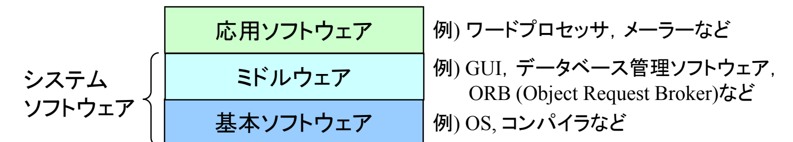
2009年度 ソフトウェア工学 講義資料

ソフトウェア工学

情報理工学部 情報システム学科 桒原 寛明

ソフトウェア

- ソフトウェア(software)
  - ✓ データ処理システムを機能させるための，プログラム，手順，規制，関連文書などを含む知的な創作 (JIS X0001)
    - プログラム
    - 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，運用マニュアル
  - ✓ プログラム，プログラムを作成する過程で得られるシステム設計書，フローチャートをはじめとする設計書，および，プログラム説明書などの関連資料
- ハードウェア(hardware)
  コンピュータ装置



ソフトウェアとハードウェアの比較

- ソフトウェア
  - ✓ 経年劣化なし
  - ✓ 導入後に修正可能
  - ✓ 製品の量産コスト，配布・流通コストは低い
    → 機能拡張，性能改善，環境適合に関する要求
    ソフトウェア進化が前提
- ハードウェア
  - ✓ 経年変化あり (磨耗，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あり
    → 機能や性能を維持することに対する要求

ソフトウェアの一般的特性

1. ソフトウェアは目に見えない製品
2. 品質の劣化なし，向上していく傾向
3. 潜在的バグが潜んでいる
4. 修正可能
5. 複写可能
6. 要求される機能は社会情勢とともに絶えず変化
7. 波及効果が生じる
8. バグは本人より第三者のほうが見つけやすい
9. 作成者の思想

ソフトウェア工学

- ソフトウェア工学(software engineering) [1968年のNATO会議]
  - ソフトウェア危機(software crisis)を打開するための
    ソフトウェア開発(software development)の技術体系および学問体系
    - 方法論(methodology)
    - 技法(technique) / 道具(tool)
    - プロジェクト管理(project management)
  - 大規模・高信頼性ソフトウェアの開発 ≠ プログラミング
- 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
  - 良いソフトウェアとは?
    高信頼, 保守が容易, 拡張が容易, 利用が簡単, 高速, ...

ソフトウェア危機

- ソフトウェア危機: 1960年代後半~
  - 「規模」の問題(1970年代):
    ハードウェアの大型化に伴う大規模ソフトウェアの必要性
    → 構造化プログラミング(構造化分析, 設計, コーディング)
  - 「量」の問題(1980年代):
    コンピュータシステムの普及に伴う開発ソフトウェア数の増大
    → 統合的ソフトウェア開発支援環境や部品化・再利用
  - 「質」「インタフェース」の問題(1990年代):
    - 社会的に重要な役割を担うシステムに対する信頼性の要求
    - ソフトウェアの大衆化に伴う使い勝手の要求
    - オープン化やネットワーク化に伴う接続性, 移行性, 互換性の要求
  - +「複雑さ」「変化」の問題(2000年代):
    - 扱う対象が専門的かつ広範囲
    - 動作環境や社会的要求が流動的あるいは急激に変動

ソフトウェアの品質特性(ISO9126)

1. 機能性(functionality): 必要な機能実装の度合い
   - 合目的性: 利用者の目的にあっているか
   - 正確性: 仕様に対して正しく動作するか
   - セキュリティ: 不当なアクセスを排除できるか
   - 相互運用性: データやコマンドがやり取りできるか
   - 標準適合性: 規格や標準にフォーマットが合致しているか
2. 信頼性(reliability): 機能が正常に動作し続ける度合い
   - 成熟性: 故障する頻度が少なくなったか
   - 障害許容性: 障害に対して許容できるか
   - 回復性: 故障したときに早く復旧できるか
3. 使用性(usability): 分かりやすさ, 使いやすさの度合い
   - 理解性: 使い方がわかりやすいか
   - 習得性: 初めてでもすぐに使えるようになるか
   - 運用性: 管理するのは楽か

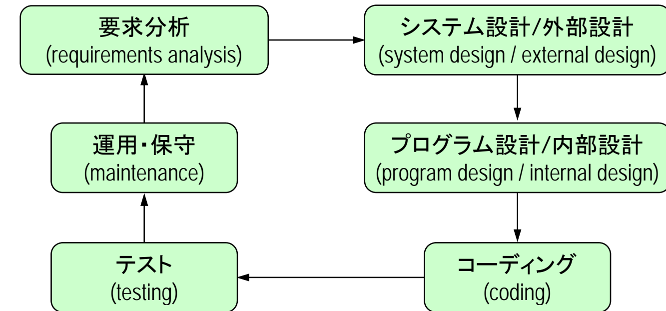
ソフトウェアの品質特性(cont'd)

4. 効率性(efficiency): 目的達成のために使用する資源の度合い
   - 時間的効率性: 処理速度が速いか
   - 資源効率性: メモリなどの資源を多く必要としないか
5. 保守性(maintainability): 改訂作業に必要な労力の度合い
   - 解析性: プログラムがわかりやすいか
   - 変更性: プログラムが変更しやすいか
   - 安定性: 変更時に障害が混入しないか
   - 試験性: テストがしやすいか
6. 移植性(portability)
   - 設置性: インストールは簡単か
   - 環境適応性: いろいろな環境(OSなど)で使えるか
   - 置換性: 他のソフトウェアに置き換え可能か
   - 規格準拠性: 規格や規約に適合しているか

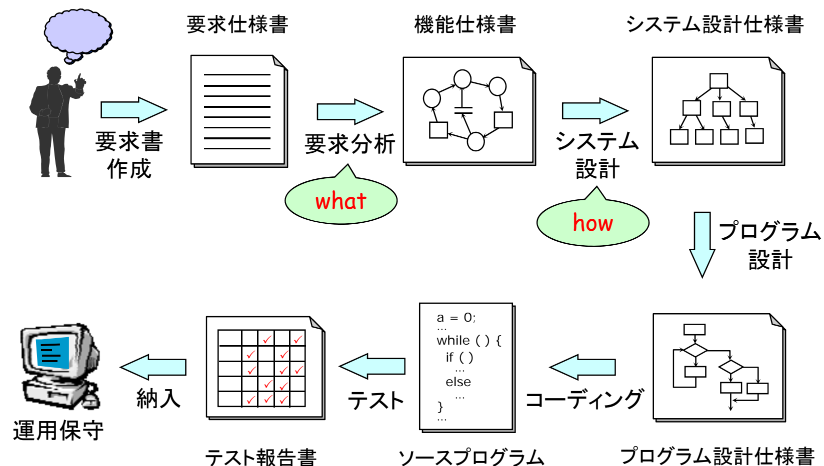
ソフトウェア開発モデル

ソフトウェア開発モデル

- ソフトウェア開発(software development)
  - ✓ 誰が(who): プロジェクト(project)
  - ✓ 何を(what): プロダクト(product)
  - ✓ どのように(how): プロセス(process)
- ソフトウェアのライフサイクル(life cycle)



ソフトウェア開発プロセス



ソフトウェア開発プロセス(分析)

(1) システムの要求書作成
- ✓ 利用者(ユーザ) が要求するシステムを整理し，自然言語で文書化
  - システム要求書(system requirements)
  - 要求仕様書(requirements specification)

(2) システムの要求分析(requirements analysis)と
システム定義(system definition)
- ✓ どのようなシステムを作成するのかを決定(分析者; analyst)
  システム要求書を分析し，要求システムを形式的に文書化
  - システム機能仕様書(functional system specification)
  - 機能仕様書(functional specification)

構造化分析(structured analysis)

ソフトウェア開発プロセス(設計)

(3) システム設計(system design)/外部設計(external design)
- ✓ システムをどのように作成するのかを決定(設計者; designer)
  モジュール(module)構成, 個々のモジュールの機能,
  モジュール間のインタフェース(interface)を決定
  - システム設計仕様書(system design specification)
  - 外部設計仕様書(external design specification)

(4) プログラム設計(program design)/内部設計(internal design)
/詳細設計(detailed design)
- ✓ 個々のモジュールの内部構造を決定(プログラマ; programmer)
  アルゴリズムとデータ構造, 処理手順を決定
  - プログラム設計仕様書(program design specification)
  - ロジック設計仕様書(logic design specification)

構造化設計(structured design)

ソフトウェア開発プロセス(実装)

(5) コーディング(coding)
- ✓ プログラム設計仕様をプログラムに変換(実装者; coder)
  具体的なプログラミング言語(program language)による記述
  - ソースプログラム(source program)

構造化プログラミング(structured programming)
- 分割・統治
- 段階的詳細化
- 3つの基本制御(逐次・選択・反復)

ソフトウェア開発プロセス(テスト, 保守)

(6) ソフトウェアテスト(software test)
- ✓ 仕様書通りにプログラムが動作するかどうかを検査(試験者; tester)
  モジューテスト(module test)/単体テスト(unit test)
  集積テスト/統合テスト(integration test)
  システムテスト(system test)/機能テスト(function test)
  出荷テスト(shipping test)/受入れテスト(acceptance test)
- ✓ ソフトウェア作成者と独立の組織
  - テスト報告書(test report): エラー(error), 故障(fault)
    → デバック(debug): エラーや故障の修正(設計者, プログラマ)

(7) ソフトウェア保守(software maintenance)
- ✓ 納入後のソフトウェアを管理(CE: customer engineer)
  運用段階で検出された故障(残存エラー)の修正
  手直し要求: 新機能の追加, 既存機能の変更, 新しい環境への適合

構造化プログラミング

- 構造化プログラミング(structured programming) [IBMの技術規範IPT]
  記述が容易 → 理解が容易な(簡単でわかりやすい)プログラムの構築技法

(1) 分割統治(divide and conquer):
大きく複雑なプログラムを小さく簡単なプログラム(モジュール:module)で合成

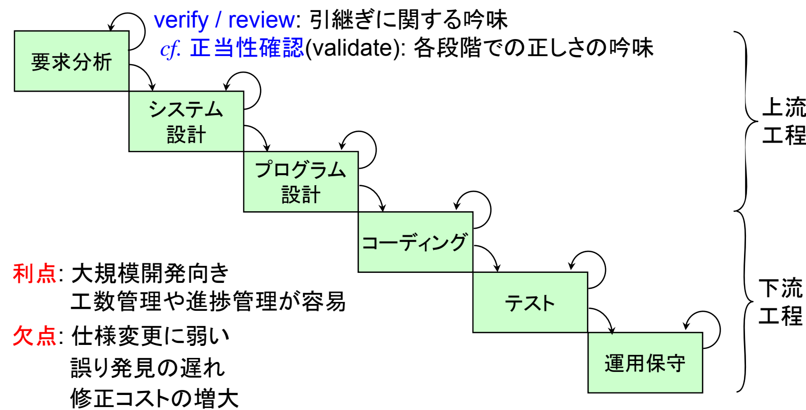
(2) 段階的詳細化(stepwise refinement):
要求プログラムを抽象データ型を仮定して作成し, 上位の抽象データ型を
下位の抽象データ型で繰り返し具体化

(3) プログラムを3つの基本制御の論理構造で構築
- 連接, 逐次(sequence)
- 選択(selection)
- 反復(iteration)

ウォータフォールモデル

- ウォータフォールモデル(waterfall model)
  トップダウンな開発プロセス

verify / review: 引継ぎに関する吟味
cf. 正当性確認(validate): 各段階での正しさの吟味



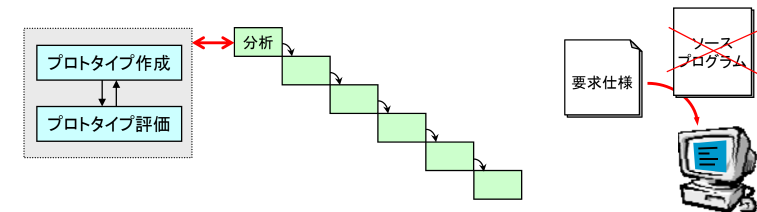
利点: 大規模開発向き
工数管理や進捗管理が容易
欠点: 仕様変更に弱い
誤り発見の遅れ
修正コストの増大

ソフトウェア開発モデルの推移

- 1960年代: 流れ図(flowchart), 開発方法論なし
- 1970年代: 構造的なソフトウェア開発, ウォータフォールモデル
- 1980年代: ソフトウェアライフサイクル有害説
  → 新しいソフトウェア開発パラダイム(paradigm)

1) ソフトウェアプロトタイピング(software prototyping)
  - ✓ システム設計時に試作品(プロトタイプ; prototype)を構築

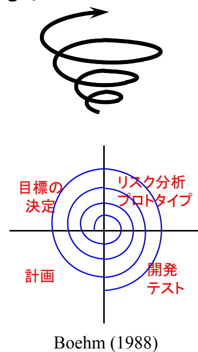
2) 操作的アプローチ(operational approach)
  - ✓ 実行可能な仕様(executable specification)の作成



ソフトウェア開発モデルの推移(cont'd.)

