ソフトウェア工学 第1回
― ソフトウェア工学概説 ―

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

担当者

- 大森 隆行

  おおもり たかゆき

  OMORI Takayuki

URL:
http://www.ritsumei.ac.jp/~tomori/lecture/se/

授業の目的

- ソフトウェア工学(software engineering)に関する基礎知識の習得

- 高品質・大規模なソフトウェアを限られた時間・費用で開発するための技術
- 要求分析・設計・実装・テスト・保守という各工程における概念・考え方・技法・用語

この授業について

- 15回(+1回)の講義
  - 16回目は復習・質問(・雑談？)のみ
    (16回目は出席の必要なし)
  - 15回の講義とは別に期末試験があります
- 講義の流れ
  - 説明 → 確認問題 → 確認問題の説明
- 評価方法
  - 日常点（レポート、出席、中間試験）
  - 期末試験

授業計画 (前半)

- 第1回 ソフトウェア工学概説
- 第2回 ソフトウェア開発モデル
- 第3回 要求分析 ぶんせき
- 第4回 構造化分析
- 第5回 オブジェクト指向分析
- 第6回 アーキテクチャ設計、
  ユーザインタフェース設計
- 第7回 モジュール設計
- 第8回 中間試験および解説

授業計画 (後半)

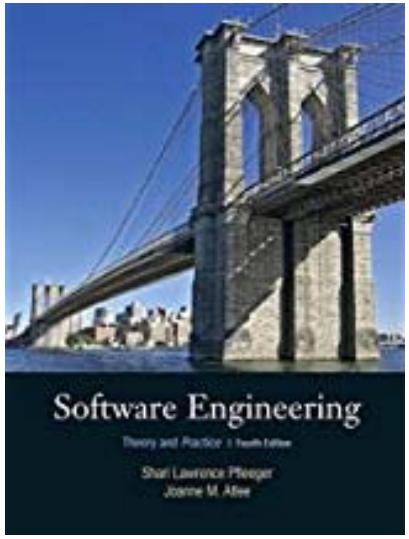
- 第9回 プログラミング
- 第10回 ソフトウェアテスト(1)
- 第11回 ソフトウェアテスト(2)
- 第12回 ソフトウェアテスト(3)
- 第13回 ソフトウェア保守と再利用ほしゅ
- 第14回 ソフトウェア開発管理
- 第15回 まとめ

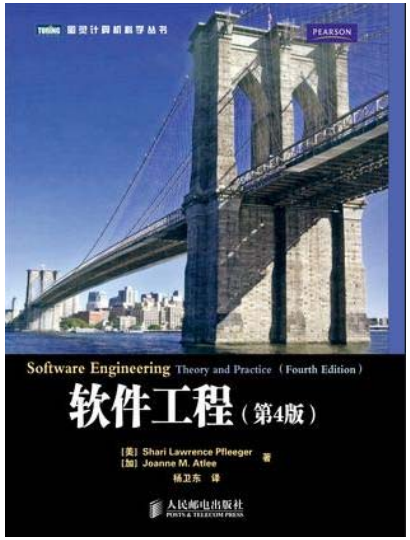
※内容や理解の程度によって多少前後することがあります

参考書

- ソフトウェア工学 高橋直久、丸山勝久 著、森北出版、2010

- ソフトウェア工学 理論と実践、シャリ・ローレンス・プリーガー 著、ピアソン・エデュケーション、2001





※この他にも多くの本の内容を紹介します

講義内容

- ソフトウェア工学概説
  - ソフトウェア開発における問題
  - ソフトウェア品質特性
  - 問題解決への取り組み

ソフトウェア (software)

きそく

- 情報処理システムのプログラム、手続き、規則、および関連文書の全体または一部分 (JIS X0001)
  - プログラム　狭義（きょうぎ）
  - 要求定義書、外部設計書、内部設計書、データベース定義書、コーディング規約、取扱説明書、… 広義
- cf. ハードウェア (hardware)
  - コンピュータの装置

応用ソフトウェア (application -)	メーラ、ブラウザ、ワードプロセッサ等
ミドルウェア (middleware)	データベース管理ソフト等
基本ソフトウェア (basic software)	オペレーティングシステム等

