第 4 章

パラダイム転換

4.1 パラダイムとは

パラダイムとは，トーマス・クーンの定義（文献(1)）によれば，

「一般に認められた科学的業績で，一時期の間，専門家に対して，問い方や答え方のモデルを与えるもの」

である。これに従えば，プログラミングパラダイムとは，

「一般に認められたコンピュータソフトウェアに関する業績で，一時期の間，専門家に対して，プログラムの書き方や読み方のモデルを与えるもの」

ということになる。具体的には，本書では，プログラミングパラダイムを以下のような意味で用いる。

プログラミングパラダイムとは：
プログラムの作り方に関する規範で，プログラムの設計手順やプログラムの構造の決め方，プログラムの記述／表現方法を規定するもの

プログラムの開発過程は，大まかには図 4.1 のようになる。この図では開発工程後半の検証過程は省略している。要求分析，設計，プログラミングの各々の工程でなんらかのパラダイムが用いられる。

要求分析では，開発しようとするシステム全体の要求仕様，すなわち「何を」(what-to-do) が決められる。しかし，要求分析段階ではあいまいな事柄が多

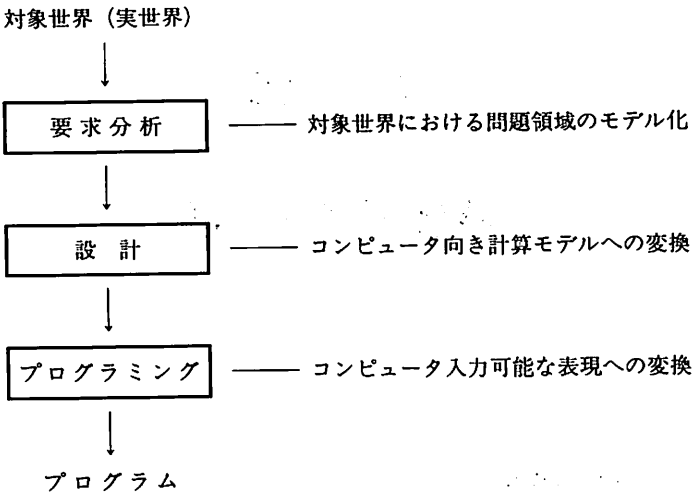

図4.1 プログラム作成過程（検証過程は省略）

く，この要求仕様を厳密に記述することは不可能に近い。そこで，少しでも厳密な要求仕様にするために，特定のモデル化技法をパラダイムとして用いる。したがって，パラダイムが異なれば要求分析技法も異なることになる。たとえば，事務処理の分野では，データの流れとそのデータを処理する機能の分割に着目し，データフロー図を用いて要求仕様を記述する構造分析技法がよく用いられている。制御システムの分野では，イベントの発生と時系列処理に着目し，状態遷移図を用いて要求仕様を記述する方法などがある。

要求仕様記述において最も重要なことは，「正しさ」であるが，これを確認するのはなかなか難しい。一般には，システムの発注者と開発者が異なることが多く，発注者と開発者が共同で「正しさ」の確認作業をしなければならない。しかし，発注者は業務の専門家ではあっても情報処理技術の専門家ではなく，その逆に，開発者は情報処理技術の専門家ではあっても，業務の専門家ではないため，一つの表現を両者が異なって解釈することもある。したがって，要求仕様は，両者にとってわかりやすいものでなければならず，要求定義技法は「わかりやすさ」の技術といえる。

設計は，対象世界の問題をモデル化した要求仕様を満たすシステムをコンピュータ上に実現するために，コンピュータで実行可能な計算モデルへ変換する作業である。ここでも種々の観点からの設計技法（設計パラダイム）がある。

特に大規模システムでは，分割統治の原則に従ってまず機能分割が行われ，設計仕様の構造化と階層化を実現する。この時，機能中心に分割していったり，データ構造に着目して分割していく技法の他，機能とデータを一体にしたオブジェクトに注目して分割していくオブジェクト指向設計法もある。設計後半のモジュール分割では，使用するプログラミング言語を意識したものになる。