3) オブジェクト指向ソフトウェア開発(object-oriented software development)
  - ✓ オブジェクトを単位としてソフトウェアを構築
    オブジェクト: 実世界の「もの」や「役割」などを抽象化したもの
  - ✓ スパイラルモデル(spiral model)
    インクリメンタル(incremental) + イテラティブ(iterative)

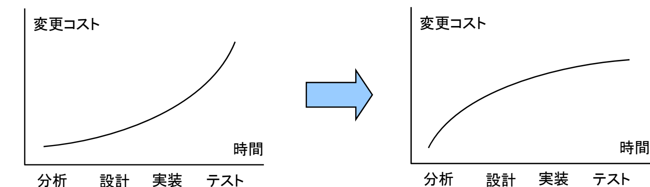
4) アジャイルソフトウェア開発(Agile software development)
  例) XP(extreme programming), SCRUM, Crystal,
  アジャイルアライアンス宣言(manifesto)
  - ✓ プロセスやツールよりも, 個人や人同士の交流を重視
  - ✓ 包括的なドキュメントよりも, 動作するソフトウェアを重視
  - ✓ 契約上の交渉よりも, 顧客との協調を重視
  - ✓ 計画に従うことよりも, 変化に対応することを重視
    注) 左側の項目を軽視するという意味ではない



Boehm (1988)

XP

- 究極のスパイラルモデル
  特徴:
  - ✓ コーディングおよびテストに重点を置く
  - ✓ 初期設計よりもリファクタリングによる再設計を重視
  - ✓ 4つの価値と12のプラクティス



Embrace Change :変化を抱擁せよ

- コミュニケーション(Communication)
- シンプルさ(Simplicity)
- フィードバック(Feedback)
- 勇気(Courage)

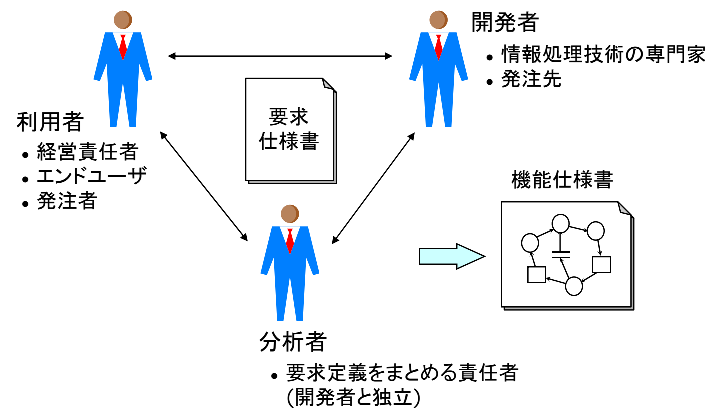
XP(cont'd.)

1. 計画ゲーム（Planning Game）
2. 小規模リリース（Small Releases）
3. 比喩（Metaphor）
4. シンプルデザイン（Simple Design）
5. テスティング（Testing）
6. リファクタリング（Refactoring）
7. ペアプログラミング（Pair Programming）
8. 共同所有権（Collective Ownership）
9. 継続的インテグレーション（Continuous Integration）
10. 週40時間（40-Hour Week）
11. オンサイト顧客（On-Site Customer）
12. コーディング標準（Coding Standards）

要求分析

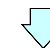
要求分析

- 開発すべきシステム全体の仕様をできるだけ厳密に定義



要求分析(cont'd)

- 要求分析の課題
  - a) 利用者の要求の曖昧さ
    - ✓ 真の利用者を特定することが困難
    - ✓ 現状の業務形態やその問題に対する認識が不十分
    - ✓ 際限ない要求
  - b) 利用者と開発者のコミュニケーションギャップ
    - ✓ 背景，知識，言葉の問題
      利用者: 業務の専門家，開発者: 情報処理技術の専門家
  - c) 開発者の技術的課題認識の甘さ
    - ✓ 開発期間や開発費用の過小評価
  - …

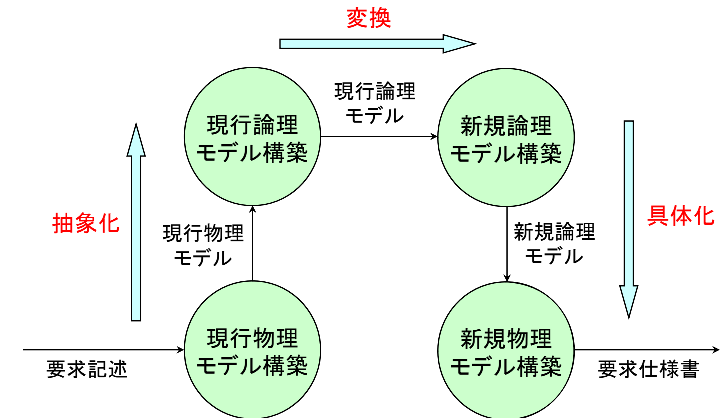


要求分析技法(モデル化技法と形式化された図式)の必要性

要求分析技法

- 分析時の観点による分類
  1) 機能:
    a) データの流れとそのデータを処理する機能に着目
      データフロー図の利用
      → 構造化分析(SA: structured analysis)
    b) ユーザの利用方法に着目
      ユースケース(use-case)とシナリオ(scenario)を利用
  2) データ: システム内のデータ構造とデータ間の制約に着目
    実体関連図を利用
  3) オブジェクト: データと機能(操作)をカプセル化して扱う
  4) プロセス: 実世界の問題の処理の流れに注目

構造化分析技法



論理的観点: どのような情報が必要であるかという要件
物理的観点: その情報を得るための仕組みに対する要件

構造化分析の手順

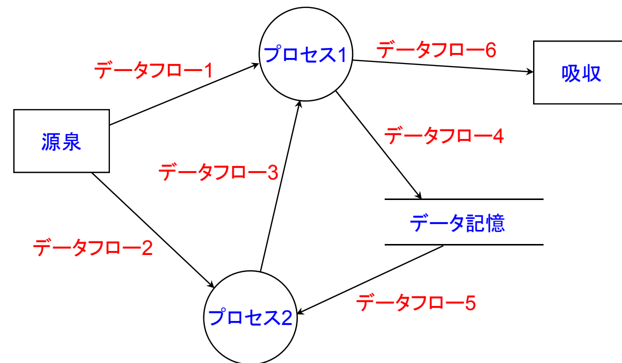
1) 現行物理モデルの構築
  現状の業務(人手部分+機械化部分)を分析
  - ✓ 物理的なレベル(ありのまま)で正確に表現
2) 現行論理モデルの構築
  本質的な機能や問題の洗い出し
  - ✓ 本質的でない部分(人, 組織, タイミング, 媒体など)を除外
  - ✓ 本質的でない部分の抽象的な概念への置換え
3) 新規論理モデルの構築
  新たに開発するシステムで解決すべき問題の洗い出し
  - ✓ 追加あるいは削除する機能の特定
  - ✓ 機械化部分と人手部分の境界の決定
4) 新規物理モデルの構築
  新規論理モデルを制約条件を満たすように具体化
  - ✓ 機械化部分と人手部分の境界の最終決定

仕様記述

- システム機能のモデル化:
  - ✓ データフロー図(DFD: data flow diagram)
  - ✓ データ辞書(data dictionary)
  - ✓ プロセス仕様書
  - ✓ ユースケース
- 蓄積データのモデル化:
  - ✓ 実体関連図(ER図: entity relationship diagram)
- 状態変化や制御手順を含む時系列動作のモデル化:
  - ✓ 状態遷移図(state transition diagram)
  - ✓ リアルタイム用データフロー図

データフロー図

- システムを階層的かつ図的にモデル化
  - ✓ システム内のデータの流れを表現
  - ✓ 事象(イベント: event)のような制御は除外
    cf. フローチャート(flowchart): 制御の流れを表現



データフロー図(cont'd)

- 構文(syntax): 4つの基本記号のグラフ表現
  1) フロー(flow): 定常的なデータの流れを表現
     矢印にデータを表す名前を付加
  2) プロセス(process): データの処理を表現
     個々のプロセスがどのような処理を行うのかを記述
     = バブル(bubble)
  3) データストア(data store): データを格納する場所
     格納するデータの名前(入出力フローのデータ名と同一)を記述
     = ファイル(file)
  4) エンティティ(entity): システムの外部にある組織や人を表現
     組織や人を表す名前を記述
     - 源泉(source): データの発生源
     - 吸収(sink): データの最終的な行き先
- 意味論(semantics): 各記号に付加された情報(名前)に依存

例題1: 業務の記述

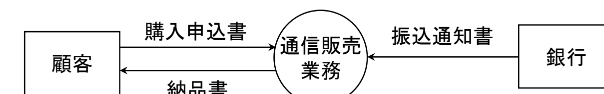
通信販売の業務(個人の顧客からの注文処理)

- 受付係は，顧客から郵便またはファックスによる購入申込書を受け取ると，この顧客が顧客ファイルに登録されていなければ登録する．次に注文票を作成し，販売係に送る．
- 販売係は，注文票を受け取ると，その注文に関する入金伝票の有無を確認し，すでに入金伝票がある場合は商品管理ファイルを参照して発送依頼票を作成し，発送依頼票を在庫管理係に送る．
- 経理係は，銀行から顧客の振込通知書を受け取ると，入金伝票を作成し，販売係に送る．
- 販売係は，入金伝票を受け取ると，その入金に関する注文票の有無を確認し，すでに注文票がある場合は商品管理ファイルを参照して発送依頼票を作成し，発送依頼票を在庫管理係に送る．
- 在庫管理係は，発送依頼票を受け取ると，商品管理ファイルを参照して在庫の有無を確認し，在庫がある場合は品物と納品書を顧客に送る．在庫切れの場合は発注処理をし，入庫後に同様の処理をする．

出典: 中所武司著，「ソフトウェア工学」，朝倉書店，1997年

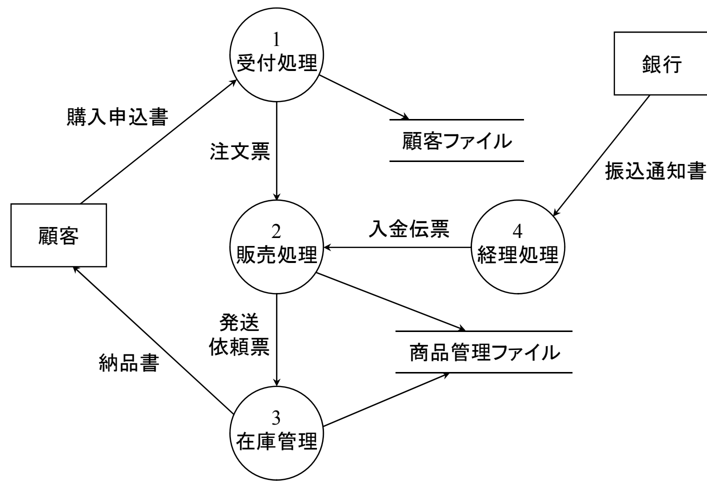
例題1: 全体文脈図

- 全体文脈図(context diagram): DFD記述の出発点
  - ✓ システム全体を1つのプロセスで表現
  - ✓ システムと外界とのデータのやり取りを表現



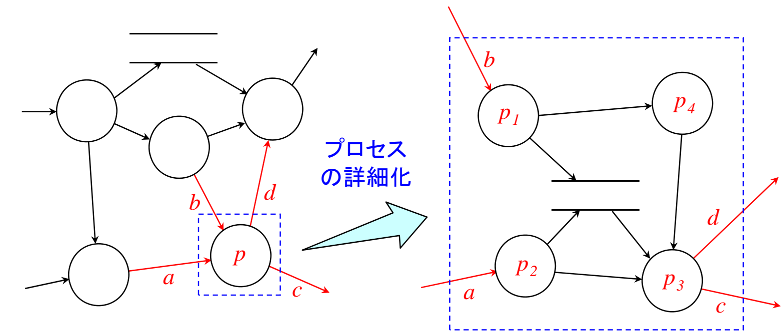
本例題における前提
- 論理レベルのDFD構築
- 業務の流れは現行モデルと同一
- 各種帳票の電子化とそれに伴う各担当の業務の自動化

例題1: DFD

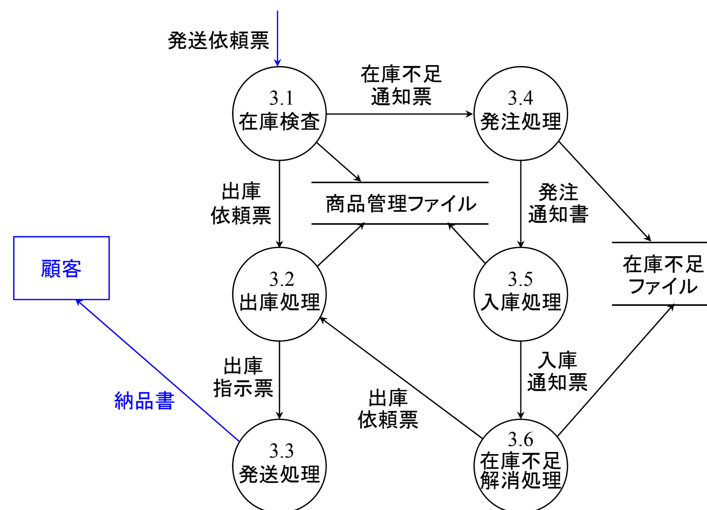


DFDの階層化

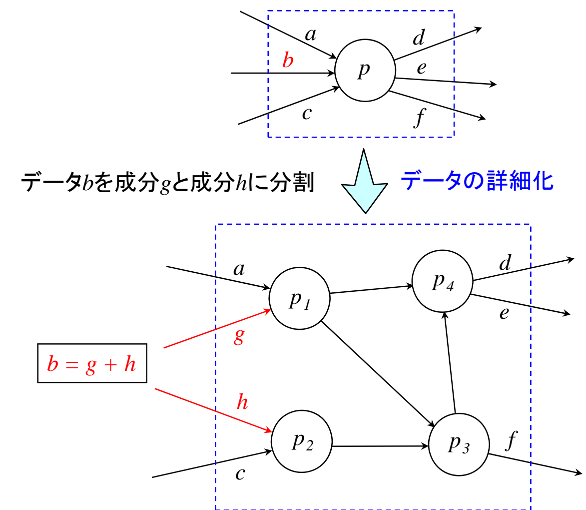
- DFDの階層化: プロセスの内部処理を詳細化
  - ✓ 着目する処理の入出力矢印は，詳細化の前後で維持



