ソフトウエア工学 93－4
（1993．7．8）

ソフトウェア仕様化・設計法の最近の研究動向

佐伯元司　　郭文音
東京工業大学 工学部 電気電子工学科

本論文では、ソフトウェアの開発効率や品質の向上を目的として、現在までに研究、開発 が進められてきたソフトウェア仕様化・設計法について概説する。さらに、これらの方法論の実用化へ向けて進められている最近の研究動向を紹介し、将来の展望について議論する。

Current Trends of the Studies on Software Specifications & Design Methods

Motoshi Saeki　　Kuo, Wen-yin
Department of Electrical and Electronic Engineering
Tokyo Institute of Technology

This paper attempts to overview the software specifications & design methods and introduce the trends of the studies of them. These researches are much impor tant since software specifications & design methods turned to practical value in these years. Furthermore we will discuss furture direction of the studies on software specifications & design methods.

1 はじめに

ソフトウェア開発には，まず要求の仕様化を行い，それに基づいて設計を行う段階がある．これらの段階は，ソフトウェア開発のライフサイクル中で先頭に位置するため，これらの段階で行う作業の質が，ソフトウェアの開発効率や開発されたソフトウェアの品質に大きな影響を及ぼす．ソフトウェアの仕様化や設計方法論，技法や支援ツールについて種々の研究がなされ，実用化への努力もなされている．本稿では，現在までに研究・開発が進められてきた仕様化・設計法について概説し，実用化へ向けて進められている研究動向を紹介し，将来の展望について議論する．なお，本稿では方法論，技法に限定し，CASE ツールなどの方法論を支援するツールの研究は扱わないことにする．

方法論を実用化するためには，まずその方法論を調査・分析し，どのような種類のソフトウェアシステムの仕様化に適しているか，弱点はどこにあるかを明らかにする必要がある．調査・分析には，以下の3つのステップが考えられる．

1. 方法論が解説されている教科書，論文を基に，作成しなければいけないプロダクトや作成の手順を分析する．
2. 実際に小規模な問題を，その方法論を用いて解き，作成プロセスに記録をとり，それを分析する．
3. 実際のソフトウェア開発現場で，大規模システムがどのように方法論を使って，開発されているかを調査・分析する．

このようなステップを踏むことにより，各方法論の適用可能性，弱点等が明らかになり，弱点を克服した新しい方法論の開発が可能になってくる．これらのステップの関係を図1に示す．

まず，第2，3節では，これまでに提案された構造化手法やオブジェクト指向法などの方法論や形式的手法を紹介する．形式的仕様記述言語は厳密には方法論とは呼べないが，最近になって，形式的仕様記述言語を用いた開発法が提案され，その成功した事例が報告されていること，ISO で次々と形式的仕様記述言語が国際標準化されてきていることなどを考慮し，3節で取り上げることにする．第4節で，上記の第1のステップの研究動向を概説する．このステップの研究でよく使用される手段は，「比較」や「分類」である．すなわち，他の方法論と比較し，差異を明らかにしたり，方法論をある観点から分類したりする．

仕様化・設計プロセスは，人的要因にかなりの部分を依存しているため，方法論を用いたとしても，人間が行なわなければならない高度な知的作業はプロセス中に残されている．このような作業は方法論の教科書には記載されておらず，それが必ずしも教科書どおりに開発作業が進められない原因にもなっている．このような作業を分析するため，研究室レベルで小規模な問題を方法論に従って記述し，そのプロセスを分析する研究が行なわれている．これが上記の第2ステップで，教科書に記載されていない作業を通して，方法論の特徴や弱点をつきとめ，支援ツールに反映させようとする研究である．これらについては，第5節で紹介する．

ソフトウェア開発は，人間集団の社会活動である．関与する人間も顧客，ユーザ，設計者，プログラマ，マネジャーと様々であり，また所属している組織やその環境も様々である．従って，方法論の適用効果もこれらの社会活動，社会環境といった要因を加味して論じなければならない．例えば，方法論の使用に関する組織の方針や方法論の教育体制などが大きな影響を及ぼすと考えられる．これらを明らかにするための研究が第3ステップで，実際の開発現場でどのように方法論が使用され，どのような効果をあげているかを調査・分析する研究である．このような研究の報告は，まだ少ないが，第6節でその一例を紹介する．

伝統的なウォータフォール型のモデルに従えば，ソフトウェア開発はまず要求分析を行ない，その後に設計を行なうという段階がある．文献 [1] では，分析は問題領域の記述を行なう作業であり，設計はその問題の解領域の記述を行なう作業としている．ウォータフォール型のモデルに従った方法論も，構造化分析法，構造化設計法といったように，分析段階を支援するものと設計段階を支援するものとに分かれている．しかし，ジャクソン開発法などのように，ウォータフォール型と異なる開発形態をとる手法では，分析と設計の区別は明確化されていない．本稿では，形式的仕様記述言語も扱うため，これらを一括して仕様化・設計法と呼ぶことにした．特に，分析法，設計法の区別が必要な箇所では，これらを使い分けることにした．

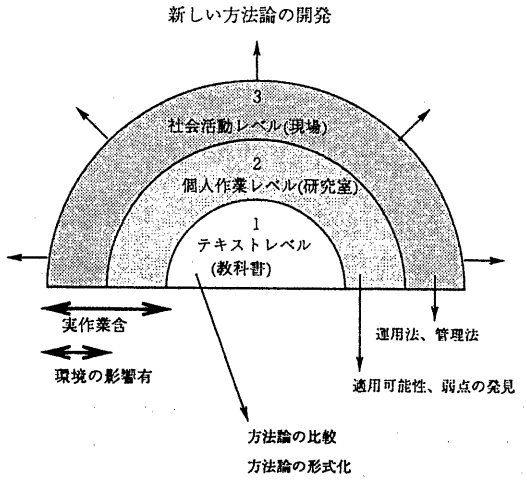


図 1: 方法論の研究動向

2 仕様化・設計法の概要

2.1 仕様化・設計法とは？

[2] や [3] でも述べられているように，ソフトウェアシステムは以下の一般的な4つの側面から捉えることができる．

1. 機能的側面 (Functional Aspect)
   対象システムが持っている機能と機能間のデータの流れ
2. 動作的側面 (Behavioral Aspect)
   対象システムの動作系列などの時間的な振る舞い
3. 構造的側面 (Structural Aspect)
   対象システムの構成要素（物理的なものから概念的なものまで含めて）とその要素間の関係，インタラクション
4. 情報的側面 (Informational Aspect)
   対象システムが扱う情報（データ）及びその構造

