アーキテクチャーと分解手法

FU14 ソフトウェア工学概論

第4回

吉岡 廉太郎

前回の内容

- 要求記述言語
  - 記述作業で用いるモデルの指針
  - 課題の性質と開発チームの経験によって使い分ける
- ドキュメント
  - 要求定義：お客さんの言葉で表現
  - 要求仕様：開発者の言葉で表現
- 確認は2段階
  - 妥当性確認：正しいシステムを作っているのか
  - 検証：システムを正しく作れているか
- 要求をテストして評価する

今日の内容

- 設計のプロセス
  - 目的と作業の特徴を理解する
- 設計（分解）手法
  - システムを分解するさまざまな方法を知る
- アーキテクチャ設計のビュー
  - システムの性質を漏らさずとらえる観点を知る

要求定義の次は設計

- システムに対する要望が決定
  - 要求定義：注文主の要望
  - 要求仕様：期待されるシステムの振る舞い
- システムをどのように実現するか
  - アーキテクチャの設計：大きな構成を決める
  - 詳細設計：データ構造やアルゴリズムを決める
- アーキテクチャの設計
  - 基本設計を行う
  - 「基本設計」、「分析」工程とも呼ばれる
  - 教科書では「モジュール化」と呼んでいる

設計のプロセス

5.1.1
5.1.2

- 「設計」とは
  - 要求をどのように実現するか考える
  - 創造的プロセス
  - このプロセスの成果物のことを「設計」と呼ぶ

- 設計のステップ
  - 初期段階の設計はアーキテクチャを決める
  - 後半の設計は個々の部品をどのように実装するかを決める

設計は創造のプロセス

- 設計は高度に知的な作業が必要
  - システムが直面する無数の状況への対応
  - 品質要求の実現（使い易さ、管理のしやすさ、…）
  - 外的制約への対応
    （規定のデータ形式、法的規制、…）

