ソフトウェア工学 第6回

アーキテクチャ設計
ユーザインタフェース設計

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

- アーキテクチャ設計
- ユーザインタフェース設計

ソフトウェアアーキテクチャ (software architecture)

- ソフトウェアの全体的な構造であり、システムの概念的な一貫性を提供する手段
  - モジュールの構造や構成
  - コンポーネント間の相互作用
  - コンポーネントが使用するデータの構造 等
- (広義には、システムの主要な構成要素とそれらの相互作用)

- 開発を方向付ける
  - 基本構造から大きく外れる改造は困難
  - 作業の分担や手順を検討する際の基準
  - 再利用資産の蓄積と活用

ソフトウェアアーキテクチャ

(例) 家の基本構造と特性

基本構造	耐火性	耐衝撃性	建設期間
わら造り	弱	弱	短
木造	中	中	中
レンガ造り	強	強	長

トレードオフを考慮して決定する

- アーキテクチャを決めると建設方法も(ある程度)決まる
  - わら → わらを集める → わらを束ねる …
  - 木 → 木を切る → 木を組む …
  - レンガ → レンガを焼く → レンガを積む …
- アーキテクチャを決めると最終製品の性質も(ある程度)決まる
  - わら造りで2階建ての家屋を建築？
  - 木造平屋を建て増しして2階、3階を積み重ねる？

ソフトウェアアーキテクチャ

(例) ソフトウェアの基本構造と特性

基本構造	信頼性	機能性	保守性
アーキテクチャA	低	低	高
アーキテクチャB	中	中	中
アーキテクチャC	高	高	低

- アーキテクチャを決めると開発手法も(ある程度)決まる
  - モジュール分割と構造化の進め方
  - プログラムとツールの再利用の進め方
  - 品質特性の評価の進め方
- アーキテクチャを決めると最終製品の性質も(ある程度)決まる
  - 求められる品質をあらかじめ考えておかなければならない
  - 他、どのような環境で動作するか 等々…

アーキテクチャに関わる品質特性の例

- 実行時の品質特性

  (例)効率性(時間的効率性)

  ・どのコンピュータにどの役割を持たせるかによって性能が変わる

  ・性能に影響を与える要因も変わる(Bであれば通信の影響有)

- 開発時の品質特性

  (例)保守性(変更性)

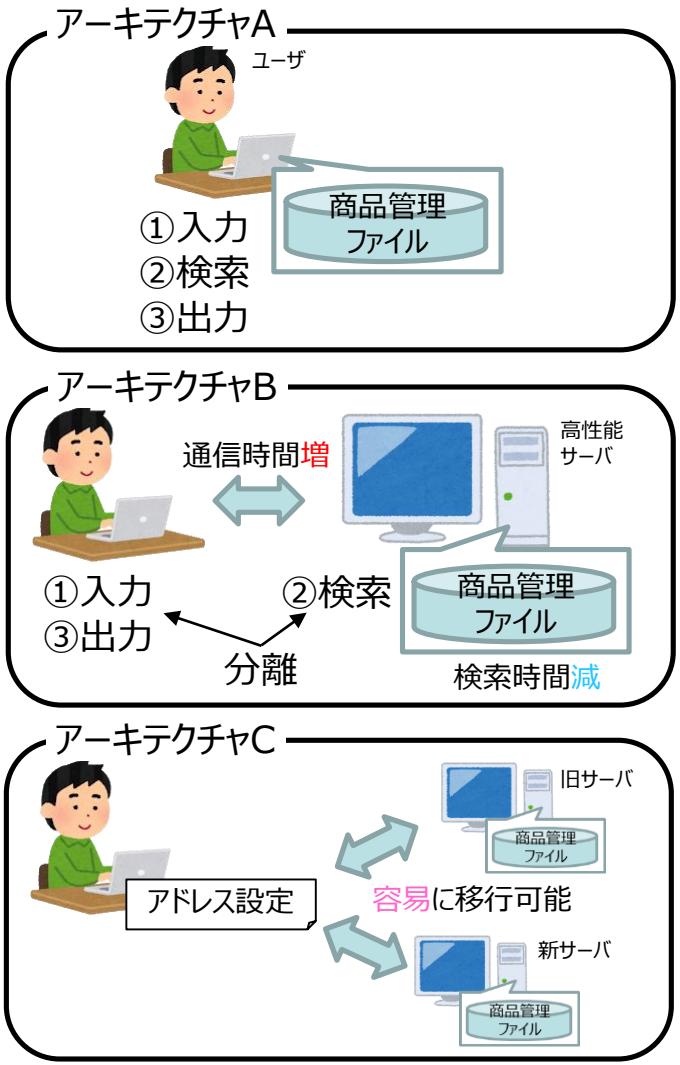
  ・商品管理ファイルの変更が想定される場合は②を分離した方が良い(Bであれば①や③に影響を与えにくい)

