ソフトウェア設計法及び演習
ソフトウェア工学概論及び演習

関澤 俊弦
日本大学 工学部 情報工学科

連絡

- 出欠
  - 3限，5限共に出席をとります
    - 出席システムに登録するのを忘れないように

復習

- 講義に関して
- 情報システム
- “ソフトウェア”と“プログラム”
- ソフトウェア開発
  - ソフトウェア開発の特徴
  - ソフトウェア設計法（ソフトウェア開発方法論）

次の時間にライセンスをインストールします．
各自，Astah* Proをインストールしたpcを持ってきてください．

- 開発工程（開発プロセス）
  - ウォーターフォールモデル
    - 要件定義，設計，実装，テスト，導入・保守
  - プロトタイプモデル
  - スパイラルモデル
  - アジャイル
- 演習

（再掲）情報システム開発の現状と問題

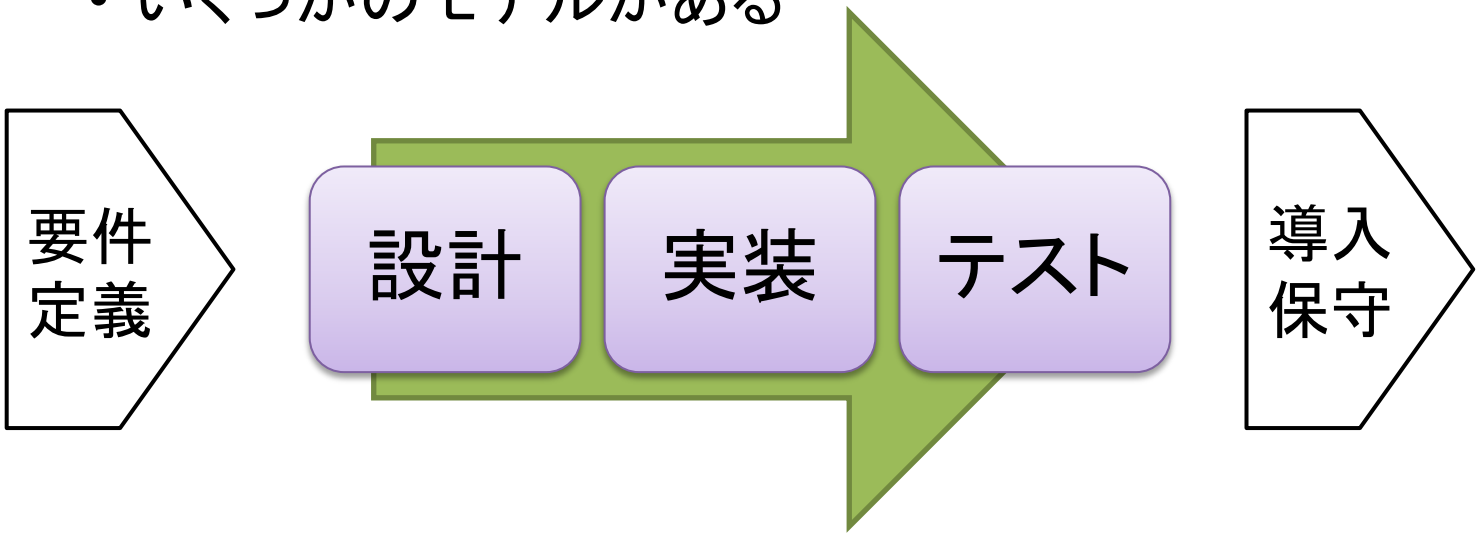
- 現状
  - 目的の変遷
  - 電子技術の進化
  - ネットワーク技術の発展
  - 対象領域の拡大
  - 影響範囲の拡大
- 問題
  - 開発期間の短縮
  - Webシステム開発技術の未成熟
  - 開発技術者の育成

情報システム開発工程 (1)

- 考え方
  - 開発に必要な作業を整理し，理解する
    - 問題となりうる個所を明確にする
  - 開発対象を情報システムとして実現する変換過程として捉える
    - 段階毎に整理し，標準化する

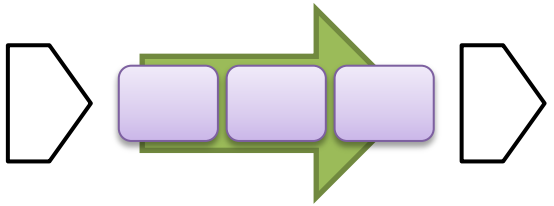
情報システム開発工程 (2)

