Vol. 35 No. 5　　情　報　処　理　　May 1994

解　説

オブジェクト指向分析・設計

4. オブジェクト指向分析・設計と従来の方法論との比較

——仕様化プロセス，モデリング，教育の観点から†——

加　藤　潤　三††

1. はじめに

従来の方法とオブジェクト指向分析，設計方法を仕様化プロセス，モデリング，教育という側面から比較した．ソフトウェア開発の現場における課題の一つは，開発者によって生産性と品質に非常に大きな差が生じ，開発すべき対象が複雑な場合はその差がより顕著になることである．これは，ソフトウェア開発に必要とされる体系化された知識と訓練が不足しており，技術者の育成が徒弟制の下で行われていることに関連する．

ソフトウェア工学は，ソフトウェアの開発に必要な体系化された知識と訓練を与え，複数のチームが共同してソフトウェアを開発するための基盤を与えることを役割としている．その中で，要求分析から実装までに人が行う種々の決定を取り上げ，それらの順序を示し，そこでの原理や技法とツール，表記法および決定の妥当性の基準を体系的に示したものがソフトウェア開発方法である．

オブジェクト指向分析・設計の方法には従来の方法の延長線上にあるものとそうでないものがある[14]．前者のほうが，方法としての特性をそなえている．ここで取り上げるオブジェクト指向分析・設計の方法は，方法の比較として適している前者に属しており知名度も高い．

また，従来の方法は，たとえば強い時間の制約が課せられる実時間システムのようなソフトウェア開発などのさまざまな課題を解決していく過程で変化してきている．オブジェクト指向分析・設計の方法の今後も同様な過程を踏むと予想できる．

従来の方法とオブジェクト指向分析・設計の方法がまったく異なるという議論の根拠に「従来の方法は機能的な言葉で考えている」ことがよくあげられる．機能をいつ仕様に組み入れ，何を使いどのように記述するかを比較すればこの議論が妥当であるかどうかが理解できる．特に，両者の何が同じで何が異なるかは，仕様化プロセスの比較でかなりの部分が明らかになる．この比較によって従来の方法を使ってきた技術者がオブジェクト指向分析・設計法も身に付けることができるかどうかも明らかになる．

比較した方法に共通な点は，複雑さの管理に抽象化を使うことである．ここでは，何をどの視点で抽象化するのかを比較してみた．

- 従来の方法として，

Realtime Structured Analysis (RSA)[1]
Jackson System Development (JSD)[2)~4)]

- オブジェクト指向分析，設計法として，

Object-Oriented Analysis, Object-Oriented Design (OOA/OOD)[5),6)]
Object-oriented Modeling Technique (OMT)[7]

を取り上げる．

次の項目の比較検討は記載していない．

- 大きな課題に対する課題の分割方針
- プロジェクト管理の枠組み
- 表記法

本稿の構成は，各方法の概要の説明，それから比較議論のようになっている．

誤解を避けるため，分析と設計をここでは次のように定義する．

- 課題領域を関心のある視点で抽象化しモデルとして記述することを分析とする．
- 分析の結果に対して実環境に関するなんらかの決定を入れたものを設計とする．
- 実環境を考えに入れたアーキテクチャを決めることは設計に入る．

† Comparison of Specification Process and Education Between Object Oriented Analysis/Design Methods and Conventional Methods by Junzo KATO (EDS Japan Technical Consulting Services.)

†† エレクトロニック・データ・システムズ(株)　テクニカル・コンサルティング部

2. 従来の方法

2.1 R S A

構造化分析[10]は，機能モデルをデータフロー図(DFD)で記述し，構造化設計[13]につないでいる．RSA は機能中心から離れ，課題領域を静的，動的，機能のそれぞれの側面からモデル化する．課題をモデル化することが分析であるという構造化分析の原理は，RSA の分析の原理と同じである．

開発者は，システム環境のモデルを使い，機能モデルを先に記述するプロセス指向か，データモデルを先に記述するデータ指向かのアプローチを決める．データ指向アプローチを格納データ主導 (stored data driven) といい，プロセス指向アプローチを変換主導 (transformation driven) という．格納データ主導アプローチで記述するデータモデルは，実体 (Entities) とそれらの間の関連 (Relationship) から構成される実体関連図 (Entity-Relationship Diagram) で表現される．

RSA は，実体関連図で静的側面を，状態遷移図で動的側面を，それからデータフロー図と制御フロー図を併合した変換構造図 (transformation schema) で機能的側面を抽象化し，課題世界のモデルを記述する．これをモデル化という．

