Ⅳ プログラム設計

到達目標

（1）プログラム構造化設計の手順を理解させる。

（2）プログラム仕様書を作成できるようにする。

（3）妥当なモジュール分割を行えるようにする。

（4）ドキュメント化手法を理解させる。

1 プログラム構造化設計

（1）プログラム設計の重要性

プログラム設計は、ウォータフォールモデルによるシステム開発工程の中で、内部設計の次の工程に位置する。

この工程では、まず、内部仕様書に基づき開発したいシステムの機能をモジュールという単位に分割する。次に、それらのモジュールごとの位置関係やモジュールの呼出し手順などを設計する。また、各モジュールの機能をドキュメント化手法で設計する。一方、システムで扱うデータの構造についても設計する。そして、これらの内容をプログラム仕様書という形で文書化するのである。

ところで、プログラム設計の手法はただ一つではない。従来、ソフトウェアの生産性が低いとされてきたのは、プログラム設計において絶対こうしなけらばならないという手法が確立されていなかったからである。ソフトウェアの場合、ハードウェアとは異なり、プログラム設計がいいかげんでも、作成したプログラムがある程度動作してしまうという特徴がある。

第一に、総ステップ数が数百行という、きわめて小規模のシステムを一人で開発するという恵まれた場合を考えてみる。この場合、システムの設計者とプログラマが同じであり、そのため、システム開発に関する考え方のくい違いというものは存在しない。しかも、ソフトウェアの量が少なくて済むために、充分なプログラム設計を行わなくとも、プログラムがうまく動作する可能性が高い。

第二に、一定程度の規模（ここでいう一定程度とは、現在市販されているアプリケーション、もしくは特定の企業で独自に開発され使用されているシステムの規模を示す。）をもつシステムを一人で開発するという場合を考えてみる。この場合も、システムの設計者とプログラマが同じであり、両者の考え方のくい違いというものは存在しない。そのため、一見、しっかりとしたプログラム設計を行わなくとも、プログラムがうまく動作すると思われがち

である。しかし、人間の記憶力には限界があり、開発作業を中断し再開する度にプログラムの全体像を理解し直すという余計な手間がかかる。この場合、第一で述べた小規模なシステム開発の場合よりも、開発スケジュールに遅延が生じることが多く、さらに、手戻りが発生する可能性が高い。例えば、有名なプログラミング言語であるＣ言語コンパイラの開発は、きわめて優秀なエンジニアが、一人で約３ヵ月かけて行ったと言われている。しかし、Ｃ言語の場合も、しっかりとしたシステム設計があったからこそ優秀な言語となり得たのである。

第三に、一定程度の規模を持つシステムを複数のメンバーで開発するという場合を考えてみる。この場合、システムの設計者とプログラマとで役割分担することが多い。これは、システムの設計者に要求されるスキルと、プログラマに要求されるスキルが異なるからである。また、システムの設計者も複数であることが多い。この場合、メンバーごとにシステムに対する考え方は食い違うものと考えなければならない。そのため、しっかりとしたプログラム設計を行わないと、分担して設計したモジュール間のインタフェースに矛盾が生じたり、システムに必要な機能が抜け落ちたりする事態が多発する。限界のある人間の記憶力をしっかりとしたドキュメントで補うことによって、開発スケジュールの遅延や手戻りの発生を防止することができる。

現在の産業界では、コンピュータの普及が驚くべき速度で進行している。そして、システムはますます高度化・大規模化し、もはや、たった一人のエンジニアで一定程度のシステム開発を行うことはまれである。もちろん、ソフトウェア開発支援ツールも発達し、以前よりもずっとソフトウェアの生産性は向上してきた。しかし、一定程度のシステム開発を行う場合は、やはりしっかりとしたプログラム設計が必要となる。

（2）プログラム構造化設計の採用

プログラム設計手法は一つではない。エンジニアが長年培った自己流の設計手法もあれば、ソフトウェアの再利用が容易に行えるため、きわめて生産性の高いオブジェクト指向設計という手法も存在する。

ここでは、プログラム構造化設計という手法に沿って考えていく。構造化設計手法は、1980 年代に全盛を迎えたプログラム設計手法である。構造化設計手法が登場するまでのシステム開発は、特に複数メンバーで開発する場合、低い生産性しか上げることができなかった。構造化設計手法は、その生産性を劇的に向上させる手法として賞賛され、多くの場面で採用されてきた。次節ではプログラム構造化設計の手法について述べる。

2　プログラム設計手順

（1）プログラム構造化設計の特徴

プログラム構造化設計は、一定程度の規模を持つシステムを複数のメンバーで開発すると

いった場合に有効な設計手法である。

この手法では、システムをモジュールという小さい単位に分割する。各モジュールは、それぞれがまとまった機能をもつ独立性の高いものとする。そして、各モジュールが階層構造を形作るように上下関係を決める。

（2）プログラム構造化設計手順

プログラム構造化設計は、表Ⅳ-1に示すような順序にしたがって行う。

表Ⅳ-1　プログラム構造化設計の順序

順序	内　　容
①	モジュールの機能分析
②	モジュール分割手法の決定
③	モジュール分割
④	モジュール構造図作成
⑤	共通データ構造設計
⑥	モジュール機能設計（インタフェース設計）
⑦	モジュール分割の妥当性の検討
⑧	モジュール詳細設計
⑨	レビュー（ウォークスルー）

