ソフトウェア工学
第4回

土田　隼之

週	授業内容・方法	週ごとの到達目標
1週	ソフトウェアの性質と開発	ソフトウェア開発の特徴および課題について少なくとも一つ上げられ、その理由を言える。
2週	ソフトウェア開発プロセス	複数の開発プロセスモデルを挙げ、それぞれの特徴を言える。
3週	要求分析	要求分析とプロトタイピングの関係性や有用性について言える。
4週	ソフトウェア設計	モジュールの結合度の低い場合と高い場合のモジュール間の依存性について述べ、モジュール結合度の低い具体例を言える。
5週	プログラミングとテスト	誤り混入をさせないためのプログラミング手法およびテスト効率を向上させる技法について言える。
6週	テストと保守	保守容易性を確保するための方策について、考察し、述べることができる。
7週	グループワーク	前半6週に関する課題を、グループワークで取り組む。
8週	中間試験	前半に習得した項目について確認する。
9週	オブジェクト指向１	身の回りのモノに関して、クラスとインスタンスという言葉を用いて説明できる。
10週	オブジェクト指向２	オブジェクト指向プログラミングの特徴について言える。
11週	ソフトウェア再利用	ソフトウェア再利用の重要性とその困難さについて言える。
12週	プロジェクト管理	プロジェクト管理の重要性を述べることができる。
13週	品質管理	品質管理手法について言える。
14週	ソフトウェア開発規模と見積もり	ソフトウェア開発規模の見積もり手法について言える。
15週	グループワーク	後半6週に関する課題を、グループワークで取り組む。
16週	期末試験	後半に習得した項目について確認する。

今日の内容

1)ソフトウェア設計とは

　#要求仕様をコーディング出来る仕様に変換する

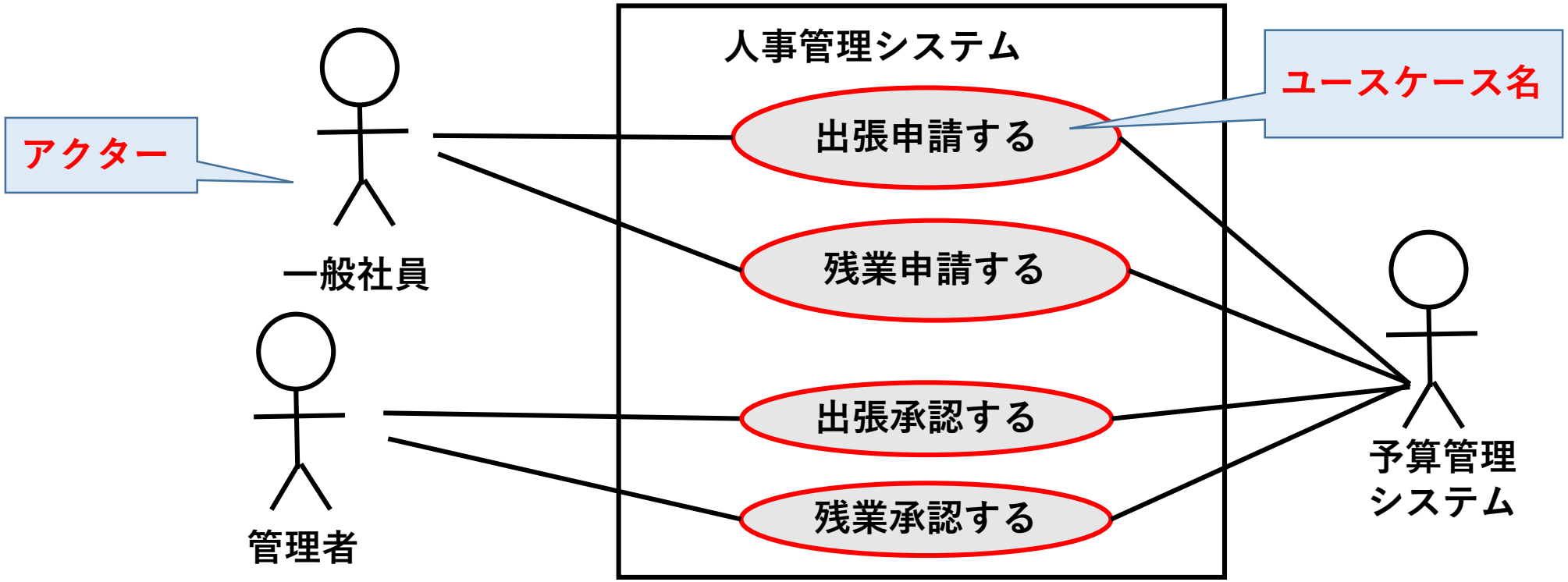
2)外部設計

3)内部設計

前回)要求仕様

ユーザや自分のアイデアの「要求仕様」をまとめる方法として、UMLのユースケース図を用いる方法がある。ユースケース図では、アクターという概念を用いて、システムの使用例を記述する。

→要求仕様は「大まかなシステム概要と、システム開発規模がわかる詳細度」で作成するため、一般的には実装するには不十分(あいまい)である。

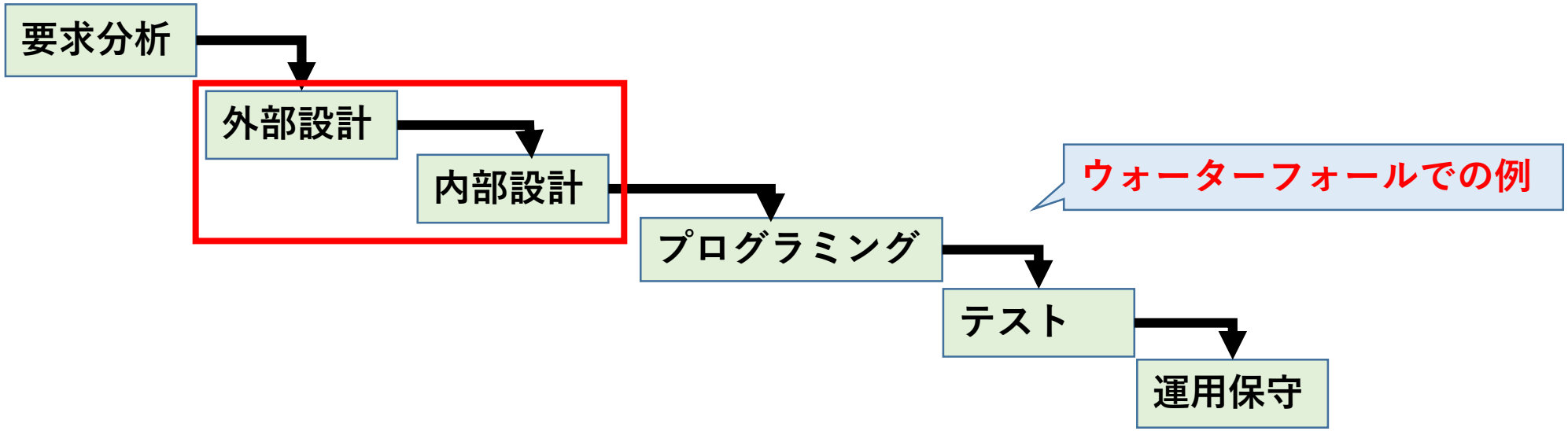


1)要求仕様とソフトウェア設計

「要求仕様」で得られた情報をもとに、実装が可能な仕様書を作成する。

a)まず、プログラミング詳細には立ち入らないシステム外部とのやりとりを扱う仕様書(内部設計書)を先に作成する。

b)その後、実装するソースコードの詳細を扱う仕様書(内部設計書)を作成する。



1)ソフトウェア設計における基本事項

要求仕様は、そのままでは実装できない
→実現方法は沢山あるが、具体的にはどうする？？
例:関数構成は？？、とり得る値の最大値は？？

- 設計とは

システム開発では、要求分析とプログラミングの間に設計と呼ばれる要求仕様からプログラム仕様に変換する段階がある。顧客の視点から、実装の視点へ転換が行われる。**アウトプットは、プログラム仕様書**となり、開発メンバ間の情報共有手段もかねる。

観点:1)顧客要求の満足、
2)開発コストの低減、
3)内部構造的な品質確保

1)ソフトウェア設計へのアプローチ

- 良い設計とは
　「要求品質を満たすソフトウェア構造を作る」こと。

品質例:機能適合性、信頼性、使用性、、、など

- 戦略
　システムの複雑さを克服することが重要である。複雑であるほど、後工程で大きな問題となるケースが多い。

a)抽象化とモデルの利用

b)分割と階層化

d)独立性

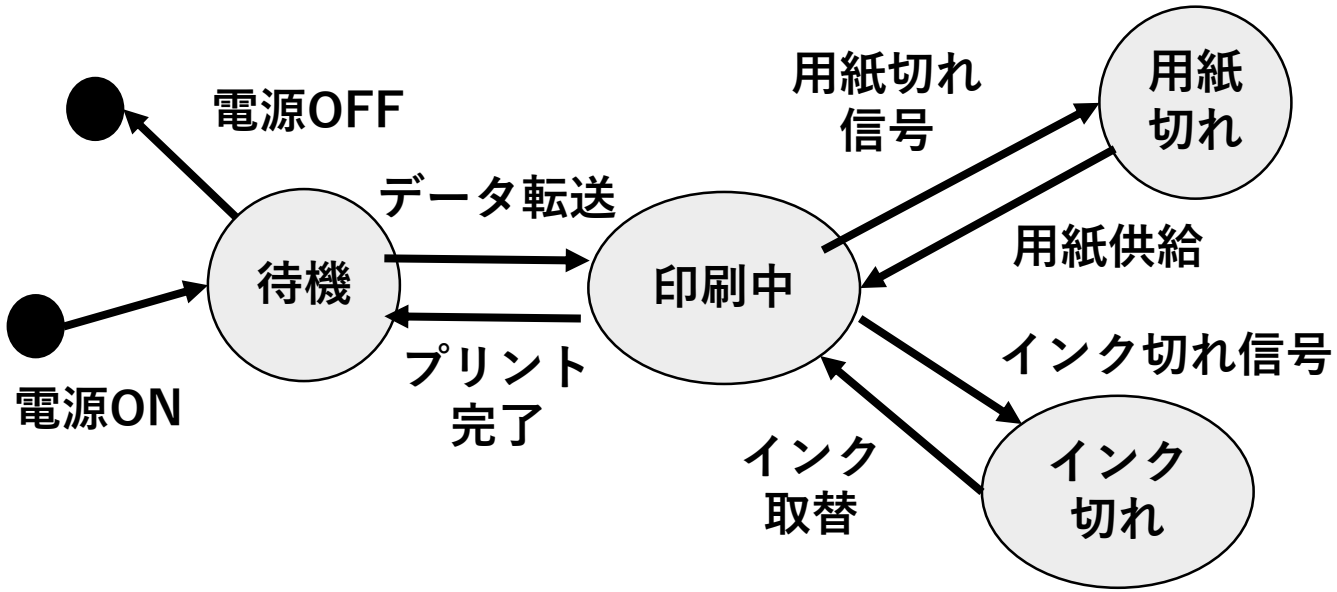
複雑さを減らしていく

1-a)抽象化とモデルの利用

- **抽象化**:検討するシステムの性質のみを浮かび上がらせるために、適切なモデルを使用して、その性質を正確に捉えると同時に、関係のない事項をそぎ落としていく作業