例題1: 詳細DFD



DFDのデータ詳細化



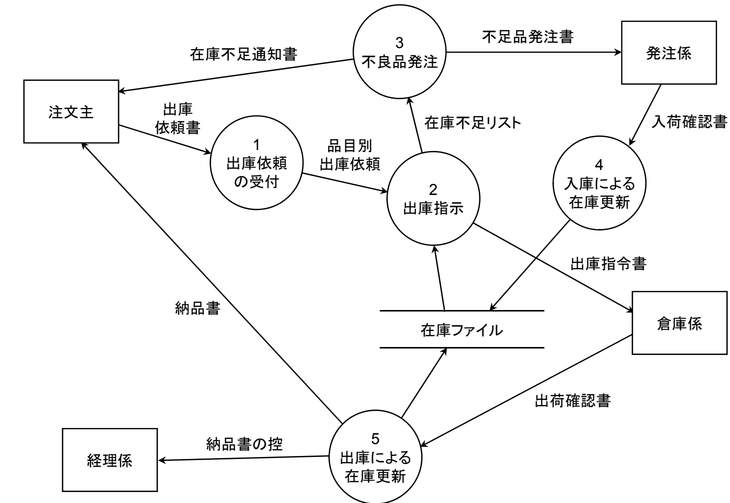
例題2: 業務の記述

酒類販売の業務

- ある酒類販売会社の倉庫では、毎日数個のコンテナが搬入されてくる。その内容はビン詰めの酒で、1つのコンテナには10銘柄まで混載できる。扱い銘柄は約200種類ある。倉庫係は、コンテナを受取りそのまま倉庫に保管し、積荷表を受付係へ手渡す。また受付係からの出庫指示によって内蔵品を出庫することになっている。内蔵品は別のコンテナに詰め替えたり、別の場所に保管することはできない。
- 空になったコンテナはすぐに搬出される。
- さて受付係は毎日数十件の出庫依頼を受け、その都度倉庫係へ出庫指示書を出すことになっている。出庫依頼は出庫依頼表または電話によるものとし、1件の依頼では、1銘柄のみに限られている。在庫が無いか数量が不足の場合には、その旨依頼者に電話連絡し、同時に在庫不足リストに記入する。そして当該品の積荷が必要量あった時点で、不足品の出庫指示をする。また空になる予定のコンテナを倉庫係に知らせることになっている。

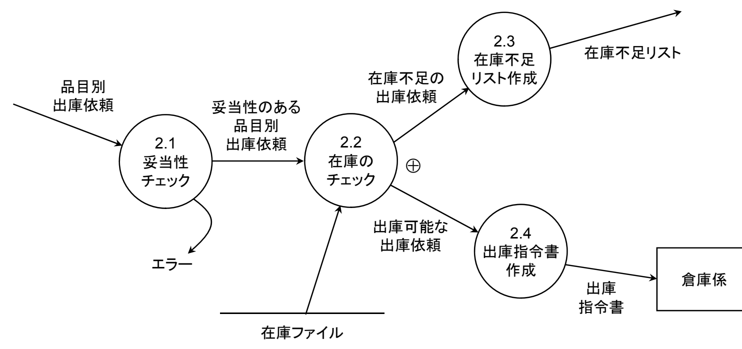
出典: 山崎利治, 「共通問題によるプログラム設計技法解説」
情報処理25-9, pp.934-962, 1984年

例題2: DFD(レベル1)



出典: 有沢誠著, 「ソフトウェア工学」, 岩波書店

例題2: DFD(レベル2)



*: 複数のデータフローの結合を表す(AND)
⊕: 複数のデータフローの分離を表す(OR)

データ辞書

- データ辞書(data dictionary): DFDに現れるデータの構造を表現
  - 等価: a=b; aはbに等しい(is equivalent to)
  - 連接: a+b; aとbからなる(and)
  - 選択: [a|b]; aまたはbのどちらかである(either-or)
  - 任意: (a); aはあってもなくてもよい(optional)
  - 反復: {a}; aを0回以上繰り返す(iterations of)
    - m{a}n; aをm回以上かつn回以下繰り返す
    - m{a}; aをm回以上繰り返す
    - {a}n; aをn回以下繰り返す

(a, b: データ要素, n, m:整数)

データ構造の例)

注文書 = 注文番号 + 顧客名 + 送付先住所 + (電話番号) + 1{注文品目}10
+ 合計 + [領収書要|領収書不要]

注文品目 = 品番 + (品名) + 単価 + 数量 + 小計

品番 = 6{数字}6

プロセス仕様

- プロセス仕様: DFD記述の終了点
  - ✓ DFDの最下層のプロセスの基本処理を表現
    = ミニ仕様(mini spec)
    - 式
    - 原因結果グラフ(cause-effect graph)，決定表(decision table)
    - 構造化言語(e.g., PDL: program description language)

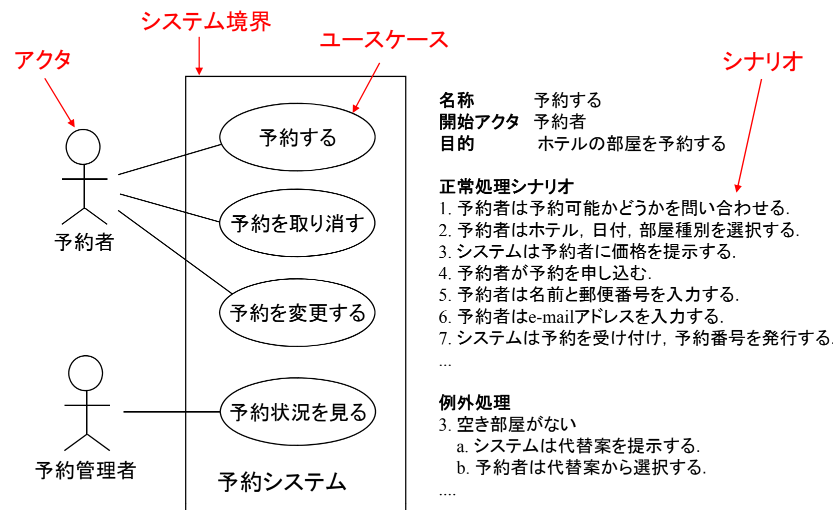
構造化言語によるプロセス仕様の例

1 以下のいづれかの処理を行う.
  1.1 もし，入力データが注文票である場合は以下の処理を行う.
    1.1.1 商品管理ファイルを参照して注文票の記載内容をチェックする.
    1.1.2 もし，記載内容不備ならば，以下の処理を行う.
      1.1.2.1 申込書不備に伴う関連処理を行う.
      1.1.2.2 販売処理を終了する.
    1.1.3 すでに入金伝票が保存されているか確認する.
    1.1.4 もし，入金伝票が保存されていなければ，以下の処理を行う.
    …

ユースケース

- ユースケース(use case) [Jacobson]:
  - ✓ システムの機能ごとに作成
  - ✓ システムの利用者側(アクタ)からみた使われ方を表現したもの
    - アクタ(actor): システムに対して利用者が果たす役割(role)
      役割ごとに異なるアクタが存在
      外部システムでもよい
      受益者(beneficiary)
  - ✓ 利用者の目的に照らして結び付けられた一群のシナリオ
    - シナリオ(scenario): 利用者とシステム間の対話を表す一連の手順
      ユースケースのインスタンス

ユースケースの例

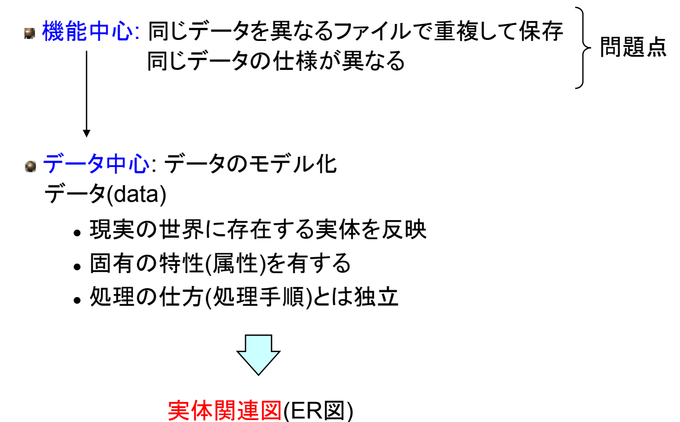


名称 予約する
開始アクタ 予約者
目的 ホテルの部屋を予約する

正常処理シナリオ
1. 予約者は予約可能かどうかを問い合わせる.
2. 予約者はホテル，日付，部屋種別を選択する.
3. システムは予約者に価格を提示する.
4. 予約者が予約を申し込む.
5. 予約者は名前と郵便番号を入力する.
6. 予約者はe-mailアドレスを入力する.
7. システムは予約を受け付け，予約番号を発行する.
…

例外処理
3. 空き部屋がない
   a. システムは代替案を提示する.
   b. 予約者は代替案から選択する.
….

実体関連図

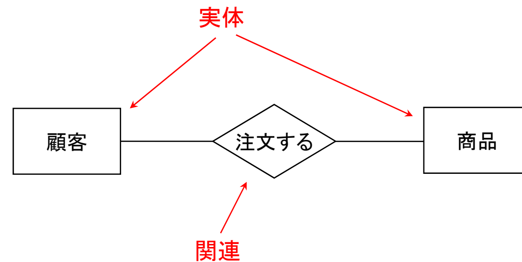


- 機能中心: 同じデータを異なるファイルで重複して保存
  同じデータの仕様が異なる
  } 問題点
- データ中心: データのモデル化
  データ(data)
  - 現実の世界に存在する実体を反映
  - 固有の特性(属性)を有する
  - 処理の仕方(処理手順)とは独立

実体関連図(ER図)

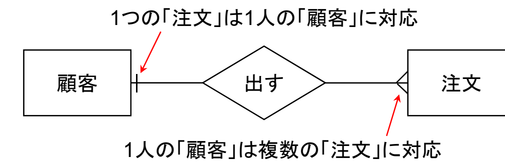
実体関連図(cont'd)

- 実体関連図(ER図)
  - ✓ 実体(entity): システム内に存在する管理対象(人，物，金，場所)
    名前をもつ四角形で表示
  - ✓ 関連(relationship): 実体間の相互の結びつき
    関連名を持つ菱形で表示

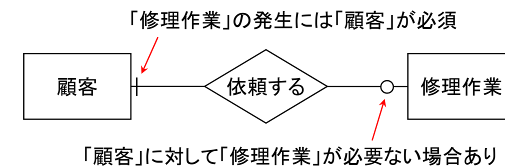


関連

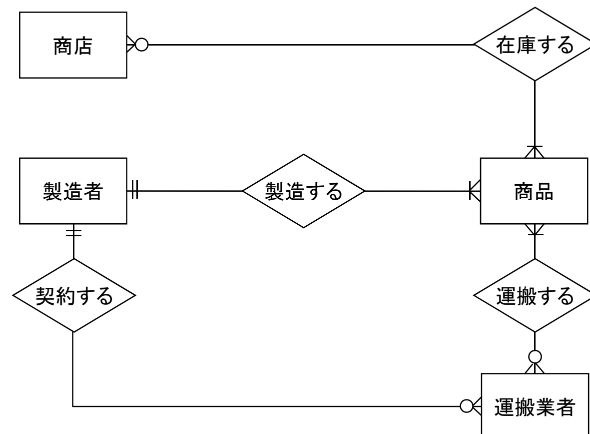
- カーディナリティ(cardinality)
  関連するオブジェクトの数を表現 (1:1, 1:N, M:N)



- モダリティ(modality)
  関連が必須であるかどうかを表現



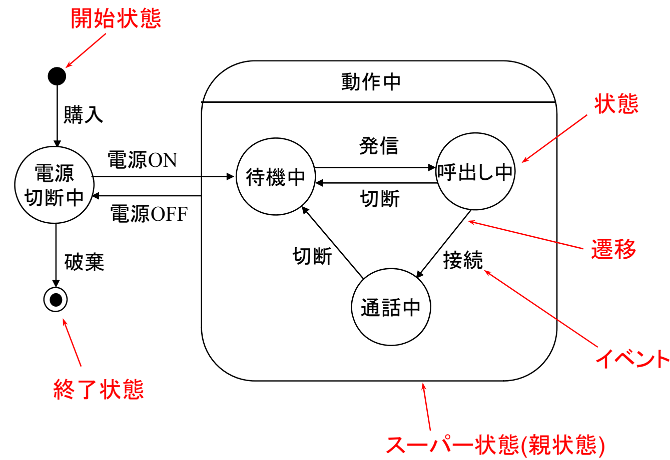
実体関連図の例



状態遷移図

- 状態遷移図/ステートチャート図(statechart diagram):
  - ✓ システムが取りうるすべての状態(state)と，そのシステムに到着したイベント(event)による状態の変化を表現
  - ✓ 外部からのイベントに対するシステムの応答を表現
    - システムは必ず1つの状態に属する(複合状態を除く)
    - イベントは一瞬
    - 遷移時間は無視
    - イベントに対する応答は現在の状態によって変化