- 運用時の品質特性

  (例)可搬性(順応性)

  ソフトウェアを別の環境へ移すときの手間

  ・サーバのアドレスがプログラム中で固定(B)
  ⇔サーバのアドレスは設定ファイルで設定(C)



アーキテクチャ設計

- アーキテクチャを決定する作業
  - 機能要求だけでなく、品質要求も満たすように基本構造を検討
  - 開発作業の方針決定や詳細設計より前に行う
  - 具体的なソフトウェアが存在しない状態で行うため、
    高度な知識・経験が必要
- 設計上の課題
  - 設計の基本方針
    - 開発期間、開発費用、開発形態、利用期間、利用形態、
      開発後の取り扱いなど評価基準の明確化
  - 評価の視点
    - 評価する視点を定め、その視点に基づく構造の表現（ビュー）
  - 評価の追跡可能性
    - トレードオフに対する判断の記録

アーキテクチャを評価する視点

- 論理ビュー
  - システムがどのような機能を持つか
  - 機能がどのような論理的構造を持つか
- 実行ビュー
  - システムがどのような実行単位で構成されるか (プロセス、タスク等)
  - 性能などの実行時の品質特性に着目
- 開発ビュー
  - システムがどのようなディレクトリ構造で管理されるか
  - 保守性や修正容易性などの品質特性に着目
- 配置ビュー
  - システムを配置したときの構造(どのコンピュータの どのプロセッサにソフトウェアを置くか等)
  - 実行時の性能、信頼性、安全性等に着目

アーキテクチャの設計手法

- 標準化された設計手法、定着した手法は存在しない
- Boschの手法（ボッシュ）
  1. 機能面からの設計
     - 機能に関する要求を満たすようにアーキテクチャを設計する
  2. 品質特性を評価
     - 設計したアーキテクチャが非機能要求を満たすか評価する
       → 満たせば設計完了
     - 要求された品質特性を満たさなければアーキテクチャ変換(3へ)
  3. アーキテクチャスタイルを用いて要求を満たすように
     アーキテクチャを変換

アーキテクチャスタイル (architecture style)

- ソフトウェアの基本的構造を表すもので、システム全体の設計に制約を課す
- システムのすべてのコンポーネントの構造を確立することが目的

- ソフトウェアの代表的な基本的構造
  - 機能分割に基づく分類
    - 階層モデル (hierarchical model)
    - クライアントサーバモデル (client-server model)
    - リポジトリモデル (repository model)
  - 制御関係に基づく分類
    - データフローモデル (data flow model)
    - コントロールモデル (control model)