- 情報システム開発工程
  - S/Wの計画（分析）から，設計，実装，テスト，導入・保守までの過程
    - いくつかのモデルがある



「ソフトウェアライフサイクルプロセス」，「ソフトウェア開発プロセス」は，「情報システム開発工程」とほぼ同義．

情報システム開発工程 (3)

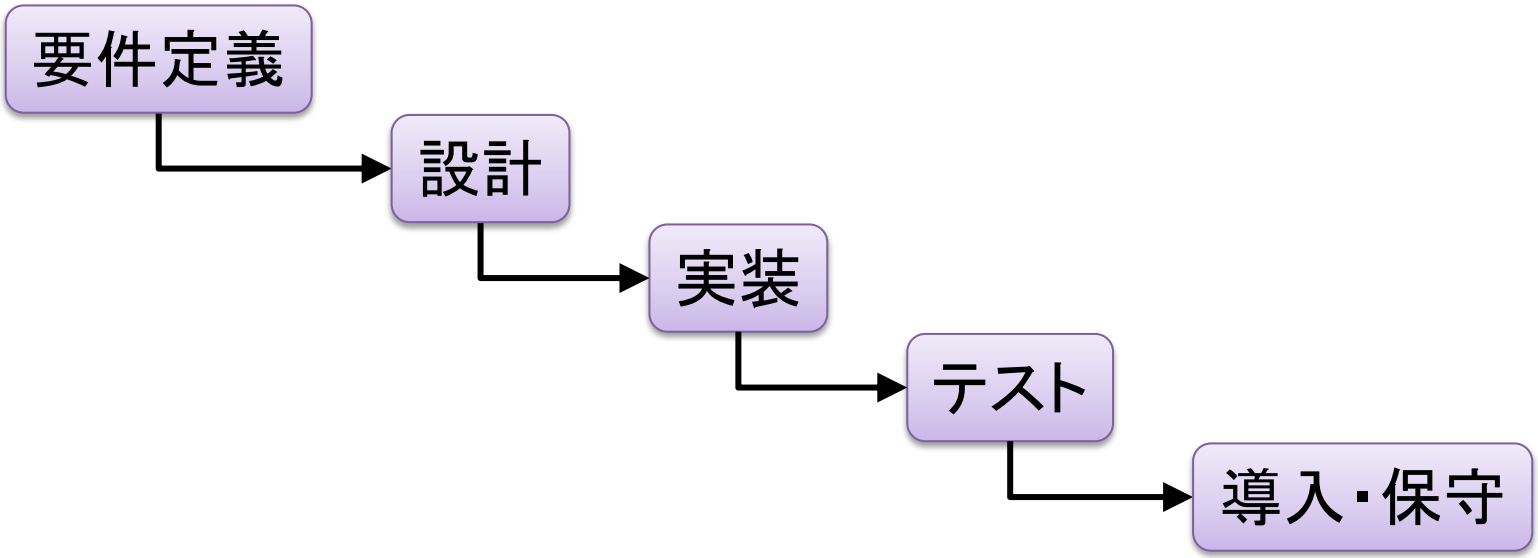


- 開発工程のフェーズ
  - 要件定義（分析）
    - 要求に基づき，仕様（機能）を明確にする
  - 設計
    - 分析に基づき，S/Wの設計書を作成する
  - 実装（開発）
    - 設計書に基づき，プログラム等を開発する
  - テスト
    - 仕様書に従って，テストを行なう
  - 導入・保守

- 開発工程（開発プロセス）
  - ウォーターフォールモデル
    - 要件定義，設計，実装，テスト，導入・保守
  - プロトタイプモデル
  - スパイラルモデル
  - アジャイル
- 演習

ウォーターフォールモデル

- 工程の手戻りを原則行わない手法
  - 各フェーズで必要な作業・成果物を明確化



水が流れるように工程が進むことからの命名.
古典的だが，現在でも広く利用されている.