プログラミングは，設計工程で十分小さく分割されたモジュール仕様をコンピュータ入力可能な表現へ変換する。ここで用いられるプログラミング言語はなんらかの計算モデル（プログラミングパラダイム）を具現化したものである。プログラムは，N. Wirth によれば，

プログラム＝アルゴリズム＋データ構造

という図式で規定される。さらにこのアルゴリズムは，R. A. Kowalski によれば，

アルゴリズム＝論理＋制御

という図式になる。したがって，プログラミングパラダイムおよびその具現化であるプログラミング言語としては，この構成要素である論理，制御，データ構造に関する視点の違いなどにより，種々のものが提案されている。コンピュータ利用の初期の段階から現在まで最も広く用いられているのは，特に制御構造を重視した手続き型パラダイムである。

この手続き型パラダイムについては次節で詳細に述べるが，フォン・ノイマンが提案したプログラム内蔵方式の最初のコンピュータ EDSAC-1 が 1949 年に完成して以来，コンピュータへの命令を 1 ステップずつ手続き的に記述するというプログラムの作り方は，この半世紀近くの間，まったく変わっていないのである。筆者は，この手続き的プログラミングスタイルこそが基本的問題であり，プログラミングスタイルに関するパラダイムシフト（パラダイム転換）が必要であると考える。

4.2 プログラミング言語の歴史

4.2.1 プログラミング言語の発展過程

プログラミング言語は，当初からソフトウェア生産性向上に大きな役割を果たしてきた技術である。プログラミング言語の研究開発に関するこれまでのマクロなトレンドを次の表4.1のように要約することができる。

表4.1 プログラミング言語の発展過程

時 期	目 的	内 容
1960年代	量的高級化	記述水準(機械語への展開率)の向上
1970年代	質的高級化	プログラミング方法論(構造化技法)の導入
1980年代	パラダイム転換	宣言的記述(手続き型から非手続き型へ)
1990年代	エンドユーザコンピューティング	脱プログラミング

(1) 高級言語化

最初の1960年代は，高価なコンピュータを効率良く利用することが重要であった。このようなコンピュータ利用の初期の段階に，プログラムの実行効率をあまり低下させないで生産効率をあげるために，アセンブリ言語からFORTRAN，COBOL，PL/Iなどの高級言語への移行が行われた。これは，機械語への展開率の向上という意味で「量的高級化」といえる。

まず1950年代後半にFORTRANとCOBOLが開発され，それぞれ科学計算用および事務処理用のアプリケーションプログラムに適用されていった。さらに，1960年代半ばには，ハードウェアやオペレーティングシステムとのインタフェースの記述にもアセンブリ言語を用いないで高級言語で記述できるように，PL/Iが開発された。

(2) 構造化

次に1970年代は，ソフトウェアの大規模化という第一次のソフトウェア危機への対応が重要であった。そのため，高信頼性と保守性を重視したプログラミングスタイルの研究が行われ，構造化技法などによりプログラムの理解容易性(「わかりやすさ」)を実現する「質的高級化」が追求された。手続き型言語に特徴的な制御構造の見直しが行われ，goto文の乱用や制御変数を用いたモジュール

間インタフェースなどの回避策が取られた。プログラムの主要な構成要素の一つであるデータ構造の表現方法についてもその抽象化が促進された。1970年頃に開発されたPascalでは，豊富なデータ型に加えて，データ型のユーザ定義機能が導入された。1970年代の終わりに開発されたAdaでは，プログラミング言語の「質的高級化」に関する70年代の主要な構造化技法が包含されている。

しかしながら，このような記述言語の高級化はソフトウェア危機のブレークスルー技術とはならなかった。1ステップずつ手続き的に記述するという点でプログラムの記述方法の基本は変わっていないからである。プログラムの実行性能を重視するかぎり，現在のフォン・ノイマン型コンピュータアーキテクチャと相性の良い手続き的記述方法に頼らざるをえない。