ソフトウェアとハードウェア

- ソフトウェア
  - 経年変化なし
  - 導入後に修正可能
  - 量産、配布・流通コストが低い

機能拡張、
性能改善等が
求められる

- ハードウェア
  - 経年変化あり(摩耗、部品の寿命)
  - 導入後の修正はほぼ不可能
  - 製品の量産および配布コストが高い

機能や性能の
維持が
求められる

ソフトウェア開発

- 顧客の要求をソフトウェアとして実現する作業
  - 良いソフトウェアを効率的に構築
  - 品質とコストのバランスが重要
- 開発技術の歴史
  - 1960年代
    - 既存業務の情報システム化
  - 1970年代　　life cycle
    - 開発計画やライフサイクルの登場
  - 1980年代
    - 開発における生産性を重要視
  - 1990年代
    - 社会の依存度が増大
    - 使い勝手の良さに対する要求
  - 21世紀
    - 複雑さへの挑戦

ソフトウェア工学 (software engineering)

- コンピュータソフトウェアを対象として、その構築、運用、保守における生産性と品質の向上を実現するための技術体系や学問体系
  - 方法論(methodology)
  - 技法(technique) / 道具(tool)
  - プロジェクト管理(project management)

- ソフトウェア工学の目的
  - 良いソフトウェアを開発すること
    → ソフトウェアの品質特性(国際規格ISO9126)

ソフトウェア危機 (software crisis)

- 背景
  - 1950年代後半以降のコンピュータの性能向上
  - 1960年代コンピュータの需要の高まり

- 技術者不足、納期遅れ、品質低下、開発費増大
  - 1968年のNATO会議で指摘

- 主要な問題
  - 生産物の増大
  - ソフトウェアに関わる人の増加
  - 期間の長期化
  - 社会的役割の変化

ソフトウェア危機 (software crisis)

- 生産物の増大
  - コンピュータの性能向上や普及によるソフトウェア大規模化
  - コンピュータの普及に伴う開発ソフトウェア数の増大
- ソフトウェアに関わる人の増加
  - ステークホルダ(stakeholder)の多様化
    - ソフトウェアの開発や利用における利害関係者
- 期間の長期化
  - 開発時間、利用時間の増大
  - 動作環境や社会的要求が変動
- 社会的役割の変化
  - コンピュータで扱う分野の拡大
  - 社会的な重要性の増大 (信頼性が求められるようになった)
  - ソフトウェアの大衆化に伴う使い勝手の向上

大規模・高信頼ソフトウェア開発に向けて

プログラミング演習（えんしゅう） ≠ 大規模で高信頼なソフトウェアの開発

练习

- 実際のソフトウェアは…
  - 大規模である
  - 複数人で開発される
  - 様々なステークホルダ stakeholder
  - 実際に利用・運用される
    - 効率性、保守性、信頼性 etc. が求められる

じっせん

- ソフトウェア工学の実践
  - 適切な理論・原理・技術に基づく方法論や、表記法・ツールを活用
  - 要求を満たす、高品質・高信頼なシステムを開発
  - 作るだけでなく、運用・保守まで含めて考える

確認問題

- 次の特徴にあてはまるものは
  ハードウェア、ソフトウェアのどちらか
  - 部品の劣化などの経年変化がある
  - 導入後に修正が容易である
  - 製品の量産のコストが安い
- ソフトウェア危機を説明した次の説明は
  正しいか、誤っているか、○か×で答えよ。
  - ソフトウェアが大規模化し、開発に携わる人が増えた。
  - ソフトウェアの開発期間、利用期間はともに短縮した。
  - ソフトウェアの社会的な重要性が増大した。
- ソフトウェアの開発や利用における利害関係者を
  意味する語を英語1単語で答えよ。

確認問題

- 次の特徴にあてはまるものは
  ハードウェア、ソフトウェアのどちらか
  - 部品の劣化などの経年変化がある　ハード
  - 導入後に修正が容易である　ソフト
  - 製品の量産のコストが安い　ソフト
- ソフトウェア危機を説明した次の説明は
  正しいか、誤っているか、○か×で答えよ。
  - ソフトウェアが大規模化し、開発に携わる人が増えた。O
  - ソフトウェアの開発期間、利用期間はともに短縮した。X
  - ソフトウェアの社会的な重要性が増大した。O
- ソフトウェアの開発や利用における利害関係者を
  意味する語を英語1単語で答えよ。stakeholder

講義内容

- ソフトウェア工学概説
  - ソフトウェア開発における問題
  - ソフトウェア品質特性
  - 問題解決への取り組み

ソフトウェアの品質特性 (ISO9126)