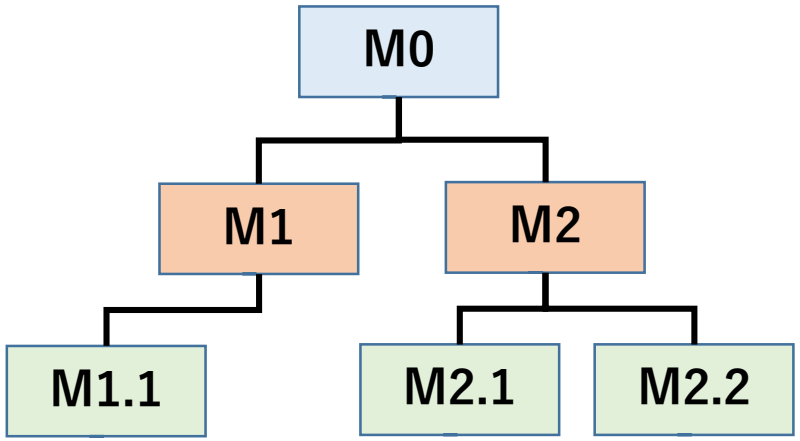
  振る舞いの記述としては、状態遷移図が用いられることも有る。

#要求仕様が文章とすると、文章を状態遷移図にして抽象化



1-b)分割と階層化

- **分割**:システムはそれを構築する要素数Nに対して2乗のオーダで複雑さが増大する。そのため、システムを複数の独立部分に分け、それぞれの部分を別々に設計していくことで、複雑さの増大を抑える手法
- **階層化**:分割を個々の部分に再帰的に適用していくことで、システムの構成要素を階層的に組織化する手法



1-c)独立性

- 分割と階層化を効率的に機能させるためには、なるべく要素間の独立性を高めることが特に重要となる。要素間の独立性が高いということは、要素内は密な結合であるが、他の要素とのインタフェースは疎かつ単純である状態を指す。

→この状態を達成することにより、設計変更時の影響範囲が狭まり、結合時のエラーが起こる可能性も少なくなる。

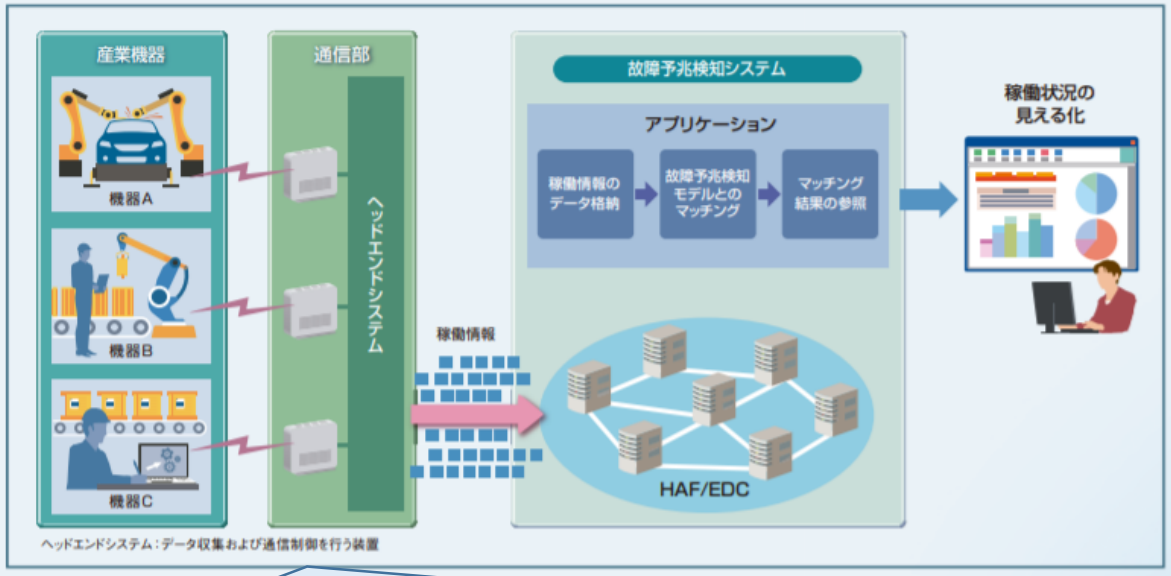
分割したコンポーネントを結合

1)ソフトウェア設計の例

機器(スマートメータ、MRI)から、情報(電力使用量やエラーメッセージ)をシステムの所定のWebAPI(URL)へリアルタイムに送付。

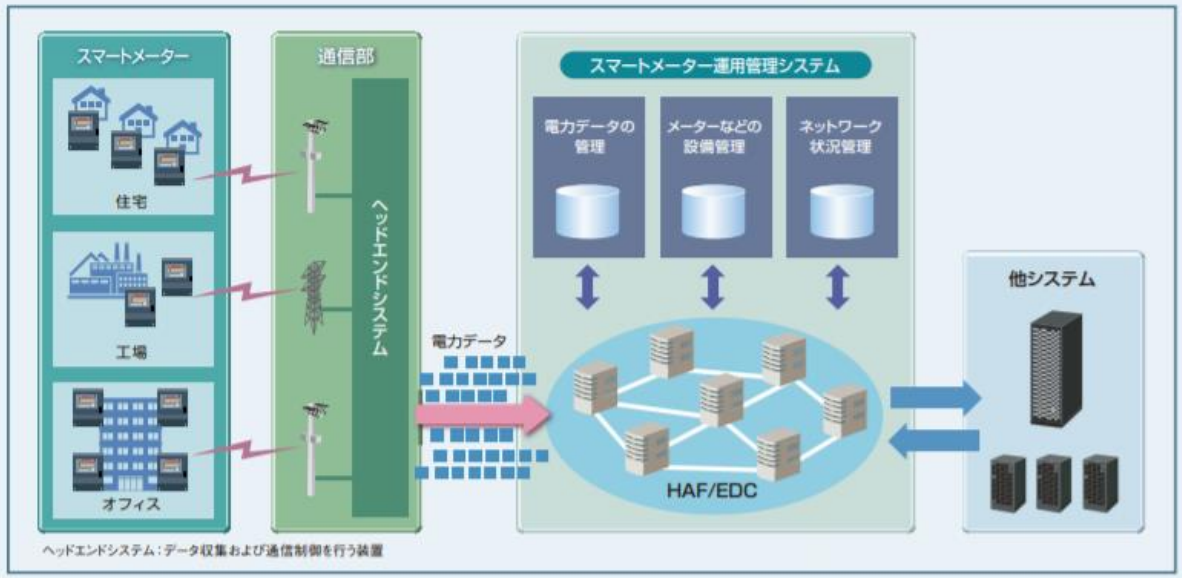
事例 2 産業機器の故障予兆検知に

世界中で稼働する産業機器から稼働情報を収集し、リアルタイムにデータを格納、分析、参照。
稼働状況に応じたメンテナンスを可能にする故障予兆検知システムに適用できます。



事例 1 スマートメーターの運用管理に

施設に設置されたスマートメーターから送信される大量の電力データを収集し、並列分散により短時間で処理。
電力データ、メーターなどの設備、ネットワーク状況を統合的に運用管理するシステムに適用できます。



機器からはエラー番号(例:321)は出る。
エラー番号から、交換が必要な場所はわかる
→まずは、エラー情報を集めるシステムを構築

→URL毎に処理(例:データ格納、取得)を決めるWebAPIを策定
例:特定URLにデータを送ると、サーバ側にデータ格納してくれる

https://www.hitachi.co.jp/products/it/society/product_solution/telecom/catalog/HAF_EDC.pdf

1)web APIの例

Yahoo!ジオコーダAPI

アプリケーションの管理

webAPI
→特定URLにhttpアクセスすると、
特定処理をしてくれる

リクエストURL

XML
https://map.yahooapis.jp/geocode/V1/geoCoder

JSONP
https://map.yahooapis.jp/geocode/V1/geoCoder

- !ローカルサーチAPI
- o!ジオコーダAPI
- Yahoo!リバースジオコーダAPI
- カセットサーチAPI
- 気象情報API
- 郵便番号検索API

概要

- Yahoo!ジオコーダAPI（以下、ジオコーダAPI）では、キーワード検索の機能を提供します
- 住所を指定して、その位置情報（緯度、経度）を出力します。

リクエストパラメーター一覧

「RESTリクエストの構築（GET）」をご参照ください。

パラメータ	値	説明
appid（必須）	string	アプリケーションID。詳細はこちらをご覧ください。
query	string	住所文字列。
ei	string	入力検索文字列のエンコード形式： • *UTF-8* - UTF-8形式（デフォルト） • *EUC-JP* - EUC-JP形式 • *SJIS* - SJIS形式
lat	float	中心の緯度。
lon	float	中心の経度。

https://developer.yahoo.co.jp/webapi/map/openlocalplatform/v1/geocoder.html

1)web APIの例2

広く使われているWebAPIサービスがどのような物かを日頃アンテナを高くしておくと、自分がAPI設計する時があれば役に立つかも

WebAPIの使い方(GETリクエスト)

GETとは

GETとは、「リソース（情報）を取得する」ためのHTTPメソッドです。ウェブサイトをブラウザーで見るとき、内部ではGETによるリクエストが行われています。Web APIも情報を取得する性質のものが多いため、GETをよく使います。