(3)　パラダイム転換

そこで1980年代には，ソフトウェアの生産性，信頼性，保守性に関する諸問題は，本質的に手続き型のプログラミングパラダイムに起因するという考えから，非手続き型の新しいプログラミングパラダイムとそれをベースにした言語の研究が盛んに行われるようになった。そのきっかけとなったのは，1980年代初めにおける，最初の本格的オブジェクト指向言語Smalltalk-80の登場と第5世代コンピュータ開発のための国家プロジェクトのなかでの論理型言語Prologの再評価であった。

しかしながら，新しいプログラミングパラダイムに関する活発な研究にもかかわらず，現在でも，実用的プログラムの開発に利用されているプログラミング言語のほとんどは，20年ないし30年前に開発されたCやCOBOLなどの手続き型高級言語である。その理由としては，過去の財産の継承，性能要求への対応，オープン化と標準化の時代のプログラムの移行性重視，などがある。

(4)　エンドユーザコンピューティング

さらに最近では，情報処理の専門家でなくてもソフトウェアを開発できるように，非手続き型プログラミングの概念をより進めて「脱プログラミング」を実現するエンドユーザコンピューティングの研究が盛んである。エンドユーザがプログラミング言語を用いないで業務のコンピュータ化を行えるための現実的な技術としては，第4世代言語，日本語プログラミング，ビジュアルプログ

ラミング，プロダクションシステム等がある。いずれも脱プログラミングのコンセプトに基づいているが，実際には「脱プログラミング言語」のレベルであり，プログラミングの概念やセンスが不要というわけではない。手続き型パラダイムからまだ脱してはいない。自ら作って使うという意味では，エンドユーザコンピューティングというよりもユーザオウンコンピューティングといった方が良い。ユーザが作らないで使うだけのOA用標準パッケージ，ハイパーテキスト，データベース検索，意思決定支援などの分野もユーザオウンコンピューティングの観点で注目していく必要がある。

4.2.2 手続き型高級言語の例

現在使用されている大半のコンピュータのフォン・ノイマン型アーキテクチャでは，1947年にJ. von Neumannが提案したプログラム内蔵方式を採用している。それ以前のコンピュータでは，コンピュータの動作を穿孔カードや紙テープで直接制御していたが，プログラムを記憶装置に記憶しておき，それをコンピュータが逐次的に実行するというプログラム内蔵方式により，コンピュータは飛躍的に高速化された。さらにデータとプログラムを記憶装置上で同じ扱いをできるようになり，プログラミング技術も飛躍的に進歩することになった。

1949年に英国のケンブリッジ大学で世界最初のプログラム内蔵方式のコンピュータEDSACが完成して以来，今日まで，プログラムを1ステップずつ記述していくという手続き型パラダイムのプログラミングが続いてきた。当初は，コンピュータの各機械命令に対応した「1」と「0」のビット列の符号を用いてプログラムを記述していた。このような機械語によるプログラミングはまもなくビット列の符号を記号化してニモニックコードで表現するアセンブリ言語の使用へと発展した。そして1950年代後半から1970年頃にかけて次々と高級言語が開発されていった。本節では，いくつかの代表的な手続き型言語の特徴について以下に示しておく。

(1) FORTRAN

1955年にIBM社のJ. Backusらが開発したもので，1966年には標準規格化された。FORTRANは，その名前がFORmula TRANslationに由来していることからもわかるように，元来，科学計算向きである。その後，1978年に規格

化されたFORTRAN 77では，構造化プログラミング用の制御文などの機能が加えられた。階乗計算プログラムの例を以下に示す。

```
      INTEGER FUNCTION FACT(N)
       FACT=1
       DO 10 I=1, N
10       FACT=I*FACT
      RETURN
      END
```

実用面では，スーパーコンピュータや汎用コンピュータなど比較的大きなコンピュータの分野で他機種への移行性が高いこと，高性能のオブジェクトプログラムを生成する最適化コンパイラがあること，ライブラリやプログラミング支援ツールが豊富である，等の利点があり，現在でもよく使われている。