状態遷移図の例



演習: システムの記述

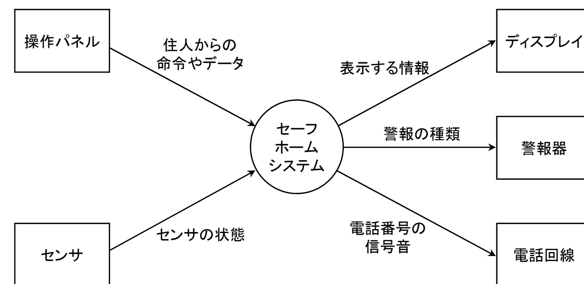
ホームセキュリティシステムの仕様

- セーフホームシステムは，設置時には住人がシステム設定をできるようにし，セキュリティシステムが接続されているすべてのセンサを監視する．操作パネルのキーパッドやキーを通して，住人とやりとりする．
- 設置時にシステムを設定するときには，操作パネルで行う．各センサに番号と種類を割りあて，システムの作動・解除を切りかえるマスタパスワードを設定し，センサイベント発生時の連絡先の電話番号を入力する．
- このソフトウェアはセンサイベントを検知した際に，システムに接続されている警報器を起動する．さらに，システム設定時に住人が指定した遅延時間を経過した時点で，監視サービスの電話番号に住所に関する情報を連絡し，検知したイベントについて報告する．電話回線への接続は，接続されるまで20秒間隔で繰り返される．
- セーフホームシステムとのすべてのやりとりは，ユーザインタフェースサブシステムによって管理される．このサブシステムは，キーパッドや特殊キーを通じて与えられた入力を読み取り，ディスプレイに入力メッセージとシステムの状況を表示する．

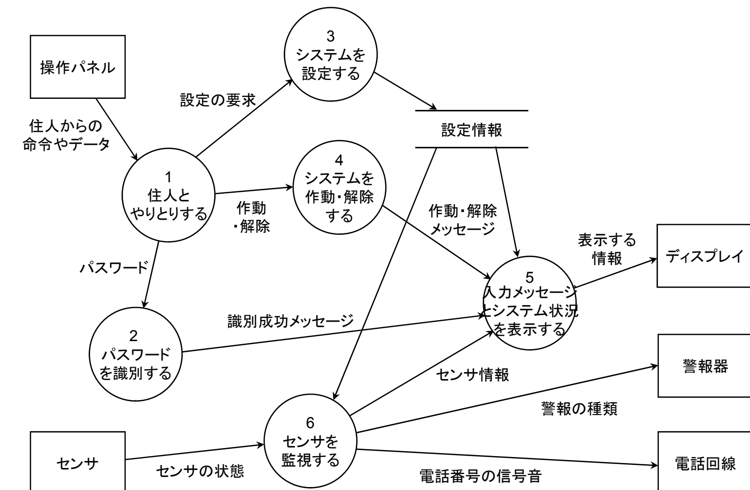
(以下 略)

出典: R. Pressman著，「ソフトウェア工学の伝統的手法」，日科技連

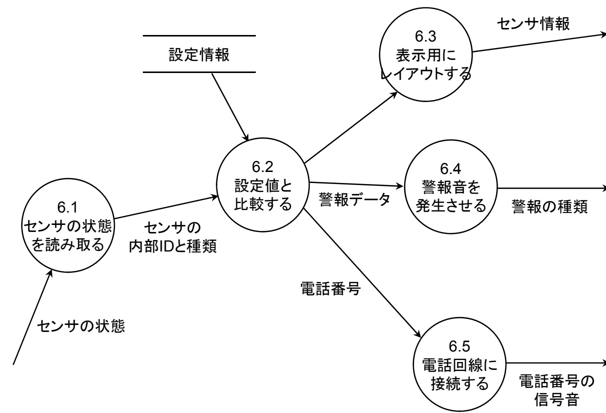
演習: 全体文脈図(レベル0)



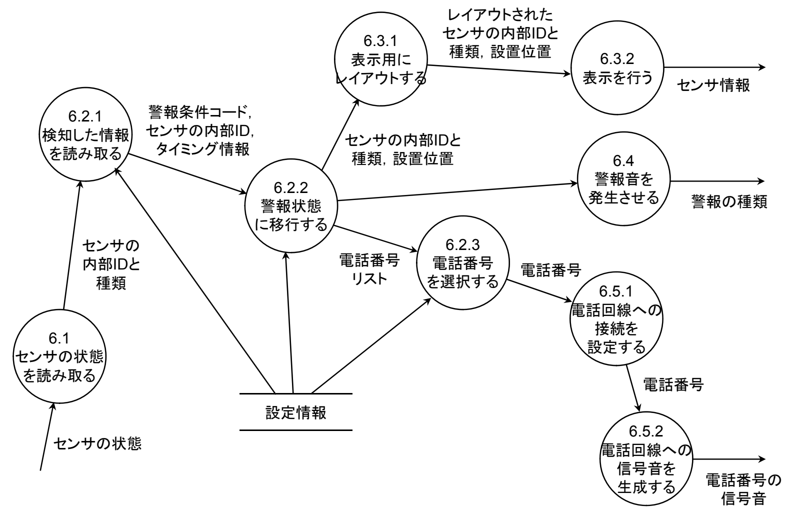
演習: DFD(レベル1)



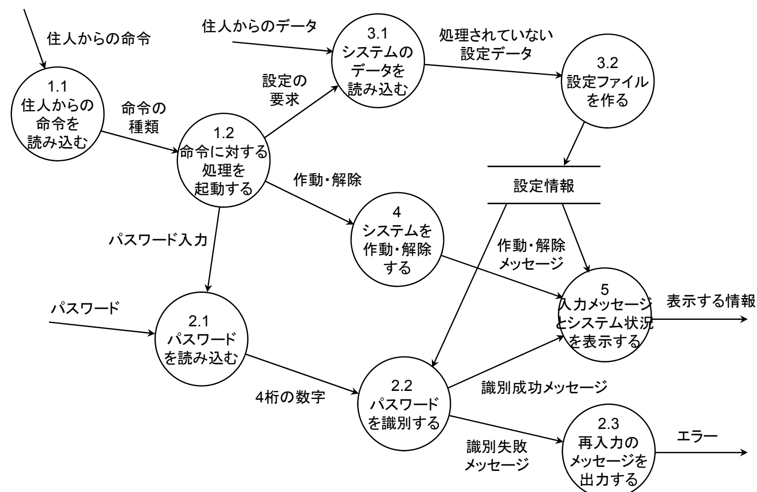
演習：DFD(レベル2)



演習：DFD(レベル3)



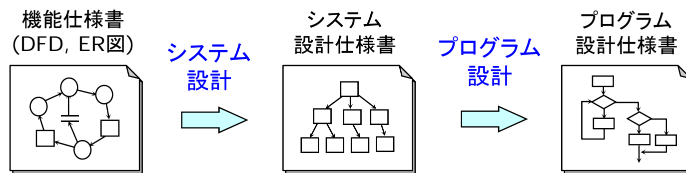
演習：DFD(レベル2)



ソフトウェア設計

設計

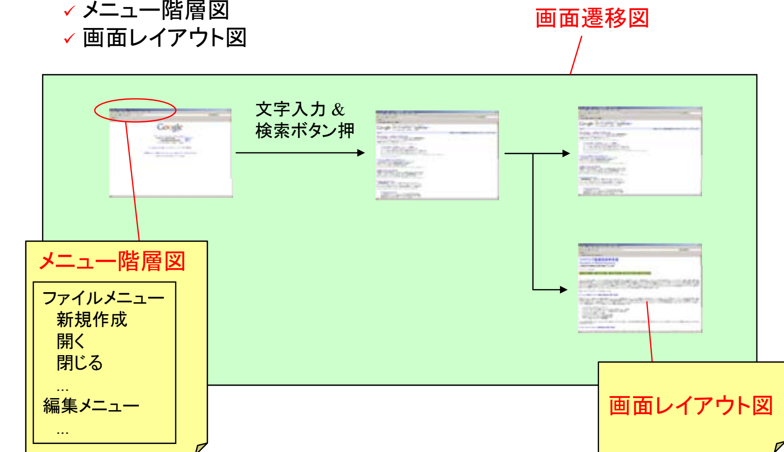
- 機能仕様書が定めるシステムをどのように実現するのかを決定



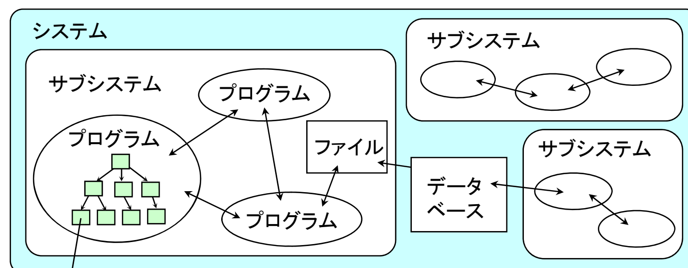
- システム設計(system design)
  - ✓ システム外部設計: システムの外部特性(環境やUIなど)を記述
  - ✓ システム内部設計: システムをサブシステムやモジュールに細分化し，それらの間のインタフェースを定義
  - ✓ データベース設計: システムに共通のデータ構造の識別と定義
- プログラム設計(program design)
  - ✓ モジュールの設計: 個々のモジュールの内部構造を決定

ユーザインタフェース設計

- GUI(graphical user interface)における
  - ✓ 画面遷移状態図
  - ✓ メニュー階層図
  - ✓ 画面レイアウト図



モジュール



モジュール(module):
機能単体あるいは関連する機能をひとまとめにしたプログラム単位

(1) 複数の文で構成され，独立して識別可能な名前をもつ
(2) コンパイルが別々にできる
(3) 決められたインタフェースを通してのみ呼出可能である

設計技法

設計技法の分類

- データの流れに基づく構造化設計
  データの流れに着目して機能を分割することでプログラム構造を決定
  - ✓ 複合設計[Myers]
  - ✓ 構造化プログラミング[Dijkstra]
- データの構造に基づく構造化設計
  システムの入出力データの構造に着目してプログラム構造を決定
  - ✓ ジャクソン法[Jackson]
  - ✓ ワーニエ法[Warnier]
- オブジェクト指向設計
  モジュールの代わりにデータと機能(操作)をカプセル化したオブジェクトを基本単位としてプログラム構造を決定
- 契約に基づく設計(DbC: Design by Contract) [Meyer]
  クライアント(client)とサプライヤー(supplier)間の義務，便益，制約を表明(assertion)で記述

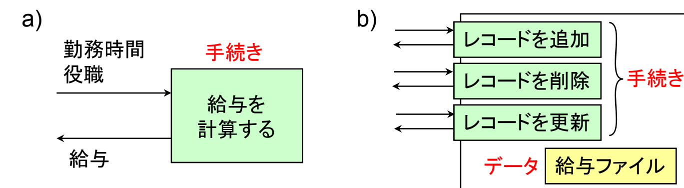
構造化設計

- 構造化設計(structured design) / 複合設計(composite design)
  システム機能をトップダウンで詳細化し，機能の階層構造を作成
  ✓ 抽象化(abstraction)の概念が有効

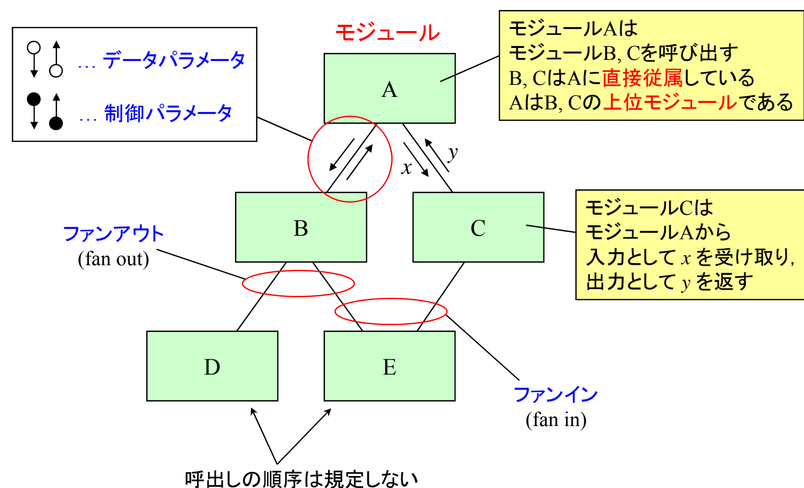
- システム設計仕様書(system design specification)
  (1) モジュール構成図(module structure diagram)
  システムを実現するモジュールの構成(静的関係)を規定
  (2) モジュール機能仕様書(module unction specification)
  個々のモジュールの機能を規定
  (3) モジュールインタフェース仕様(module interface specification)
  モジュールが外部から呼び出させるときのインタフェースを規定