従って，方法論をこれら4つのどの側面からの仕様化を支援してくれるかで分類することができる．

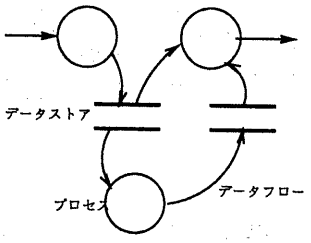
仕様化・設計法の方法論は，仕様書を作り上げるために，何をどのような手順で作っていけばよいかを教えてくれる．従ってすべての方法論は，

- プロダクトの視点 ： どんなプロダクトが作られるか，つまりプロダクトの構造．方法論が支援しているプロダクト（中間生成物も含む）は，記法まで考えるとさまざまであるが，本質的なものは表1のようにまとめられる．この表は，分析段階の生成物のみを対象にしたものである．
- 作業の視点 ： どんな作業をどの順序で行なわなければならないか．直観的には，最初に行う作業の結果を基に後続の作業が進められていくため，最初に行う作業が重要であると思われる．

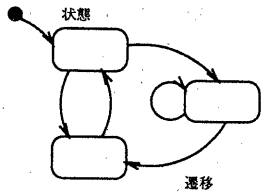
表 1: 代表的なプロダクト

機能的側面	動作的側面	
データフロー図 (DFD)	状態指向	イベント指向
	状態遷移図 (STD) メッセージシーケンスチャート	イベント系列図（実体構造図など）
	ペトリネット	

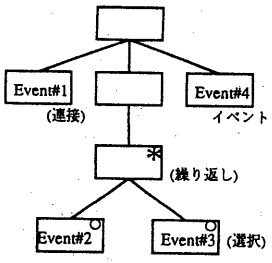
構造的側面		情報的側面
静的構造	動的構造	実体関連図 (ERD)
実体関連図 (ERD)	オブジェクト通信図	データ構造図 (Jackson Tree など)



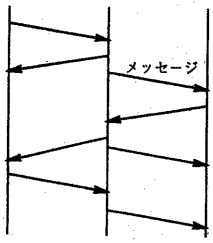
1)データフロー図



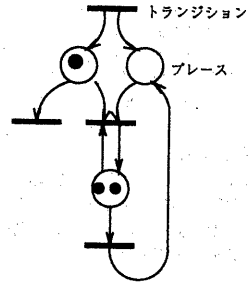
2)状態遷移図



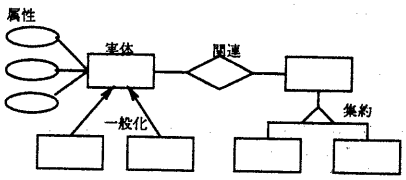
3)イベント系列図



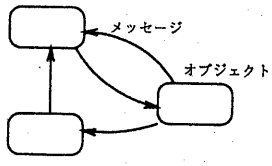
4)メッセージシーケンス
チャート



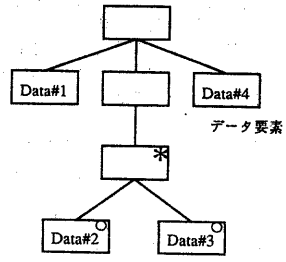
5)ペトリネット



6)実体関連図(拡張)



7)オブジェクト通信図



8)データ構造図

図 2: プロダクトの記法

の二つの視点で整理することができる．

例えば，Coad&Yourdon のオブジェクト指向分析法 [4] では，表2に示すような作業手順でプロダクトを作成していく．もちろん，すべての手法がプロダクトや作業手順の両方を支援しているわけではない．どちらか片方しか支援していない手法もある [1]．

これまでに提案された方法論を，プロダクト，作業の観点から上述の4つの側面のどれに主眼をおいているかで分類すると，表3のようになる．

2.2 構造化手法

構造化手法では，分析段階は構造化分析法 (Structured Analysis)，設計段階は構造化設計法によって支援する．

構造化分析法 [5] は対象システムの要求を，システムが持っている機能，機能間でのデータの流れで捉える手法で，1970年代始めに構造化設計法とともに提案された．当初のねらいは，データとデータを処理するプロセスとをデータフロー図を用いて，図的に表現することにあったようである．その後，DeMarco によって，データ辞書やミニスペックの概念が追加され，現在までに種々の拡張が行なわれてきている．その最も代表的なものは，実時間システムを記述するために，Ward らや Hartley らによる状態遷移図の導入であろう [9]．彼らは，データフローとともにプロセス間の制御信号のフロー（コントロールフロー）を記述し，制御信号フローのタイミングを状態遷移図で記述する．Ward らの手法は，同じデータフロー図中に制御信号フローを混在させるのに対し，Hatley らはこれらを別の図として分けて書く手法をとっている．状態遷移図で複雑なシステムを記述する際の欠点として，状態数が膨大になってしまうことが挙げられる．Harel らは AND 分割，OR 分割と呼ばれる状態の階層化機構を導入した状態遷移図 State Chart を提案し [22]，データや制御信号の流れを表した図ともに用いた [2]．

構造化設計法 [7] は，得られたデータフロー図中のプロセスをプログラムモジュールに系統的にマッピングしていく手法で，トップダウン設計，モジュラリティを高めることをねらいとしている．手法としては，変換分析法とトランザクション分析法の2つがある．構造化設計を行なった結果得られるものは，モジュール，モジュール間の呼びだし関係（サブルーチンコール），モジュール間での入出力データの受渡し関係を表したモジュール構造図である．