(2) COBOL

1959年にコンピュータ関係のユーザとメーカで組織するCODASYL (the COnference on DAta SYstems Language)委員会で開発されたもので，1968年に標準規格化された。比較的簡単な構文で事務処理プログラムを自然語（英語）風に記述できる。1970年代後半には構造化プログラミング用の機能がつけ加えられた。階乗計算の例を以下に示す。

```
PROCEDURE DIVISION
    USING N FACT.
  MOVE 1 TO FACT.
  PERFORM
    VARYING I FROM 1 BY 1 UNTIL I>N
    COMPUTE FACT=I*FACT
  END-PERFORM.
```

実用面では，FORTRAN と同様に他機種への移行性が高いこと，プログラミング支援機能が豊富であることなどの利点がある。コンピュータの利用目的がこれまで圧倒的に事務処理が多かったことからよく使われている。特に定形業務向けのパッケージ化や開発手順の標準化と自動化が進んでいる。

(3) Algol

1960 年頃に開発され，Algorithmic Language に由来する ALGOL60 という名前で規格化されている。FORTRAN などの既存の言語では，意味仕様の曖昧さから細かい部分の仕様がコンパイラ毎に異なるという不便があったために，Algol では，言語仕様を厳密に定義したきれいな言語とする努力がなされた。その結果，プログラム内で使用する変数の名前とデータ型はすべて明示的に宣言する機能，変数や手続きの名前の有効範囲を明確にする機能，制御構造をわかりやすくする構造化プログラミング機能，手続き呼び出しにおける実引数と仮引数の関係を明確にする機能などが導入された。階乗計算の例を以下に示す。

```
integer procedure factorial(n);
  value n;
  integer n;
      factorial:=if n=0 then 1
                        else n × factorial(n-1)
```

実用面では，標準的な仕様定義が難しい入出力機能を言語仕様の対象外としたことやよいコンパイラが少なかったことからあまり普及していない。しかしながら，Algol で導入された重要な概念の多くが，PL/I や Pascal など，その後に開発された言語に大きな影響を与えた。

(4) BASIC

1964 年にダートマス大学の T. E. Kurtz らによって開発された言語で，1978 年に標準規格化されている。この言語は，修得が容易で，コンピュータを簡単に利用できるようにすることを目的にしていた。階乗計算の例を以下に示す。

```
10 INPUT N
20 LET F=1
30 FOR I=1 TO N STEP 1
40 LET F=I*F
50 NEXT I
60 PRINT F
70 RETURN
80 END
```

実用面では，タイムシェアリングシステム下で動くものが最初であった。その後，マイクロコンピュータの発展と共に普及し，パソコン上で動く小さなプログラムなどに用いられてきた。そのため，処理系もインタプリタ型が主流であったが，マイクロコンピュータの高性能化と共に実用的な規模のプログラム開発にも用いられるようになり，コンパイラが開発され，言語仕様が拡張されている。

(5)　PL/I

1965年にIBMコンピュータのユーザ団体であるSHAREが中心になって開発した言語で1979年に標準規格化されている。PL/Iは，科学計算から事務処理まで広範囲な応用プログラムを記述できるようにするために，FORTRAN, COBOL, Algolの機能を包含した仕様にすること，およびハードウェアやオペレーティングシステムの機能をアセンブリ言語を用いないで直接利用できるようにすることを意図したために，言語仕様はかなり大きくなっている。階乗計算の例を以下に示す。

```
FACTORIAL: PROC(N) RETURNS(FIXED BIN(31));
  DCL N FIXED BIN(31);
  DCL I FIXED BIN(31);
  DCL F FIXED BIN(31);
    F=1;
    DO I=1 TO N;
      F=I*F;
    END;
    RETURN(F);
END FACTORIAL;
```

実用面では，適用範囲が広く，ほとんどのプログラムがこの言語だけで記述できる利点があるが，コンパイラが大きくなるため大きなコンピュータでないと利用できないことや不用意なプログラミングをするとオブジェクトプログラムの効率が極端に悪くなることがあるので注意を要する。