要件定義

- 目的
  - システムに対する要求を明らかにする
  - システム仕様を作成する
    - 要求を的確に反映していることが重要
- 成果物
  - 要件定義書
  - 各種のモデル

要件定義

- 分析項目
  - 業務モデルの作成
  - データモデルの作成
  - 非機能要件の定義
  - システムスコープの明確化

要件定義: 業務モデルの作成

- 機能とデータ(入出力)の処理を明確にする
  - 分析結果はプロセスモデルとして文書化
    - 開発作業の基盤となる
  - プロセスモデルの表現方法
    - データフローダイアグラム (Data Flow Diagram: DFD)
      - Lesson04で解説予定

要件定義: データモデルの作成

- データ間の関連性を明確化する
  - データにのみ着目してデータモデルを作成
    - データベース設計等の基本情報として使用する
  - データモデルの表現方法
    - エンティティリレーションダイアグラム (Entity Relationship Diagram: ER図)
      - Lesson05で解説予定

要件定義: 非機能要件の定義

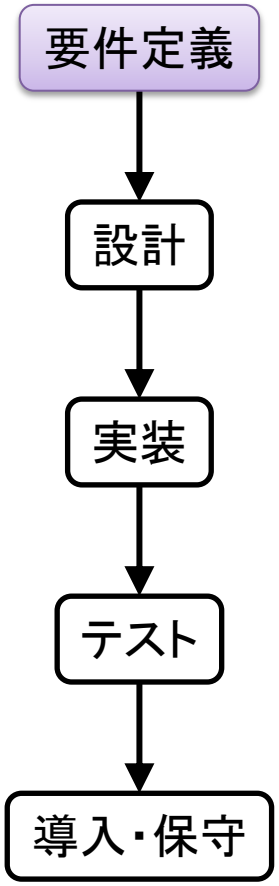
- 非機能要件とは，機能やデータ以外の全般
  - 性能，信頼性，セキュリティなどが含まれる
    - 例:「機能的には十分でも応答速度が遅いことは許されない」など
- 非機能要件がある場合，仕様に記述する

要件定義: システムスコープの明確化

- システム化する/しない対象の明確化
  - すべての業務を情報システム化する方が良いとは限らない

要件定義: まとめ

- 要件定義の成果物
  - プロセスモデル
  - データモデル
  - 非機能要件
  - 制約条件，システム構成
  - 開発計画



設計

- 目的
  - システムの設計を行なう

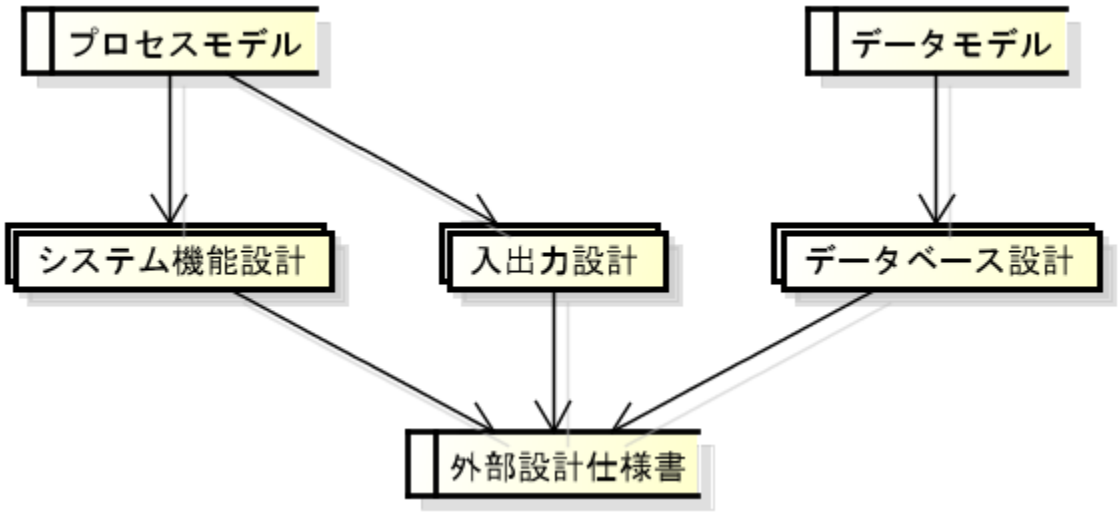
- 設計フェーズの詳細
  - 外部設計
    - システムとユーザ間のインターフェースの設計
  - 内部設計
    - システムの内部構造の設計
  - 詳細設計