Gomaa らは構造化設計法を実時間システム用に拡張した設計法 DARTS(Design Methods for Real Time Systems) を提案している [11]．現在では，タスクの構造化やモジュールの情報隠蔽機構が拡張されている [23]．

2.3 オブジェクト指向的手法

現実世界を「オブジェクト」なるものでとらえていくことから始まる開発法が，オブジェクト指向的手法である．オブジェクトが持っている特徴として，Rumbaugh らは 1) クラス概念，2) 多態 (Polymorhisim)，3) 継承 (Inheritance) をあげている [13]．オブジェクト指向的手法は，上記の特徴（の一部もしくは全部）を活かした手法である．これまで，オブジェクト指向分析法・設計法は種々のものが提案され，プロダクトの記法や作業手順も様々である．文献 [1]，[24] に種々のオブジェクト指向的手法を詳細に渡って比較した例があるので，そちらを参考にされたい．

オブジェクト指向分析法では，ほとんどの手法がオブジェクトの識別作業から始めるように支援している．識別されたオブジェクトはクラスとして抽象化され，最終的には実体関連図で表現される．オブジェクト間の関係は，実体関連図中の関連 (relationship) で表現されるが，オブジェクト指向法特有の一般化/特殊化関係 (Generalization/Specialization：継承関係)，集約関係 (Aggregation) といった関係を特別な記法で表現しているものもある．

オブジェクトが持っている属性や機能，他のオブジェクトとの関係，オブジェクトの振舞いなどがモデル化されていき，データフロー図，状態遷移図，イベント系列図，メッセージシーケンスチャート，オブジェクト通信図などで記述されていく．どのような順序でモデル化がなされるか，どのような記法を用いるかは，手法ごとに異なる．

2.4 ジャクソンシステム開発法

ジャクソンシステム開発法 (JSD)[17] は，分析段階から実現段階までを一貫して支援するための方法論である．機能仕様の変更に強いシステムを構築するため，まず対象とする現実世界のモデル化を行い，できあがったモデルに対し必要な機能を付加し，実際の稼働環境に合わせて実現を行うという3段階を踏む．これらは，それぞれ Modeling Phase, Network Phase, Implementation Phase と呼ばれる．Modeling Phase では，現実世界中の実体 (Entity) と実体に起こる動作 (Action あるいは Event という) を抽出し，実体ごとの動作系列を図2のイベント系列図で記述する．この図は，実体構造図 (Entity Structure Diagram) という．Network Phase では，この実体を並列に動作するプロセスとみなし，各機能を表すプロセスとデータを交信しあう．プロセスとプロセス間データの交信は，システム仕様図 (System Specification Diagram) と呼ばれる一種のデータフロー図で記述される．データの交信には，受信側プロセス，送信側プロセスのどちらに主導権があるかで，状態ベクトル結合とデータストリーム結合の2種類がある．

2.5 データ指向的手法

データ指向的手法は，現実世界をシステムが扱うデータで捉え，そのモデル化から始めていく手法である．データベースアプリケーションに使用される情報モデリング手法 (Information Modeling, Data Modeling) は，扱うデータを実体関連モデルで捉える手法である．オブジェクト指向分析法との違いは，データと考えられるもののみを実体 (Entity) としてモデル化する点である．

分析段階で，データの構造を連接，選択，繰り返しの3つの操作でモデル化し，設計を行っていく手法がデータ構造指向的手法である．この手法の最大の特徴は，分析段階で得られたデータ構造からプログラムのモジュール構造を得るためのシステマティックな設計段階の手法にある．この手法の一つであるジャクソン法 (Jackson Structure Programming：JSP) は，入力データ，出力データを連接，選択，繰り返しでモデル化し，各々，図2のデータ構造図（正式にはジャクソン木という）で表現する．2つの図のノード間にある条件を満足する対応関係を求め，入力データの構造図と対応関係を基にモジュール構造を抽出する．対応関係が見つからない場合は，構造不一致 (Structure Clush) と呼ばれ，状況に応じて対策が用意されている．

3 形式的仕様記述言語

形式的手法 (Formal Method) とは，集合論，論理や代数といった数学的な体系を基礎としてソフトウェア開発を行う手法である．これらの数学的な体系は主にソフトウェア (プログラムだけでなく仕様書といった成果物を含む) を記述するのに用いられる．これまで，記述された開発対象システムの各種の性質を数学的体系が持っている演繹体系を用いて証明できることが，形式的手法を用いることの利点とされている．例えば，形式言語を用いてソフトウェアの仕様を記述し，記述された仕様から対象システムが持っている性質を演繹したり，仕様から演繹規則を用いてシステマティックにプログラムを合成・導出したり，開発されたプログラムが仕様を満たしているかというプログラムの正当性を証明するのに使用したりすることができるとされている．これまでに，種々の仕様を記述するための形式言語，形式的仕様記述言語

表 2: 方法論の例 (Coad&Yourdon の OOA)

	作業手順	作成するプロダクト
1	オブジェクト，クラスの識別	実体（クラス・オブジェクトを表す）のみからなる実体関連図（クラス・オブジェクトレイヤ図）
2	クラス間の構造・関係の定義	関連を追加した実体関連図（構造レイヤ図）
3	サブジェクトの定義	実体をグループ化した実体関連図（サブジェクトレイヤ図）
4	属性の定義	実体に属性を追加した実体関連図（属性レイヤ図）
5	サービスの定義	実体にサービスを追加した実体関連図（サービスレイヤ図）