2.1.1 仕様化プロセス

仕様化は以下に説明するエッセンシャルモデル (essential model) 作成，実現モデル (implementation model) 作成の順で進められる．エッセンシャルモデルは**表-1** で示されている成果物を記述し，**表-2** の手順で記述される実現モデルはエッセンシャルモデルを実行環境に適したモデルへ変換したモデルである．

(1) エッセンシャルモデルの作成プロセス

システムの外からシステムと会話する環境と，そのインタフェースをシステムコンテキスト図で定義する．さらに，システムの外部で発生し，システム内部の状態やシステムからの応答を引き出す出来事を外部事象として識別し事象一覧表に記述する．システムの範囲 (scope) はこの両者で決まる．これが環境モデルである．

環境モデルと利用者の要求から，振舞いモデルを導き出す．システム内部の状態を変化させ，システム内部の状態から応答を作るシステムであれば，データ指向アプローチを選択する．

表-1 エッセンシャルモデルの記述順序

1. 環境モデル (environment model) の記述
   - 成果物：システムコンテキスト図
     - 事象一覧表
2. 振舞いモデル (behavioral model) の導出
   - 成果物：振舞いモデル
     - 事象一覧表
     - 実体関連図 (データ構造図)
     - 状態遷移図
     - 変換構造図
       - データフロー図 (Data Flow Diagram)
       - 制御フロー図 (Control Flow Diagram)
     - データ辞書
     - 変換仕様書 (ミニスペック)
3. 上位レベルのモデル化
   - 成果物：上位レベルの変換構造図
4. 下位レベルのモデル化
   - 成果物：下位レベルの変換構造図

データ指向アプローチでは，データ構造図の作成の後，状態遷移図で個別の実体の動的な側面を表現する．環境からのデータや制御メッセージを授受するプロセスのネットワークが変換構造図である．まず，実体への作用を表現するレベルで記述する．それから変換構造図のプロセスをグループ化したものを上位の変換構造図とし，上位の階層を作る．最上位はシステムコンテキスト図である．最後に，変換プロセスを JSP (Jackson Structured Programming)[11] の基本法で解ける程度まで下位レベルに展開し，変換仕様書（ミニスペック）としてこの最下位のプロセスの仕様を記述する．このように，構造化分析と異なるもう一つの点は，トップダウンまたはボトムアップのいずれのアプローチでもないことである．

(2) 実現モデルの作成プロセス

実装するハードウェア，オペレーティングシステムなどの基盤になるソフトウェアが与えられたとき，性能要求を満たすようにエッセンシャルモデルを分割しそれぞれをプロセッサに割り当てる．次に，プロセッサ別に割り付けられた変換構造図をタスク（バッチ，オンライン，対話，インタラプトハンドラ）に分割する．それから，エッセンシャルモデルと外部のインタフェースを決定し，モデルとして追加する．後は，タスク管理と，データベースを設計し，最後に変換構造図を構造図へ変換しタスクの構成モジュール階層と，モジュールの仕様を決定する．構造化設計のモジュールの独立性の尺度として使われる結合度 (coupling)，モジュールの内部のまとまりの尺度として使われる凝集度 (cohesion) を使い，モジュール化の妥当性を検査する．

表-2 実現モデルの記述手順

1. プロセッサモデルの構築
2. タスクモデルの構築
3. インタフェースモデルの構築
4. システムサービスモデルの構築
5. モジュール化

表-3 JSD の開発手順

1. アーキテクチャ段階（モデル化フェーズ）
1.1 実世界の抽象化
1.2 事象の抽出
1.3 実体の構造定義
1.4 モデル・プロセス定義
1.5 実体の属性定義
1.6 実体のオペレーション列挙
1.7 実体の内部構造定義
1.8 データモデルへの変換
1.9 性能要求の評価
1.10 システムの物理アーキテクチャの概要
・プラットフォーム
・既存のシステムとのインタフェース
1.11 サブシステム分割
2. ネットワーク仕様化段階（ネットワークフェーズ）
2.1 対話機能の追加
2.2 情報機能の追加
2.3 入力サブシステムの設計
外部インタフェース：対話，エラー処理
3. 物理設計段階（実現フェーズ）
実行環境を考慮した設計
3.1 データ設計
3.2 プロセッサへのプロセスの割当
3.3 運用などの実装に必要となる処理の定義
3.4 性能評価
4. 構築段階（実現フェーズ：構築とテスト）
4.1 プログラムの完成
4.2 コード化とテスト
5. 導入段階
完成したシステムの実装
ユーザの受け入れテスト
（略）

