

アーキテクチャーの品質

FU14 ソフトウェア工学概論

第6回

吉岡 廉太郎

前回の内容

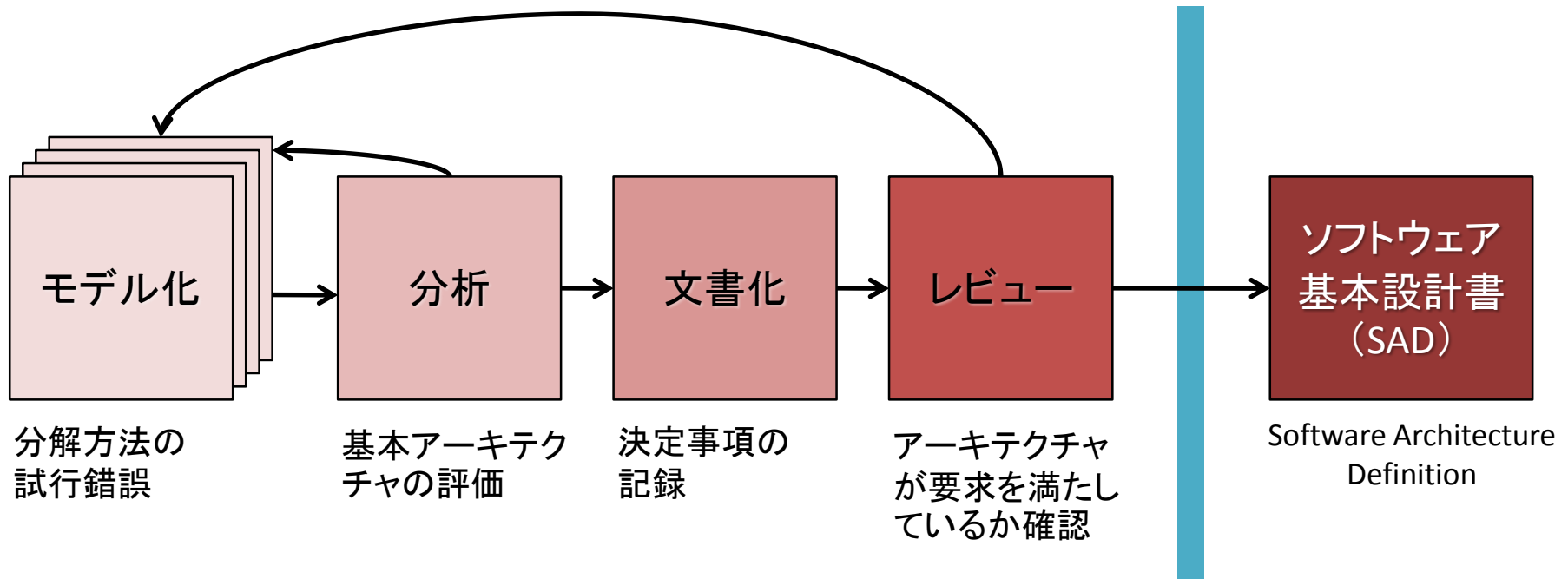
- アーキテクチャーの様式
 - システムの全体構成を考えるためのモデル
 - 様式でシステムの“性質”が決まる
 - 最低限知っておくべき、一般的常識、共通言語
 - 通常は、複数の様式を組み合わせる

今日の内容

- 設計のプロセス
- 品質を満たすための基準
 - ソフトウェアユニットの構成法が種々の“品質”に及ぼす影響

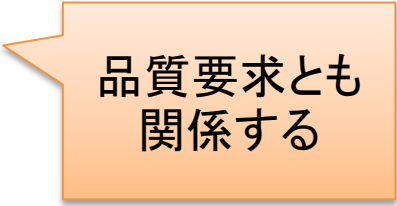
設計のプロセス

- ソフトウェアシステムの設計は反復作業
- 最終成果物は、基本設計書



品質の追求

- アーキテクチャーの様式によって得られる効果は全体的なもの
- より細かくアーキテクチャーの善し悪しを検討しなければならない
- アーキテクチャーを評価する“品質観点”
 - ① 更新性
 - ② 性能
 - ③ セキュリティー
 - ④ 信頼性
 - ⑤ 堅牢性
 - ⑥ 可用性
 - ⑦ ビジネス価値



品質要求とも
関係する

更新性

- 設計は変更が容易であることが好ましい
- 変更によって受ける影響の違いの種類
 - ➡ 直接的に影響を受けるもの
 - 変更に合わせて責任範囲が変わる
 - ➡ 間接的に影響を受けるもの
 - 責任範囲は変わらないが、実装を見直す必要がある

更新性(2)

- 直接的影響を受けるソフトウェアユニットを最少にする
 - 変更が見込まれるユニットをまとめる
- 変更を見込む
 - 変更の可能性が高い設計上の条件を特定し、一つのソフトウェアユニットにカプセル化する(納める)
 - 変更があった場合そのユニットのみに変更を収めることができる
- 集合性を高める
 - 同じ役割に属するソフトウェアユニットをまとめて管理する
 - 役割が変わった場合の変更範囲を最小化できる
- 一般化する
 - ソフトウェアユニットが一般的であればあるほど、その中身を変えずに、入力の変更だけで対応できる可能性が高くなる
 - 抽象化、継承の親クラス、など

更新性(3)