設計: 外部設計

- システムとユーザ間のUIの設計
  - 機能設計
    - 用件定義で作成されたモデルを詳細化する
  - 入出力設計
    - 機能設計で称された機能に対して，関連する入出力データを設計する
    - Webアプリケーションでは画面設計が重要となる
  - データベース設計
    - 論理設計: 業務視点のデータベースの設計
    - 物理設計: 計算機視点のデータベースの設計

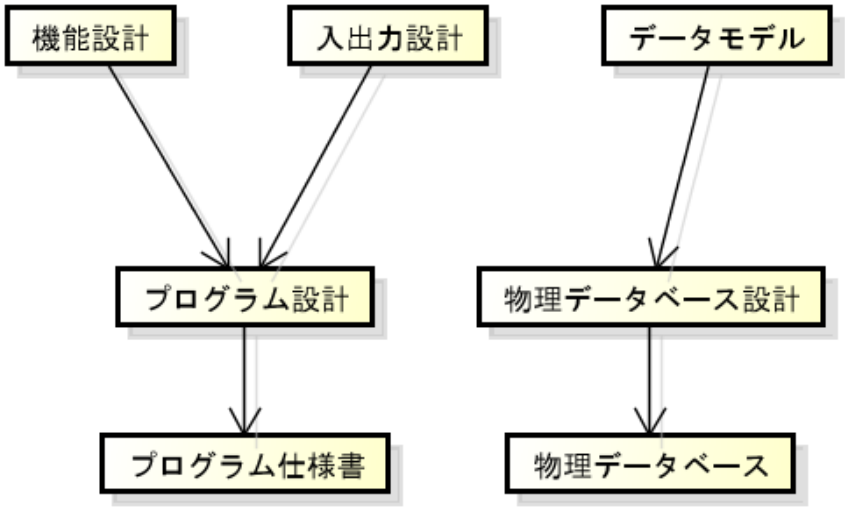
設計: 外部設計

- 要件定義と外部設計の関係
  - 要件定義: プロセスモデル，データモデル
  - 外部設計: 機能設計，入出力設計，DB設計



設計: 内部設計

- システムの内部構造の設計
  - 外部設計を実現するための内部構造の設計
  - プログラム仕様書として定義
    - 入出力設計を元に，処理方法・画面と処理プログラムの関係を明確にする



実装

- 目的
  - 内部設計で作成したプログラム仕様書を元に，プログラム設計・コーディングを行なう

テスト

- 目的
  - 開発したシステムが要件定義で作成されたシステム仕様書通りに動作するか確認する

導入・運用

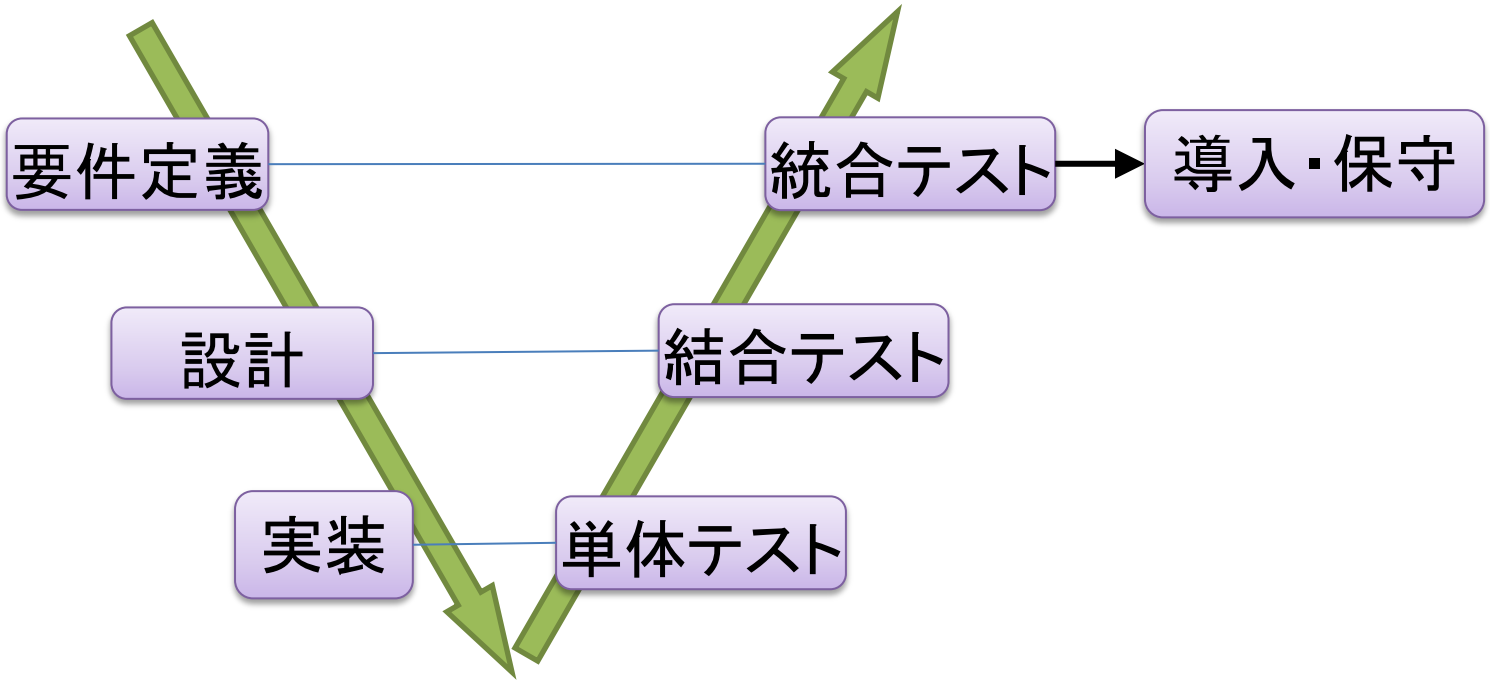
- 目的
  - ユーザに対してシステムを導入する
    - 操作手順書などの作成
    - ユーザの教育