2.1.2 教育のための配慮点

構造化分析／構造化設計，データモデル化（実体関連図），状態遷移図，JSP の基本的な知識があれば理解しやすい．教育を実践する場合，実体と関連，属性の区別が明瞭でないことや分析の精度と課題の理解度によって変わることを認識しておく必要がある．状態遷移図は，変換構造図の作成段階でオブジェクトの動的側面の記述から制御の記述に視点が変わることにも触れると良い．

2.2 J S D

JSP は，正規表現の構文解析アルゴリズムを応用したプログラム構造の設計技法であり，JSP 基本法と JSP 基本法で解けるようにプロセスを分割するプロセスネットワークの設計法からなる．

一方，JSD は JSP の概念を拡張し，並行処理やクラスとインスタンス，抽象データ型といった概念を取り入れ，オブジェクトの概念を直接使わないにもかかわらずオブジェクト指向分析・設計に近いソフトウェア開発方法である．その仕様はネットワーク，プロセス，抽象データ型の3階層 (layer) から構成される．

2.2.1 仕様化プロセス

JSD では，ネットワーク仕様化までを仕様化といい物理設計以降を実現という．開発プロセス名が改版のつど変わっている[9]．最近の LBMS-JSD[4] に基づいて説明する（**表-3**）．仕様化は，プロジェクトの始まり (Project Initiation)，課題世界をモデル化するアーキテクチャ段階 (Architecture Stage)，機能を記述するネットワーク仕様化段階 (Network Specification Stage)，プログラム構造やレコードレイアウトを決める物理設計段階 (Physical Design Stage)，設計したものをコード化する構築段階 (Construction Stage)，構築したものをテストする導入段階 (Installation Stage) となる．

アーキテクチャ段階は，モデル化とアーキテクチャの概要決定に分かれる．モデル化は，さらに役割 (role) といわれる実体の構造の定義までと，モデルプロセスの定義の2段階に分かれる．前者は実世界の動的側面の抽象化であり，ソフトウェアシステムの範囲が決まり，後者はその具体化である．

モデルプロセスは互いに独立に並存し，プロセス間の同期は共通事象によって示す．同期処理など並行処理を明示的に記述するのはネットワーク仕様化段階である（**表-4**）．ここでは，実体の定義をモデルプロセスに，属性は変数に，事象はインタフェースに読み変える．JSD には，モデルプロセスから実体関連図への変換法があり，静的モデルも導出できる．

並列に動作するプロセスのネットワークを完成させるのがネットワーク仕様化である．ここでは，機能や外部とのインタフェース，時間的制約を仕様化する．通信プリミティブには非同期通信としてデータストリーム結合，状態ベクトル結合用のプリミティブがあり，同期通信として制御付きのデータストリーム結合用のプリミティブがある．

JSD では論理設計過程がない．ネットワーク仕

表-4 ネットワーク仕様化の作業

- 機能別にネットワーク図を作成し，機能を仕様化する．
  必要に応じてプロセスを追加する．
  -システムの状態を変える事象を出力する対話機能
  -システムの外部に出力する情報機能
- ユーザとの会話を入力サブシステムとして仕様化
- エラー処理，再利用可能なプロセスの決定
- システムへの性能要求の仕様化

表-5 JSD の設計

1. 物理設計
   - データ設計
     -状態ベクトル分離，
     -レコード，データベース設計
   - プロセッサへのプロセスの割当
     -ネットワークの分割とプロセス分割
     -ネットワークの階層構造への変換
   - 運用などの実装で必要な処理の定義
     -セキュリティ
     -リカバリ
     -スケジューラ
   - 性能評価
2. プログラム設計
   - スケジューラなどの追加プログラムの完成
   - ネットワーク仕様のプロセスのコード化

様化段階（表-4）が機能のモデル化のほかに，それに対応する内容をもつ．明らかに設計として分けているのは物理設計（**表-5**）だけである．

2.2.2 教育のための配慮点

並行処理プログラミング，データモデル化（実体関連図），状態遷移図，JSP について基本的な知識があると理解が早い．教育の実践の場合，実体の抽出を先行すると不必要に多くの実体候補を抽出するため，事象の抽出を先行させると良い．なお，動的側面からのアプローチは，その考えが理解でき使いこなせるまでに時間がかかることも留意しておくと良い．