例として、ルビ振りAPIを使って「明鏡止水」の読み方を調べてみます。
GETの場合は、ブラウザーで以下のURLを見ることでもAPIの結果をプレビューできます。
ただし「<あなたのアプリケーションID>」の部分は登録したアプリケーションIDに置き換えてください。
（アプリケーションIDの登録方法はご利用ガイドをご覧ください）

```
http://jlp.yahooapis.jp/FuriganaService/V1/furigana?appid=<あなたのアプリケーションID>&sentence
```

webAPIでは、URLのパラメータ(?以降)で処理の追加情報を渡す

リクエストパラメーター一覧

「RESTリクエストの構築（GET）」をご参照ください。

パラメータ	値	説明
appid（必須）	string	アプリケーションID。詳細はこちらをご覧ください。
query	string	住所文字列。
ei	string	入力検索文字列のエンコード形式： • *UTF-8* - UTF-8形式（デフォルト） • *EUC-JP* - EUC-JP形式 • *SJIS* - SJIS形式
lat	float	中心の緯度。
lon	float	中心の経度。

https://developer.yahoo.co.jp/appendix/request/rest/get.html

https://e-words.jp/w/URL%E3%83%91%E3%83%A9%E3%83%A1%E3%83%BC%E3%82%BF.html

今日の内容

1)ソフトウェア設計とは

　#要求仕様をコーディング出来る仕様に変換する

2)外部設計

3)内部設計

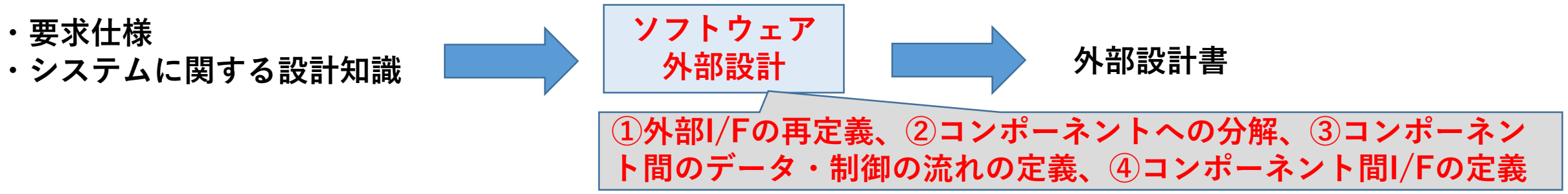
2)外部設計

システムを外部から見たときのシステムの振る舞いや構成を定義する工程

→UIや、システムが持つべき機能や性能、構成を定義

#プログラムの書き方などは、外部設計の段階では考えない

要求仕様から外部設計書を作成する。外部インタフェース(I/F)を再定義し、システムを構成するコンポーネントに分解して、要求仕様を満たすようにデータと制御の流れを決定する。



2)外部設計の作成例

システムの各側面ごとに個別設計を行って、それらをまとめて外部設計書とするのが一般的である。構築システム毎に、具体的な手順は異なる。

a:業務フローの作成

b:サブシステムへの分解

c:画面や帳票のレイアウト作成

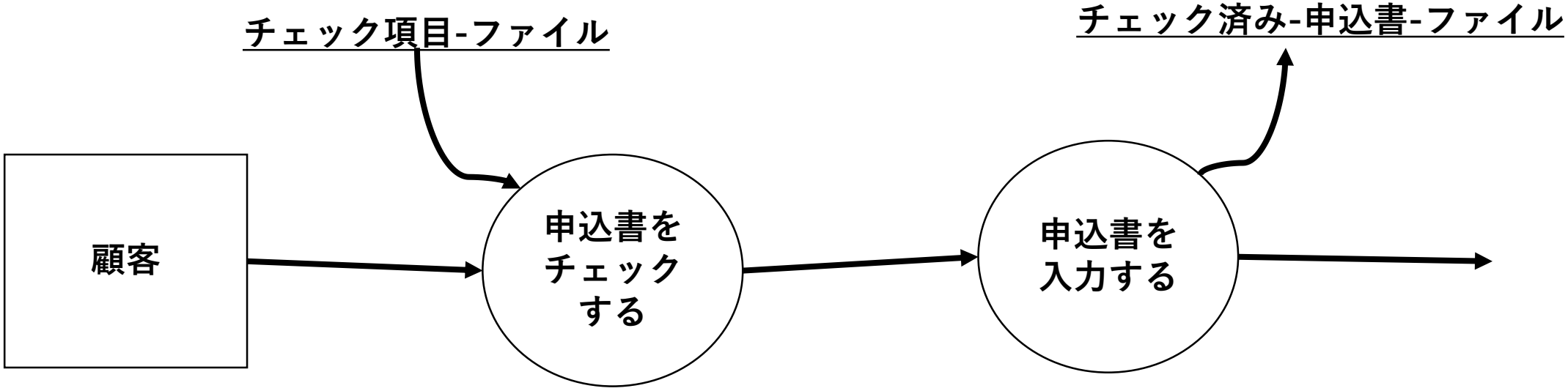
d:コード設計

e:システムインタフェース設計

2)外部設計の流れ (a:業務フローの作成)

システム化の対象である業務を把握するため、業務フローを作成する。システム化しない部分も含めて、すこし広い範囲について業務フローを作成することで、システム全体に対するシステム化範囲の位置付けが読み取れる。

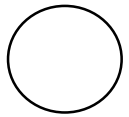
業務フローを記載する図法としてDFD(Data Flow Diagram:データフロー図)がある。DFDは、業務やシステムにおけるデータの流れを表現する記法である。



2)外部設計の流れ (a:業務フローの作成)

データフロー:データの流れを表す。矢印はデータの送り元から送り先へ向かって引く。異なるデータは別の矢印として表す。処理を始めるきっかけを表す目的には使用しない。

プロセス:データに対する処理を表す。プロセスに入るデータフローを、プロセスから出ていくデータフローへ変換する働きを持つ。



ファイル:一時的なファイルの保存場所を表す。コンピュータ上に存在しなくとも、「申込書受領箱」のような業務上存在するファイルも表せる。

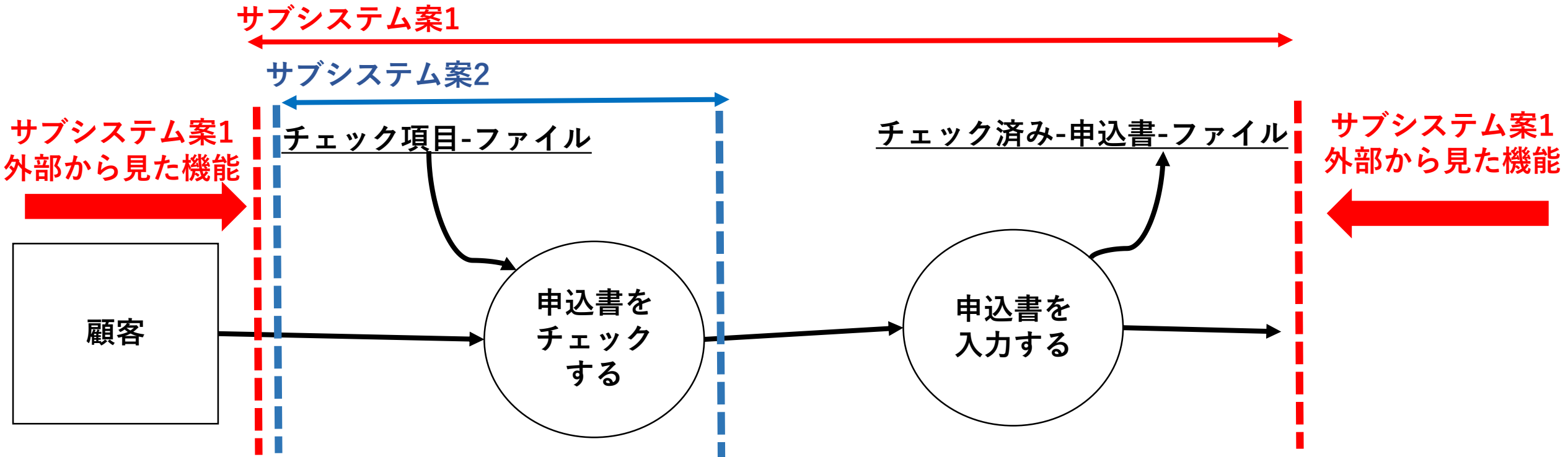
データ源泉とデータ吸収:データ源泉は、データが発生する場所を表し、データ吸収はデータがたどり着く場所を表す。記述しようとするDFDの範囲外からデータが入り、それがまた範囲外に出るとき、それぞれをデータ源泉、データ吸収で表現する。

2)外部設計の流れ (b:サブシステムへの分割)

システムをいくつかのサブシステムに分割し、サブシステム単位で開発するのが一般的である。サブシステムへの分割は、業務フローを基に、業務上の作業単位やシステム開発の効率などを考慮しながら行う。

その後、分割したサブシステムごとに、外部から見た機能を定義する。

サブシステムの中に複数の機能が含まれる事があるが、その場合はサブシステムをさらに分割するのが一般的である。



2)外部設計の流れ (c:画面・帳票レイアウト)

ディスプレイに表示する画面構成や、帳票(プリントアウトする用紙の構成)などのユーザインタフェースの定義は外部設計で行われる。

帳票レイアウト:一般に固定されており、帳票のデザイン、各項目の表示内容、表示桁数、表示するデータのタイプ、その項目は必須か任意かなどを記述する。

画面レイアウト:1)情報の表示だけでなく、2)ユーザの入力を受け付け、入力を画面表示に反映させる双方向性を持つ。ある操作中に、複数の画面に遷移することもあるが、画面レイアウトでは、この遷移も記述する。

2)外部設計の流れ (d:コード設計)

システムが取り扱う情報のうち、書き表し方が複数あったり、入力者によって表記にバラつきが生じたりする可能性がある情報はコード化を検討する。

コード化:ある内容に対して、番号や英数記号で作った特定のコードを割り当てる事#例:性別、都道府県名(兵庫:30,大阪:29)、商品名など

外部設計の段階で、値とコードの対応を定義するのが一般的である。

2)外部設計の流れ (e:システムインタフェース設計)