階層モデル

- 機能等を複数の層で捉えたモデル

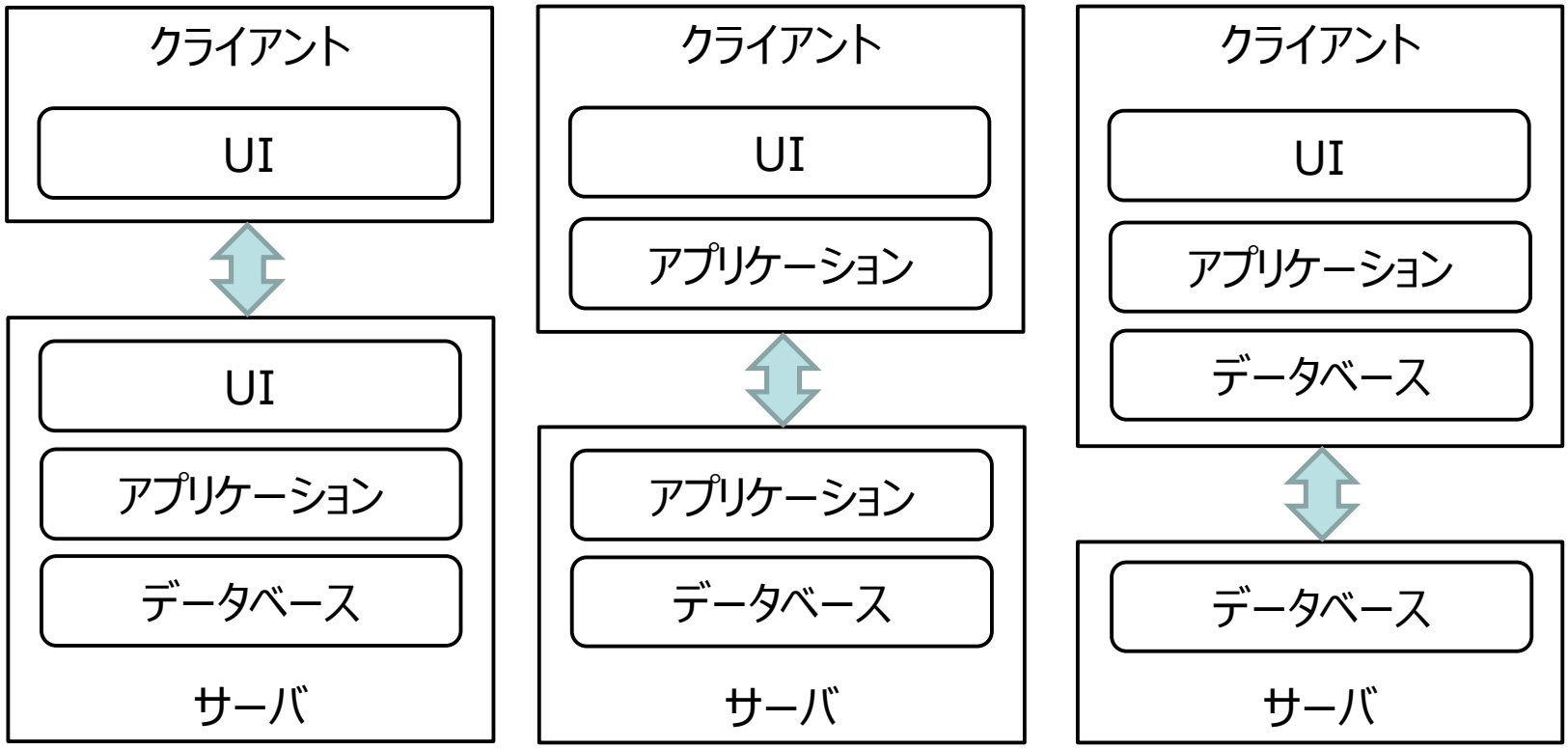
ISO 7層モデル

Webアプリケーション
3層モデル

仮想マシンモデル

クライアントサーバモデル

- データと処理機能をクライアントとサーバに分けて運用する分散システムモデル