2.3 従来の方法の共通点

RSA, JSD に共通する特徴をまとめると次のようになる．

- 初めに課題領域のモデルを記述し仕様化する．
- 実体とその属性を使い機能を記述する．
- 実現とは仕様を変換し実装させることである．
- 実現の前にアーキテクチャを決定する．
- サブシステムの分割は事象を参考にする．
- システムへの実際の入力／出力は別サブシステムとして扱い，モデルでは仮想入力／出力を使う．

3. オブジェクト指向分析・設計法

3.1 OOA/OOD

OOA は情報モデル化とオブジェクト指向プログラミング言語から派生した分析法である[5]．OOD は，表-7 で示す4つのシステム構成要素を，実現で使うオブジェクト指向プログラミング言語やデータベース管理システム，実装環境での並行処理制御に適したように決定する方法である．OOA/OOD は**表-6** に示す5種類のレイヤと**表-7** に示す4種類の構成要素からなる．

OOD の成果物はシステムを5個のレイヤに分けて記述したレイヤモデルに対して実現に関する設計上の決定を追加したものである．なお，構造化設計におけるモジュール化の尺度を拡張し，結合度をメッセージ結合 (Message Connection)，汎化 (Generalization) の良好度の尺度として，凝集度をサービスやクラス，汎化-特殊化 (Generalization-Specialization) の良好度の尺度として用いる．

3.1.1 仕様化プロセス

（1） 分析プロセス

表-6 のレイヤの記述順に沿って説明する．

L1：オブジェクトを抽出する

課題領域を記述している文章や図式などから名詞を選び，それをオブジェクトの候補とする．OOA の与えているガイドラインに従ってオブジェクトの候補からオブジェクトを決定する．

L2：オブジェクト間の構造を抽出する

分類構造 (Classification Structure) と組立構造 (Assembly Structure) を構造という．L1 で選択したオブジェクト間の関係を，それぞれ一般化一特殊化 (Generalization-Specialization) と全体一部分 (Whole-Part) の関係を用いて記述したものである．

表-6 レイヤモデル (layers Model) の構成

L1：クラス＆オブジェクトレイヤ (Class & Object layer)
L2：構造レイヤ (Structure layer)
L3：サブジェクトレイヤ (Subject layer)
L4：属性レイヤ (Attribute layer)
L5：サービスレイヤ (Service layer)

表-7 OOD で作成する構成要素

PD：課題領域 (Problem Domain) の構成要素
HI：人との対話 (Human Interaction) の構成要素
TM：タスク管理 (Task Management) 構成要素
DM：データ管理 (Data Management) 構成要素

L3：サブジェクトを抽出する

OOA のモデルの概観を与えるため，オブジェクトをいくつかのグループにまとめる．このグループをサブジェクトという．通常，複数のサブジェクトからシステムは構成される．オブジェクトのグループであるサブジェクト間の結合はメッセージ通信を使い定義する．L2 のオブジェクト間の構造を参照し，サブジェクト間の相互依存と相互作用を最小にするようにグループ化する．

L4：属性を定義する

課題領域から各オブジェクトの属性になるデータ項目を抽出し，該当するオブジェクトに割り付ける．オブジェクトの具体例であるオブジェクトのインスタンス間の結合と，1対1や1対多，多対1，多対多といったインスタンス間の結合の多重性 (multiplicity)，結合の両端のインスタンスの存在が必須であるかどうかを定義する存在性の制約 (participation) を記述する．ここで，静的モデルが完成する．

L5：サービスを定義する

サービスとは，オブジェクトが受け付けるメッセージで起動され，外部から観察できる振舞いをいう．サービスは，事象によって値が変化する属性や応答の関係，オブジェクトの履歴，基本的なサービスとしての機能から構成される．ここで，動的モデルや機能モデルが完成する．

（2） 設計プロセス

OOD では，課題領域のオブジェクトとクラスを実装する言語に合わせて詳細化し，ユーザインタフェースやデータ管理，タスク管理を追加する．

PD：課題領域の設計

C++，Smalltalk のようなプログラミング言語に適合するように OOA のモデルを「PD：課題領域」の構成要素に変換する．

HI：人との対話の設計

GUI などの外部インタフェースを構成するクラスを設計する．

TM：タスク管理の設計

並行処理プロセスの制御と複数のプロセッサ間の制御部分を設計する．

DM：データ管理の設計

データ管理の設計とは，オブジェクトサーバを設計することである．フラットファイル，リレーショナルデータベース管理システム，オブジェクト指向データベース管理システムのようなデータ管理ソフトウェアの選択に合わせてデータ管理部分を設計する．

