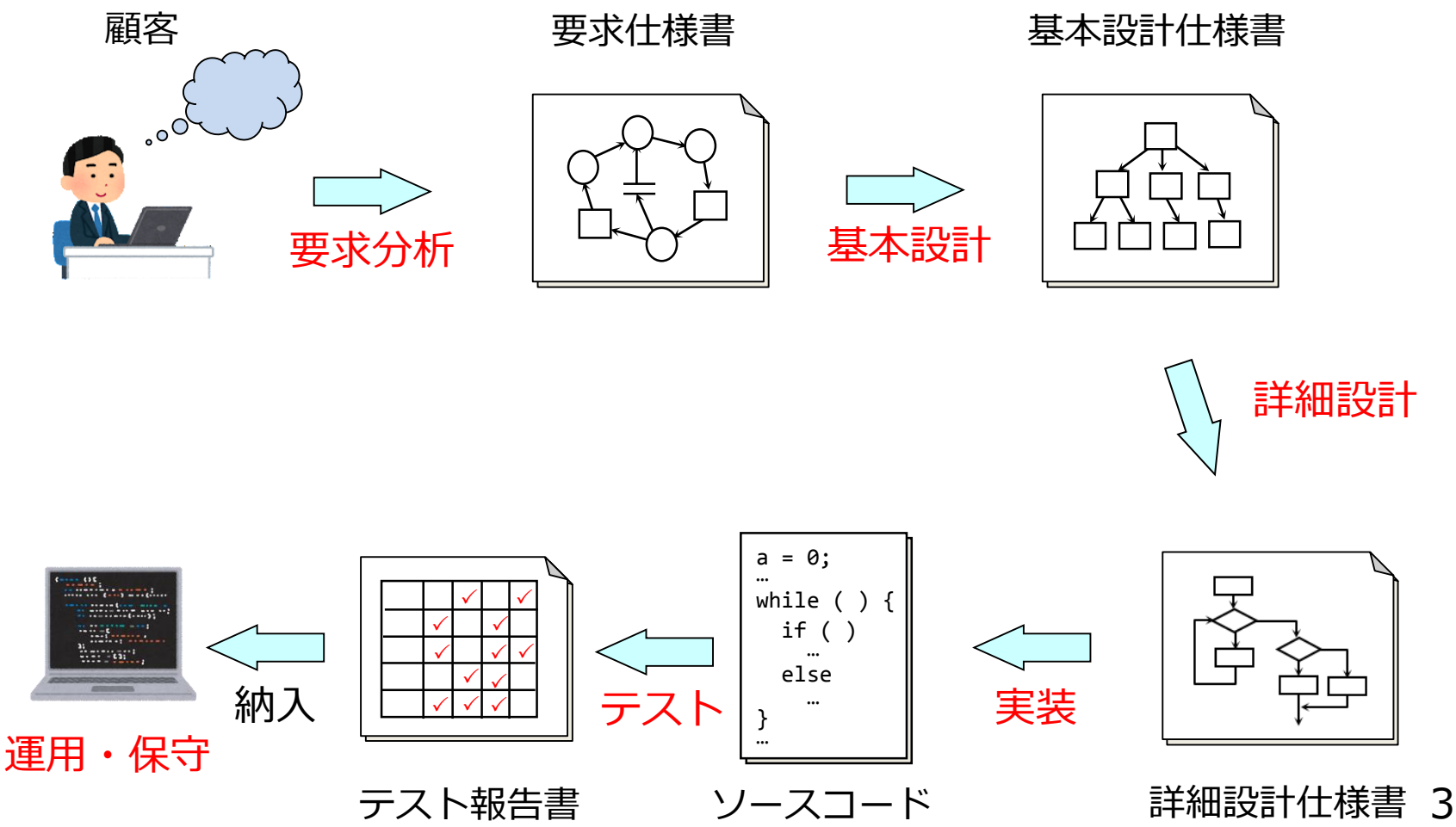
ソフトウェア開発モデル

ここではソフトウェア開発(の方法や手順)