1. 機能性(functionality)：必要な機能が実装されているか
   - 合目的性：利用者の目的に合っているか
   - 正確性：仕様に対して正しく動作するか
   - 相互運用性：他のシステムとやり取りできるか
   - セキュリティ：不当なアクセスを排除できるか
   - 標準適合性：ソフトウェアの機能が法規、規格、業界標準を遵守しているか

品質特性

品質副特性

仕様（しよう）手段方法

2. 信頼性(reliability)：機能が正常に動作し続けるか
   - 成熟性：障害時にソフトウェアが停止しないか
   - 障害許容性：障害時に機能を提供し続けられるか
   - 回復性：故障したときに素早く復旧できるか
   - 標準適合性：ソフトウェアの信頼性が法規、規格、業界標準を遵守しているか

ソフトウェアの品質特性 (ISO9126)

3. 使用性(usability)：利用者にとって使いやすいか
   - 理解性：使い方が理解しやすいか
   - 習得性：初めてでもすぐに使えるようになるか
   - 操作性：ユーザインタフェースが使いやすいか
   - 魅力性：利用者にとって魅力があるか
   - 標準適合性：ソフトウェアの使用法が法規、規格、業界標準を遵守しているか

4. 効率性(efficiency)：目的達成のために使用する資源は適切か
   - 時間的効率性：応答時間が短いか、処理速度が速いか、指定されたスループットが確保できるか
   - 資源効率性：メモリやネットワークなどの資源を余計に消費しないか
   - 標準適合性：ソフトウェアの性能が法規、規格、業界標準を遵守しているか

ソフトウェアの品質特性 (ISO9126)

かいてい

5. 保守性(maintainability)：改訂作業に必要な労力は少ないか 可维护性
   - 解析性：変更箇所を特定しやすいか
   - 変更性：プログラムが変更しやすいか
   - 安定性：変更時にその影響が予想外の箇所に及ばないか
   - 試験性：変更時にテストがしやすいか
   - 標準適合性：ソフトウェアの保守性が法規、規格、業界標準を遵守しているか

6. 移植性(portability)：他の環境へ移しやすいか
   - 順応性：別の環境に移す際の手間は少ないか
   - 設置性：インストールしやすいか
   - 共存性：同じ環境で他のソフトウェアと共存できるか
   - 置換性：他のソフトウェアに置き換え可能か
   - 標準適合性：ソフトウェアの可搬性が法規、規格、業界標準を遵守しているか

確認問題

- ISO9126において定められているソフトウェアの品質特性について、以下の説明に当てはまるものを下の語群から選べ。
  (1) 利用者にとって使いやすいか
  (2) 必要な機能が実装されているか
  (3) 他の環境へ移しやすいか
  (4) 改訂作業に必要な労力は少ないか
  (5) 機能が正常に動作し続けるか
  (6) 目的達成のために使用する資源は適切か

機能性、信頼性、使用性、効率性、保守性、移植性

確認問題

- ISO9126において定められているソフトウェアの品質特性について、以下の説明に当てはまるものを下の語群から選べ。
  (1) 利用者にとって使いやすいか使用性
  (2) 必要な機能が実装されているか 機能性
  (3) 他の環境へ移しやすいか 移植性
  (4) 改訂作業に必要な労力は少ないか 保守性
  (5) 機能が正常に動作し続けるか 信頼性
  (6) 目的達成のために使用する資源は適切か 効率性

機能性、信頼性、使用性、効率性、保守性、移植性

講義内容

- ソフトウェア工学概説
  - ソフトウェア開発における問題
  - ソフトウェア品質特性
  - 問題解決への取り組み

問題解決への取り組み

- 分割統治と構造化
- 抽象化とモデリング
- 要求分析
- 追跡可能性
- 経験の蓄積と再利用
- 系統的な評価と管理

分割統治と構造化

- 分割統治 (divide and conquer)
  - 大きな問題を独立性の高い小さな問題に分割する
  - 小さな問題をそれぞれ解(ほど)く
  - 解を取りまとめて、本来の大きな問題の解を得る

- 分割統治のための手法 (例)
  - ソフトウェアの構造化：
    大きな問題をたくさんの小さな要素に分割した後に、
    それらの要素の間の関係を分かりやすく整理すること
    - 絡み合って複雑な問題も、細かく分けて考えれば
      構造を理解できる
    - ソフトウェアを“部品”に分割する
    - 構造化分析、構造化プログラミング等