外部とのデータのやり取りがあるシステムでは、システムインタフェースの設計を行う。システムインタフェース設計では、外部のシステムに渡すデータや受け取るデータの仕様を定義する。

例えば、外部とやり取りするファイルの仕様である「ファイル仕様」
やり取りするデータの内容である「データ交換仕様」がある。

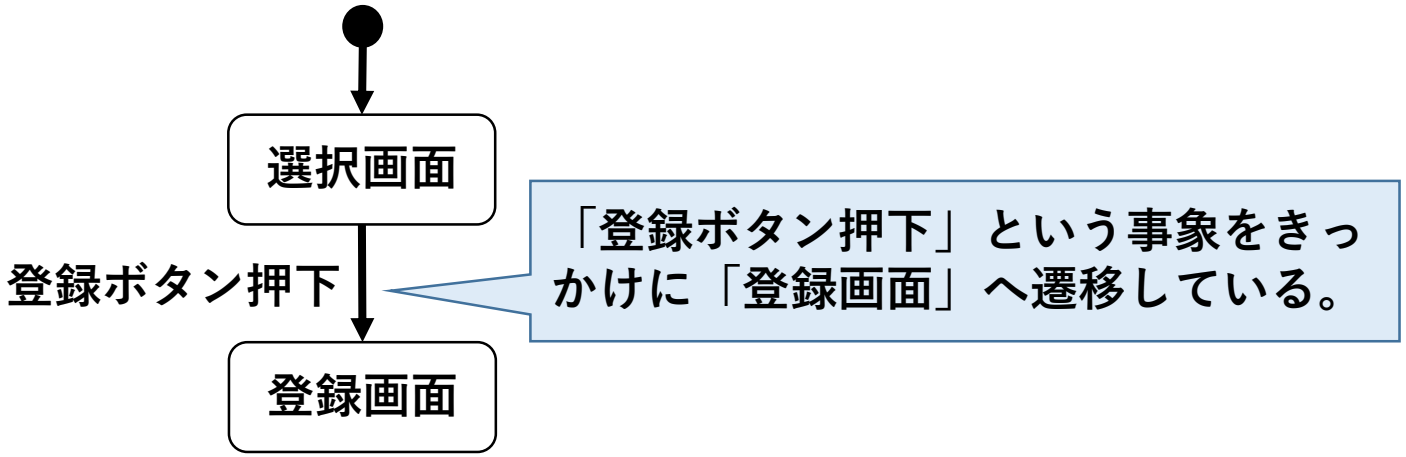
ファイル仕様:文字コード、改行コード、フィールド構成、値のタイプ、空データの可否、最大レコード数などを定める。
データ交換仕様:使用するネットワーク媒体の種類、トランスポートプロトコル、データの形式などを定める。

2)外部設計の流れ (画面遷移図の作成例)

具体例として、UI画面の遷移を扱う画面遷移図について考える。

以下の流れで作成する。

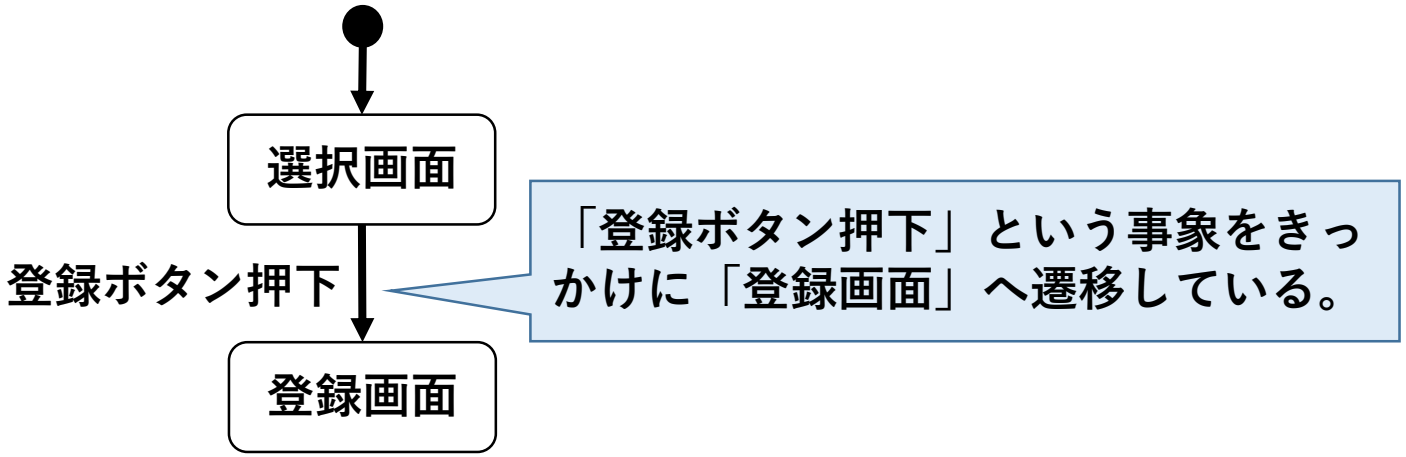
・「ユースケース記述を実現する画面」と「画面が含む項目」を洗い出す
　画面一覧の作成　#画面項目、ボタン、リンク
・ユースケース記述を実現する画面遷移を洗い出す
　画面遷移図 #例えば、UMLのステートマシン図で記述される



2)ステートマシン図

システムやクラスの「状態」と「遷移」を表現するための図として、ステートマシン図がある。「状態」とは、開発対象のシステムを理解する上で重要となる状況に対して名前をつけたもの。例えば、登録画面などが状態となる。

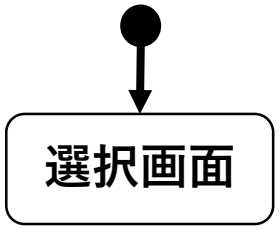
システムは、動いている間も特定の事象をきっかけとして状態が変わり続ける。この「状態が変わること」を「遷移」という。



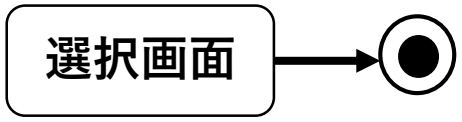
2)ステートマシン図

単純状態:システムなどの状態を表現する要素が「単純状態」であり、単に「状態」ともいう。例えば「選択画面」は状態である。

開始疑似状態:システムの最初の状態を「開始疑似状態」といい、黒く塗りつぶされた丸で表す。「疑似状態」とは、遷移の制御を行うための疑似的な状態を意味する。



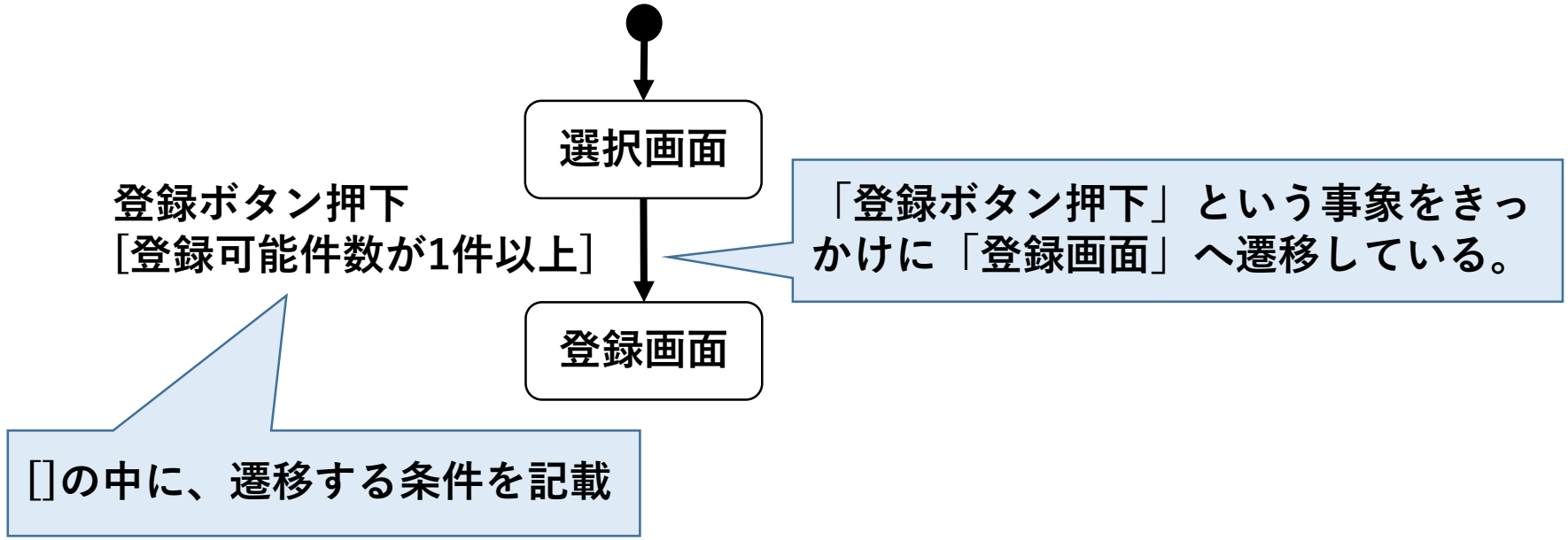
終了状態:遷移の終了を「終了状態」といい、内側が黒く塗りつぶされた二重丸の要素で表す。



2)ステートマシン図

外部遷移:別の状態に遷移することを「外部遷移」という。以下は、「選択画面」で「登録ボタン押下」があったときに「登録可能件数が1件以上」ならば「登録画面」に変わることを表現している。

#細かい遷移条件は内部設計となることも。



課題4-1外部設計の作成(締切:5/25)

前回作成した「航空会社の顧客管理システムのユースケース」の業務フロー(DFD)と画面遷移図を作成してください。複数のユースケースを作成した場合、必ずしもすべてのユースケースについて、業務フロー(DFD)と画面作成図を作成しなくて良いです。

まずは、個人で作成し、その後に隣の人と作成したユースケース図を見せあってください。

提出方法:PDF

課題4-2外部設計の作成(締切:5/25)