ウォーターフォールモデルの特徴

- 利点
  - 開発の本質的な工程を表わす
    - 製品を作成するときの基本パターン
  - 大規模開発に適している
    - 事前にしっかり開発計画を作成する必要があるため
- 欠点
  - 時間がかかる
    - 入念な準備は時間を要する
  - 開発作業が一方通行である
    - 開発途中で手戻りが発生するとやり直しとなる

V字開発モデル

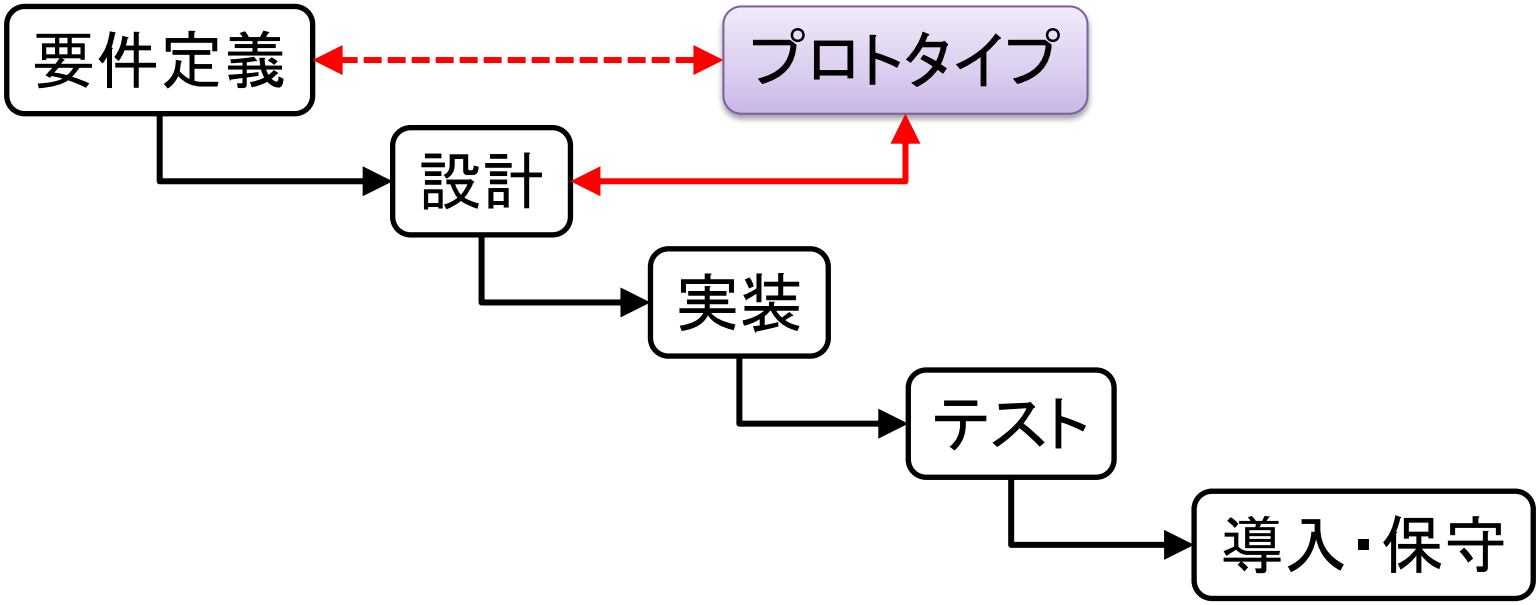
- ウォーターフォールモデルと基本的に同じ
  - 対応するフェーズを関連付ける
  - 利点・欠点: ウォーターフォールモデルと同