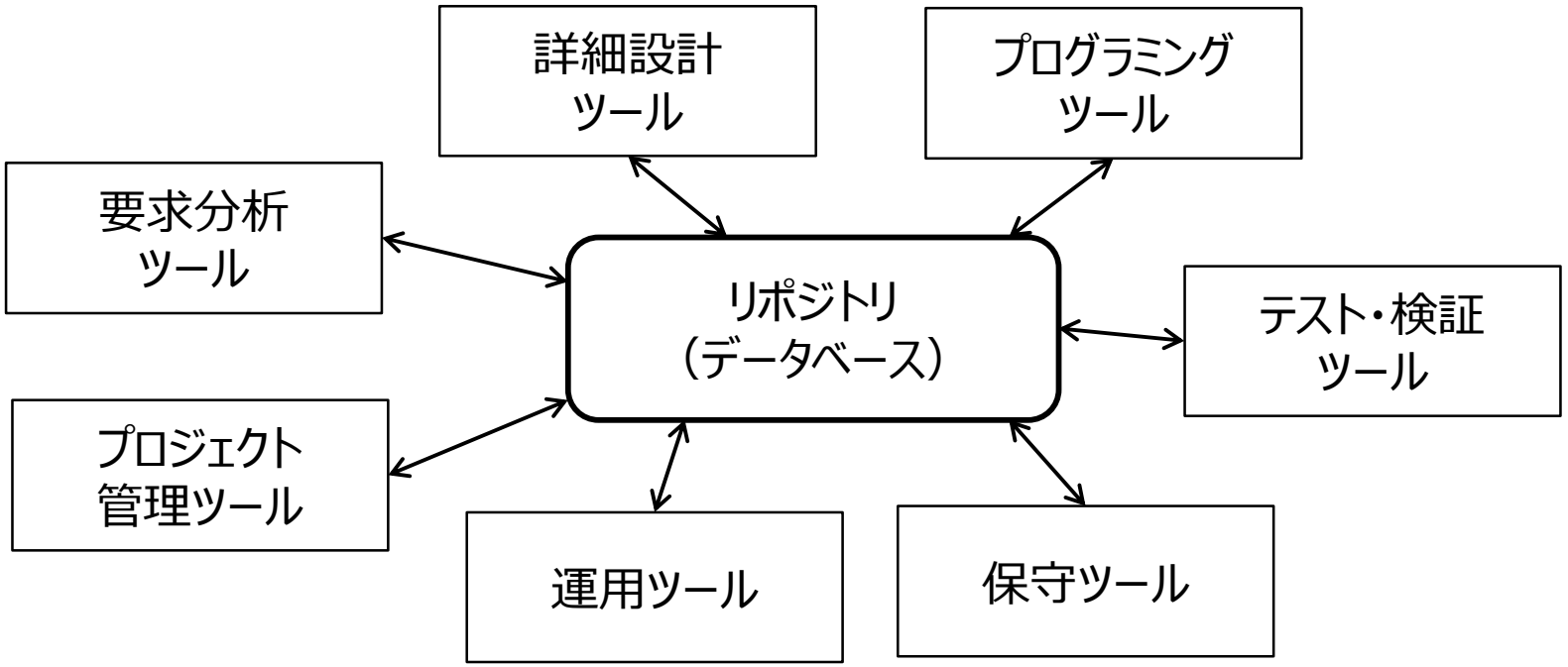


シンクライアント (thin client) モデル

ファットクライアント (fat client) モデル

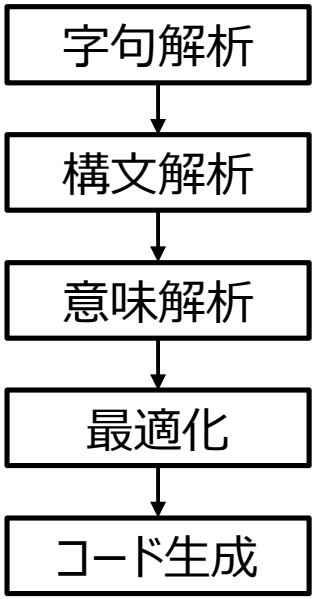
リポジトリモデル

- データ中心モデルとも呼ぶ