(6) Pascal

1970年頃にチューリッヒ工科大学のN. Wirthによって開発された言語で，1983年に標準規格化された。学生にプログラミング技法を教えるのに適した言語として開発されたため，言語仕様が簡潔なわりにデータ型が豊富で，構造化プログラミングも可能になっている。コンパイラも1パスコンパイル方式でコンパクトに作れるように言語仕様が工夫されている。階乗計算をループおよび再帰関数を用いて記述した例の各々を以下に示す。

```
function FACTORIAL(N:integer):integer;
  var F,I:integer;
  begin
    F:=1;
    for I:=1 to N do F:=I*F;
    FACTORIAL:=F
  end
```

```
function FACTORIAL(N:integer):integer;
  begin
    if N=0 then FACTORIAL:=1
           else FACTORIAL:=N * FACTORIAL(N-1)
  end
```

実用面では，マイクロコンピュータの発展と共に急速に普及したが，標準規格化が遅れたために，実用的な規模のプログラムを開発するために機能拡張された部分に互換性がないことがある。また，実行時に配列変数のインデックスの範囲が正当か否かのチェックを行うため，オブジェクトプログラムの性能は必ずしも良くない。

(7) C

1972年頃にBell研究所のD. Ritchieによって開発された言語で，1989年にANSI標準規格となった。この言語は，アセンブリ言語の代わりとなるシステム記述言語として開発されたため，他の高級言語に比べて細かい記述が可能で，オブジェクトプログラムの実行効率も良い。階乗計算の例を以下に示す。

```
factorial(int n){
  if(n==0) return(1);
  return(n * factorial(n-1));
}
```

実用面では，この言語がUNIXの記述に用いられたことからもわかるように，UNIXと共に普及してきた。機種間での移行性が高いという利点が大きい。

4.3 パラダイム比較論

従来の手続き型高級言語の特徴は，一言でいえば，性能を気にしながら「作りやすさ」を追求することであった。高価なハードウェアを効果的に利用するためにプログラムの実行性能が最重要課題であったため，フォン・ノイマン型アーキテクチャと相性のよい手続き型パラダイムからはずれることなく，制御構造とデータ構造の高水準化と「分割統治」の原則に従ったモジュール機能の導入が行われた。このような量的高級化は，確かにプログラムの作りやすさに大いに貢献したが，1970年代に始まった「規模」と「量」と「質」に関するソフトウェア危機に十分対応できるものではなかった。

その後，プログラムの信頼性と生産性の観点から書きやすさよりも読みやすさ，作りやすさよりもわかりやすさが重視され始めた。作りやすさの追求からわかりやすさの追求へパラダイム転換が起こった。このような背景の中で，今日まで多くの新しいプログラミングパラダイムが提案され，適用されてきた。本書では，その以下のような代表的なものについて，後章で説明する。

① 構造化パラダイム（5章）
② 宣言型パラダイム（6章，7章）
③ 計算モデル的パラダイム（8章，9章）
④ マルチパラダイム（10章）
⑤ 構文的パラダイム（11章）

最初の構造化パラダイムは，あくまでも手続き型パラダイムの枠内で制御構造とデータ構造のわかりやすさ，モジュール構造のわかりやすさを追求したものである。宣言型パラダイムは，手続き型言語プログラムの内部変数がもたらす計算の副作用を避け，仕様記述をそのままプログラムとすることによりわかりやすさを実現するアプローチである。計算モデル的パラダイムは，フォン・ノイマン型アーキテクチャの演算装置と記憶装置からなる計算モデルにとらわれず，新しい計算モデルを導入して，それに基づいたプログラミング方法を提案するものである。マルチパラダイムは，単一パラダイムの美しさよりも，むしろ性質の異なる複数のパラダイムを融合することにより，複雑なシステムのわかりやすさの向上を狙うものである。以上の①から④のカテゴリのパラダイ

ムについては次の第2編で説明する

最後の構文的パラダイムは，手続き型か非手続き型かは問わず，構文的観点からわかりやすさを追求するものである。これは，パラダイムとしては少し弱いが，わかりやすさの観点でインパクトが大きいので本書ではプログラミングパラダイムの1つとして扱う。この詳細は第3編のエンドユーザコンピューティングの中で述べる。