抽象化

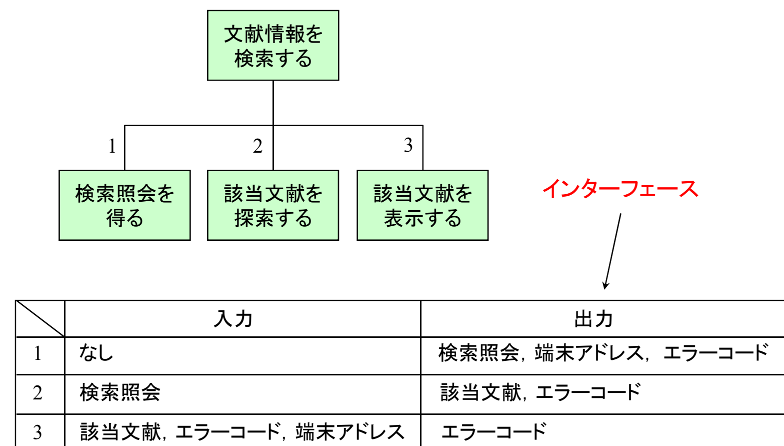
- 抽象化(abstraction) ≅ 情報隠蔽(information hiding)
  a) 手続きの抽象(procedure abstraction):
  手続きの使い方を手続きの実装から分離すること
  b) データの抽象(data abstraction):
  データの使い方をデータの実装から分離すること
  → カプセル化(encapsulation)
  c) 制御の抽象化(control abstraction):
  プログラムの制御構造を内部的な詳細から分離すること
  例) 3つの基本制御構造で表現 → 構造化プログラミング



モジュール構造図



モジュール構造図の例

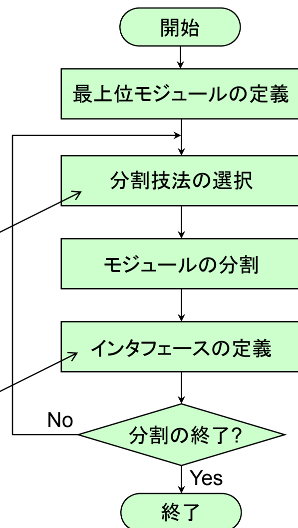


	入力	出力
1	なし	検索照会，端末アドレス，エラーコード
2	検索照会	該当文献，エラーコード
3	該当文献，エラーコード，端末アドレス	エラーコード

構造化設計の手順

構造化設計の作業[Constantine]

1) モジュールの機能の定義
2) モジュールの階層構造の決定
3) モジュール間のインタフェースの決定

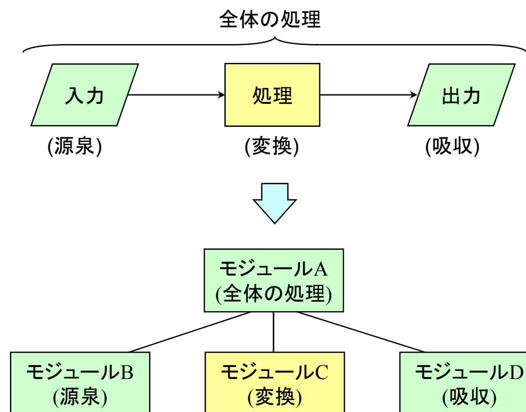


モジュールの分割技法
(1) 源泉/変換/吸収分割(STS分割)
(2) トランザクション分割(TR分割)
(3) 共通機能分割

受け渡しされる入力と出力の情報を定義

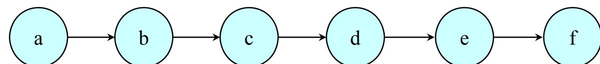
源泉/変換/吸収分割

- 源泉/変換/吸収分割(STS分割:Source/Transform/Sink decomposition)
  - ✓ 機能を入力から出力への変換とみなして分割

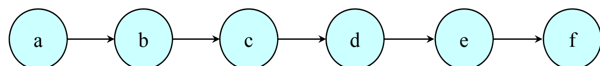


STS分割の手順(1)(2)

(1) 問題構造図の記述
- ✓ 与えられた仕様から，3~10個の機能で問題を記述する



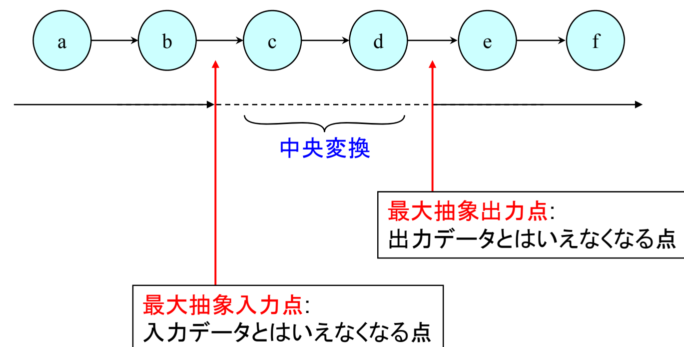
(2) 主要データの識別
- ✓ 問題のなかの主要な入出力データの流れを明らかにする



主要な入力データ　主要な出力データ

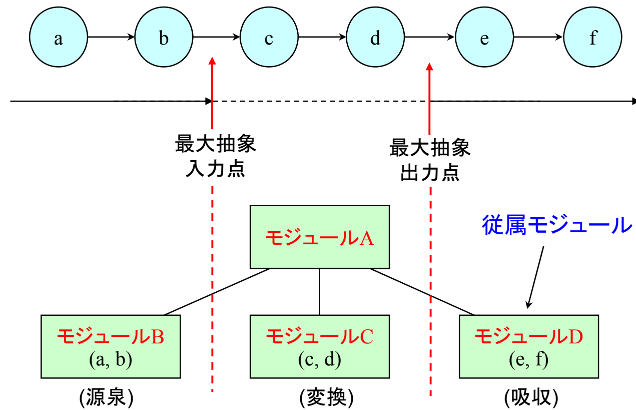
STS分割の手順(3)

(3) 最大抽象点の発見
- ✓ 入力側から入力データを順方向にトレースし最大抽象入力点を見つける
- ✓ 出力側から出力データを逆方向にトレースし最大抽象出力点を見つける



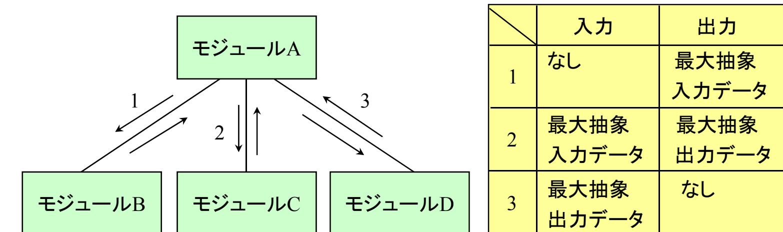
STS分割の手順(4)

(4) 直接従属モジュールの定義
- ✓ 最大抽象入力点および出力点を区切りに源泉/変換/吸収部分に分割する



STS分割の手順(5)(6)

5) モジュール間インタフェースの定義
- ✓ 下位モジュールを中心にその入出力データを決定する



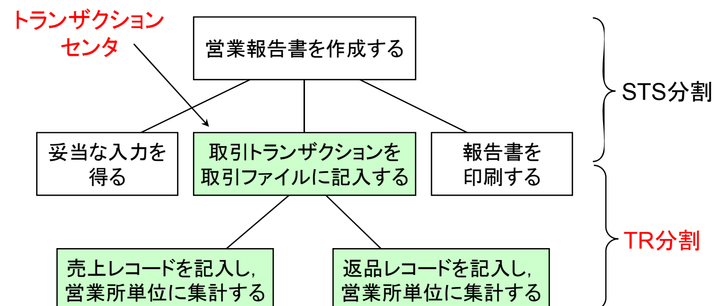
	入力	出力
1	なし	最大抽象入力データ
2	最大抽象入力データ	最大抽象出力データ
3	最大抽象出力データ	なし

6) 分割の繰返し
- ✓ モジュールB, C, Dについて同様の手順を繰り返す

トランザクション分割

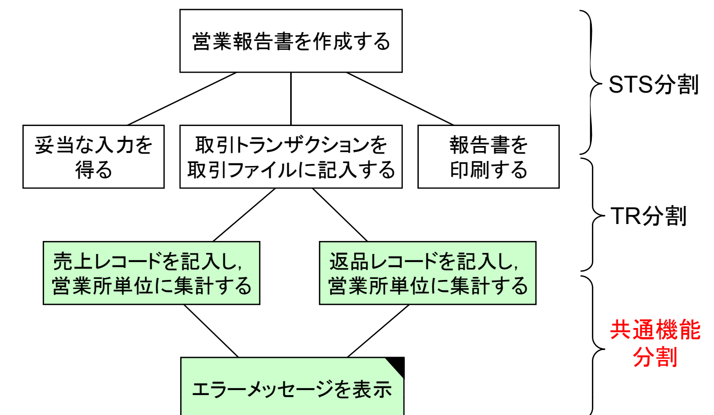
- トランザクション分割(TR分割: transactional decomposition)
  - ✓ 分岐するトランザクション処理ごとにモジュールを設定する

  トランザクション: 1つの処理の単位
  e.g., データベースへの読み込みと更新操作のまとまり

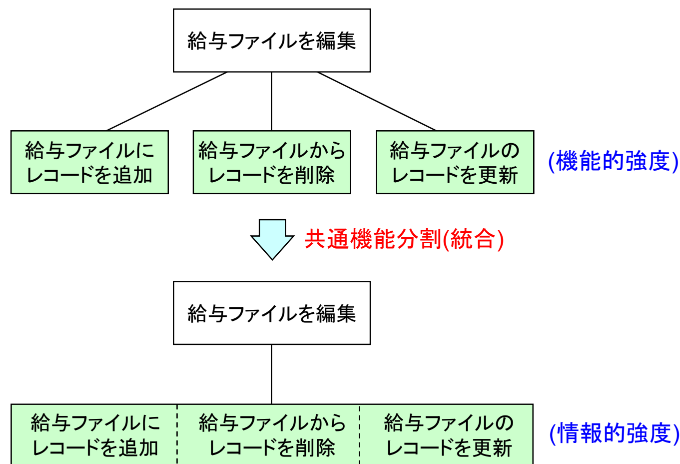


共通機能分割

- 共通機能分割(functional decomposition)
  - ✓ 複数のモジュールに含まれる共通の従属機能を取り出して定義

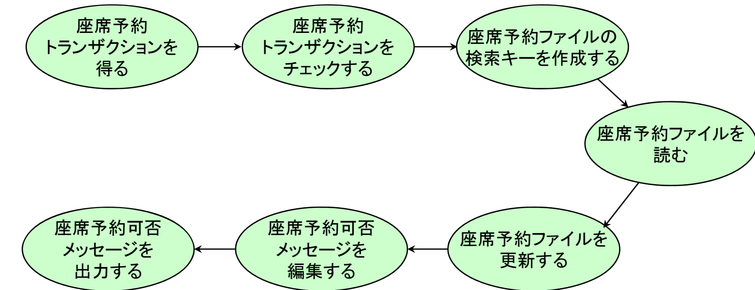


共通機能分割(cont'd)



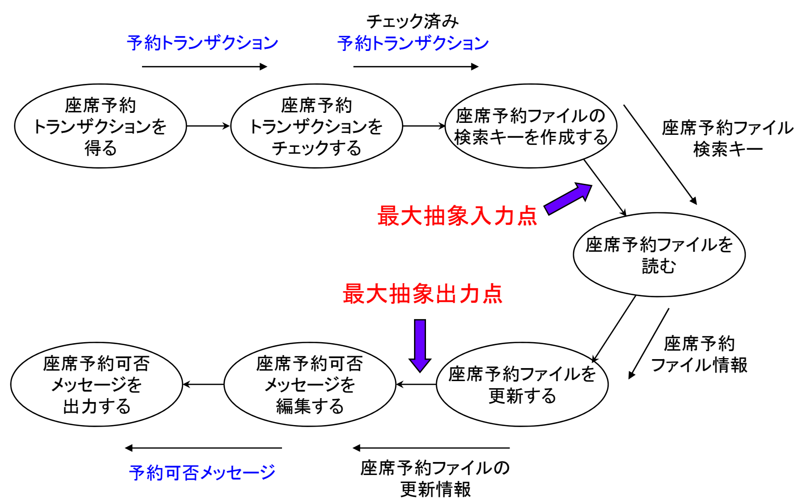
例題: 問題構造図の記述

- 座席予約システム
  (概要) 端末から座席予約トランザクションを受け取り, 座席予約ファイルの座席情報を基に予約の可否を調べ, 座席予約ファイルの更新を行う. また, 端末に予約可否のメッセージを表示する.