- 複数のサブシステムが共有データ(リポジトリ)を介してデータ交換する

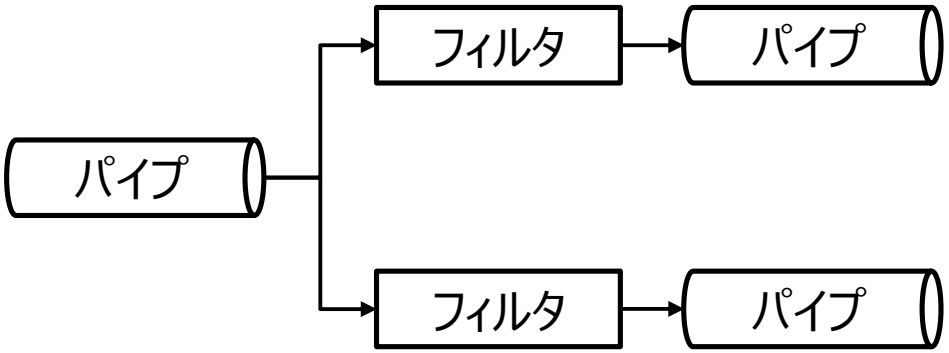


データフローモデル

- 入力データを処理して出力を生成する変換機能により構成される



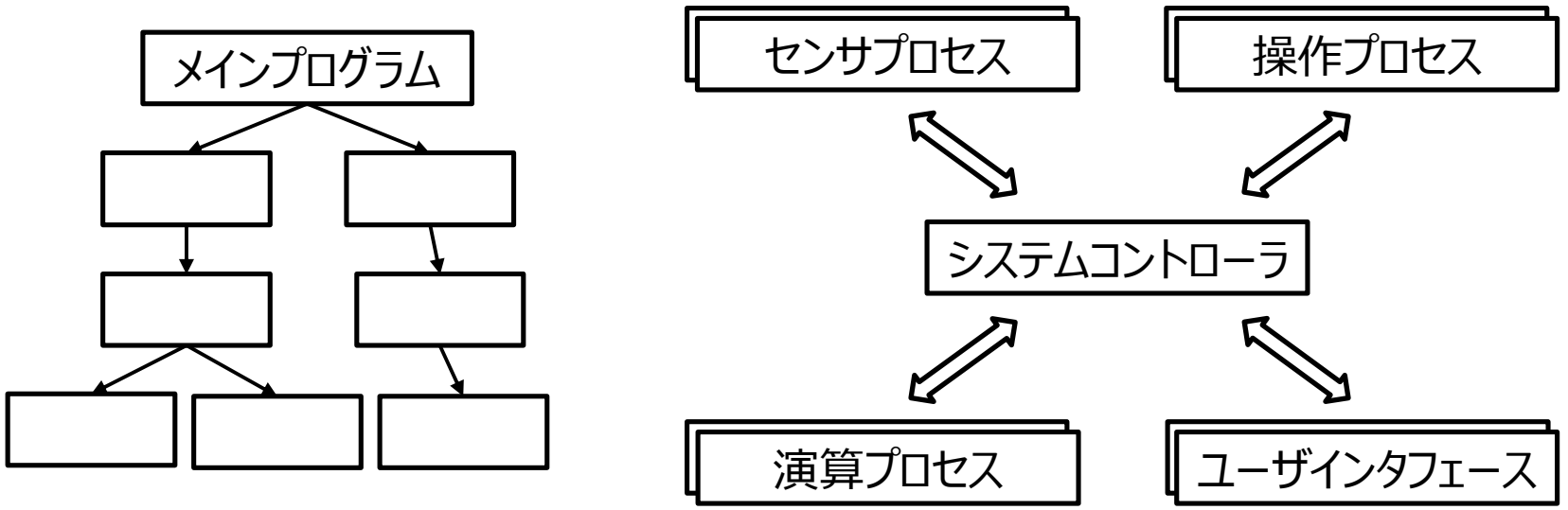
コンパイラ
(compiler)