表 3: これまでに提案された方法論

		分析法	設計法
機能的側面 (動作的側面)	構造化手法	構造化分析法 (SA：DeMarco[5], Gane&Sarson Yourdon[6] など)	構造化設計法 (SD：Yourdon&Constantine[7], Myers[8] など)
		実時間システム用構造化分析法 (Real-Time SA：Hartley, Ward[9], Ward&Mellor[10], Harel[2] など)	DARTS[11]
構造的側面	オブジェクト指向法	オブジェクト指向分析法 (OOA：Shlaer&Mellor[12], Coad&Yourdon[4], OMT[13] など)	オブジェクト指向設計法 (OOD：Booch[14], Coad&Yourdon[15],Wirfs-Brock[16] など)
		ジャクソンシステム開発法 (JSD)[17]	
情報的側面	データ指向的手法	実体関連モデル (Chen[18])	
		情報モデリング手法 (Martin[19] など) データ構造指向的手法 (Warnier[20], Jackson の JSP[21] など)	

が開発されてきた．これらは，一般的に議論されているように，大きく分類すると，以下の２つに分けられる．

1. モデル指向 (Model Oriented)
   数学の構成的な概念，例えば集合，関数，直積などを用いて対象システムの抽象的なモデルを作ることによって仕様を記述していくための言語で，代表的なものに VDM[25], Z [26], CSP[27], CCS[28], ペトリネット [29] などがあげられる．

2. 性質指向 (Property Oriented)
   対象システムのモデルを作り上げるのではなく，システムの満たすべき性質を記述していくことによって，仕様を記述するための言語である．形式論理を用いて性質を公理として記述するものと，代数の等式として制約を記述するものとがある．前者の例としては，Larch, Iota, Temporal Logic を用いる手法があり，後者の代表的な例は OBJ3, Clear[30], Act on[31] などの抽象データ型の代数的仕様記述言語があげられる．

その他に，両者の性質を併せ持つ言語も存在する．例えば，1988 年に ISO で規格化された LOTOS(Language of Temporal Ordering Specification)[31] は，モデル指向の CCS と性質指向の仕様記述言語 ACT one の両者の系統を組む言語である．LOTOS は，OSI の通信システムの仕様記述に用いられ，システムの動作に関する部分は，動作式 (Behavior Expression) と呼ばれる CCS をベースにした言語で，システムが扱うデータおよびそのオペレーションは抽象データ型の代数的仕様記述言語 Act one で記述する．なお，文献 [32] に形式的仕様記述言語の詳細概説がされているので，そちらも参照されたい．

4 方法論の比較・分析・モデル化

4.1 方法論の比較・分析

Pressman は，方法論の比較のポイントとして，

1. 対象領域の分析の機構
2. 機能やふるまいをどう記述するか
3. インタフェースの定義
4. 問題の分割機構
5. 抽象化の支援法
6. 問題の本質的な観点からの表現とインプリメンテーションの観点からの表現との差異

を挙げており [33]，これらをベースに，Monarchi らが２３のオブジェクト指向分析・設計法の比較検討を行なった [1]．この検討の結果，オブジェクト指向法の共通の利点・弱点（例えば，対象システムのミクロなレベルからマクロなレベルまでの種々の見方を合わせたり，統合したりするわかりやすい手法を持っていないなど）が洗い出せたとしている．

Fichman らは，構造化手法などの伝統的な手法と各種のオブジェクト指向的手法がサポートしているものの中から，分析法については１１の項目，設計法に関しては１０の項目を選び，比較を行った．例えば分析法の場合，比較の項目として，実体（オブジェクトに該当）の識別・分類，一般–特殊化や部分– 全体といった関係，状態や状態遷移がサポートされているかなどが挙げられている．彼らの比較の目的は，構造化手法やデータ中心手法が現場でよく使用されている状況を踏まえ，これらの手法からオブジェクト指向法へ移行した際の開発プロセスや資源を大きく変更しなければいけないかどうかを調べることにあった．結果は，Martin の手法などのデータ指向的手法からではオブジェクト指向分析法へ移行するのにはそれほど大きな変更がないが，

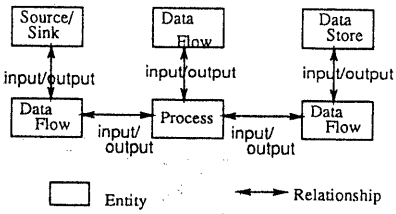


図 3: データフロー図のモデル化例

構造化分析法からでは根本的に大きな変更が必要であることがわかったとある．設計法では，データ中心設計法，構造化設計法どちらもオブジェクト指向設計法へ移行するのに大きな変更が必要であるという結論が得られた．

ジャクソンシステム開発法，リアルタイムＳＡ (Ward & Mellor) とオブジェクト指向分析法 (OMT, Coad) を，手順，作り上げるモデルの違い，教育の観点から比較を行なったものに，[34] がある．

4.2 方法論のモデル化

方法論の記述を支援するために種々の CASE ツールが開発されている．しかし，通常は１つの CASE ツールでは１つの方法論しか支援できない．１つのツールで複数の方法論を支援するために，CASE ツールの生成系，メタ CASE の開発が行われている．これらの開発のためには，方法論をモデル化し，形式的に記述することが必要である．この形式的な記述のためのモデルをメタモデルという．実体関連モデル (Entity Relationship Model) をメタモデルとして，データフロー図を記述した例を図 3に示す．ソフトウェア仕様化/設計法のために，いくつかのメタモデルが提案されている．表 4はそれらのメタモデルを次の視点から比較したものである．

1. 記法 :設計法で用いるグラフィカルな記法である．それぞれの設計法に固有の記法がある．たとえば，SA ではプロセスを丸で表す．この視点ではメタモデルが概念の記法をグラフィック化できるかどうかを考えている．Viewpoints Approach[35] と MetaEdit[36] は，ER モデルを使って設計法の概念のノーテションをグラフ化して表せる．しかし，簡単なグラフしか表せないので，たとえば， Rumbaugh の OOA[13] などのダイヤグラム のようなやや複雑な記法になると，表現しにくくなると思われる．