「課題2-2レポートで作成した内容」について業務フロー(DFD)と画面遷移図を作成してください。#記載内容が複雑な場合、おおまかな詳細度で良いです。

まずは、個人で作成し、その後に隣の人と作成したユースケース図を見せあってください。

提出方法:PDF

今日の内容

1)ソフトウェア設計とは

　#要求仕様をコーディング出来る仕様に変換する

2)外部設計

3)内部設計

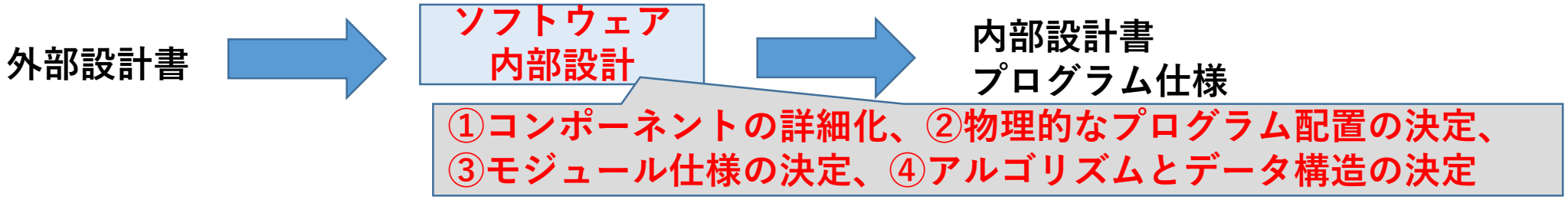
3)内部設計

外部設計書を基に、システムの具体的な実現方法を定義する工程

→具体的には、データ構造やモジュール分割(構造化設計)を定義

#プログラムをどうかくか、を内部設計で定める。

外部設計で定められた外部設計書を詳細化して、プログラム仕様を作成する。各コンポーネントをプログラムモジュールレベルの記述まで詳細化する。プログラム言語の特徴も考慮し、アルゴリズムとデータ構造も決定する。



3)内部設計の作成例

内部設計書も外部設計書と同様に、まず個別の設計を行った後に、それらをまとめるという手順で作成するのが一般的である。構築システム毎に、具体的な手順は異なる。

a:画面・帳票の詳細設計

b:外部インタフェースの詳細設計

c:処理ロジックの詳細設計

d:メッセージの詳細設計

3)内部設計の作成例

a:画面・帳票の詳細設計)外部設計書に記載した画面・帳票レイアウトや画面遷移図を基に、必要な項目や説明を追加して、プログラムが製造できるレベルまで詳細化する。#細かい遷移条件など

b:外部インタフェースの詳細設計)外部とのインタフェースについては、基本となる情報はすべて外部設計書に記述してあるはずだが、データの送信者と受信者、1回に送信する最大件数、インタフェースを使用する周期など、プログラムを製造できるレベルまで必要な情報を追加して記述する。

c:処理ロジックの詳細設計)機能仕様のロジックをプログラミング可能なレベルまで詳細化する。詳細化の粒度は、プログラミングする人のレベルにより異なる。

d:メッセージの詳細設計)メッセージの種類をエラー、情報提供、ログなどに分類する。画面に表示する項目として、以下などを定義する。

・標準的なメッセージのダイアログサイズと構成

・メッセージ番号(例:404NotFound,403Forbidden)

3)処理ロジックの詳細設計(モジュール分割)

設計戦略(抽象化、分割、独立性)をプログラム設計に関して具体化したもの。設計における複雑さを軽減するだけでなく、検証段階における部分的なテスト・デバッグが可能となるため、段階的かつ効率的に品質管理可能となる。

モジュール:基本的には、分離可能な最小のプログラム単位

プログラム記述を分割した開発単位

「各設計法の考え方(モジュール結合度など)」が、実業務で出てくることはあるが、そのまま適用されることはあまりないかも

- **複合設計法:**モジュール分割をデータの変換過程に着目して行う手法。
- **データ構造分割法:**入力データ構造と出力データ構造に着目して処理の手順を検討する方法。

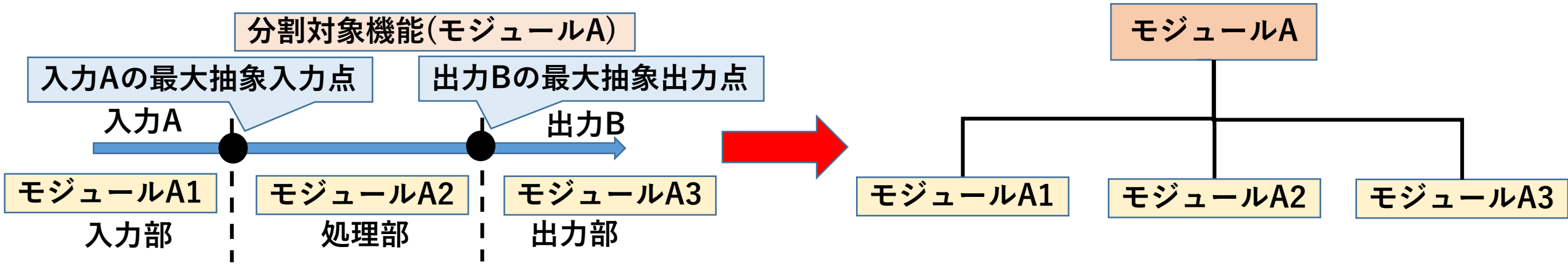
3)複合設計法(STS分割:Source Transform Sink)

STS分割:機能をデータの変換過程と捉え、1つの機能を入力、処理、出力の3つの機能に分割する方法。

手順1)対象とする機能への入力と出力を明確にする。

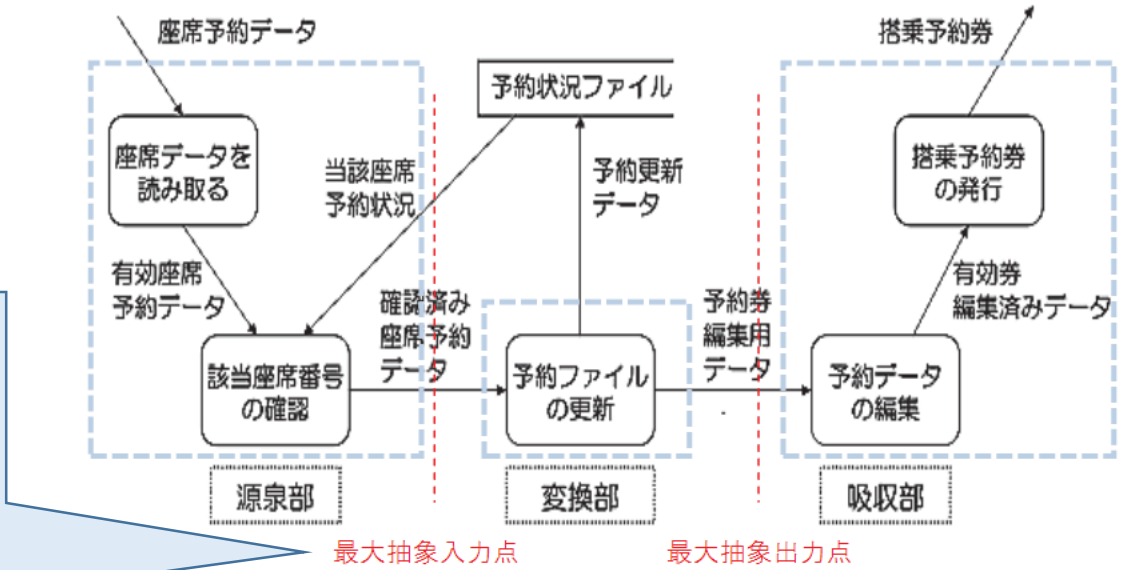
手順2)主なデータの流れに着目し、入力データの抽象化が最大になる点(最大抽象入力点)と出力データとしての抽象化が最大になる点(最大抽象出力点)を決める。

入力から最大抽象入力点までを入力部、最大抽象入力点から最大抽象出力点までを処理部、最大抽象出力点から出力までを出力部として、3つのモジュール構造が得られる。

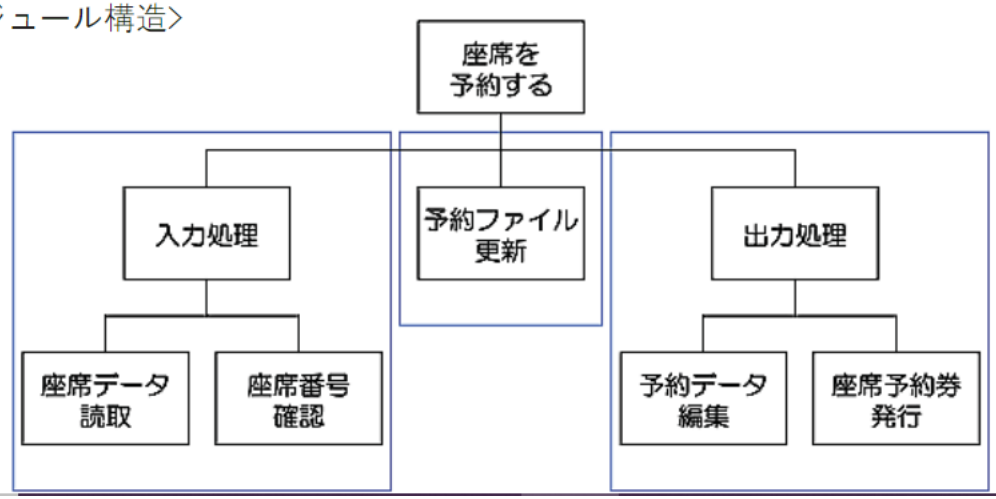


STSによる搭乗座席予約の例

「確認済み座席予約データ」は元の「座席予約データ」を予約状況ファイルを更新可能なように変換したものである。そのため、元データとデータ構造が異なる可能性が高く、抽象度が高いと見込まれる