表-8 OMT のモデル

・オブジェクトモデル： オブジェクトとクラス，クラス間の構造
・動的モデル：状態遷移図
・機能モデル：データフロー図

3.1.2 教育のための配慮点

オブジェクト指向プログラミング言語，データモデル化（実体関連図），状態遷移図の基本的な知識があれば説明しやすい．導入の部分で教える余裕があると後が円滑に進む．教育の実践の場では，分析と設計で同じ表記法を使っているので一度表記法でつまずくと後まで分からないことに注意する．なお，結合度と凝集度は，構造化設計のモジュール化の例を引用しながら解説すると分かりやすい．

3.2 O M T

従来の，特に RSA や最近の構造化分析の方法にオブジェクト指向の概念を混合した方法である．OMT は課題世界を**表-8** で示す三つの視点でモデル化する．

3.2.1 仕様化プロセス

（1） プロジェクトの始まり

要求をまとめ，課題の範囲や必要な項目，アプリケーション上の分脈，仮定，性能要求を課題 (Problem Statement) として作成する．

（2） 分　析

ユーザへのインタビュー，課題領域の知識，実世界の経験からオブジェクトモデル，動的モデル，機能モデルを作成する(**表-9**)．オブジェクトモデルの作成は，「クラスの抽出，それからクラス間結合の記述，次にクラスへの属性の付与，この後もう一度継承関係を詳細化する」という第1段階と，「機能モデルの作成の後，オブジェクトモデルに操作の定義を追加する」という第2段階からなる．動的なモデルの作成では，オブジェクト別にその状態遷移を記述する．このとき，典型的な対話順序のシナリオを作成することもある．なお，必要によっては，シナリオ作成にユーザインタフェースのフォーマットをモックアップとして作ることもある．機能モデルの作成では，オブジェクトモデルと動的モデルを参照してデータフロー

図を使い機能を仕様化する.

（3） システム設計

課題と分析結果から，共通のアーキテクチャを参考にしてレイヤとパーティションからなるシステムアーキテクチャを設計する（**表-10**）.

このとき，次のような共通のアーキテクチャを与えている.

- バッチ変換
- 連続変換
- 対話インタフェース
- 動的シミュレーション
- リアルタイムシステム
- トランザクションマネージャ

（4） オブジェクト設計

システムアーキテクチャに合わせて性能要求を満たすようにモデルを変換し，最終的にはモジュールとしてパッケージ化する（**表-11**）. このとき，再利用可能なオブジェクトを継承を使い派生させる. 上位のオブジェクトとの差分を下位のオブジェクトで設計すれば良いという方法で再利用することが基本である. 内部データ構造，アルゴリズム，制御や関連の実装がオブジェクト設計で決まる.

3.2.2 教育のための配慮点

構造化分析と設計，データモデル化（実体関連図），状態遷移図を通常使っている人には，比較的分かりやすい方法である. さらに，オブジェクト指向プログラミング言語の基本的な知識があると継承やメッセージ通信の概念が分かりやすい. システム設計では，システム設計の経験がないと分かりづらい. 初心者のみを教えるときは，身近なシステム例を用意しておくと良い.

3.3 オブジェクト指向分析・設計法の特徴

OOA/OOD および OMT に共通している特徴をまとめると以下のようになる.

- オブジェクト指向プログラミング言語を実装言語としている. さらに，オブジェクト指向プログラミング言語のほかに，Cなどの手続き型言語も扱える.
- 実世界を静的，動的，変換の各モデルで記述する.
- 機能のモデル化は後半になる.
- オブジェクトに属性と操作がカプセル化されている.

表-9 OMT のモデル化の手順

1. オブジェクトモデルを構築する.
1.1 オブジェクトクラスを抽出する.
1.2 データ辞書の準備.
1.3 結合を抽出する.
1.4 属性の定義.
1.5 継承を詳細化.
1.6 アクセス経路をテストする.
1.7 反復してオブジェクトモデルを作る.
1.8 クラスをモジュールにグループ化する.
2. 動的モデルを構築する.
2.1 シナリオの準備.
2.2 オブジェクト間の事象を抽出する.
2.3 事象の流れ図を作成する.
2.4 状態遷移図を作る.
2.5 オブジェクト間の事象を照合し一貫性を検証する.
3. 機能モデルを構築する.
3.1 入力，出力の値を抽出する.
システムの境界を示し，コンテキスト図を作図する.
3.2 機能依存性を示すためデータフロー図を構築する.
自明の処理になるまでデータフロー図を詳細化する.
3.3 最下位のレベルのプロセスの仕様を記述する.
3.4 オブジェクト間の制約を抽出し記述する.
3.5 最適化の基準を仕様化する.
4. 動的モデルと機能モデルが完成後，操作を追加.
5. 分析を繰り返す.