- 課題の分解
  - 扱いやすい/理解しやすい単位に分ける
  - 分け方＝アーキテクチャ（モジュール化とも呼ばれる）

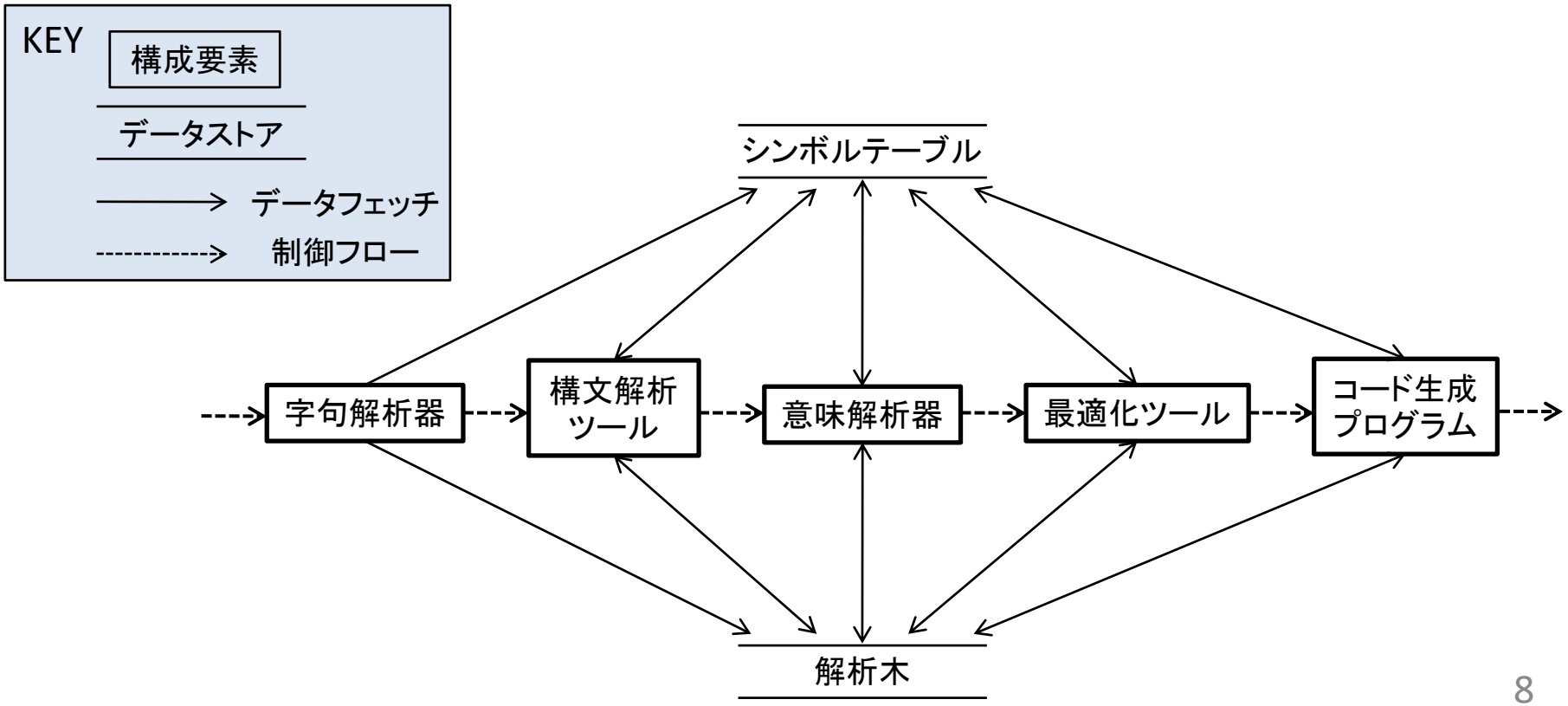
設計の進め方

- 過去の設計から学び、活用する
  - 同様の問題に対する過去の解決策を再利用する
  - 状況に合わせて変更するだけで解決できる事もある

- 既存の解決策を活用する方法
  - 複製（クローン）
    設計/コードをほとんどそのまま使う
  - 参照モデル
    アプリケーション分野ごとにシステムの分解方法を示した汎用的なアーキテクチャを当てはめる

参照モデルの例

- 例:コンパイラーの参照モデル
  - 他には、OS、データベース、通信ネットワークなど



参照モデルの限界

- 新しい課題の場合、そのまま適用できることは少ない
  - 実世界のシステムは複雑で2つと同じシステムはない
  - 特定分野に特化した参照モデルでは対応しきれない

- より一般化された解決策：アーキテクチャ・スタイル
  - 分野に依存しないソフトウェア・アーキテクチャ
  - 複数のスタイルを組み合わせて、目的のソフトウェアを実現することができる