- モデル(model)：検討すべき項目に関連する重要なものを抜き出して記述したもの
- ソフトウェア開発モデル：
  どのようにソフトウェア開発を進めるかを
  モデル化したもの
  - 複数の開発プロセス(process, 過程)から構成される
  - 各プロセスでは、作業の成果をまとめたもの
    ＝成果物(artifact, 教科書ではproduct)を作る
- 特に、どのようなプロセスで開発するかをモデル化したものを、ソフトウェアプロセスモデル(software process model)(または ライフサイクルモデル(lifecycle model)) と呼ぶ

ソフトウェア開発プロセス

ソフトウェア開発の作業工程



要求分析 (requirements analysis)

- 利用者(ユーザ)のシステムへの要求を分析
  - 要求獲得
    - 利用者(ユーザ)の要求を獲得
      - e.g., 利用者がそのシステムで何をするのか
  - 要求記述
    - システムへの要求を文書化
    - 利用者の要求を分析し、システムの仕様(specification)を文書化
      - 要求をすべて実現できるわけではないことに注意 (コスト等の制約、技術的に不可能 etc.)
  - 成果物：要求仕様書 (requirements specification)

設計 (design)

- 基本設計
  - 外部設計、システム設計とも呼ぶ
  - 外部から見たシステムの振る舞い
    (ユーザインタフェース, user interface) や
    システム全体の基本構造
    (アーキテクチャ, architecture) を決定
  - 成果物：基本設計書

  - 要求仕様書に基づいて行われる
    (以下のプロセスでも同様に、前の工程の
    成果物に基づく)

設計 (design)

- 詳細設計
  - 内部設計、プログラム設計とも呼ぶ
  - システムをモジュール(部品)に分割し、
    それぞれのモジュールの機能や
    データ構造を決定する
  - 成果物：詳細設計書
    - データフロー図(DFD)やUML等々、
      様々な種類の文書がある

実装 (implementation)

- 詳細設計書に基づいてプログラムを作成
  - プログラミング言語(programming language)を使う
- コーディング(coding)とも呼ぶ

- 成果物：ソースコード (source code)
  プログラム (program)

テスト・検証 (testing, V&V)

- テスト
  - 仕様通りにプログラムが動作するかを確かめる
    - 単体テスト：モジュールごとに行う
    - 結合テスト：複数のモジュールを結合して行う
    - システムテスト：システム全体で行う
    - 受け入れテスト：ユーザが行う
- 検証
  - 正当性の確認 (verification)
    - システムが仕様通りに動作するか確かめる
  - 妥当性の確認 (validation)
    - システムが要求通りに動作するか確かめる
- 成果物：テスト報告書

運用・保守 (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

確認問題

- どのようにソフトウェア開発を進めるかをモデル化したものを何と呼ぶか。
- 以下はソフトウェア開発のプロセスについて説明したものである。それぞれの説明に該当するものを下の語群から選べ。
  - 利用者のシステムへの要求を分析する
  - 外部から見たシステムへの振る舞いを決定する
  - プログラムを構成する部品の機能を決定する
  - プログラミング言語を使用してソースコードを記述する
  - プログラムが仕様通りに動作するかを確かめる
  - 納入後のソフトウェアを維持・管理する