<モジュール構造>

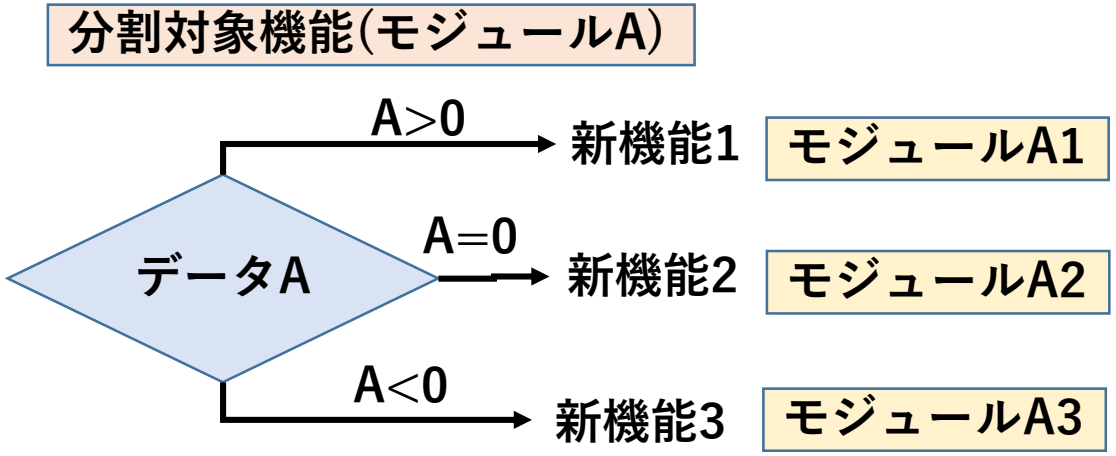


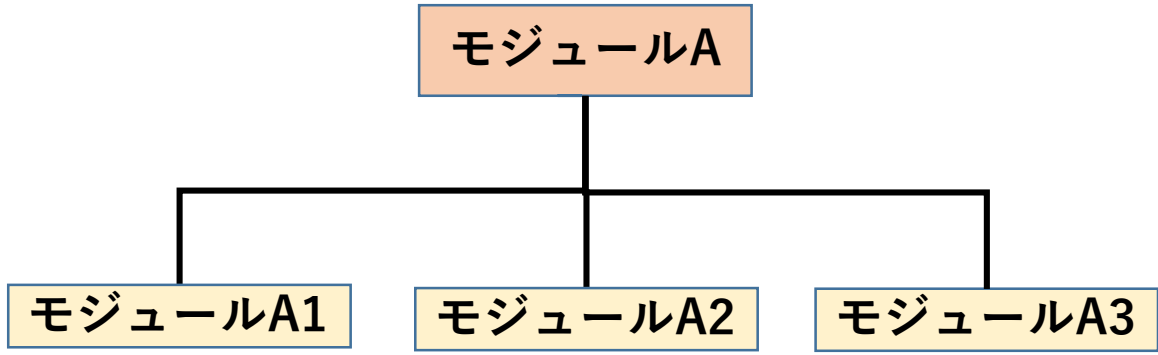
https://www.ipa.go.jp/files/000056493.pdf

3)複合設計法(TR分割:TRansaction)

TR分割:入力データに応じて処理が変わる場合に用いるモジュール分割方法。例えば、入力データAの値によって、処理内容が変わる場合などが該当する。各処理内容を「分割されたモジュール」として、データによって処理を分けるモジュールを「親モジュール」とする。

#STS分割技法と組合せて用い、主にSTS分割技法でうまく分割できない場合に適用する。



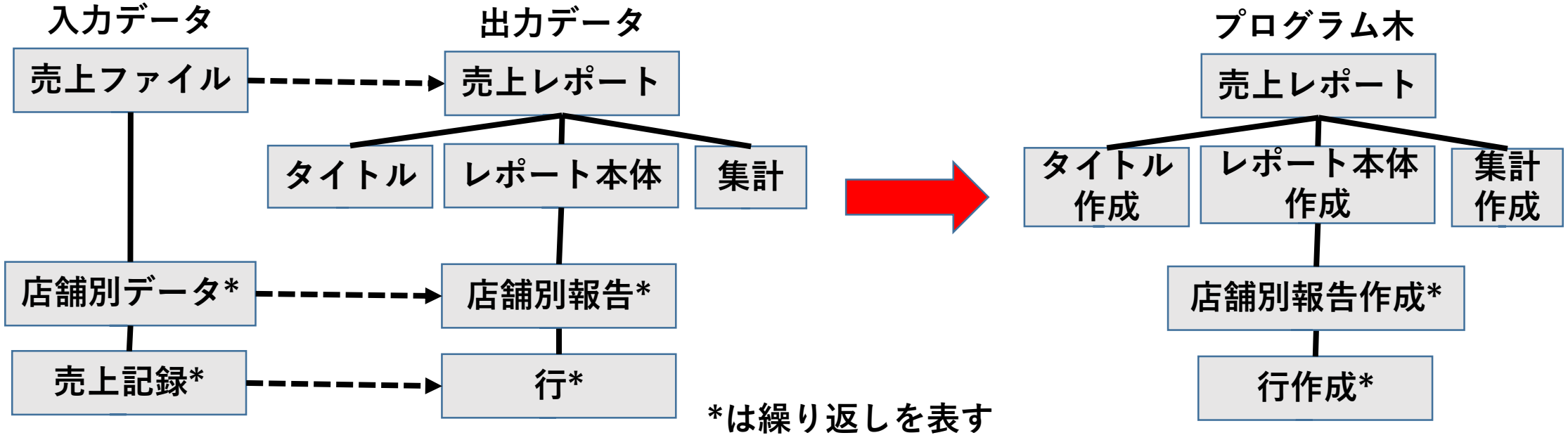


3)データ構造分割法

- データ構造分割法:入力データ構造と出力データ構造に着目して処理の手順を検討する方法。入出力データの構造からプログラム構造を決める手法として、ジャクソン法がある。

手順1)入力データおよび出力データを、基本、繰返し、選択(分岐)、(基本の逐次実行である)連接の4つの構造要素を用いて表す。

手順2)入出力データの対応関係を取り、入力データをマージしてプログラム木を作る。完成したプログラム木がプログラムのモジュール構造となる。



3)データ構造分割法の例

[支払い記録の集計]

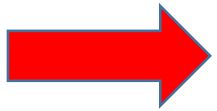
顧客の支払い状況の取りまとめを、支払い記録から集計するプログラムを作成する。

(a)支払いファイルPFILEは、顧客番号、日付、支払額のレコードから構成される。PFILEは顧客番号ごとに昇順で並んでいるとする。

(b)レポートファイルEPFILEを作成する。レポートは、1)出力形式に整形された日付、支払額、2)顧客番号、各顧客の総支払額、3)全ファイルの総額を表す。

(a)支払いファイルPFILE

顧客番号	日付	支払額
0001	2022.5.12	30000
0003	2022.5.22	13000
0003	2022.8.12	23000



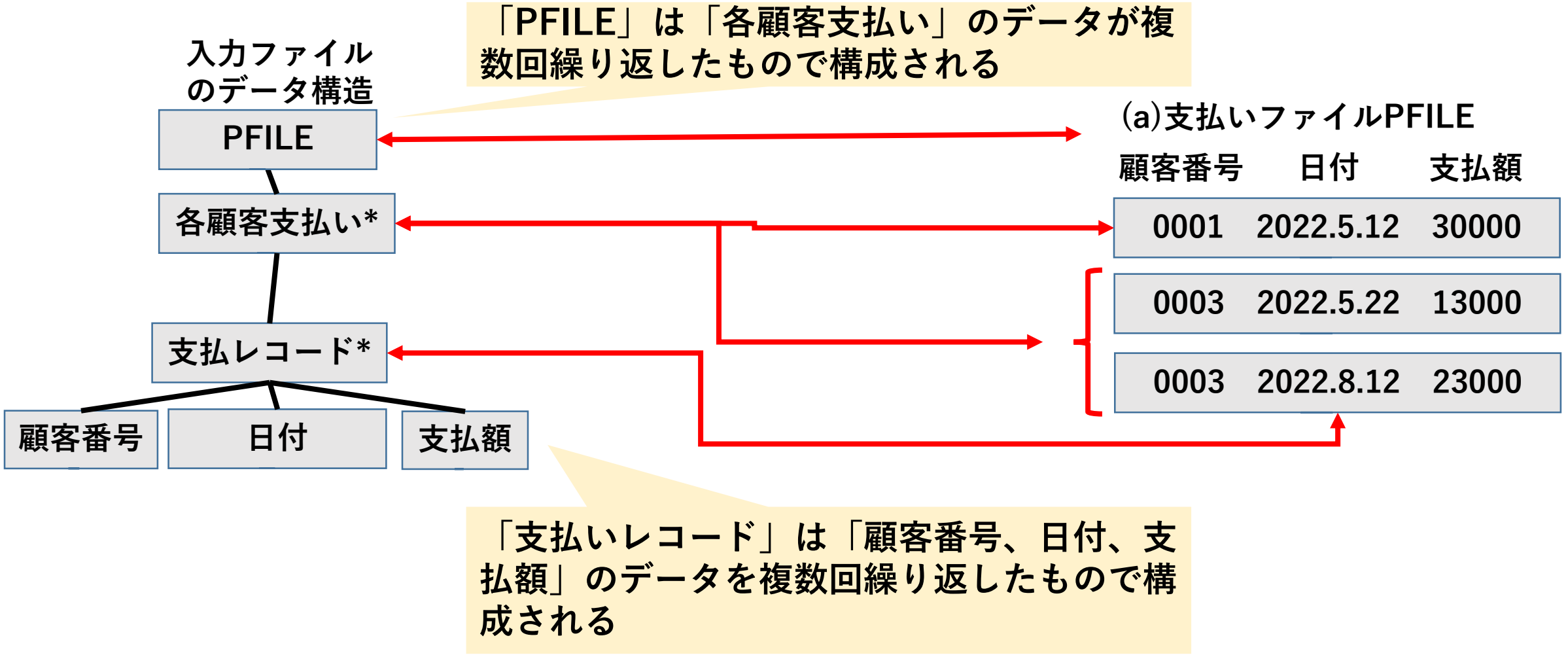
(b)レポートファイルEPFILE

```
May. 12, 2022    ¥30000
        0001 total ¥30000
May. 22, 2022 ¥13000
Aug. 12, 2022 ¥23000
        0003 total ¥36000

                Full total ¥66000
```