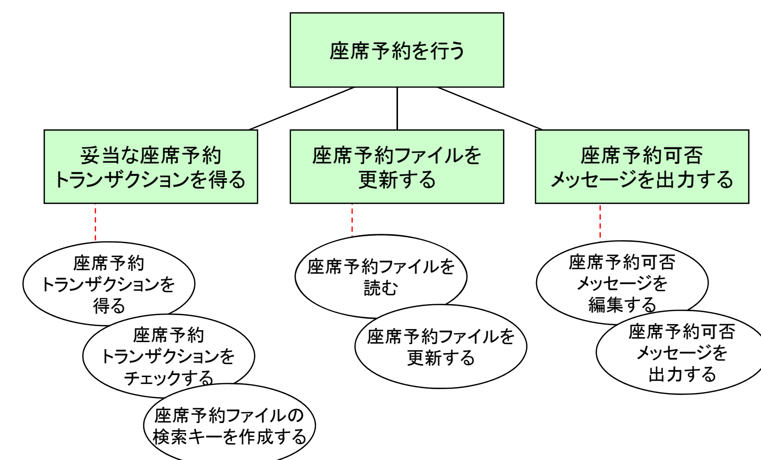


出典: 電子開発学園著,「新版ソフトウェア工学」, SCC出版局

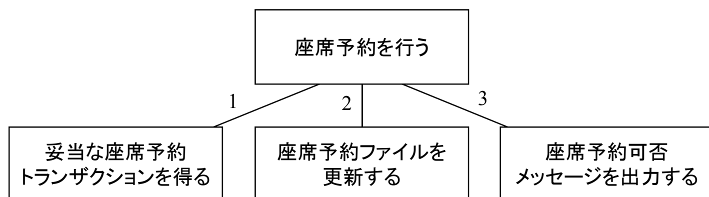
例題: 主要データの識別と最大抽象点の発見



例題: 従属モジュールの定義

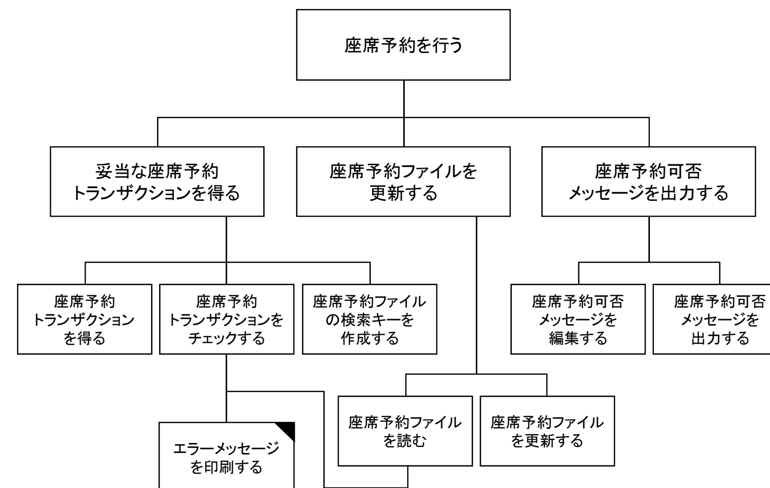


例題: インタフェースの定義

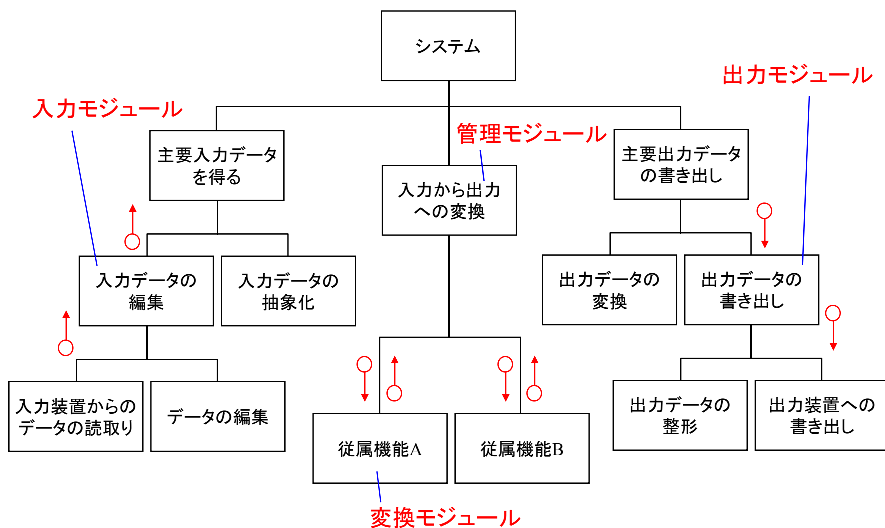


	入力	出力
1	なし	座席予約ファイル検索キー
2	座席予約ファイル検索キー	座席予約ファイルの更新情報
3	座席予約ファイルの更新情報	なし

例題: モジュール構成図



モジュールの一般的構造



演習: システムの記述(再)

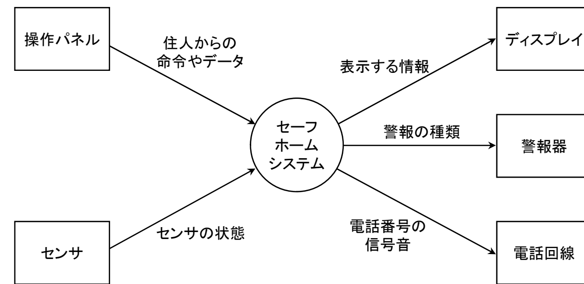
ホームセキュリティシステムの仕様

- セーフホームシステムは，設置時には住人がシステム設定をできるようにし，セキュリティシステムが接続されているすべてのセンサを監視する．操作パネルのキーパッドやキーを通して，住人とやりとりする．
- 設置時にシステムを設定するときには，操作パネルで行う．各センサに番号と種類を割りあて，システムの作動･解除を切りかえるマスタパスワードを設定し，センサイベント発生時の連絡先の電話番号を入力する．
- このソフトウェアはセンサイベントを検知した際に，システムに接続されている警報器を起動する．さらに，システム設定時に住人が指定した遅延時間を経過した時点で，監視サービスの電話番号に住所に関する情報を連絡し，検知したイベントについて報告する．電話回線への接続は，接続されるまで20秒間隔で繰り返される．
- セーフホームシステムとのすべてのやりとりは，ユーザインタフェースサブシステムによって管理される．このサブシステムは，キーパッドや特殊キーを通じて与えられた入力を読み取り，ディスプレイに入力メッセージとシステムの状況を表示する．

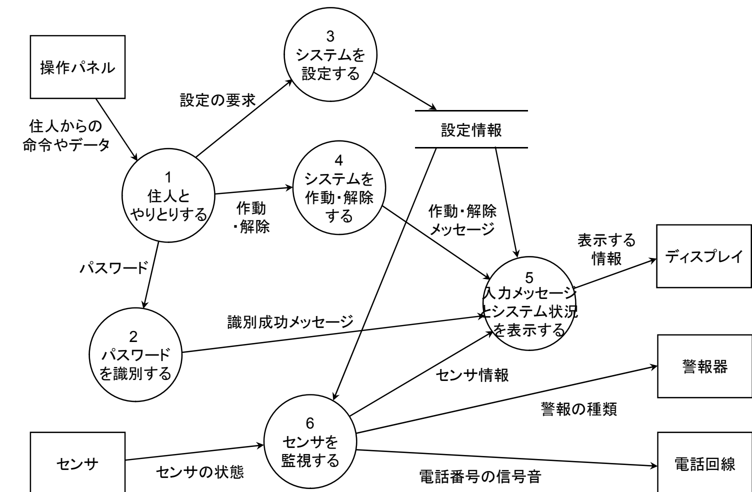
(以下 略)

出典: R. Pressman著，「ソフトウェア工学の伝統的手法」，日科技連

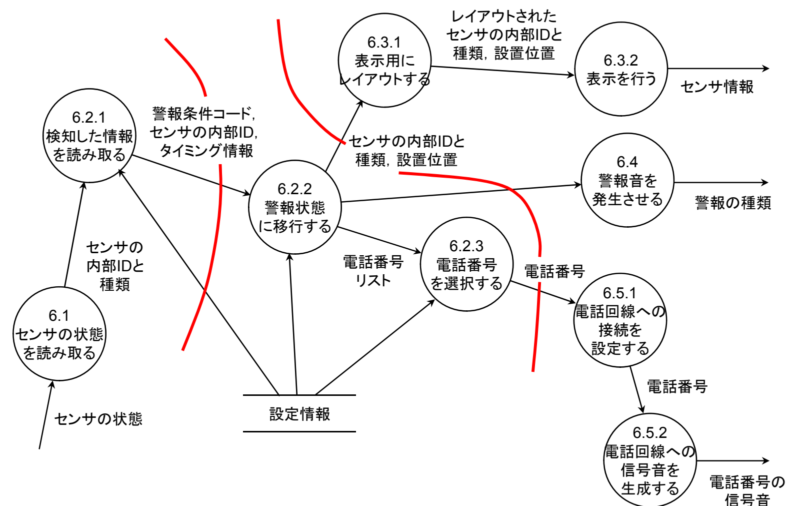
演習：全体文脈図(レベル0)



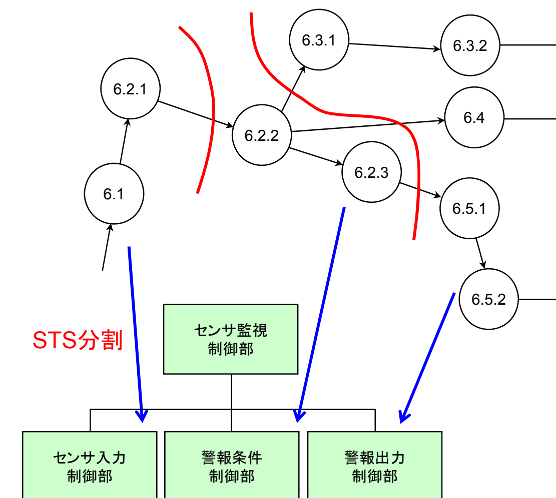
演習：DFD(レベル1)



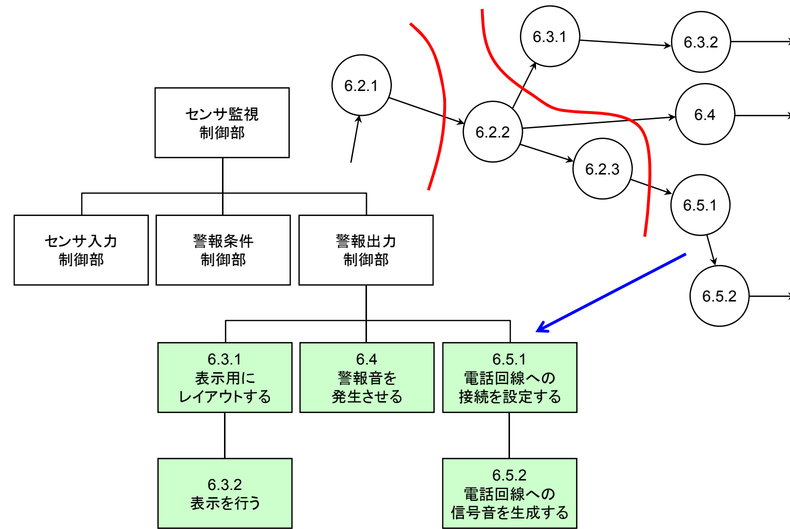
演習：DFDとモジュール分割(1)



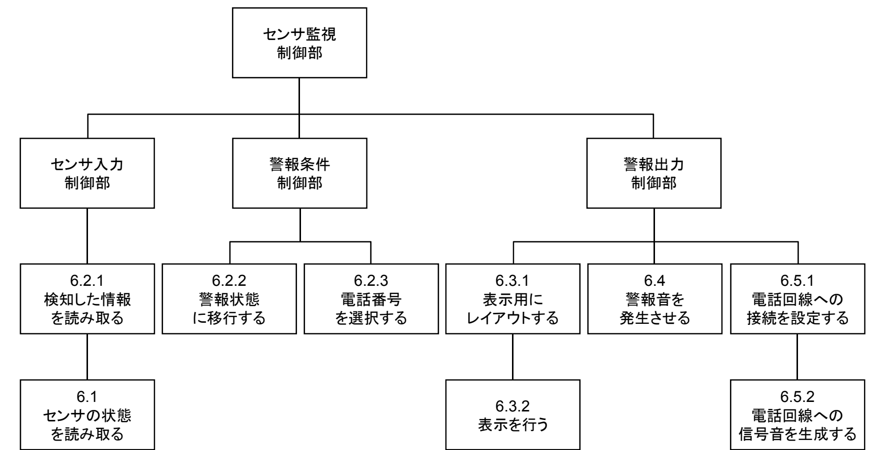
演習：モジュール構成図(1-a)



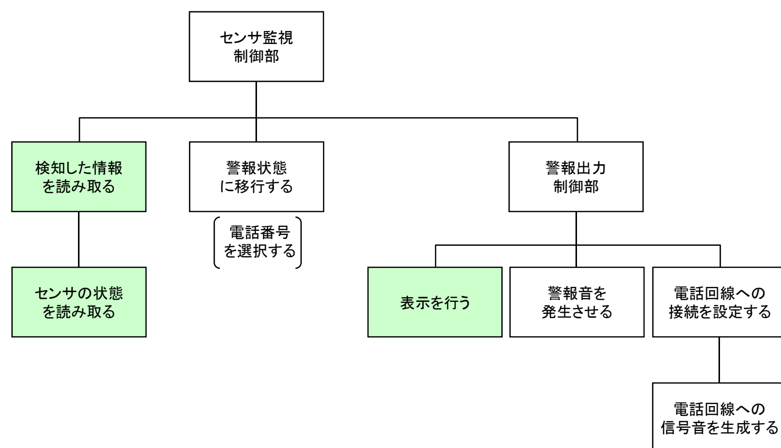
演習: モジュール構成図(1-b)



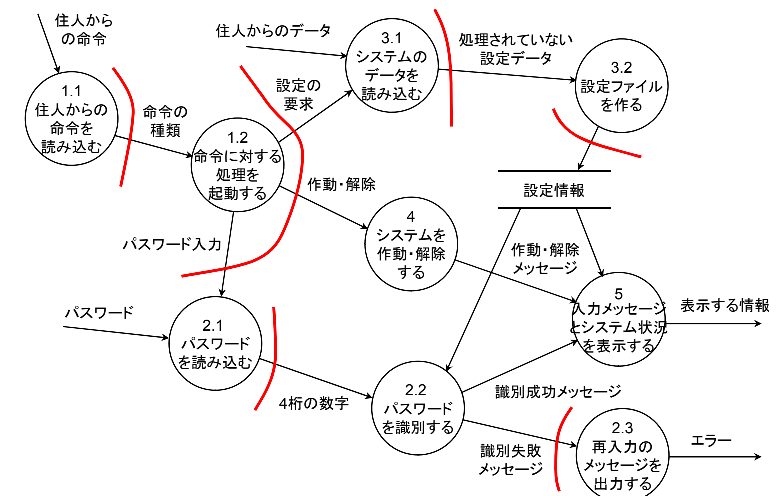
演習: モジュール構成図(1-c)