アーキテクチャ・スタイルの例

9.4.1

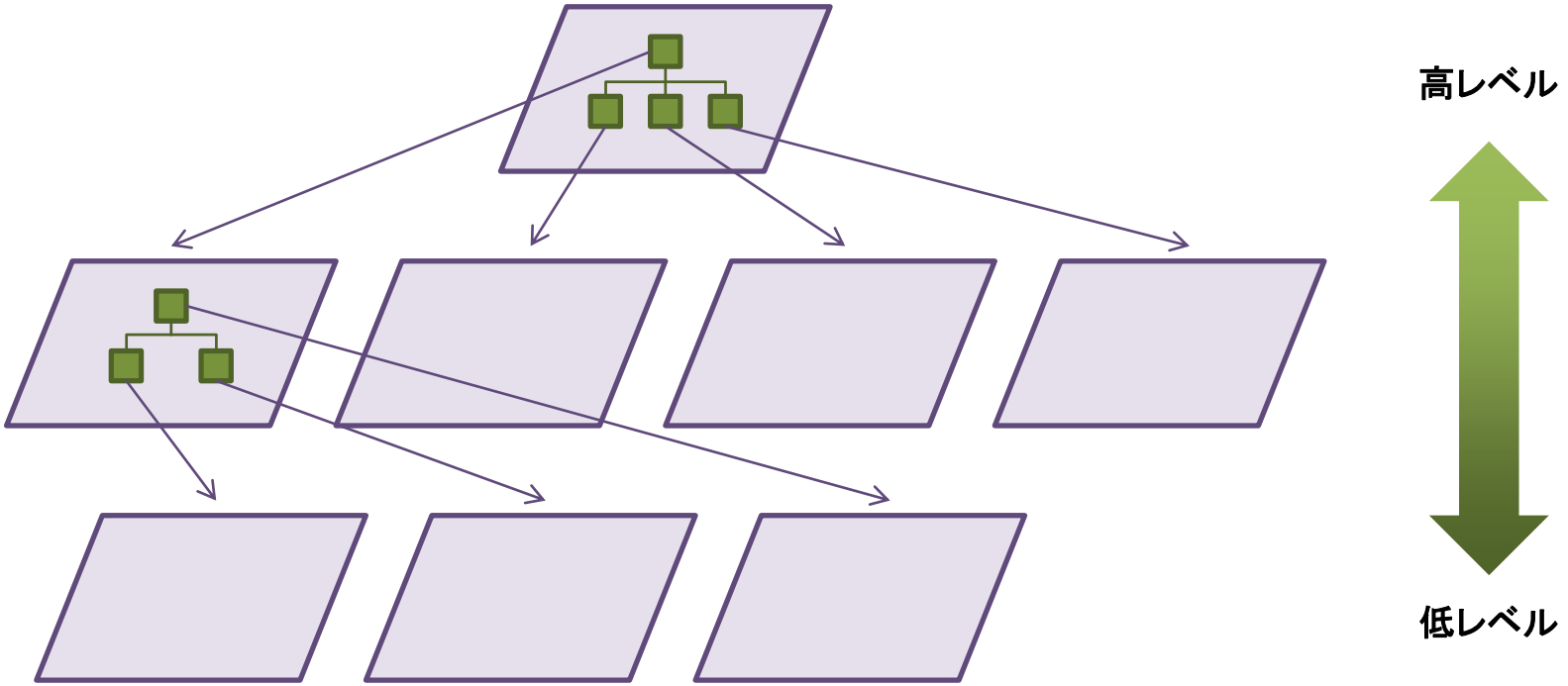
- アーキテクチャの選択肢を提案するガイドライン

デザインパターン	• 汎用性の高い解決策を示す • 低レベルの設計に対する具体的解決策
デザイン コンベンション	• 一定の価値基準に基づいた、設計の指針やアドバイス • 従うべき基準と設計の方法を示す
デザイン プリンシプル	• 良い設計の基準を示し、どのように設計するかまでは決めない • 既存の解決策ではなく新しい設計を産み出すのに適する

5.3.2

トップダウンによるシステム設計

- システムを段階的に分解しながら設計する
  - 徐々に理解を深める
  - レベルが下がるにつれ詳細化する



分解手法を用いた設計

- 既存の解決策が当てはまらない課題にも有効
  - 分解手法を用いて整理する

システムの分解
＝モジュール化

- 分解手法の種類
  - 機能分解
  - データ指向分解
  - プロセス指向分解
  - イベント指向分解
  - オブジェクト指向分解

機能分解

- 機能または要求の単位で分ける
  - 一つの単位をモジュールと呼ぶ
- 高レベル設計
  - 要求仕様書に記載された機能をモジュール化する
- 低レベル設計
  - 各モジュールをより小さなモジュールに細分する
  - モジュール間の呼び出し関係を表す

データ指向分解

- データの単位でモジュール化する
- 高レベル設計
  - 概念的なデータ構造を表す
- 低レベル設計
  - モジュール間のデータ分布
  - 各データと概念的データ構造との関係を記述