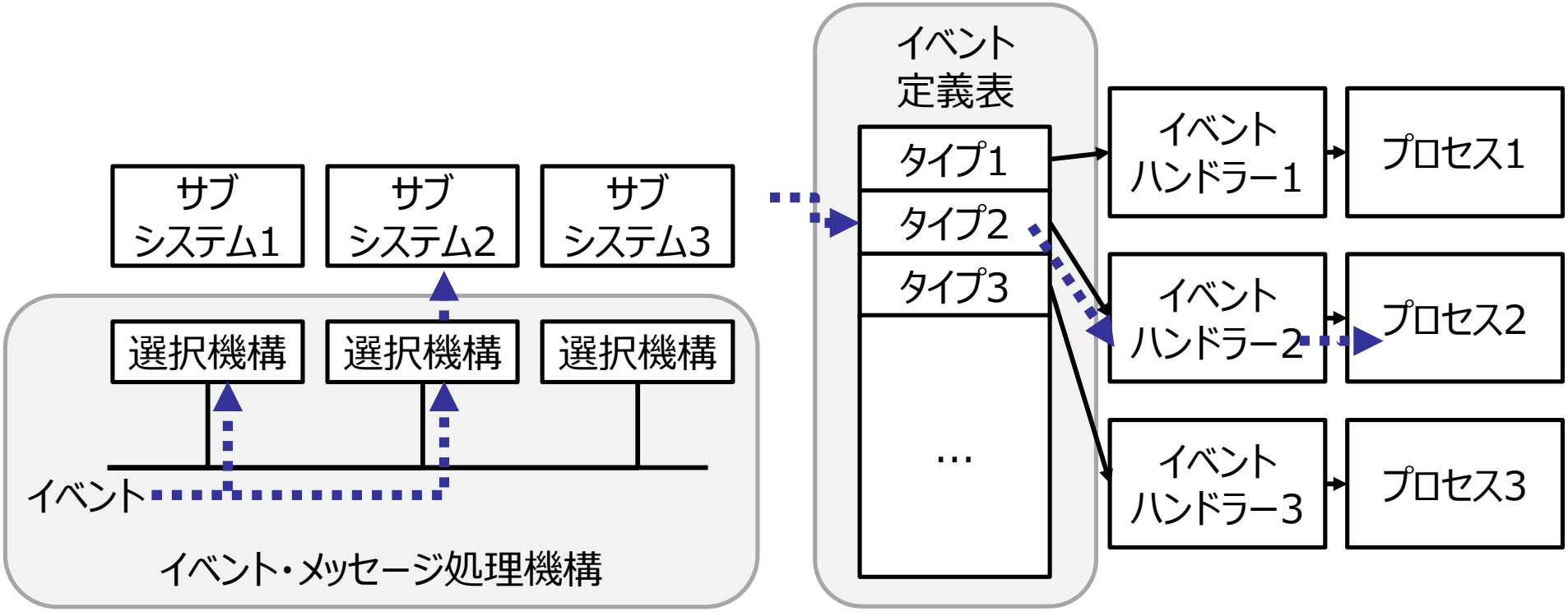
パイプとフィルタ
(pipe & filter)

コントロールモデル

- サブシステム間のコントロールに注目したモデル
- 集中型コントロールモデルとイベント駆動型コントロールモデルに分けられる



コントロールモデル



ブロードキャストモデル
(broadcast model)

割り込み駆動型モデル

イベント駆動型コントロールモデル

確認問題

- ソフトウェアアーキテクチャに関して説明した下の各文は正しいか。○か×で答えよ。
  - アーキテクチャと開発手法は互いに独立であるため、開発手法決定後にアーキテクチャは容易に変更できる。
  - 最終的なソフトウェアの品質はアーキテクチャの影響をあまり受けない。
- 以下の各説明に合うソフトウェアの基本的構造(アーキテクチャスタイル)の名称を答えよ。
  - 機能を複数の層で捉えたモデル
  - データと処理機能をクライアントとサーバに分けて運用する
  - 複数のサブシステムが共有データを介してデータ交換する
  - 入力データを処理して出力を生成する変換機能を複数連ねることで構成
  - サブシステム間のコントロール(制御)の形態に着目