表-10 OMT のシステム設計の手順

1. システムをサブシステムに分割する.
2. 課題に固有の並行処理を抽出する.
3. プロセッサとタスクにサブシステムを割り付ける.
4. データストアの管理のアプローチを選択する.
5. 大域的な資源のアクセスの扱い方を決める.
6. ソフトウェアの制御の実現の選択.
7. 境界条件の扱い方を決める.
8. トレードオフの優先度を与える.

表-11 OMT のオブジェクト設計の手順

1. クラスに三つのモデルを組み合わせる.
2. 操作の実現のため，アルゴリズムを設計.
3. データのアクセスパスの最適化.
4. 外部との対話との制御の実現.
5. 継承を増加させるためクラス構造を調節.
6. 関連（association）を設計.
7. オブジェクトの表現（representation）を決定.
8. クラスと関連をモジュールにパッケージ化.

- オブジェクト間の相互依存関係をオブジェクト間の結合（関連）でモデル化する.
- 継承（inheritance），集約（aggregation）を明示的に記述する.
- レイヤとパーティションという概念でアーキテクチャを抽象化する.
- 手順の各段階ごとにガイドラインが用意してあり，適用にあたってかなり役に立つ.
- 実装するデータ管理システムにリレーショナルデータベースを取り上げ，設計したモデルと

の対応関係を詳細に説明している．

4. 比較と議論

4.1 仕様化プロセス

仕様化プロセスの順序を**表-12**に示す．RSAに最も近いのがOMT，それからOOA/OODになっている．一般に，後で記述されるモデルは前に記述したモデルに制約される．たとえば，JSDは初めに動的側面に着目し，必要なら静的なモデルを作る．概してJSDの静的なモデルは他の方法よりも実体の数は少ない．JSDの仕様そのものが並行処理の枠組みの上で記述されることによる．これは，JSDが同期問題をもつ複数プロセッサのシステムやリアルタイムシステムに適している[5]ことと関連する．

4.2 仕様記述の視点

Winograd[12]は記述すべき領域を課題領域 (subject domain)，対話領域 (domain of interaction)，実現領域 (domain of implementation) に分けることを提案した．すなわち，実世界がさまざまな側面をもっており，抽象化できる側面をいくつかの視点から分離させることによって，課題領域の複雑さを管理するという提案である．ソフトウェア開発は，必要な側面を切り出して個別に記述し，最終的にはこれらの側面を順序立てて合成するという手順を踏む．このような側面の分離とそれらの合成の仕方と順序，および記述物の妥当性の検査基準がソフトウェア開発方法の要である（**表-13**）．OMTではオブジェクト設計で合成するほかは，他の方法はすべて実現で合成する．ユーザインタフェースとはユーザと計算機との対話の記述である．比較したすべての方法が対応する視点をもっていることが分かる．

表-12 仕様化プロセスの比較

	RSA	JSD	OOA	OMT
範囲 (scope) の決定	1	1	1	1,3
オブジェクトの定義	2	2	1	1
オブジェクト間の関連	3	6	2,4	2
オブジェクト間の属性	3	4	4	2
オブジェクトの状態遷移	4	3	5	4
オブジェクトへの操作	5	5	5	5
グループ化	6	7	3	3
ユーザインタフェース	D	8	D	4

Dは設計プロセスに該当する項目である．

表-13 視点の比較

	RSA	JSD	OOA	OMT
課題領域	状態遷移図 実体関連図	モデルプロセス 実体関連図	クラスと オブジェクト	オブジェクトモデル 動的モデル
機能	データフロー図 制御フロー図	ネットワーク 機能プロセス	サービス	機能モデル
ユーザ	インタフェース モデル	入力サブシステム	ヒューマン インタフェース	対話インタフェース

表-14 分析におけるモデルの種類と構成要素

方法	構成要素	モデルの種類
JSD	プロセス ネットワーク 抽象データ型	モデルプロセス 実体関連図（モデルプロセスから派生）
RSA	実体関連図，データ辞書 状態遷移図 データフロー図＋制御フロー図 ミニスペック	エッセンシャルモデル
OOA	クラスとオブジェクト サービス記述	レイヤモデル L1, 2, 3, 4, 5
OMT	オブジェクト 状態遷移図 データフロー図 制御フロー図 ミニスペック	オブジェクトモデル 動的モデル 機能モデル