保守、テスト、基本設計、詳細設計、
要求分析、実装

講義内容

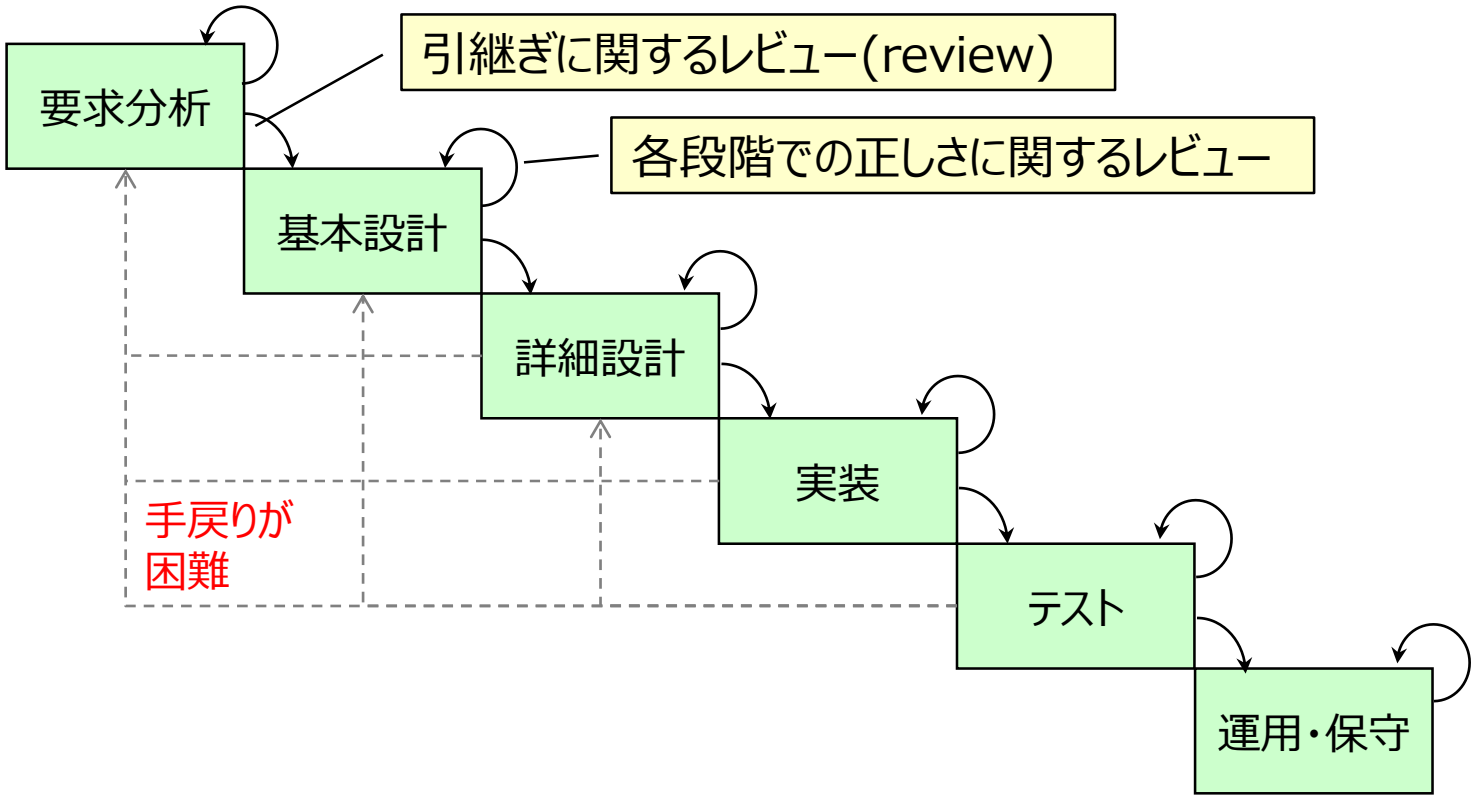
- ソフトウェア開発モデル
  - 概説、ソフトウェア開発プロセス
  - ソフトウェアプロセスモデル

ソフトウェアプロセスモデル

- ソフトウェアをどのようなプロセスで開発するかをモデル化したもの
  - ウォーターフォールモデル
  - 使い捨てプロトタイピング
  - スパイラルモデル
  - 進化型プロトタイピング
  - アジャイルプロセスモデル

ウォーターフォールモデル

- 滝(waterfall)に例えられるプロセスモデル
- トップダウン(top-down)な開発プロセス
  - 要求分析から運用・保守まで一直線に進む
  - 工程の後戻り(手戻り)は原則、想定しない