2. 作業手順 : 複数の設計法でソフトウェア仕様書の開発をサポートするためのものである．これはこのツールではどの設計法でどんな作業が行なわれるかというような情報を持っている．この視点では，メタモデルで設計法の作業手順の情報を定義する機能と，実際の設計作業を記録する機能を評価する．つまり，この視点は設計法の中で生成された生成物の構造を注目するものである．Viewpoints Approach は *work plans* と *work records* によってその作業の情報を記録できるが，それらは簡単すぎて非形式的であるため，設計法の記述や作業を有効に表すことができないと思われる．

3. 階層構造 : 階層構造には二つの種類がある．ひとつは同じ設計法で生成された生成物の階層構造で，もうひとつは異なる設計法で生成された生成物の階層構造である．

4. プロダクトの制約条件の記述 : 同じ設計法でも，異なる設計法でも生成された生成物の間に矛盾が生じることがあり得る．この条件の記述から生成物の一貫性をチェックすることができる．Brinkkemper は三つの側面 (event, task, data)

からメタモデルのモデリングのテクニックを提案した．彼はER モデルと一階述語論理 に基づいてその三つ側面から，それぞれのメタモデルを記述した．生成物の制約は述語論理によって書かれるが，実際のシステムには，そのような制約が効かないかもしれない．

5. 適用範囲：メタモデルの適用性について評価する視点である．メタモデルでどんな設計法を記述できるかが，適用性を決める要素の一つである．例えば，Brinkkemper のメタモデルは，一部分の設計法について明確に記述する能力をもっているが，情報システムでは，データ・フロー・ダイアグラム Data Flow Diagrams (構造分析の一部) と個体/関連 Entity Relationship モデリングのみである．Jordan らが，データ・フロー・ダイアグラム，Coad's OOA や State Transition Diagram[37] について議論していた．かれらのメタモデルはその三つの設計法にしか適用されないが，もし他の設計法を加えるとそのモデルがますます複雑になってしまう．

6. メタモデルの基礎：それぞれのメタモデルが何に基づいて作られたかを検討する．ほとんどのメタモデルは 個体/関連モデルか，それとも，個体/関連モデル の変形に基づいて作られたものである．述語論理の利用はメタモデルの記述能力を向上させることができるけれども，それとともに，有効なシステムを実現するのが難しくなると思われる．MetaEdit[36] はユーザにメタモデルを定義する機能を提供している．つまり，MetaEdit にはメタメタモデルが存在する．しかし，メタ設計法のガイドがないのでユーザにとっては相応しいメタモデルを作るのは困難である．

5 実験的手法によるプロセスの評価

小規模な問題を設定し，それをある方法論に従って仕様を作成するという実験を行ない，作成プロセスを分析するという手法は，その方法論の利点・欠点を調べる上で重要である．このような分析を通して，教科書には書かれていない重要な作業やそのやり方も明らかになることもある．実験のやり方としては，大きく分けて，

1. 分野の違う複数の問題を 1 つの方法論で解き，その分野への適用可能性を調べる，
2. 共通の問題を設定し，それを複数の方法論で解き，そのプロセスを比較する，

の 2 つが行なわれてきた．

5.1 オブジェクト指向法

Twente 大の Aksit らは，ネットワークデータベース，化学プロセスコントロール，知的チュータシステム，分散オフィスシステム，オブジェクト指向言語のプログラミング環境，分散オペレーティングシステムなどの様々なシステムの１２の開発プロジェクトを取り上げ，既存のオブジェクト指向法では解決解決できなかった問題点を分析した [40]．彼らは，抽出した問題点を現実世界をオブジェクトで捉える際の問題，オブジェクト間の関係（Classification や Aggregation など）に関する問題，オブジェクト間のインタラクションに関する問題の 3 つに分類した．第 1 のカテゴリでは問題領域から再利用可能な構造を導き出すことが難しい，2 のカテゴリではクラス間の継承がスーパークラスが持っている属性や演算をオーバーライドしたり拡張したり機構しか持っていないという問題点がよく見られたとしている．3 のカテゴリの問題点はよく見られ，オブジェクトを複数の見方で捉えることができない[1]，データ構造の一貫性，トランザクション，問い合わせといったデータベースが持っている概念を持っていない，メッセージのやりとりという低レベルのオブジェクト間インタラクション機構しかなく，高レベルのもっと抽象的なインタラクションを定義できない，といった問題点が挙げられている．特に最後のインタラクションの抽象化機構の欠如の問題は，6 つのプロジェクトで検出された．

本位田らは，Coad&Youdon のオブジェクト指向分析法をヘリコプターの自動着陸誘導システムの分析に適用し，その作業記録の詳細を形式的に表現した [41]．彼らの用いた手法は，分析作業を階層的に表現し，作業の入出力，つまり作業の機能的な側面をデータフローモデルで，動作的側面をフローチャートで表現した．これにより，教科書に出ていないような方法論の特徴を捉えることができるとしている．

5.2 方法論，仕様記述言語の比較

1987 年に開かれた Fourth International Workshop on Software Specification and Design で，分野の異なる 4 つの共通問題（図書館，暖房システムの制御，テキスト処理，リフトの制御）が設定され，それを出席者が独自の方法論・言語を用いて解くという試みがなされた [42]．Wing は，図書館問題を解いた１２の論文の方法論・言語を最終成果物である形式的仕様を基に比較分析を行なった [43]．彼女の分析手法は，問題文に含まれている曖昧な部分や不完全な部分がどのように取り扱われているかを調べるやり方で，形式的手法を用いることにより，曖昧な部分や不完全な部分の存在が識別でき，対処できることがあることを指摘している．また，比較した形式的手法は大きな差がなかったことも述べられている．

古宮らは，上記の共通問題を各種の方法論・言語で解いたときのプロセスを比較検討し，方法論・言語の実プロセスから見た際の分類を行なっている [44]．彼らは，対象システムのどの概念要素から識別していったかで，4 つに分類し，データとイベント概念を先に識別していくようなプロセスでは，ある特定の作業間で後戻りが顕著に見られ，それらの作業での強力な支援が必要であることを述べている．