4.3 モデルの種類とその構成要素

モデル化の作業中は，モデル間の整合性を常に考えなければならない．システム開発では要求が開発途中で変化することが多い．表記法の一貫性とそれが統合されている必要がある．表記法の種類や構成要素の種類が多いと変更の管理が煩雑になりがちである．逆に，表記法の種類が多く構成要素も多いと何を記述するのかが比較的容易に分かるので利点になるときもある．したがって，応用するときは，記述すべき単位と構成要素のトレードオフを考える．OMT，RSAは多くの構成要素と表記法をもち，JSD, OOA/OODは構成要素の種類は少なく，表記法も簡単である（**表-14**）．

4.4 各手法を構成する原理

各手法にはそれを構成するいくつかの技術的要素がある．これらを開発の原理として考え，比較した(**表-15**)．

実世界は，並存し互いに通信するオブジェクトからなる．したがって，並行処理 (Concurrency) の概念が実世界のモデルの中に意識的に組み入れられていることは重要である．オブジェクト指向の特徴であるカプセル化 (encapsulation)，メッセージ通信 (message passing)，継承 (inheritance) を取り上げた．オブジェクト指向分析・設計法では，再利用できる構成要素の組織的な抽出に継承を使う．JSD では抽象データ型が，RSA では機能の共通性が，再利用の原理である．オブジェクト指向のオブジェクトの並存，カプセル化，メッセージ通信は，分散システムの並行性やデータの局所性，メッセージ通信と対応する．オブジェクト指向が分散システムの設計に使える根拠になっている[16]．しかし，分散システムでは，分散されたオブジェクトには個別の時刻があること，通信時間の遅れや物理的制約による性能への影響が無視できないことが追加される．すなわち，分散システムでは時刻の取扱い，性能要求，時間の仕様化も重要な項目である．JSD のみが同期や時間といった分散システムで重要な概念を明示的に扱っている．これをオブジェクト指向設計で補うとすれば，それは分散システムにオブジェクト指向設計を特化することになり，オブジェクト指向設計の一般性は犠牲になる．

複数の人数で開発する場合必須になるグループ化 (module/subject)，オブジェクトの整合性を宣言する制約 (constraint) を比較項目に含めた．JSD は，必要なモデルプロセスをネットワークに集めるというグループ化でサブシステム化する．グループ化の表記法を特に設けていない．大規模システムの特徴の一つは，グループ化でも対応が困難なほどのサブシステムの数とサブシステムの範囲が広いことである．しかし，グループ化がかなり役立つことも確かである．

表-15 各手法を構成する原理

	RSA	JSD	OOA/D	OMT
並行処理	*	*		*
カプセル化		*	*	*
メッセージ通信		*	*	*
グループ化	*	*	*	*
継承			*	*
制約	*	*	*	*

表-16 望ましい予備知識と経験

	RSA	JSD	OOA/D	OMT
構造化分析／構造化設計	*		*	*
データモデル	*	*	*	*
状態遷移図	*	*	*	*
実体関連図	*	*	*	*
ジャクソン法 (JSP)	*	*		
オブジェクト指向プログラミング言語			*	*

4.5 望ましい予備知識と経験

勉強するにあたってあらかじめ知っておくと便利な知識と経験を比較した (**表-16**)．プログラミング言語は，ソフトウェアの素材でもある．実現で使われるプログラミング言語の知識をもつことは設計における判断を確実なものにするという意味で役に立つ．

5. おわりに

RSA, JSD, OOA/OOD, OMT は，体系化された開発方法であるという意味では，いずれも同じである．

従来の方法は実践を通じて時間をかけて洗練され改良されてきたことを考えると，オブジェクト指向分析・設計法の洗練はこれからである．従来の開発方法での応用経験や知識がオブジェクト指向分析・設計の応用において，新しい概念や方法の導入時の開発者の反応への対処などで，役立つ．オブジェクト指向分析・設計法は従来の方法の影響を受けており，これらの知識はオブジェクト指向分析，設計を理解する上で有用である．

従来の手法では機能的な言葉で考えているという指摘は，誤ってはいない．従来の方法では実体の抽出が重要であったために，オブジェクトやクラスに相当する概念に気付いていた．そして，それらと機能の仕様化を組み合わせてモデル化していた．しかし，オブジェクトという概念を使わなかったという点でどうしても機能的な言葉に頼らざるをえなかったといえる．