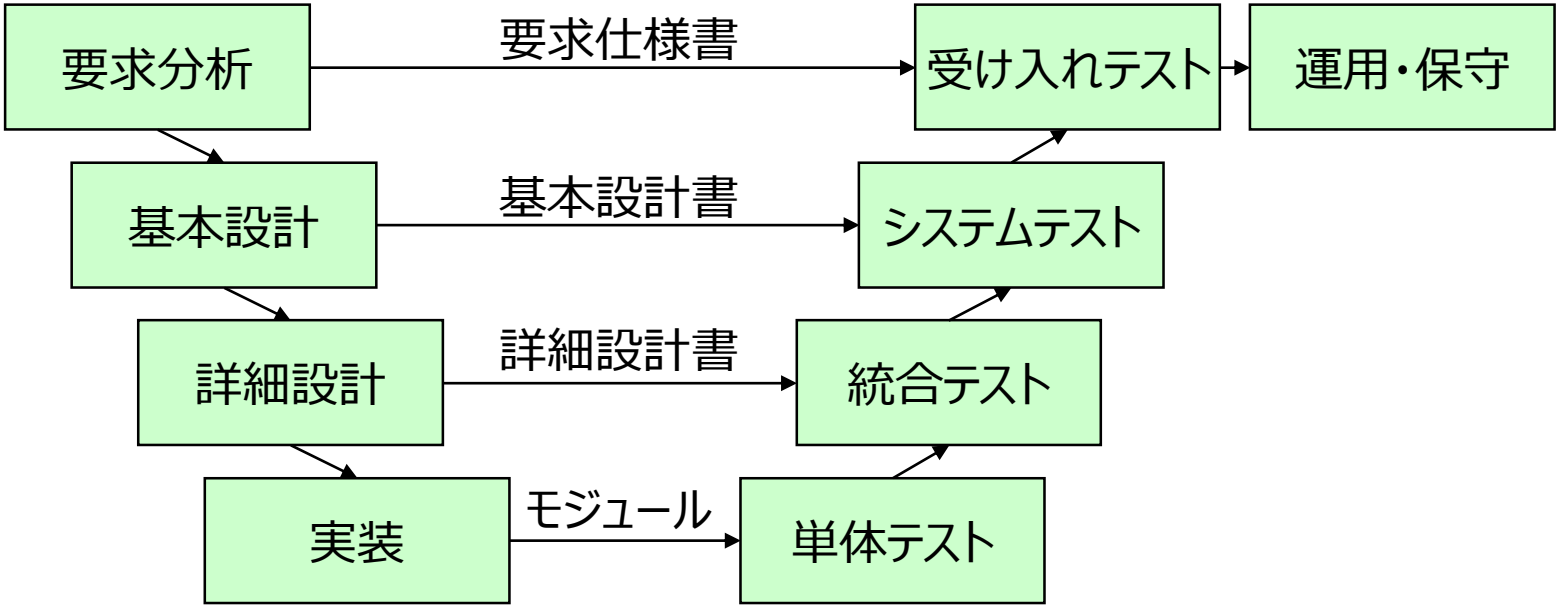


ウォーターフォールモデル

- 利点
  - シンプルで分かりやすい
    - 要求仕様が明確な(後から手戻りが発生しない)場合に有効
- 欠点
  - 仕様変更に弱い
    - 多くのシステム(特に開発・利用期間が長い場合)では仕様変更が発生する
  - 問題発見が遅れる
    - 利用者は開発の終盤まで製品に触れられないため要求の誤解に気付くのが遅れる
    - 要求分析の時点でシステムが存在しないため、詳細な要求(UI(ユーザインタフェース)等)を決定しにくい
    - 実装段階で初めて実現困難な要求であることに気付くことがある
  - 問題発見が遅れると修正コストが増大