- 開発工程（開発プロセス）
  - ウォーターフォールモデル
    - 要件定義，設計，実装，テスト，導入・保守
  - プロトタイプモデル
  - スパイラルモデル
  - アジャイル
- 演習

プロトタイプモデル

- 設計前にプロトタイプを作成・評価する
  - システムの一部や核となる部分を対象とする



ウォーターフォールの改良型

プロトタイプの必要性と効果

- ニーズの把握
  - システムの稼働イメージで要求を確認できる
- 正確なレビュー
  - 要求に対して，曖昧さがない評価ができる
- 機能の検証
  - 機能の実現可能性を検証できる
- 操作性の評価
  - システムの操作性を事前に評価できる

プロトタイプモデルの開発工程

- 基本的にはウォーターフォールモデルと同じ
  - 設計フェーズ
    - 機能設計を元にプロトタイプを作成する
      - UI，簡易データベースなどを含む
  - 考慮する点
    - 単純性: システムの本質を表わすプロトタイプの作成
      - プロトタイプの作成にかかわるコストを抑える
    - 拡張性: 拡張・修正が容易なプロトタイプの作成
    - 対話性: 試行錯誤が可能な画面設計
    - 表示性: システム稼働時をイメージの作成

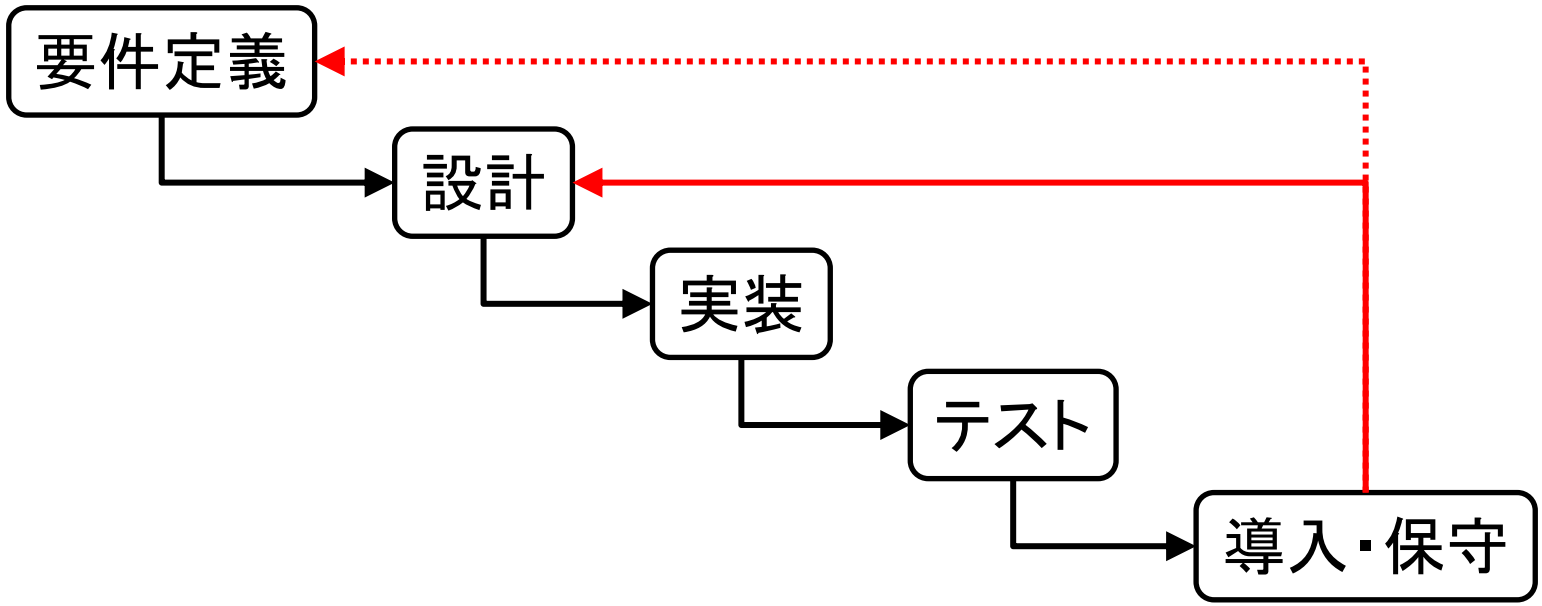
プロトタイプモデルの特徴

- 利点
  - 要件定義に不明確な点を含むシステムの開発に適する
    - 表2.2「プロトタイプが効果的なシステム特性」を参照（教科書 p.26）
- 欠点
  - 要件定義が明確なシステムの開発には不向き
    - プロトタイプを作成する意義が薄い

- 開発工程（開発プロセス）
  - ウォーターフォールモデル
    - 要件定義，設計，実装，テスト，導入・保守
  - プロトタイプモデル
  - スパイラルモデル
  - アジャイル
- 演習

スパイラルモデル

- システム全体をサブシステムに分割し，サブシステムの開発を繰り返す




スパイラル（渦巻き）に見えることからの命名．

スパイラルモデルの適用 (1)

- サブシステム毎の開発
  - システムをサブシステムに分割する
    - サブシステム間の独立性が高い方が望ましい（大規模システムに向く）