抽象化とモデリング

- 抽象化 (abstraction)
  - 対象とする事物から本質的でない部分を取り除くこと
  - 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する（記述したものをモデルと呼ぶ）
- モデリング（modeling）
  - ソフトウェアに関する様々な項目に対して、それらの本質を浮かび上がらせるように抽出する
  - 大きく複雑なシステムであっても、詳細にとらわれずに本質的な部分に着目できる
  - 複数のモデルの存在
    - 項目ごとに別のモデルがある
    - 同じ項目でも異なる視点からモデリングできる

要求分析

- 要求（requirements）
  - ソフトウェアが、どのような制約のもとで、どのようなサービスを提供するか
  - 明快かつ詳細に決定され、文書として厳密に定義されているのが望ましい
- 要求分析
  - システムのサービスと制約を探り出し、分析し、文書化し、検査する過程
  - 十分に行わないと、ソフトウェア開発の失敗の原因となる
- 要求工学 (requirements engineering: RE)
  - 要求をまとめるための学問・技術体系
  - ソフトウェアとして何を作るべきかを定める

ついせき
追跡可能性 (traceability)

- ソフトウェア開発では数多くの「判断」がなされる
  - 例えば、要求仕様や設計仕様
  - 一部は成果物(artifact)として文書化される
  - 開発プロセスに応じて、様々な成果物がある

→「どのような判断をしたのか」
「なぜその方法・方式を選択したのか」等を残す必要がある

- 追跡可能性
  - ある成果物から他の成果物の対応箇所や、大元の判断を辿ることができる性質
  - 例) プログラムの不具合を発見したとき、原因となった設計仕様書の記述箇所、要求仕様書の記述箇所、要求の理由を辿れる
  - 例) 要求仕様書を変更したとき、その要求によって実現された設計仕様書の機能や性能、プログラムの記述を辿れる
  - 利用者・分析者・設計者・プログラマが判断した「理由」まで辿れると良い

経験の蓄積（ちくせき）と再利用

- 開発経験(成功、失敗)の蓄積
  - 共通化して蓄積、集約して「次の開発」に再利用
  - 方法論やツールが開発される
    → 様々なプロジェクトや企業で使われる
    → 経験として蓄積され、さらに改良され、
    さらに広く使われる
- ソフトウェア開発において
  「経験的に良いと確認された方法」を
  標準化(standardize)して誰もが使えるようにする
- SWEBOK (Software Engineering Body of Knowledge)
  - ソフトウェア工学基礎知識体系
  - ソフトウェア工学に関連する知識を体系的にまとめたもの
  - 最近の研究の成果を反映して、2014年にver.3策定

系統的な評価と管理

- 大規模なソフトウェア
  → 多数の技術者が共同で開発を進める
  → 個人のソフトウェア開発と異なり
  系統的な管理(management)と評価(evaluation)が必要
  例えば...
  - プログラムの妥当性検証、テスト
  - ソフトウェアの規模の見積もり
  - ソフトウェア開発計画の策定
  - プロジェクトの進捗管理
  - 進捗(しんちょく)遅れへの対策
  - 振り返り

確認問題

- 以下の説明にあてはまる語句を答えよ
  - 大きな問題を独立性の高い小さな問題に分割すること
  - 対象とする事物から本質的でない部分を取り除くこと
  - 対象とする事物から本質的な部分を抜き出して記述したもの
  - ソフトウェアがどのような制約のもとで、どのようなサービスを提供するかを表すもの
  - ソフトウェアを構成するある成果物から他の成果物の対応箇所を辿ることができる性質
  - ソフトウェア工学に関連する知識を体系的にまとめたもの

確認問題

- 以下の説明にあてはまる語句を答えよ
  - 大きな問題を独立性の高い小さな問題に分割すること 分割統治
  - 対象とする事物から本質的でない部分を取り除くこと 抽象化
  - 対象とする事物から本質的な部分を抜き出して記述したもの モデル
  - ソフトウェアがどのような制約のもとで、どのようなサービスを提供するかを表すもの 要求
  - ソフトウェアを構成するある成果物から他の成果物の対応箇所を辿ることができる性質 追跡可能性
  - ソフトウェア工学に関連する知識を体系的にまとめたもの SWEBOK

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
- 「ソフトウェアエンジニアリング
  基礎知識体系 -SWEBOK V3.0-」
  松本吉弘 訳、オーム社、2014