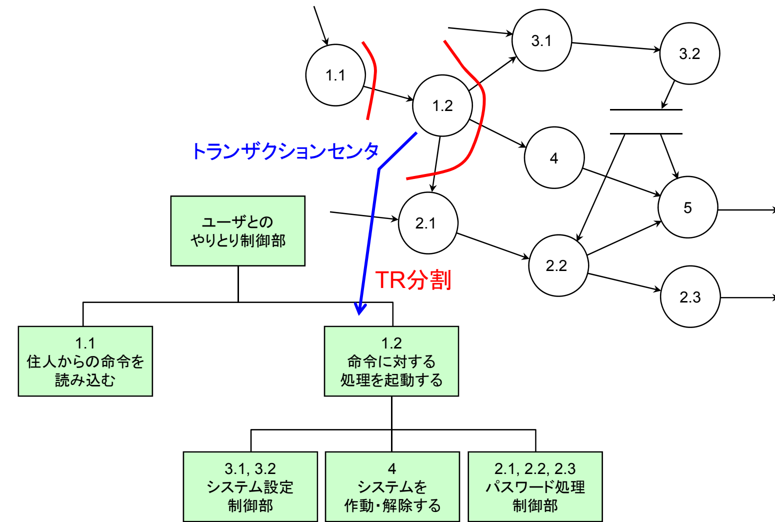
演習: モジュール構成図(1-d)



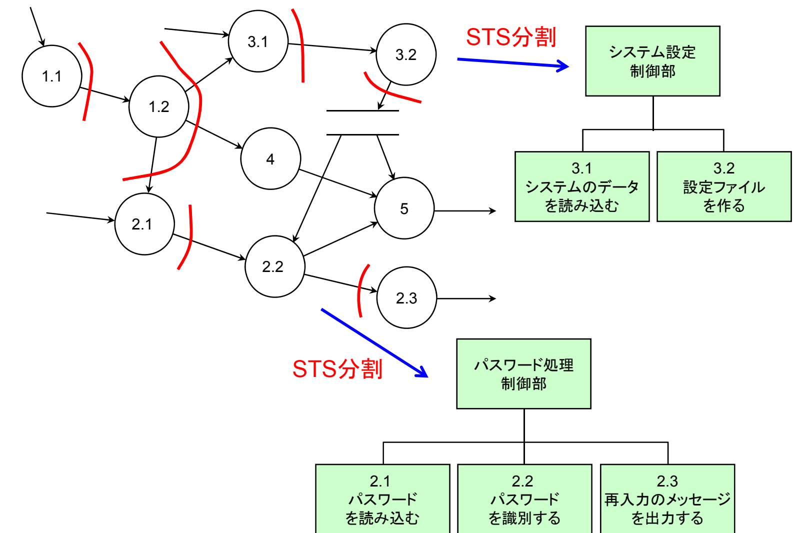
演習: DFDとモジュール分割(2)



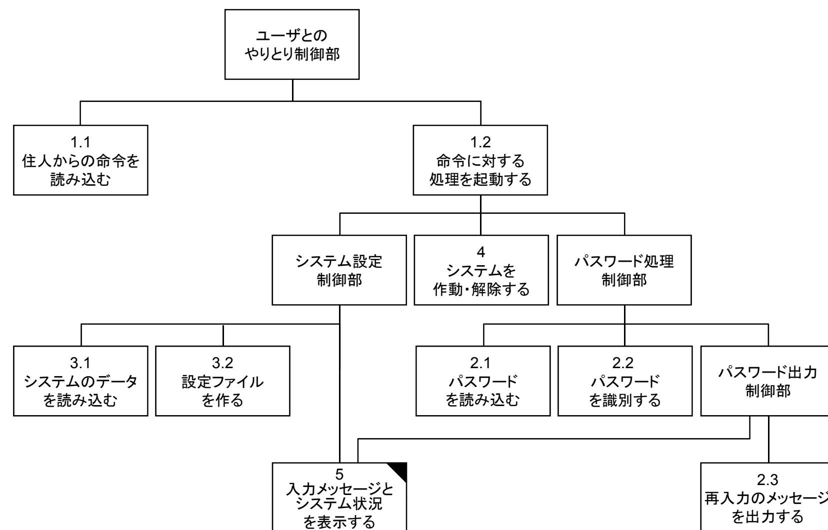
演習: モジュール構成図(2-a)



演習: モジュール構成図(2-b)



演習: モジュール構成図(2-c)



設計の評価基準

- 分割の観点
  - モジュールの大きさ: モジュールを構成する文の数
  - モジュールの簡潔さ: 汎用化の尺度
- 独立性の観点
  - モジュール強度: 個々のモジュール内部での関連の尺度
    = モジュール凝縮度
  - モジュール結合度: 異なるモジュール間の関連の尺度
- 階層構造化の観点
  - 制御範囲と影響範囲
  - ファンイン(fan in)とファンアウト(fan out)

モジュール強度

- モジュール強度(module strength) / モジュール凝縮度(module cohesion)
  - ✓ モジュール内に存在する構成要素(文や機能)の関連の強さ

(1) 暗号的強度, 偶発的強度(coincidental strength)
　特定の機能を持たず，偶然に集められたモジュール

(2) 論理的強度(logical strength)
　見かけ上同じ機能を果たすが，実際には関連した複数の機能を集めたモジュール

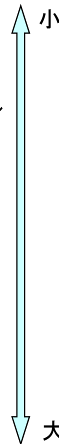
(3) 時間的強度(temporal strength)
　特定の時期に連続して実行される機能を集めたモジュール

(4) 手順的強度(procedural strength)
　逐次的に実行される関連のある機能を集めたモジュール

(5) 連絡的強度(communication cohesion)
　手順的強度，かつ，同じデータを入力あるいは出力する機能を集めたモジュール

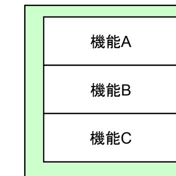
(6) 情報的強度(informational strength)
　特定のデータ構造を扱う複数の機能を集めたモジュール

(7) 機能的強度(functional strength)
　単一の機能を実行するモジュール

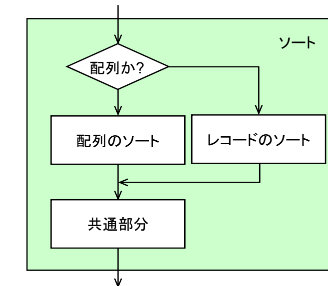


モジュール強度の例(1)～(3)

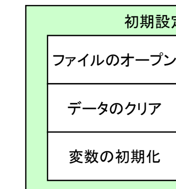
(1) 暗号的強度



(2) 論理的強度

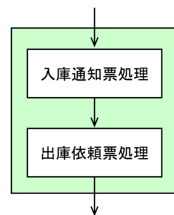


(3) 時間的強度

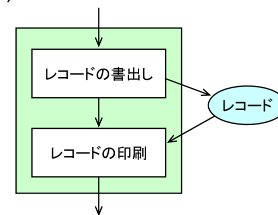


モジュール強度の例(4)～(7)

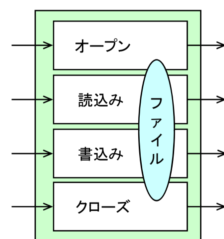
(4) 手順的強度



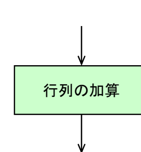
(5) 連絡的強度



(6) 情報的強度



(7) 機能的強度



モジュール結合度

- モジュール結合度(module coupling)
  - ✓ モジュール間に存在する構成要素(文や機能)の関連の強さ

(1) 内容結合(content coupling)
　モジュールどうしがデータや手続きを共有する
　一方のモジュールが他方のモジュールの内容を直接参照する

(2) 共通結合(common coupling)
　モジュールどうしが共通領域にあるデータを参照する

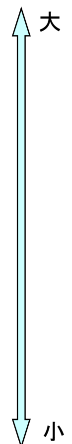
(3) 外部結合(external coupling)
　モジュールどうしが外部に存在するデータ領域を参照する

(4) 制御結合(control coupling)
　呼出しモジュールが呼び出されたモジュールの制御を引数を通して指示する
　データ共有はなし

(5) スタンプ結合(stamp coupling)
　共通領域にないデータの構造体(未使用データ)を引数として受け渡す

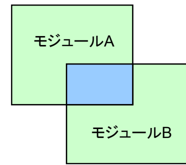
(6) データ結合(data coupling)
　必要なデータだけを引数として受け渡す

(7) 無結合

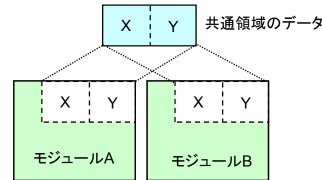


モジュール結合の例(1)～(3)

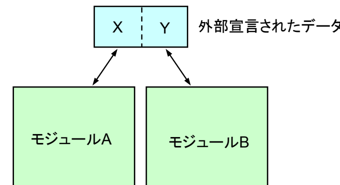
(1) 内容結合



(2) 共通結合

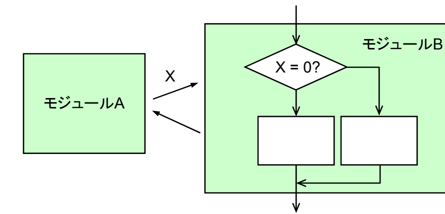


(3) 外部結合

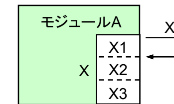


モジュール結合の例(4)～(6)

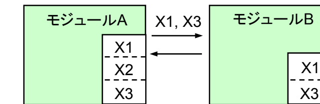
(4) 制御結合



(5) スタンプ結合



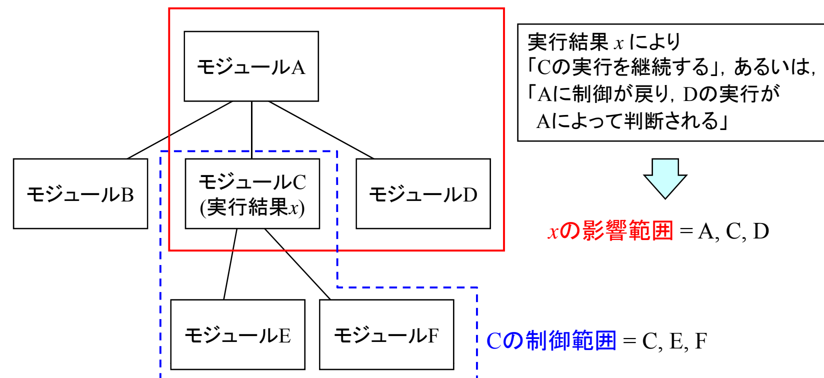
(6) データ結合



制御範囲と影響範囲

- 制御範囲: 対象モジュールとそのモジュールに従属するすべてのモジュール
- 影響範囲: 対象モジュールの実行結果により実行されるすべてのモジュール

影響範囲 ⊆ 制御範囲となるように修正



データ構造に基づく設計

- 入力データ構造と出力データ構造に着目し，プログラムの論理構造を決定
  - ✓ プログラム = 入力データから出力データへの変換
  - ✓ データは特定の処理手順とは独立
    - → データ中心設計
  - ↔ データの流れ(機能)に着目

(1) ジャクソン構造分割(Jackson法)

(2) ワーニエ法(Warnier法)

ジャクソン法

- 入力データ構造と出力データ構造の対応関係から
  プログラムの論理構造を決定

手順:

(1) データ構造の定義
　入力データ構造と出力データ構造を分析し，
　4つの構成要素(基本，連続，繰返し，選択)でデータ構造図を作成

(2) データ構造の対応付け
　入力データ構造図と出力データ構造図の構成要素間の対応関係を決定
　構造が不一致のとき，中間のデータ構造を導入

(3) プログラム論理構造の決定
　出力データ構造を基に論理構造を定義

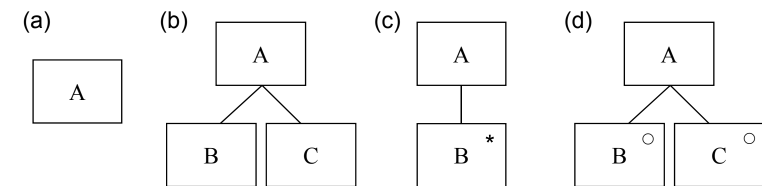
データ構造図の構成要素

(a) 基本
　これ以上分割できない構成要素．1つのデータ項目が相当

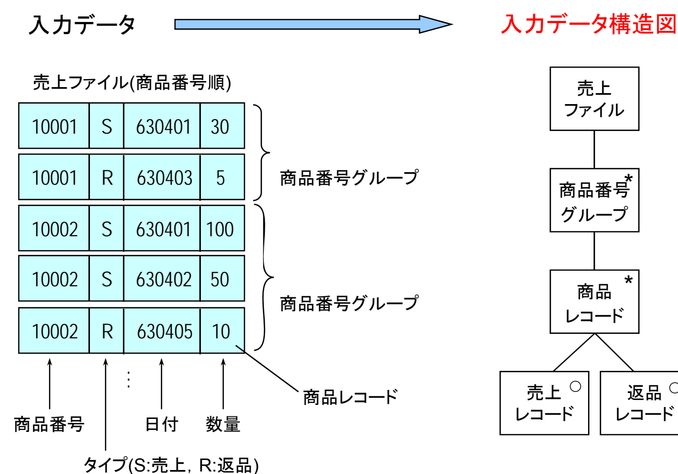
(b) 連続
　異なる複数の基本要素からなる構成要素で，それぞれの構成要素は
　1度だけ順番に現れる．複数データ項目を持つレコードに相当