6 観察的手法による実際のプロセスの評価

方法論がソフトウェアの生産性を上げるのに役に立っているか，方法論の弱点は何か，といったことを調べるために，実際に方法論が現場でどのように使用されているかを調査した研究もいくつかある．以下の節では，構造化分析法とオブジェクト指向分析法と形式的仕様記述言語の調査例を紹介する．

6.1 構造化分析法

構造化分析法は，かなり現場に普及していると思われるにも関わらず，どのように使用されているかの調査/分析，およびその結果の公表例が少ない．Bansler らは，デンマークの 3 つの組織のソフトウェア開発セクションをインタビューし，実際の開発プロジェクトの中で構造化分析法がどのように使用されたかを調べた [45]．その結果，構造化手法のテキストなどに載っている分析手順どおりには行なっていない，つまり部分的にしか方法論を使用しておらず，どの部分を使ったかも組織，プロジェクトごとに異なっていたということが判明した．具体的には，既存システムの分析を行なわなかったり，論理モデルを構築していなかったり，コンピュータ処理部分だけしか DFD を書かなかったり，他の手法（実体関連図や画面レイアウトを書いたり，ユーザインタフェース部分は第 4 世代言語を用いてプロトタイプを作ったりしたなど）を用いたりしたなどの現象が見られた．さらに書かれた DFD はユーザとのコミュニケーションはほとんど使用されなかった．これらの原因として，著者らは分析対象となった

[1] 例えば，先生，生徒というクラスを考えたときに，先生のあるインスタンスが他の先生の授業を受けるというケースも現実にはあり得る．このときのインスタンスは生徒という見方で捉えることが自然である．これが，1 つのオブジェクトを先生，生徒という複数の見方で捉えた例である．

表 4: メタモデルの比較

Meta Model	記法	作業手順	階層構造	制約条件の記述	適用範囲	メタモデルの基礎
Viewpoints[35]	○	△[1]	○	○	○	Template with 5 slots (in cl. ER)
Jordan & Davis[37]	–	–	–	–	–	ER
MetaEdit[36]	○	–	–	–	○	OPRR
MetaView[38]	–[2]	–	–	–[4]	○	ER
Brinkkemper[39]	–	–	–	○	△[3]	ER + Predicate Logi c
Method Base	×[3]	○	○	×[4]	○	ER

–: 記述されてない　　×: 含まれてない

ER: Entity Relationship Model
OPRR: Object Property Role Relationship Model (Extended version of ER model)
1): 非公式的な記述だけである.
2): ツール環境ではグラフィック的な記法を表せるが, メタモデルの本体には含まれてない.
3): 情報システム開発のメソッドのためだけである.
4): 制約条件の記述がツール環境で定義される, 又は, ツールによって定義される.

プロジェクトの規模（大規模ではなかった），構造化手法の教育の不備，組織の方針，方法論の本質的な弱さなどを挙げている．構造化分析法の後段は，構造化設計法が用いられるのが通常であるが，分析対象となった事例では複雑な部分にしか構造化設計法が用いられず，ほとんどの部分が直接コーディングされた．

6.2 オブジェクト指向分析法

オブジェクト指向の概念に基づいた設計法を使って実際のシステムを作成した四つの事例について考察する．表5にそれらの事例の概要を示す．これらの例から，オブジェクト指向の設計法の長所は再利用であることが分かる．一つのオブジェクトを作成すれば，構造が似ているオブジェクトはそのオブジェクトを拡張することで，容易に作成できるためと考えられる．さらに，再利用性があることにより，品質管理が行ない易くなったり，信頼性が高くなったり，実行効率がよくなったりする．しかし，欠点としては，“対象オブジェクトが不明確”，“モデルの間の矛盾”と“インスタンスの間の矛盾”が挙げられている．これらは現実世界の‘もの’を分析単位の‘オブジェクト’に写像する際に生じる問題である．このような問題が生じる理由はオブジェクト指向開発法を大規模システムに適用する場合の経験やモデルの蓄積が不十分であるからと考えられる．

6.3 形式的仕様記述言語

形式的仕様記述言語が開発された当時，1970,80年代前半には，その難しさのため，いずれも小規模の例題しか扱われず，実際規模の問題には適用されなかった．ところが，実際の開発プロジェクトに形式的仕様が使用され，成功を収めた例がここ数年の間にいくつか報告され，それがかなりの数にのぼってきた．また，OSIの通信システムの仕様を形式的に記述するために，LOTOS, ESTELLE, SDLといった形式的仕様記述言語が，国際標準として次々と規格化されてきた．本節では，形式的仕様記述が実際のソフトウェア開発の現場でどのように使用され，評価されているかを見てみよう．

ソフトウェア開発に形式的仕様を導入した実例は種々報告されており，開発対象となったソフトウェアもオシロスコープの制御プログラム，CASEツール，患者のモニタリングデータの実時間データベースシステム，地下鉄の制御システムなど，多岐に渡っている．これらの実例を調べ，分析した結果が文献 [50] に述べられている．これによると，形式的手法の使用法は，仕様を形式的に書いただけのプロジェクトから実際に検証を行なったプロジェクトまでがある．前者の例は，仕様の中に含まれる誤りを仕様記述段階でできるだけ検出する目的がある．

文献 [51] で述べられている CASE プロジェクトは，SSADM(Structured System Analysis and Design) を支援する CASE ツールを開発するプロジェクトで，仕様記述にはZが用いられた．できあがった仕様は，文書にして約 340 ページ（ただし，コメント付き），約 280 のオペレーションが約 550 の Z スキーマによって定義されている．彼らは，この仕様から直接，手でコーディングを行い（一部自然言語で設計仕様を書き，それをコーディングしたものもある），約 58000 行の Objective C によるソースを開発した．このプロジェクトでは，正当性の証明や数学的なプログラム合成・導出手法は用いなかったが，形式的仕様を用いることにより，仕様レベルのエラーを簡単に発見することができたということが報告されている．