  - 各サブシステムをウォーターフォールで開発する
    - 開発済みのサブシステムと繋げつつ開発する
    - プロトタイプを作成することもある

スパイラルモデルの適用 (2)

- システムの品質改善
  - システム全体を一度開発する
  - システムの問題点を洗い出す
  - 問題があればシステムを作り直す (Scrap and Build)

スパイラルモデルの特徴

- 利点
  - 要件定義ですべてを定義することが難しいシステム開発に適する
  - 各サブシステムの開発規模が小さい
  - 表2.3「スパイラルモデルの特徴」を参照（教科書 p.28）
- 欠点
  - 一般に，時間・コストがかかる

- 開発工程（開発プロセス）
  - ウォーターフォールモデル
    - 要件定義，設計，実装，テスト，導入・保守
  - プロトタイプモデル
  - スパイラルモデル
  - アジャイル
- 演習

アジャイルモデル（アジャイル開発）

- 開発対象を多数に分割し，機能毎に開発する手法の総称
  - 開発サイクルを繰り返す点は，スパイラルモデルと共通
    - 反復型（イテレーション）開発とも呼ばれる
- "アジャイルモデル"は総称
  - エクストリーム・プログラミング
    (XP: eXtreme Programming)

アジャイルモデルによる開発

- 開発サイクル
  - システムを小さい機能に分割する
    - 一般に，スパイラルモデルよりも小さく分割する（1～4週間で開発できる程度と言われる）
  - 各機能を，要件定義からテストまで行なう
- 実装優先
  - プログラム作成を優先し，動作させる（要件定義・設計・テストも行なう）
- 文書化
  - 開発者が同じ場所で作業し文書を減らす

アジャイルモデルの特徴

- 利点
  - 小規模開発に適する
  - 表2.4「アジャイルモデルの特徴」を参照（教科書 p.31）
- 欠点
  - 大規模システムに適用することが難しい
  - 設計等の文書が十分とならない可能性がある（本来はあってはならない）

- 開発工程（開発プロセス）
  - ウォーターフォールモデル
    - 要件定義，設計，実装，テスト，導入・保守
  - プロトタイプモデル
  - スパイラルモデル
  - アジャイル
- 演習