講義内容

- アーキテクチャ設計

- ユーザインタフェース設計

ユーザインタフェース設計

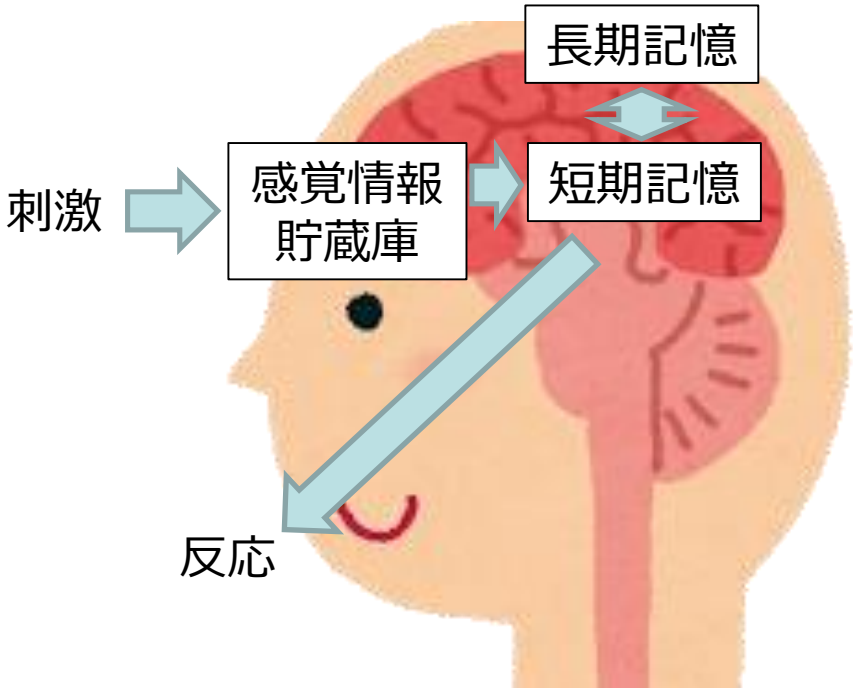
- ユーザインタフェース (UI: user interface)
  - ユーザとシステムの対話を仲介
- ユーザインタフェース設計
  - 設計指針に基づいてユーザインタフェースを定める作業
  - ユーザの負担を最小限にすることが目的
    - システムがユーザの意図を正しく把握する
    - ユーザに実行結果を分かりやすく提示
    - 使いやすい入力方式（どこに何を入れるのか分かりやすく）
  - 人間の認知機能を考慮する必要
    - 一度に多くの情報を提示しすぎない
    - 追従可能な速度での画面の切り替え等
    - ユーザモデルの構築も重要
      - 例：上級者、中級者、初心者ごとにUI設計

人間の情報処理モデル

(1) 目や耳から入った情報は、
非常に短い間、ほぼそのままの形で
感覚情報貯蔵庫に保存される
- ほとんど変わらない情報が続いて入ると連続した映像として認知
- 短い間隔で異質な情報が入ると情報欠落

(2) 20秒程度短期記憶に保存
- 頭の中で高速に利用可能
- 保存可能なチャンク(chunk)は7±2個

(3) 長期記憶にある知識を使って
符号化されて長期記憶に格納される
- 容量は膨大
- 引き出しに時間がかかったり、失敗する (引き出しに工夫が必要)



設計ガイドライン

- 観点１：対話の道具
  - 親しみやすいUI
    - 日常的な概念を使う(例：文書、フォルダ)
  - 操作手段・用語・コマンド形式の一貫性
- 観点２：対話の進め方
  - フィードバックの提供
    - ユーザの入力に対するシステムの応答
  - 驚きを最小にする
    - ユーザのメンタルモデル(mental model:ユーザの頭の中のどのように動作するかのイメージ)に沿った振る舞い
- 観点３：エラーへの対応
  - エラーからの回復を容易にする
    - 例：アンドゥ(undo)機能
  - ユーザへのガイダンス
    - 例：ヘルプ機能、オンラインマニュアル

※イメージ：頭の中で思い浮かべる姿･形。心像、形象。

設計プロセス

- 他の要求同様、UIに関しても要求分析が重要
  - UIの変更がソフトウェアの基本構造に影響を及ぼすこともある
    → できるだけ早い段階でのユーザによる評価が必要

- プロトタイプによる評価
  - 要求分析時に、ユーザが行う操作を理解し、
    UIプロトタイプを作成
    - ユーザ評価で満足できる結果になるまで改善を繰り返す
  - 実行可能なプロトタイプを作成
    - ユーザ評価で満足できる結果になるまで改善を繰り返す
  - 最終的なシステムの実現
    - ユーザと一緒に最終評価