V字モデル

- ウォーターフォールモデルの
  テスト・検証プロセス

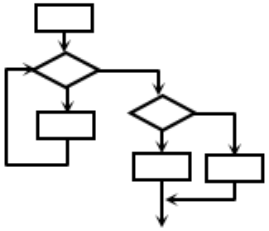


- 各プロセスと対応付けてテスト・検証
  が行われる

確認問題

- 以下の各文の空欄を埋めよ
  - 要求分析から運用・保守まで一直線に進むことを想定したプロセスモデルを、滝に例えて、(1) と呼ぶ。
  - (1) の欠点として、前の工程での誤りが見つかったときに前の工程に戻ること(= (2))を想定していないことが挙げられる。
  - ウォーターフォールモデルのテスト・検証プロセスはプロセスを図示したときの形状から、(3) と呼ばれることがある。

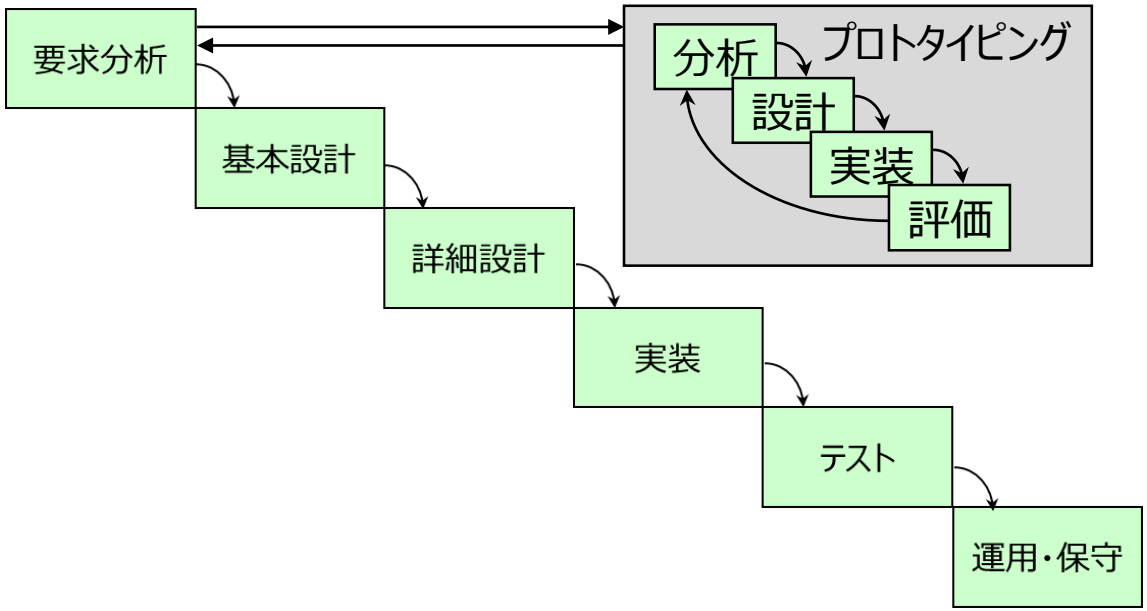
開発モデルの推移


フローチャート

- 1960年代：フローチャート(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
  - ウォーターフォールモデル
- 1980年代：ソフトウェアライフサイクル有害説
  - エンドユーザ不在の開発、重要な判断を初期に確定
    → 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
  1. 開発の初期段階から実行可能なプログラムを部分的に作成
  2. プログラムを実際に動作させて機能を確認
  3. プログラムを改善
  4. 2, 3を繰り返し
     - 使い捨てプロトタイピング (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)