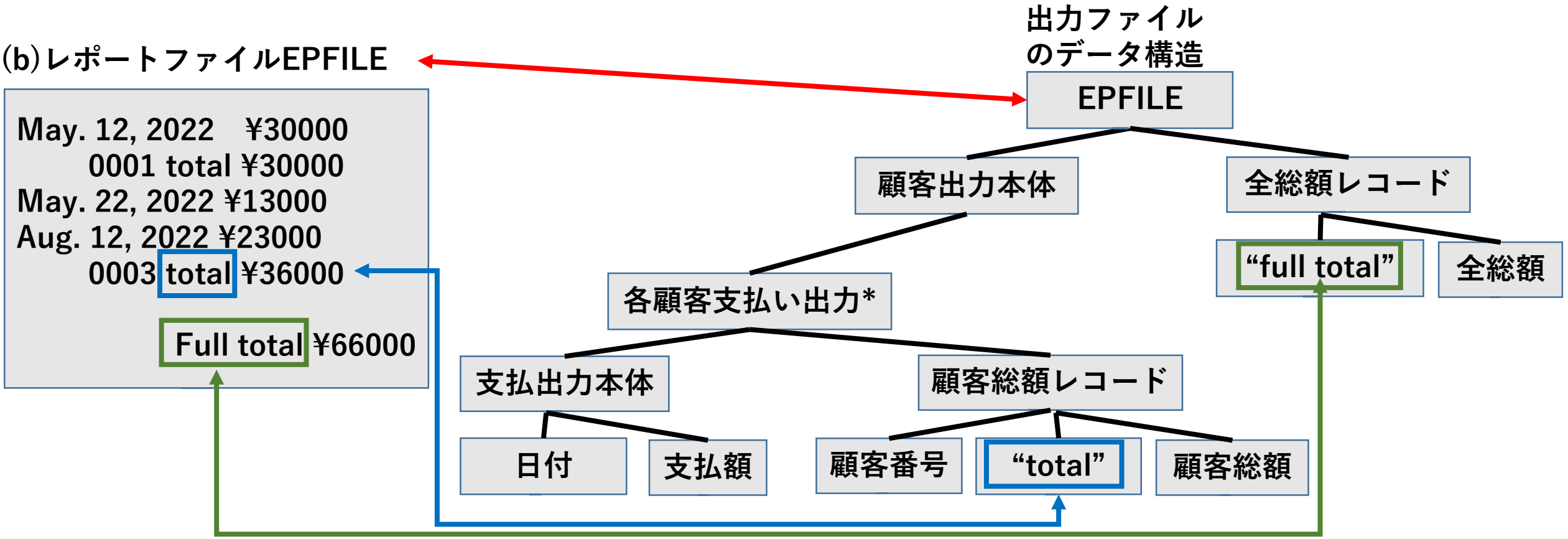
3)データ構造分割法

プログラムは、入力ファイルを出力ファイルに変換するものである。支払記録の集計の「入力ファイル」と「出力ファイル」に着目する。



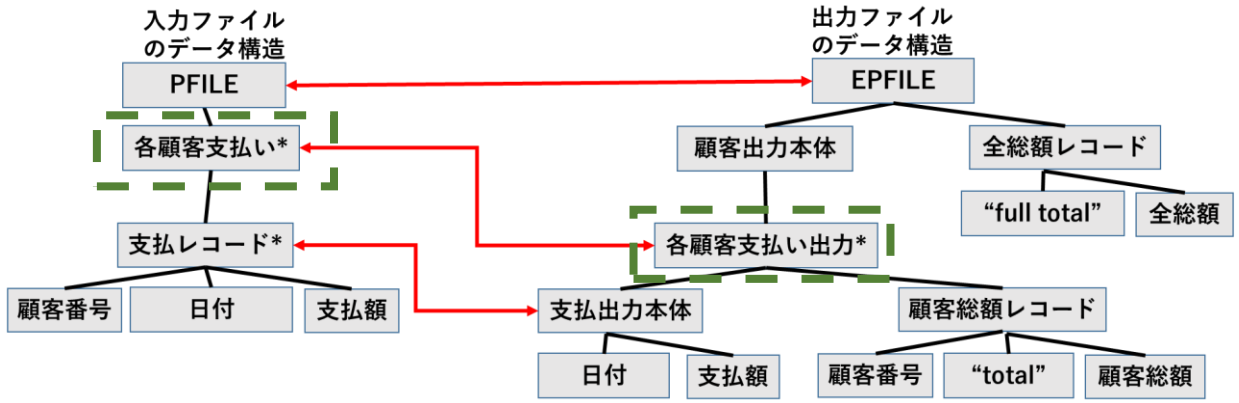
3)データ構造分割法

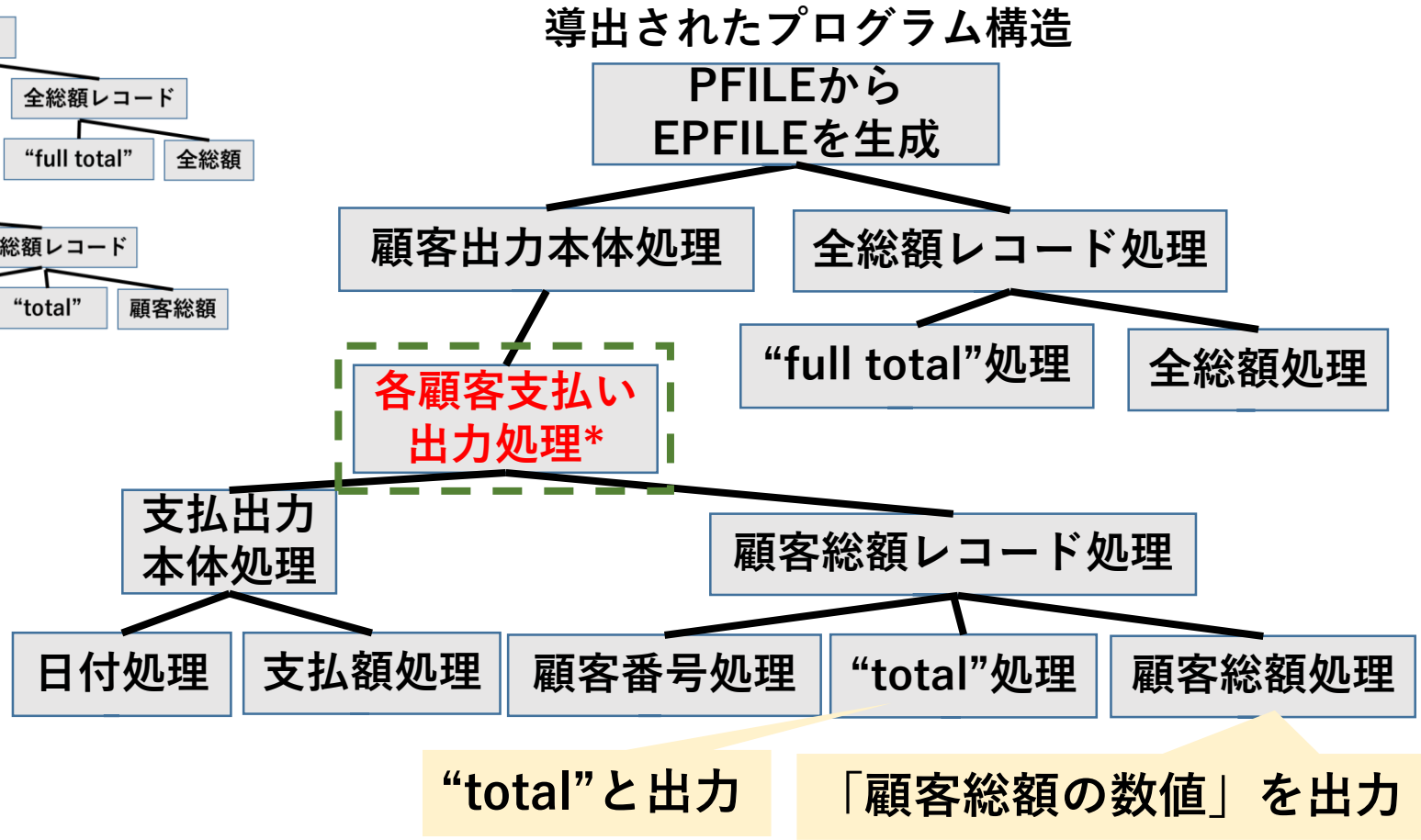
プログラムは、入力ファイルを出力ファイルに変換するものである。支払記録の集計の「入力ファイル」と「出力ファイル」に着目する。



3)データ構造分割法

入力データと出力データを統合したものが、プログラム構造となる。例えば「各顧客支払い」を入力として「各顧客支払出力」を出力する処理となる。





課題4-3外部設計の作成(締切:5/25)

「支払い記録の集計」のプログラム構造を用いた場合の、プログラム例をC言語で記載せよ。なお、各モジュール(例:顧客出力本体処理)を関数名とし、関数の具体的な処理の内容は記載しなくて良い。ただ、モジュール内にモジュールがある場合は、そのモジュールを呼び出す粒度は記載する事。

提出:テキスト(PDF,ワードファイルなど)で提出ください。

3)内部設計の例

SQL文を実行するための主な機能として下記がある。

(1)SQL文解析:受信SQL文を解析し、使われているSQL句を明らかにする。

(2)SQL文最適化:解析結果から、処理量減のため受信SQL文の変更やデータアクセスに索引を使うかの判断などを行い、クエリ実行プランを作成する。

(3)クエリ実行エンジン:上記で作成されたクエリを実行する。

```
select *
from tableA,tableB,tableC
where tableA.a1=tableB.b1
 and tableB.b2=tableC.c2
```

データベース

SQLの解析

SQLの最適化

クエリ実行
エンジン

抽出カラム: すべて
操作対象表:tableA,tableB,tableC
結合条件: tableA.a1=tableB.b1
tableB.b2=tableC.c2