プロセス指向分解

- システムで実行されるプロセスをモジュール化

- 高レベル設計
  - システムの主要なタスクを明らかにする
  - タスクを実行プロセスに割り当てる
  - 各タスク間の連携を記述する

- 低レベル設計
  - プロセスをより詳細に記述する

イベント指向分解

- システムで発生するイベントをモジュールに割り当てる

- 高レベル設計
  - 想定される入力イベントを列挙する

- 低レベル設計
  - システムの状態を列挙し、イベントがどのように状態遷移を引き起こすか表現する

オブジェクト指向分解

- オブジェクトをモジュールに割り当てる

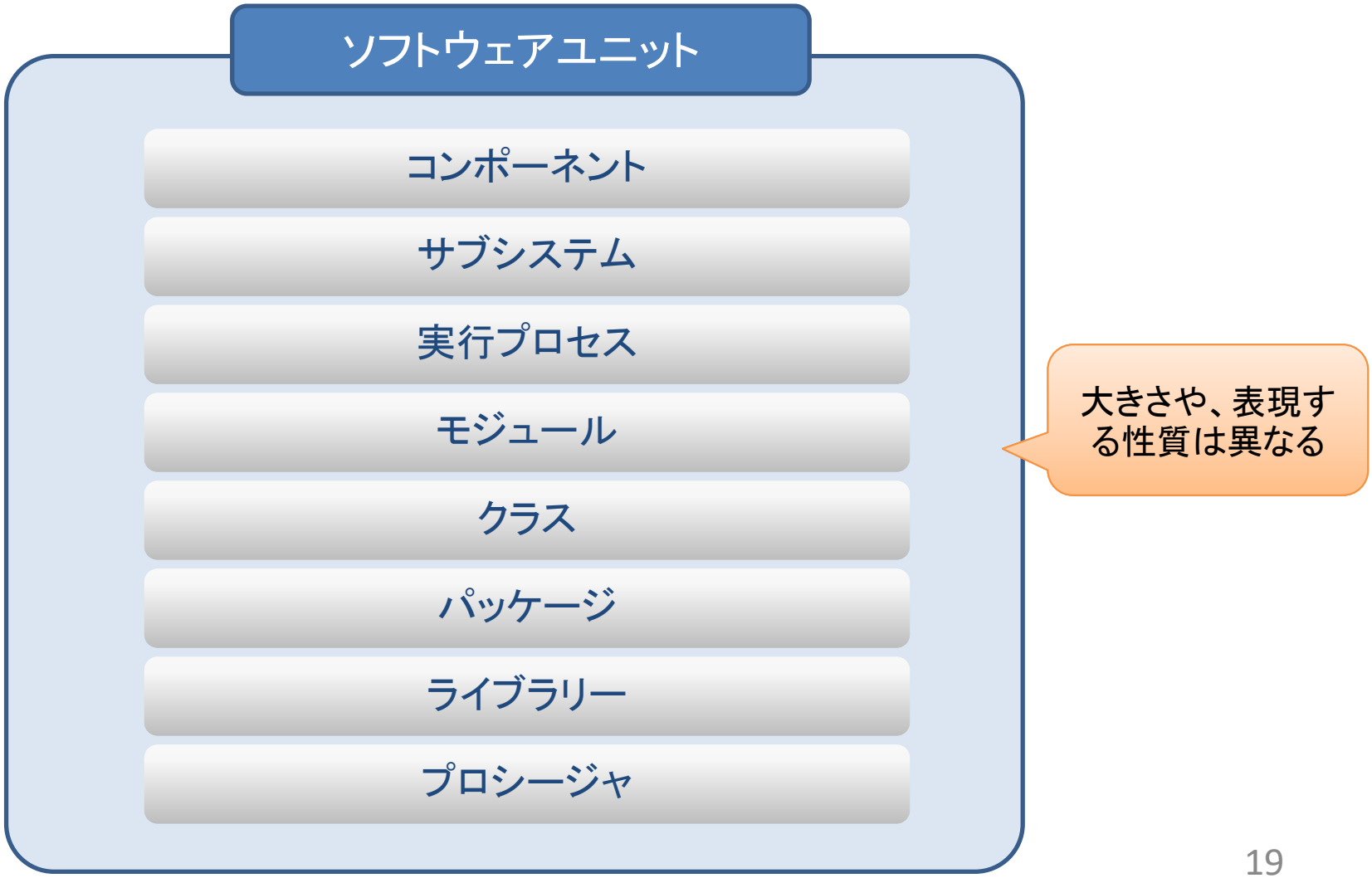
- 高レベル設計
  - システムのオブジェクトの種類とオブジェクト間の関係を記述する