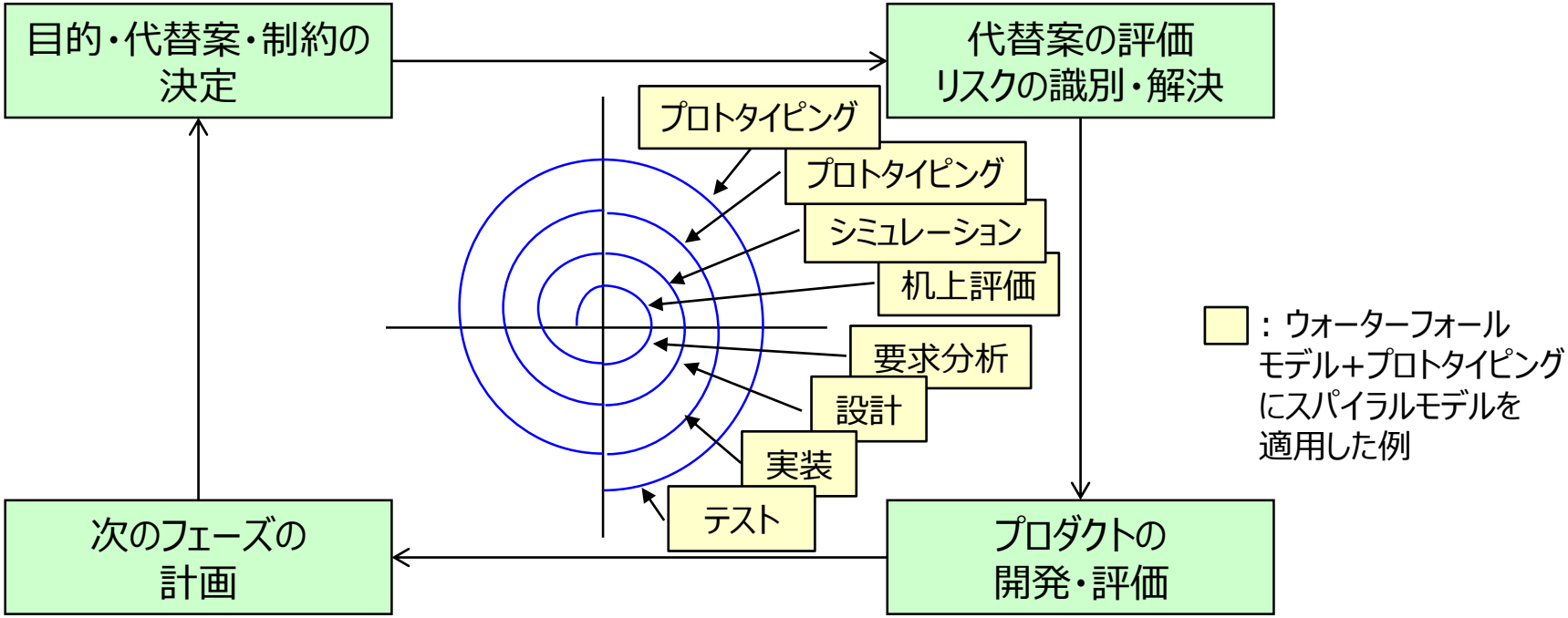
使い捨てプロトタイピング

- プロトタイプ(prototype)：品質確認等のために作る試作品
- プロトタイピング(prototyping)：
  実際にソフトウェアを構築する前に試作品を作り、
  問題点を明らかにする手法
  - UI等の確認
  - 素早く作って、素早くフィードバック(feedback)
- 使い捨てプロトタイピング：
  - 開発の初期段階で作成したプロトタイプをその後使わない



スパイラルモデル

- 下の4フェーズを繰り返して、段階的に開発を進める



- スパイラルモデル自体はプロセスモデルではなく、他のプロセスモデルと組み合わせて用いる(既存のプロセスモデルを改善する)

進化型プロトタイピング

- プロトタイプに修正を加えていき、最終的なソフトウェアとするプロトタイピング手法
  - 機能が明確な部分から開発
  - すべての仕様を確認しながら作るため、大幅な手戻りを予防できる
  - 分析・設計・実装を繰り返す
    - iterative (反復的)
      - すべての部分を一度完成させ、プロトタイプとして提供
      - リリース(release)ごとに各部の完成度を高めていく
    - incremental (漸進的)
      - ソフトウェアを独立性の高い部分に分割
      - リリースごとに機能を追加する
    - 両者を組み合わせて使うのが普通