従来の方法もオブジェクト指向分析・設計法も静的側面と動的側面，機能的側面を抽象化することでは同じである．しかし，OMT のアーキテクチャや OOA/OOD の人間とのインタフェースの扱いは従来の方法にはない部分である．

教育という面からみると，RSA または JSD のような従来の方法を使ってきた技術者が，今後 OOA/OOD や OMT を学習しそれを応用することはさほど困難なことではない．また，従来の方法を知らない技術者には OOA/OOD や OMT が前提としている RSA や JSD の知識をいかに体系化し簡潔に教えるかが課題として残る．従来の方法もオブジェクト指向の方法も実践を通じて理解し使えるようになる．教育だけでは不十分である．したがって，実践用のマニュアルを教材とは別に用意することを薦める．

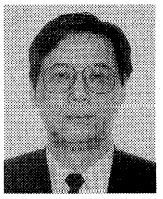
素材であるプログラミング言語が豊かになれば，それらを上手に使える方法が有効であることは自明である．オブジェクト指向分析・設計法はオブジェクト指向プログラミング言語を視野に入れた方法であり，この意味でこれからのソフトウェア開発法の主流になると考えられる．

今後の課題として，分散システムの開発にオブジェクト指向分析・設計が使われるためにはアーキテクチャと時間の扱い方の方針が必要になってくる．JSD の時間の扱い方が一つの参考になると思われる．

参考文献

1) Ward, P. and Mellow, S.: Structured Development for Real-Time Systems Vol. 1-3, Yordon Press (1986).
2) Jackson, M.: System Development, Prentice-Hall (1982). [邦訳：システム開発 JSD 法（大野旬郎，山崎利治訳）共立出版 (1989)].
3) Cameron, J. et al.: Tutorial JSP & JSD The Jackson Approach to Software Development 2nd Ed., IEEE (1889).
4) LBMS: LBMS Jackson System Development, John Wiley & Sons (1992).
5) Cord, P. and Yourdon, E.: Object-Oriented Analysis, Prentice-Hall (1990). [邦訳：オブジェクト指向分析（OOA）第2版（羽生田栄一監訳）トッパン (1991)].
6) Cord, P. and Yourdon, E.: Object-Oriented Design 1991, Prentice-Hall (1991).
7) Rumbaugh, J. et al.: Object-Oriented Modeling and Design, Prentice Hall (1991). [邦訳：オブジェクト指向方法論 OMT モデル化と設計（羽生田栄一監訳）トッパン (1992)].
8) Meyer, B.: Object-Oriented Software Construction, Prentice-Hall (1988).
9) 加藤，大野：JSD 実用システムへの応用，BIT, Vol. 22, No. 9, pp. 90-98, 共立出版 (1990).
10) DeMarco, T.: Structured Analysis and System Specification, Prentice-Hall (1979). [邦訳：構造化分析とシステム仕様（渡辺純一訳）日経 BP, (1986)].
11) Jackson, M.: Principles of Program Design, Academic Press (1975). [邦訳：構造的プログラム設計の原理（鳥居宏次訳）日本コンピュータ協会 (1980)].
12) Winograd, T.: Beyond Programming Languages, CACM, Vol. 22, No. 7, pp. 391-401 (1979).
13) Yourdon, E. and Constantine, L.: Structured Design, Prentice-Hall (1978).
14) Fisherman, R. and Kemerer, C.: Object-Oriented and Conventional Analysis and Design Methodologies, IEEE Computer, pp. 22-39 (Oct. 1992).
15) Monarchi, D. and Puhr, G.: A Research Typology for Object-Oriented Analysis and Design, CACM, Vol. 35, No. 9, pp. 35-47 (1992).
16) Andleigh, P. and Gretzinger, M.: Distributed Object-Oriented DATA-SYSTEMs Design, Prentice-Hall (1992).

（平成5年9月3日受付）



加藤 潤三（正会員）

1971 年岡山大学理学部物理学科卒業．同年日本ユニバック(株)入社．顧客システムの開発，ソフトウェア開発方法の開発と適用および支援ツールの試作に従事し，1991 年同社を退社．現在，エレクトロニック・データ・システムズ(株)にてテクニカル・コンサルティング部部長．ソフトウェア工学，特に CASE ツール，ソフトウェアプロセスモデル，分散システムの設計，オブジェクト指向ソフトウェア開発等のソフトウェア開発方法，ツールの開発に興味を持つ．