検証を行なうことにより，プロダクトの質を挙げた開発事例もたくさん報告されている．IBM の Cleanroom を用いた開発例や ESPRIT プロジェクトの RAISE(Rigorous Approach to Industrial Software Engineering) を用いた事例などが挙げられる．Cleanroom による開発では，まず対象ソフトウェアの仕様が，Stimulas-Response モデルをベースにして形式的に記述される．仕様を記述するのは仕様記述チーム (Specification Team) が行なう．その後，仕様をインクリメント (Increment) と呼ばれるセグメントに分割し，インクリメントごとに設計，コーディングを行なっていく．設計，コーディングを行なうのは開発チーム (Development Team) であり，さらにこのチームはプログラムが仕様を満たしていることの検証を行なう．開発されたソフトウェアは確認チーム (Certification Team) によって統計的な手法によるテストが成され，信頼性がチェックされる．Linger は，Cleanroom を用いた開発プロジェクト 15 例を調査し，テスト段階で平均して 3.3 個/KLOC のエラーしかなかったと報告している．従来の開発法では 30–50 個/KLOC のエラーが検出されたことと比較すると高品質であることを述べている [52].

形式的手法の整備および実際の開発への適用は，かなり進んできたと言えよう．しかし，ソフトウェアの仕様記述が現実世界のモデルを作り上げるという作業である限り，いかにしてモデルを作り上げていけばよいかというモデル化の方法論が重要であるには変わりはない．つまり，形式的仕様記述言語を用いてどのように仕様を書いていけばよいのかが問題である．これに関しては，形式的仕様言語の Specification Style の研究 [53]，Z にオブジェクト指向設計法を取り込んだ HOOD[54] などの方法論の研究が課題である．

7 おわりに

本稿では，現在までに提案された仕様化・方法論を概説し，それらに対して行われている研究を紹介した．1970 年代初めに登場した構造化分析手法と違って，オブジェクト指向分析法などに代表されるように，最近の方法論は複数の側面（機能的側面，動作的側面，構造的側面，情報的側面）から対象をモデル化するの

表 5: オブジェクト指向設計法の事例

方法	分野	期間	言語	長所	欠点	その他
OOD	顧客の資料システム (CRIS)[46]	360 日 + 24 日*	PL/I	プロセスの数とメモリが少なくなる	–	IBM のメインフレーム上で CRIS が稼働
ROOD	ディジタル移動通信システム [47]	–	Ada	再利用性がある.	対象のオブジェクトが不明確	–
OMT	画像ファイリングシステム [48]	15.5ヶ月* + 進行中	–	拡張性がよい, 信頼性が高い, 実行効率がよい.	モデルの間の矛盾, インスタンスの間の矛盾	開発ツールは MacDraw
OOD	総合ソフトウェア管理システム (CAPS)[49]	–	C 言語	再利用性が高い, 品質管理し易い.	–	–

OOD: Object-Oriented Development
ROOD: Real-time Object-Oriented Development
OMT: Object Modelling Technique
CRIS: Customer-Related Information System
–: 明確に書かれてない場合
*: オブジェクト指向の教育の時間を含む

が普通である. また, 旧来の方法論も複数の側面からモデル化できるように拡張されてきた. 複数の側面からのモデル化は効果的であろうが, その反面ユーザは種々の記法や概念を学習しなければならない. 教育の問題等含めて, 実際の組織での適用事例の調査・分析の研究がますます重要になってくるであろう.

参考文献

[1] D.E. Monarchi and G.I. Puhr. A Reserach Typology for Object-Oriented Analysis and Design. *CACM*, Vol. 35, No. 9, pp. 35–47, 1992.

[2] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, and A. Shtul-Trauring. STATEMATE : A Working Environment for the Development of Complex Reactive Systems. In *Proc. of 10th ICSE*, pp. 396–406, 1988.

[3] B. Curtis, M. Kellner, and J. Over. Process Modeling. *Commun. ACM*, Vol. 35, No. 9, pp. 75–90, 1992.

[4] P. Coad and E. Yourdon. *Object-Oriented Analysis.* Prentice Hall, 1990.

[5] T. DeMarco. *Structured Analysis and System Specification.* Yourdon Press, 1978.

[6] E. Yourdon. *Modern Structured Analysis.* Yourdon Press, 1989.

[7] E. Yourdon and L.L Constantine. *Structured Design : Fundamentals of a Discipline of Computer Program and Systems Design.* Prentice-Hall, 1979.

[8] G.J. Myers. *Composite/Structured Design.* Van Nostrand, 1978.

[9] P. Ward. The Transformation Schema : An Extension of the Data Flow Diagram to Represent Control and Timing. *IEEE Trans. on Soft. Eng.*, Vol. 2, No. 12, pp. 198–210, 1986.

[10] S.J. Mellor and P.T. Ward. *Structured Development for Real-Time Systems I, II ,III.* Yourdon Press, 1986.

[11] H. Gomaa. A Software Design Method for Real Time Systems. *CACM*, Vol. 27, No. 9, pp. 938–949, 1984.

[12] S. Shlaer and S.J. Mellor. An Object-Oriented Approach to Domain Analysis. *ACM SIGSOFT Software Engineering Notes*, Vol. 14, No. 5, pp. 66–77, 1989.

[13] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lonrensen. *Object-Oriented Modeling and Design.* Prentice-Hall, 1991.

[14] G. Booch. Object-Oriented Development. *IEEE Trans. on Soft. Eng.*, Vol. 12, No. 2, pp. 211–221, 1986.

[15] P. Coad and E. Yourdon. *Object-Oriented Design.* Prentice-Hall, 1991.

[16] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software.* Prentice Hall, 1990.

[17] M.A. Jackson. *System Development.* Prentice-Hall, 1983.