- 間接的影響を受けるソフトウェアユニットを最少にする
 - 依存性を最少化する
- 疎結合
 - 結合を弱めることで変更が伝播することを防げる
→共有するデータ、関数の呼び出し、など
- インタフェースを変更しない
 - ユニットが自身のインタフェースだけを通して他のユニットと通信をしていれば、変更はそのユニットの境界を越えない
- インタフェースの追加
 - ユニットが新たなデータやサービスを提供しなければならないとき、新たなインタフェースを追加して対応すれば、既存のインタフェースに変更は生じない

性能

- 処理スピードや容量などシステムの性能属性を評価する
 - 応答時間
 - 一件の要求への応答にかかる時間は？
 - スループット
 - 1分間にいくつの要求に応答できるか？
 - 負荷
 - 応答時間やスループットを低下させずに対応できるユーザの数は？
- 性能向上の方法
 - リソースを有効活用する
 - リソース割当の効率化
 - 早い者勝ち: 要求を受け取った順番に処理する
 - 優先順位付け: 要求に指定された優先順位の順番で処理する
 - 期限順: 要求に求められる期限が早い順番で処理する
 - リソースへの要求を減らす

セキュリティ

- アーキテクチャーでセキュリティを考慮する
- **耐性**: 攻撃を阻止する能力
 - アーキテクチャーの設計で考慮できること
 - セキュリティ上の対策をすべて設計に含める
 - 悪用できる弱点を最少にする
- **回復力**: 攻撃から素早く簡単に回復する能力
 - アーキテクチャーの設計で考慮できること
 - 機能を断片化して攻撃の広がりを阻止する
 - 機能を素早く回復できる考慮

信頼性

- ソフトウェアシステムを信頼できる条件
 - 要求された機能を、想定される条件下で、正確に果たすこと
 - ソフトウェアの中身にエラーが無いことと関係する
- 「誤り」と「障害」
 - 人のエラーで起きるのが誤り
 - 要求された機能から外れることが障害
 - ➡ 誤りを防いだり許容できるようにすることでソフトウェアの信頼は向上する

信頼性(2)

■ 受動的誤り検知

- － 実行時に誤りが起きるのを監視する

■ 能動的誤り検知

- － 定期的に誤りの兆候を探したり、障害の発生を予測する

□ 例外

- － 期待されない動作の原因となる状態

□ 例外処理

- － 例外の発生を処理してシステムを正常な状態に戻す仕組み

□ 一般的な例外の種類

- － サービスの提供に失敗した
- － 要求と異なるサービスを提供した
- － データを破壊してしまった
- － 定められた条件(セキュリティー条件など)に違反する動作をした
- － デッドロックに陥った

信頼性(3)

• 障害回復

– 障害に即時に対応し、被害を最小化する

トランザクション取り消し

- 途中で障害が起きても安全に取り消せる単位で処理をまとめて(トランザクション)管理する

チェックポイント／ロールバック

- 現在の状態をチェックポイントとして保存し、その後にシステムで問題が起きたら、そのポイントまでシステムの状態を戻す(ロールバック)

バックアップ

- 障害のあるユニットをシステムが自動的にバックアップと置き換える

制限付きサービス

- 障害発生前の状態に戻り、障害に影響されない範囲のサービスを提供する

修正・継続(自己治癒)

- 問題を検知し、症状を治して継続する

報告

- 障害発生前の状態に戻り、障害が起きたことを例外処理ユニットに報告する

堅牢性

- 堅牢なシステムの条件
 - 不正な入力や予期しない条件の中でも正確に動作する
 - 環境や他のユニットに問題が発生しても、正確に動作を続ける
- “相互不信”が前提
 - 他のユニットに障害が起きることを想定する
- ➡ 堅牢性を実現する手法は信頼性の手法とは異なる
- ➡ 回復手段は信頼性の場合と同じ
 - トランザクションの中止
 - チェックポイントへのロールバック
 - バックアップの起動
 - 制限付きサービスの提供
 - 自己治癒
 - 例外の発生を報告

ユーザビリティ

- ユーザビリティとは、ユーザによるシステム操作がどれだけ容易かを表す
 - ユーザインタフェースは、他の機能等とは分離して、独立したソフトウェアユニットに割り当てるべき
 - ユーザが起点となるコマンドの中には、アーキテクチャーの配慮が必要なものもある：例) undo
 - システムが起点となる活動の中には、環境のモデルを必要とするものがある：例) 時間指定の処理

ビジネス価値

- ビジネス・ゴールとは
 - システムが発揮しなければならないビジネス上の特性
 - 開発コストの最小化、開発工期の短縮、など
- 開発vs.購入
 - 開発工期、費用の節約
 - 信頼性の確保
 - 既存コンポーネントを利用した場合の制約
- 初期開発費用vs.管理コスト
 - 更新性の高いシステムにして費用を削減
 - 複雑なシステムは納期が遅れる: 競合他社に顧客を奪われる
- 最新技術vs.枯れた(安定した)技術
 - 新たに身につけるのはコストがかかり、納期も遅れる
 - 新しい技術を学ぶか、新しい人材を雇う
 - 将来的には、いずれは獲得しなければならい技術もある

本日のまとめ

- システムが満たさなければならない品質要求を考慮した選択
 - “品質観点”で評価し、様式を比較検討する
 - 信頼性、堅牢性、セキュリティの対策が大きい

次回講義の事前学習:

7.4.4