UIの評価

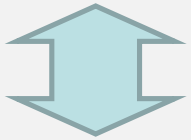
- 一般に人的要因が影響する性能の客観的評価は困難
- Shneidermanによる測定可能な人的要因
  - 学習時間
    - 必要な操作の使用方法を覚えるために必要な学習時間
  - 実行速度
    - タスクの遂行に要する時間
  - エラーの割合
    - タスクの遂行中に発生した(人間の)エラーの種類と回数
  - 長期記憶
    - 一度習得したタスク遂行に必要な知識を覚えている程度
  - 満足度
    - システムを使用してどの程度気に入ったかという主観的評価

※タスク：ここでは、「人間が行うひとまとまりの作業」の意味

対話方式 1/2

- コマンド言語：ユーザはキーボードからコマンドを入力する
  - 利点
    - 安価な端末で実現可能
  - 欠点
    - コマンドを覚えなければならない
    - タイピング能力が必要

CUI: character user interface



GUI: graphical user interface

- メニュー選択：ユーザはメニュー項目から選択して実行
  - 利点
    - ユーザのエラーが少ない
    - 覚えておかないといけないことが少ない
    - 選択過程を追跡することで状況に応じた支援が可能
  - 欠点
    - 選択項目が多い場合は選択が困難
    - 複数のコマンドを組み合わせて動作させることが困難
    - 経験豊富なユーザはマウスの使用を煩わしく感じる

対話方式 2/2

- 直接操作
  - オブジェクトを常に表示しておき、マウス等で操作可能
  - 操作の結果が素早く表示され、かつ、可逆的
    → 初心者でも習得が容易
- (例) Windowsデスクトップ環境
  - 利点
    - 操作が直感的でわかりやすい
    - フィードバックがすぐに得られる
  - 欠点
    - 適切なメタファ(比喩, metaphor)で
      表現することが難しい
    - プログラムが複雑になる
    - ハードウェアに対する要求が大きい
    - 操作の繰り返しや履歴の追跡が
      コマンド言語やメニュー選択と比べて困難

出力画面の設計

- 設計における注意点
  - ユーザの関心
    - 関心のあるものを的確に表示する
  - 情報の価値
    - 表示することに価値があるのか、
      価値がどのように変化するのか
  - 表示方法の特徴
    - テキスト(文字列) vs ビジュアル(視覚的)、
      アナログ vs デジタル
    - 色の表示法
      - 色の数・一貫性、色による強調・状態表現
    - 大量情報の可視化
      - Shneidermanの指針：
        「まず全体の概略を示し、その中から重要な情報の拡大と選別をして、ユーザの要求に応じてさらに詳細な情報を提示しなさい」

確認問題

- 次の説明に合う語を答えよ。
  - ユーザとシステムの対話を仲介する機構
  - ユーザの入力に対するシステムの応答
  - ユーザが持つ、ソフトウェアがどのように動作するか等のイメージ
  - ソフトウェアの試作品のうち、特にUIの評価を目的としたもの
  - 計算機のUIにおける比喩

確認問題

- 空欄に当てはまる語を答えよ。
  - 人間の情報処理モデル
    - 目や耳から入った情報は、非常に短い間、
      ほぼそのままの形で(　1　)に保存される。
    - (　1　)から抽出された情報は、
      20秒程度(　2　)に保存される。
      保存可能な情報は(　3　)±2チャンク程度とされている。
    - さらに、一部の情報は符号化されて(　4　)に蓄えられる
      (　4　)は(　2　)と比べると容量が大きいが、
      引き出しに時間がかかったり、失敗したりする。
  - Shneidermanの指針
    - 「まず全体の(　5　)を示し、その中から重要な情報の
      拡大と選別をして、ユーザの要求に応じて
      さらに(　6　)な情報を提示しなさい」

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
- 「実践ソフトウェアエンジニアリング」
  ロジャー・プレスマン 著、西康晴、榊原彰、内藤裕史 監訳、
  古沢聡子、正木めぐみ、関口梢 翻訳、日科技連、2005