[18] P. Chen. Entity-relationship Model : Towards a Unified View of Data. *ACM. Trans. on Database Systems*, Vol. 1, No. 1, pp. 9–36, 1976.

[19] J. Martin. *Information Engineering I, II, III.* Prentice-Hall, 1990.

[20] J.D. Warnier. *Entrainement àla Programmation LCP.* Les Editions d'Organisation, 1984.

[21] M.A. Jackson. *Principles of Program Design.* Academic Press, 1975.

[22] D. Harel. STATECHARTS: A Visual Formalism for Complex Systems. *Science of Computer Programming*, Vol. 8, pp. 231–274, 1987.

[23] H. Gomma. Structuring Criteria for Real Time System Design. In *Proc. of 11th ICSE*, pp. 290–301, 1989.

[24] 本位田真一. オブジェクト指向分析・設計総論. オブジェクト指向分析・設計チュートリアル, pp. 1–16. 情報処理学会, 1993.

[25] C.B. Jones. *Systematic Software Development Using VDM.* Prentice Hall, 1986.

[26] J.M. Spivey. *The Z Notation — A Reference Manual.* Prentice Hall, 1987.

[27] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[28] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[29] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice Hall, 1981.

[30] J.V. Guttag, E. Horowitz, and D.R. Musser. Abstract Data Types and Software Validation. *Commun.ACM*, Vol. 21, No. 12, pp. 1048–1064, 1978.

[31] ISO 8807. *Information processing systems — Open Systems Interconnection — LOTOS — A formal description tecnique based on the temporal ordering of obsevational behaviour*, 1989.

[32] J.M. Wing. A Specifier's Introduction to Formal Methods. *Computer*, Vol. 23, No. 9, pp. 8–24, 1990.

[33] R.S. Pressman. *Software Engineering : A Practitioner's Approach - third edition.* McGraw-Hill, 1992.

[34] 加藤潤三. 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 -. オブジェクト指向分析・設計チュートリアル, pp. 51–65. 情報処理学会, 1993.

[35] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints : A Framework for Integrating Multiple Perspectives in System Development. *International Journal of Software Engineering and Knowledge Engineering*, Vol. 1, No. 2, pp. 31–58, 1992.

[36] K. Smolander, K. Lyytinen, V.P. Tahvanainen, and P. Marttiin. MetaEdit — A Flexible Graphical Environment for Methodology Modelling. In *Proc. of 3rd International Conference CAiSE91, LNCS 498*, pp. 168–193, 1991.

[37] A.K. Jordan and A.M. Davis. Requirements Engineering Metamodel : An Integrated View of Requirements. In *Proc. of 15th COMPSAC*, pp. 472–478, 1991.

[38] P. Sorenson, J. Tremblay, and A. McAllister. The Metaview System for Many Specification Envirnments. *IEEE Software*, Vol. 2, No. 5, pp. 30–38, 1988.

[39] S. Brinkkemper. *Formalisation of Information Systems Modelling.* Thesis Publisher, 1990.

[40] M. Aksit and L. Bergmans. Obstacles in Object-Oriented Software Development. Technical Report Memoranda Informatica 92-15, Unversiteit Twente, 1992.

[41] Y. Kishimoto, S. Honiden, N. Kotaka. Formalizing Specification Modeling in OOA. *IEEE Software*, Vol. 10, No. 1, pp. 54–66, 1993.

[42] *Problem Set for the 4th International Workshop on Software Specification and Design: Proc. of 4th International Workshop on Software Specification and Design*, 1987.

[43] J.M Wing. A Study of 12 Specifications of the Library Problem. *IEEE Software*, Vol. 5, No. 4, pp. 66–76, 1988.

[44] S. Komiya, M. Saeki, S. Honiden, S. Ohtsuki, H. Horai, J. Kato, A. Ohmori, and K. Ohmaki. An Experimental Analysis for Classifying Specification Processes. In *Proc. of 5th International Conference on Software Engineering and Knowledge Engineering (SEKE'93)*, 1993.

[45] J.P. Bansler and K. Bϕdker. A Reappraisal of Structured Analysis : Design in an Organizational Context. *ACM Trans. on Information Systems*, Vol. 11, No. 2, pp. 165–193, 1993.

[46] T. Morgan J. Davis. Object-Oriented Development at Brooklyn Union Gas. *IEEE Software*, Vol. 10, No. 1, pp. 67–74, 1993.

[47] 梶原清彦. ディジタル移動通信システムへの適用. オブジェクト指向分析・設計チュートリアル, pp. 93–106. 情報処理学会, 1993.

[48] 山城明宏, 吉田和樹, 斉藤 悦生, 入内島 裕子. OMT 法による画像ファイリングシステムの分析. オブジェクト指向分析・設計チュートリアル, pp. 67–81. 情報処理学会, 1993.

[49] 鈴木啓之. 設計支援ツール開発へのオブジェクト指向開発. オブジェクト指向分析・設計チュートリアル, pp. 83–91. 情報処理学会, 1993.

[50] S. Gehart, D. Craigen, and T. Ralston. Observations on Industrial Practice Using Formal Methods. In *Proc. of 15th ICSE*, pp. 24–33, 1993.

[51] A. Hall. Seven Myths of Formal Methods. *IEEE Software*, Vol. 7, No. 5, pp. 11–19, 1990.

[52] R.C. Linger. Cleanroom Software Engineering for Zero-Defect Software. In *Proc. of 15th ICSE*, pp. 2–13, 1993.

[53] C.A. Vissers, G. Scollo, and M. Sinderen. Architecture and specification style in formal descriptions of distributed systems. In *Protocol Specification, Testing and Verification VIII*, pp. 189–204, 1988.

[54] R.D. Giovanni and P.L. Iachini. HOOD and Z for the development of Complex Software Systems. In D. Bjorner, C.A.R. Hoare, and H. Langmaack, editors, *VDM'90 : VDM and Z - Formal Methods in Software Development*, pp. 262–289, 1990.