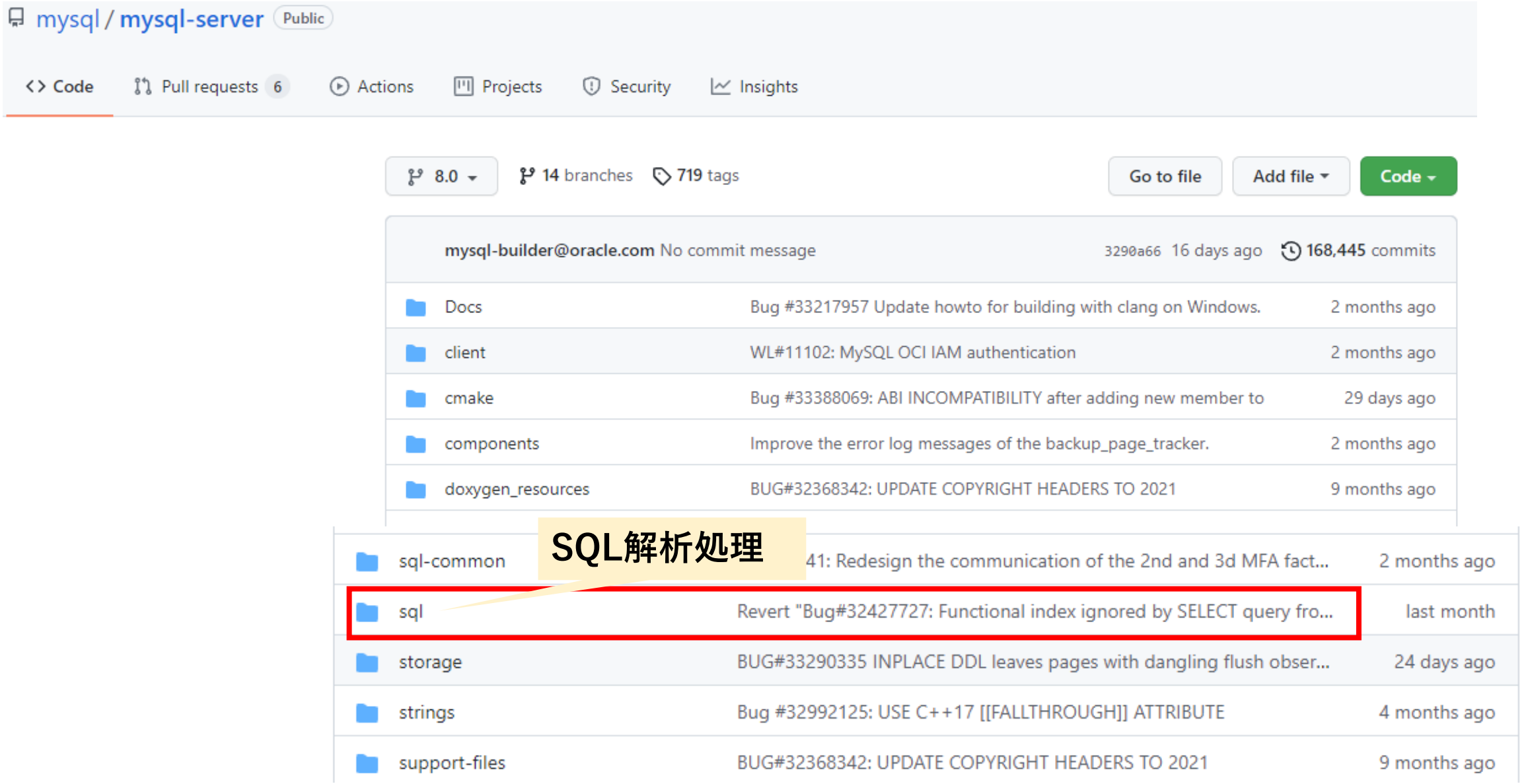
処理コスト10
tableA tableB tableC

処理コスト15
tableA tableB tableC

処理コスト5
索引使用
tableA tableB tableC

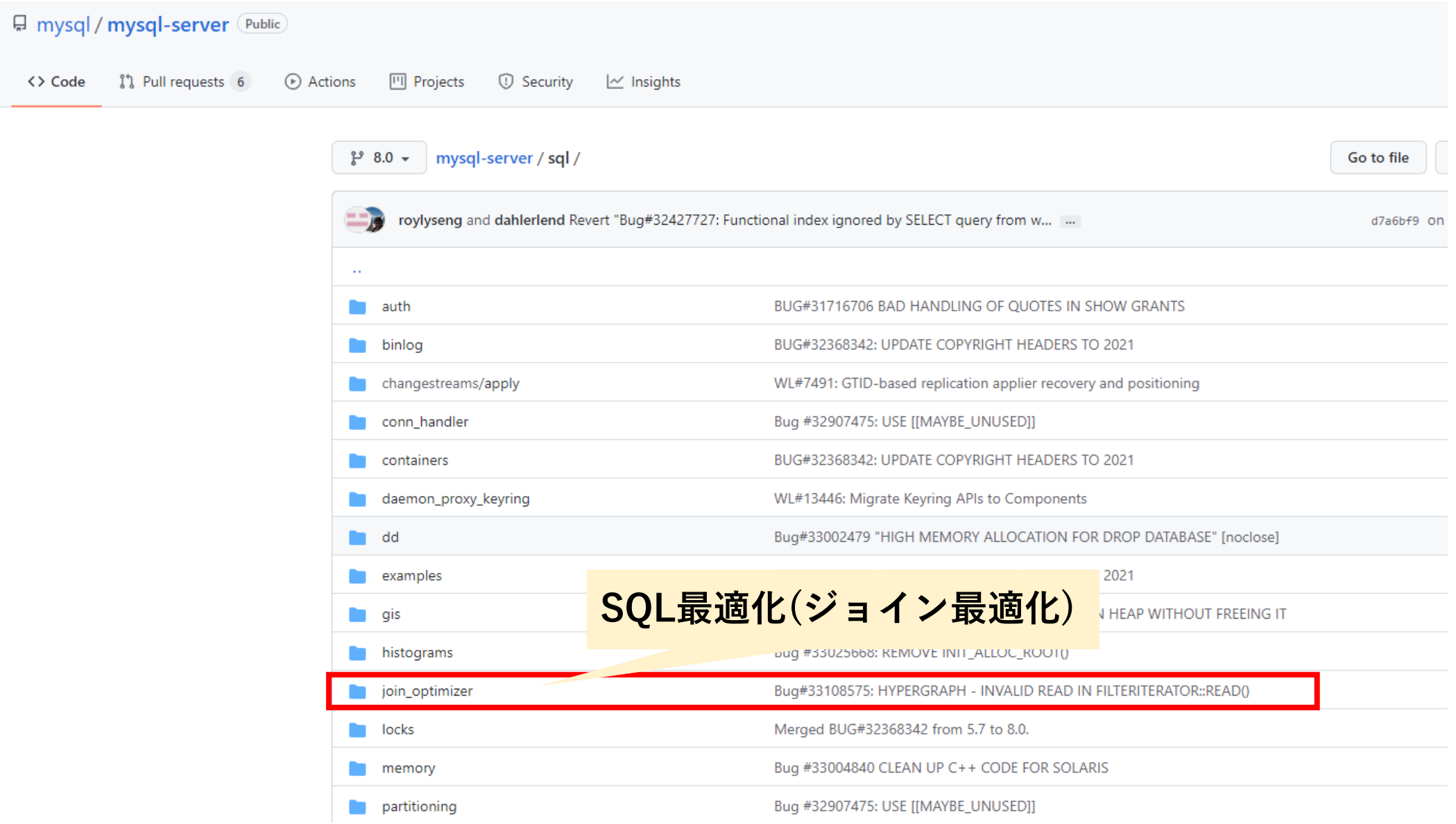
・・・

3)内部設計の例



https://github.com/mysql/mysql-server

3)内部設計の例



https://github.com/mysql/mysql-server/tree/8.0/sql

3)モジュール分割の評価

モジュール分割の良否は、設計の良否に直接影響するため非常に重要である。関連項目として、下記がある。

1)モジュールの大きさ:人間の認知範囲を反映させることができる程度の大きさにするとよい。例えば、プログラム100行以内、ディスプレイ2画面程度で見ることができる程度など。

2)情報隠蔽の度合:モジュールの使用とは関係ないモジュール内部の設計事項を隠すこと。モジュール使用者は、不要な情報を理解する必要がなくなり、修正時も他モジュールへの影響度が小さくなる。

3)モジュール間結合度:2つのモジュール間の結合の強さをモジュール間結合度といい、低いほど良い設計といわれる。

4)モジュール強度:モジュール内の命令群が深く関わりあっているかの強さ。

3)モジュール間結合度1

モジュール間結合度は、2つのモジュールの間の結合が密である程度を示す。モジュール間結合度の定義として下記がある。結合度は低い方が望ましい。

a)内容結合:あるモジュールが、他のモジュール内の構成要素を直接参照したり、変更する結合をいう。例えば、GOTO文で直接他のモジュールに移動する場合である。

b)共通結合:複数のモジュール間で、ある領域(共通/外部変数)を参照している結合をいう。1つのモジュールがデータ領域を変更すると他のモジュールに影響を与える。

c)外部結合:あるモジュール内で外部参照可能であると宣言されたデータ領域を、他のモジュールが直接参照する結合をいう。

3)モジュール間結合度2

モジュール間結合度は、2つのモジュールの間の結合が密である程度を示す。モジュール間結合度の定義として下記がある。結合度は低い方が望ましい。

d)制御結合:あるモジュールが他のモジュールを呼び出す時、制御のためのフラグやパラメータを渡す。呼び出すモジュールは、呼び出されるモジュール内の制御を理解している。

e)スタンプ結合:共有データ領域にはない同じ構造のデータ(構造体)を受け渡す結合をいう。構造内のデータ順序とデータ型をモジュール間で知っている必要がある。一方の構造体の仕様を変更すると、他方も修正が必要になる。

f)データ結合:データの必要な部分だけを、呼出しモジュールへの引数として渡す結合をいう。呼び出されるモジュールと呼び出すモジュールのデータ構造に関する特別な知識は必要ない。

3)モジュール強度1

- 1つのモジュール内の機能間の関連性をモジュール強度という。モジュール強度が弱いときは、さらなるモジュール分割を吟味する必要がある。

a)暗号的強度:モジュール間で機能の関連がない場合。お互いに全く関係なく複数の機能を実行している。

b)論理的強度:関連する複数の機能を持つモジュールで、制御変数により機能を選択する。

c)時間的強度:機能要素間の関連は薄いが、一群の機能要素がある時間的順序関係に沿って起動される。

3)モジュール強度2

d)手順的強度:モジュールを構成する各機能の起動には順序関係があり、それに従いまとめられている。

e)連絡的強度:各機能の起動に順序があり、かつ各機能が共通のデータを参照または変更する。

f)機能的強度:単一機能として考えることができる機能群からなるモジュールで1つのモジュールで機能を1つ実現している場合である。

g)情報的強度:同じ内部データを扱う機能を集め、かつ1つのモジュールの機能は単一の場合。オブジェクト指向のカプセル化の概念に近い。