- 低レベル設計
  - 各オブジェクトの属性と操作を詳細化する

分解手法の選択方法

- システムの性質に応じて選択
  - システムを特徴付ける仕様は何か
    - 機能、オブジェクト、…
  - システムのインタフェースの種類
    - 入力イベント、データ・ストリーム、…
- 実際の設計では、複数の手法を用いる
  - 複数の手法で記述し、比較検討する
  - 設計のレベルに応じて異なる手法を使う

ソフトウェアを構成する単位



分解(設計)の評価基準

- モジュール化(されている)
  - システムの各機能がそれぞれただ一つのソフトウェアユニットで実行され、それらユニットの入力と出力が明確に定義されていること
- 明確さ
  - ソフトウェアユニットのインタフェースが正確にそのユニットの外見上の振る舞いを定義していること

5.8.1

横断的な特徴を捉える

- 分解手法の長所
  - システムをプログラム上の単位に分解する

- 複数のソフトウェアユニットにまたがる横断的な特徴を捉えられない
  - 複数のコンピュータに分散した処理やサービスの関係
  - システムが提供するサービスと、それらのサービスの連携、など…

- システムの実装上の分解に縛られない表現方法
  - アーキテクチャーのビュー

5.7.2

アーキテクチャー設計におけるビュー

5.7.2

論理ビュー

分解

- システムをプログラム可能な単位で記述する
- 一般的に階層的な表現になる
- 複数のモデルを使って表すことができる

分解手法で見てきたとおり

依存関係

- ソフトウェアユニット間の依存関係を記述する
- プロジェクトの計画に役立つ
- ソフトウェアユニットに設計変更を加えた場合の影響範囲を評価するのに役立つ

一般化

- ソフトウェアユニットの一般化・特化関係（継承関係）を記述する
- 抽象または拡張可能なソフトウェアユニットを設計する際に役立つ
- 検証しやすい、再利用しやすい

5.7.2

その他ビュー

実行

- システム内の実行順序を、コンポーネントとそれらを結ぶコネクターで記述する
- 各コンポーネントは一つの実行の単位を表す
- コネクターは、コンポーネント間の通信手段（通信路、データ、…）を表す

開発

- ソフトウェアユニットとそれを実装したソースコードの対応を記述する
- ソースコードの検索を助ける

配置

- 実行時の単位となるソフトウェアユニットを物理的リソース（プロセッサ、記憶装置、…）に対応づける
- 品質に関わる属性（パフォーマンス、信頼性、セキュリティー、…）について分析するのに役立つ

役割分担

- システムの設計を、開発チームメンバーに割り当てる作業として分解して記述する
- プロジェクト管理において、リソースの確保と進捗の確認に役立つ

本日のまとめ

- 設計は創造のプロセス
  - 大きく、複雑なシステムを、扱いやすい単位に分解してとらえる
  - 既存のモデルを当てはめるだけで完成することはまずない
  - 設計は要求に対する理解を深めるプロセスでもある
  - 工夫と反復が不可欠
- 分解手法
  - システムをソフトウェアユニットに分解する具体的ガイドライン
- アーキテクチャの観点
  - 一つの分解手法だけでは捉えきれないシステムの性質を明らかにする
  - 検討漏れが無いかを検証するのに役立つ

次回講義の事前学習：
5.7.1, 5.7.2 , 9.4.1, 9.4.3