(c) 繰返し
　同一の構成要素が繰返し現れる構成要素．

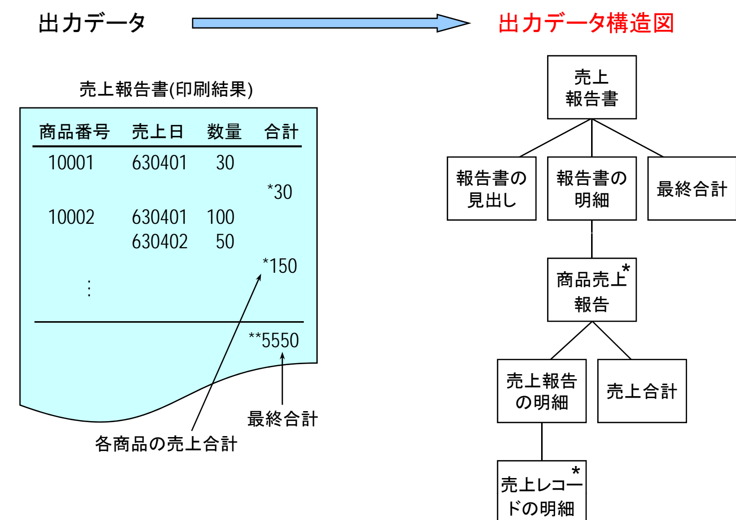
(d) 選択
　複数の構成要素のうちどれか1つを選択する構成要素．



入力データ構造図の例

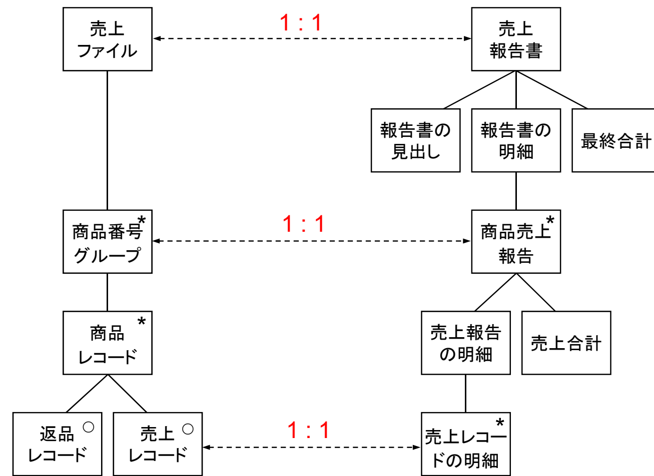


出力データ構造図の例

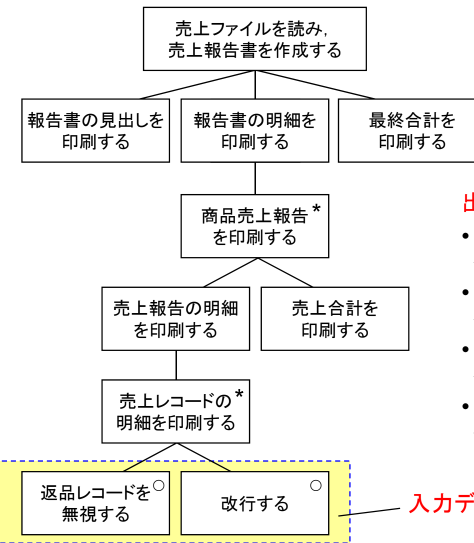


構成要素間の対応

入力データ構造図　　出力データ構造図



プログラム構造



入力データ構造図からの補正

出力データ構造図に対応

- データ構造図の基本
  → プログラムの1命令
- データ構造図の連続
  → 逐次処理
- データ構造図の繰返し
  → 繰返し
- データ構造図の選択
  → 選択

ワーニエ法

- 入力データ構造と出力データ構造から直接プログラムの論理構造を決定

手順:

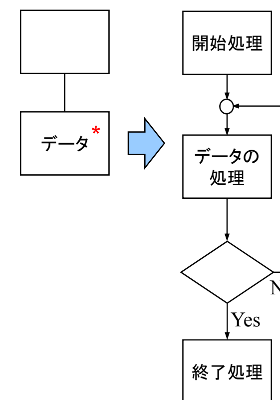
(1) データ構造の定義
　入力データ構造と出力データ構造を分析し，
　4つの構成要素(基本，連続，反復，選択)でデータ構造図を作成

(2) プログラム論理構造の決定
　入力データ構造を基にプログラムの論理構造を決定する．

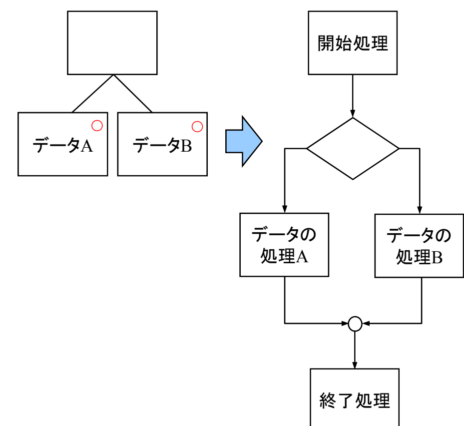
- 繰返しのデータ構造 → 繰返しの論理構造
- 選択のデータ構造 → 選択の論理構造
- 入力データ構造にあり，出力データ構造になし → 無視
- 入力データ構造になし，出力データ構造にあり → 入力データの加工
- 各論理構造の前後に開始部と終了部を付加

データ構造とプログラム論理構造

(a) 繰返し

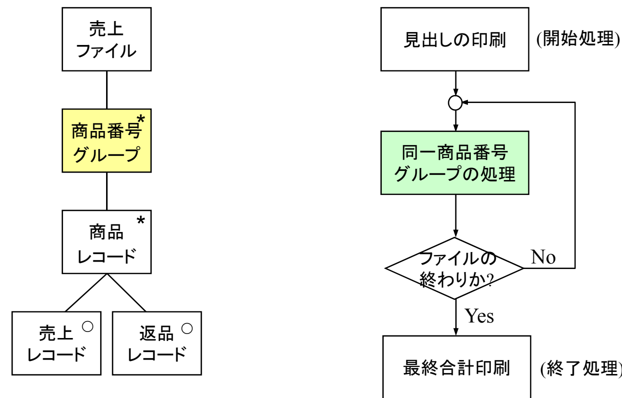


(b) 選択



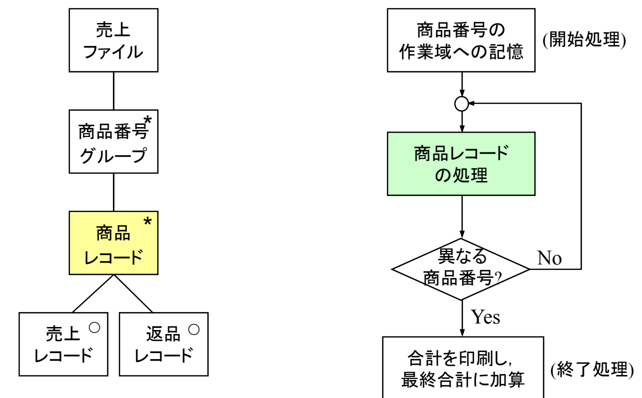
プログラム論理構造の例(1)

入力データ構造図 → プログラム論理構造



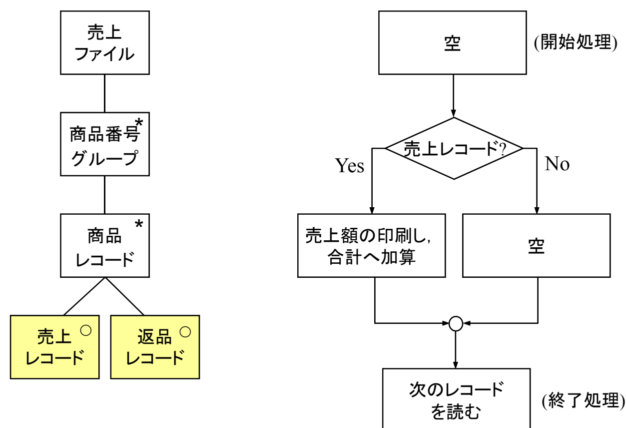
プログラム論理構造の例(2)

入力データ構造図 → プログラム論理構造

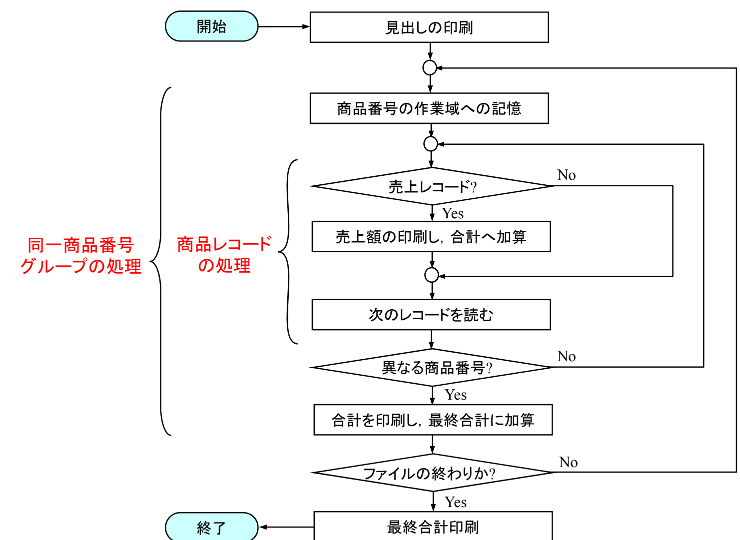


プログラム論理構造の例(3)

入力データ構造図 → プログラム論理構造



プログラム論理構造の例(全体)



データの正規化

- データの正規化/標準化 = データの部品化
  - ✓ データは他のデータと重複する部分を持たない
  - ✓ 各データは固有の役割を果たす

正規化手順:

(1) データ項目間の相互関連性の分析

(2) 最も簡潔なデータ項目の組に分解し，レコードの単位とする

- 1次正規化: 繰返し部分の消去
- 2次正規化: キーに1:1で従属する属性だけに分解
- 3次正規化: キー以外の従属関係を分解

非正規形データ

人事技能レコード								
社員番号	社員名	部門コード	部門名	部門長名	技能保有			
					コード	技能名	経験年数	レベル
10100	A	500	開発1	P	DB1 DB2 PG1	データベース データベース プログラミング	3.0 1.0 2.0	4 2 3
10200	B	500	開発1	P	DB1 PG1 PG2	データベース プログラミング プログラミング	3.0 3.0 1.0	4 4 2
10300	B	600	シス1	Q	PG1	プログラミング	2.0	2
⋮								

1次正規形データ

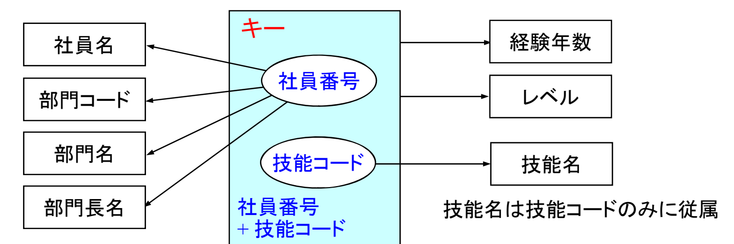
- 非正規形データ
  - 技能記録 = 社員番号 + 社員名 + 部門コード + 部門名 + 部門長名 + { 技能コード + 技能名 + 経験年数 + レベル }

社員番号	社員名	部門コード	部門名	部門長名	技能保有			
					コード	技能名	経験年数	レベル

社員に対する繰返し　　技能に対する繰返し

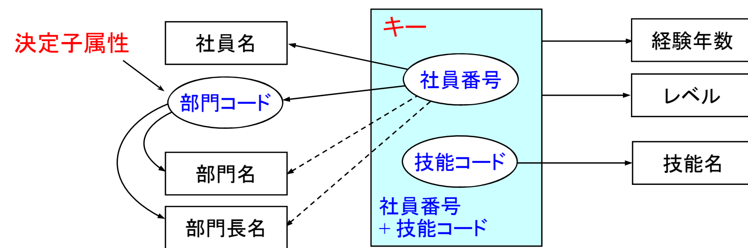
- 1次正規形データ
  - キー
  - 社員 = 社員番号 + 社員名 + 部門コード + 部門名 + 部門長名
  - 保有技能 = 社員番号 + 技能コード + 技能名 + 経験年数 + レベル
  - 連結キー

2次正規形データ



- 2次正規形データ
  - 社員 = 社員番号 + 社員名 + 部門コード + 部門名 + 部門長名
  - 保有技能 = 社員番号 + 技能コード + 経験年数 + レベル
  - 技能 = 技能コード + 技能名

3次正規形データ



- 3次正規形データ
  - 社員 = 社員番号 + 社員名 + 部門コード
  - 部門 = 部門コード + 部門名 + 部門長名
  - 保有技能 = 社員番号 + 技能コード + 経験年数 + レベル
  - 技能 = 技能コード + 技能名