確認問題

- 以下の各記述に合う開発モデルおよびプロセスモデルを下の語群から選べ
  - 要求分析から運用・保守まで一直線に進む
  - 目的の決定、リスクの識別、プロダクトの開発、次のフェーズの計画の4フェーズを繰り返す
  - プロトタイプを用いるもののうち、作成したプロトタイプを改善して最終製品とするもの
  - プロトタイプを用いるもののうち、初期の分析に用いた後、プロトタイプを使わないもの

ウォーターフォールモデル、使い捨てプロトタイピング、
進化型プロトタイピング、スパイラルモデル

アジャイルプロセスモデル

agile [Adj]機敏な

- 変化に迅速に対応することを目指す
- 開発対象を小さな機能に分割し、各機能を迅速に開発
- 実行可能なソフトウェアを随時提供する
- 進化型プロトタイピングを突き詰めたようなもの

- 単なる開発手順というよりは、
  開発の手順や方法を考えるための枠組み
- 代表的なアジャイルソフトウェア開発手法
  - エクストリーム・プログラミング (XP: Extreme Programming)
  - スクラム (SCRUM)

エクストリーム・プログラミング(XP)

- 開発時における具体的なプラクティス(行動指針)を規定

計画ゲーム	短期リリースのための見積もり、計画
短期リリース	意味のあるリリースを短期間に行う
メタファ	システムの動きを表現する比喩を導入
シンプルな設計	無駄のない設計を行う
テスト	テストを記述し、自動的にテストを行う
リファクタリング	コードの見直しを積極的に行う
ペアプログラミング	2人1組でコーディングを行う
共同所有	コードは全員で共有する
継続的インテグレーション	修正のたびに統合し、テスト実行可能状態を継続
週40時間労働	働き過ぎは良い結果を生まない
オンサイトユーザ	ユーザが開発に参加する
コーディング規約	コードの記述方法に関する約束を決める

ペアプログラミング

- 2人1組でプログラミングを行う手法
  - 役割分担
    - ドライバー：コンピュータへの入力、設計の書き下ろし
    - ナビゲーター：ドライバーの作業を監視し、単純ミスや方向性の間違いを見付ける

※ミス：誤り、失敗。mistake

- 利点
  - 間違いに早期に気付くことができる
    → 作業効率の向上、プログラムの品質向上
  - プログラムについての知識が共有される

継続的インテグレーション (continuous integration)

- 近年では、継続的デリバリ(continuous delivery)と合わせて、CI/CDと言われることが多い
  - ソフトウェアに対する変更を素早く安全に稼働中のサービスに反映
    - プロセスの自動化や標準化が重要
  - 利用者への影響を低減
- CI
  - 新しい機能を実装
    → ローカル変更を共有ブランチにマージ
    → テストを繰り返し、正しく動くことを保証
- CD
  - CIによって作られたコンテナイメージを本番環境に自動的にデプロイ(deploy, 配置)

確認問題

- 以下の各文の中でアジャイルプロセスモデルの説明として正しいものを選べ
  - 仕様は一度決めるとその後変更しないものとみなす
  - 開発対象のソフトウェア全体を一度に完成させる
  - 開発の初期段階でユーザ(顧客)が実行可能なソフトウェアを入手できる
- 次の各文の説明に合う語をカタカナで答えよ
  - アジャイル開発手法の一つで、開発時における具体的なプラクティスを規定している。XPとも表記される
  - 2人1組でプログラミングを行う手法で、XPのプラクティスの一つにもなっている

参考文献

- 「ソフトウェア工学」
  高橋直久、丸山勝久 著、森北出版、2010
- 「要求工学」
  大西淳、郷 健太郎 著、共立出版、2002
- 「ソフトウェア工学」
  岸知二、野田夏子 著、近代科学社、2016
- 「わかりやすいアジャイル開発の教科書」
  前川直也、西河誠、細谷泰夫 著
  ソフトバンククリエイティブ、2013
- 「ペアプログラミング」
  ローリー・ウィリアムズ、ロバート・ケスラー 著
  長瀬嘉秀、今野睦 訳、ピアソン・エデュケーション、2003