演習2-1: Astah* Proのライセンス登録

- 登録手順（**指示されたときのみ実施**）
  - ポータルから，ライセンスファイルをダウンロードする（"JUDE_License_User_Professional.xml"）
  - Astah* Proをインストールしたディレクトリに，1.のファイルをコピーする
  - Astah*の[ヘルプ]-[バージョン情報]で，ライセンスが登録されていることを確認する

ここを確認

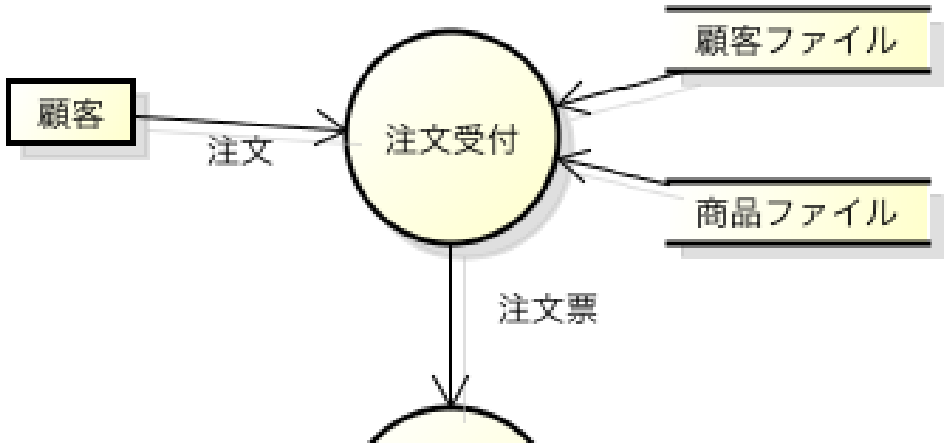
演習2-2: Astahを使ってみる

- 次のC言語のフローチャートをAstah*を用いて描画せよ

```
while (s != 0){
  if (s > 0) {
    s--;
  } else {
    s++;
  }
}
```

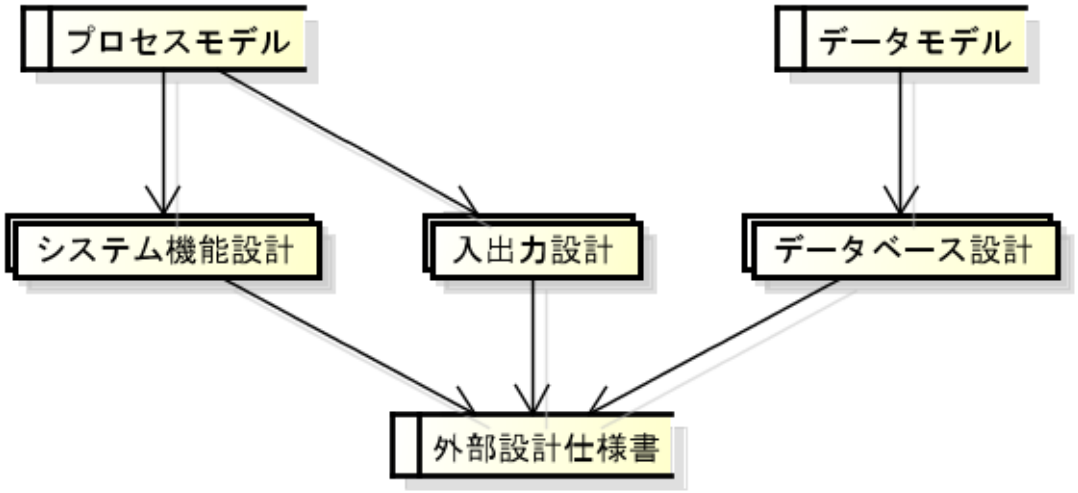
演習2-3: プロセスモデルの図を描く

- Astah*を用いて，教科書p.12の図2.2にあるプロセスモデルを自分で描け
  - 「データフロー図（DFD）」で記述
    - 図中の記号の意味はLesson04で取り上げる予定



演習2-4:「外部設計」の図を描く

- 次に，教科書p16の図2.5にある「外部設計」の図を自分で描け
  - 「データフロー図（DFD）」で記述
    - 右クリックメニューの「図の表記」から「デマルコ」→「ゲイン/サーソン」に変更する必要あり



演習がおわったら

- 今回は特に提出物なし
- 次回の演習のためにastah* proのファイル（例:「ソフトウェア設計法／lesson02_1.asta」）を保存しておくこと