イ　モジュールの機能分析

まず、システムが必要とする機能を列挙する。そして、それらの機能が分割できないか、統合できないかを検討する。もし、複数の機能で共通の部分が見つかったら、それは後で共通モジュールとして統合する。また、ある機能があまりにも大きい場合は、適当な大きさの複数個の機能に分割する。

例えば、以下のような機能は共通モジュールの候補となる。

(イ) 文字・数字の入力機能

(ロ) 文字・数字の表示機能

(ハ) ファイルの入出力機能

(ニ) エラーメッセージ表示機能

(ホ) エラー処理機能

（注）　新しいシステム開発環境では、これらの共通モジュールの候補となるような機能

をオペレーティングシステム自体が既に提供していることが多い。例えば、文字・数字の入力・表示機能は、ＵＮＩＸ上のＧＵＩ（グラフィカルユーザインタフェース）である Motif や MS−Windows などでは標準的に提供されている。また、エラー処理機能も、C++言語や ObjectPASCAL 言語などでは例外処理としてプログラム文法上で提供されている。

ハ　モジュール分割手法の決定

モジュールを分割するために最適な手法を決定する。現在では、様々なモジュール分割手法が提案されており、対象となるシステムの内容によって最適な手法を採用すればよい。モジュール分割の具体的な手法については、Ⅳの３で詳しく述べる。

ニ　モジュール分割

②で決定したモジュール分割手法にしたがってモジュール分割を行っていく。モジュール分割の最大の目的は、モジュールを適切なサイズにすることによってシステムの複雑さを減少させることである。一般に、簡単なシステムの場合でも、モジュールのサイズが大きいと全体を概観することが難しくなり、複雑に見えることが多い。サイズが大きいものでも、機能単位で細かく分割することによって理解しやすくなる。

適切なモジュールの大きさは、場合によって異なる。後に述べるモジュール機能の文書化技法にも依存するが、一つのモジュールが約２００～３００ステップになるように分割することが従来から行われてきた。モジュールがこれ以上のステップになった場合には、モジュールの中に複数の機能が含まれていないかをチェックすることを奨める。

（注）　システムを数多くのモジュールに分割すると、動作時に別のモジュールを呼び出す回数が増え、そのオーバーヘッドによりシステムの性能が低下することがある。これは、モジュールを呼び出すときの引数の受け渡しにスタックを使用しているため、そのオーバーヘッドを考慮したものである。確かに、リアルタイム性を要求される制御系のシステムでは、これを無視することはできない。しかし、コンピュータの性能は急速に改善されており、モジュール分割によるオーバーヘッドを考慮する必要性は低下していると考えられる。

ホ　モジュール構造図作成

モジュールを分割する場合は、それらのモジュールの呼び呼ばれ関係を決める。例えば、Ａ、Ｂ、Ｃという三つのモジュールが存在したときに、ＡはＢとＣを呼び、ＢはＡから呼ばれ、ＣもＡから呼ばれるといった関係である。

この呼び呼ばれ関係を、全てのモジュールに対して記述した図のことをモジュール構造図と言う。

モジュール構造図作成の具体的な内容については、Ⅳの４で詳しく述べる。

ヘ　共有データ構造設計

モジュールとモジュールとの間で、データのやりとりを行う方法は二つある。

第一は、モジュールを直接呼び出すときに、引数でデータのやりとりを行う方法である。この方法は、モジュール間の独立性を高めることができるため、プログラム構造化設計では最も多く使用される方法である。特に、リエントラント（再入可能）なソフトウェアを作成する場合には、必ず使用しなければならない方法である。

第二は、共有データ域を設け、複数のモジュールがそこを参照・変更することによってデータのやりとりを行う方法である。この方法は、データの変更を行ったのがどのモジュールであるかが明確にならず、モジュール間の従属性が強まるという問題がある。また、リエントラントなソフトウェアを作成する際には、データ参照のみしか行ってはならないので、一般には、なるべく使用しないこととされている方法である。しかし、この方法をどうしても使用しないと実現できない場合、あるいは、この方法を使用した方が効率がよい場合が存在する。例えば、初期処理で値を設定した後、二度と値を変更しないデータを全てのモジュールから参照する場合には、わざわざそのデータを引数でやりとりするのは冗長である。また、エラー処理で取得したエラーデータを格納するような場合は、共有データにした方が処理が単純化される。そこで、モジュール分割ができた時点で、共有データ構造の設計を行っておく。ここで設計したデータは、モジュール間では引数を使った受け渡しを行う必要がない。

ト　モジュール機能設計（インタフェース設計）

モジュールの呼び呼ばれ関係は、モジュール構造図で明確になった。また、モジュールごとの機能も明確になった。そこで、モジュール仕様書を作成する。後に、プログラマがこのモジュール仕様書をもとにコーディングを行うため、内容を正確に記述しなければならない。

モジュール機能設計の詳細については、Ⅳの５で詳しく述べる。

チ　モジュール分割の妥当性の検討

ここまでのプログラム設計を行ってきたところで、既に行ったモジュール分割が果たして妥当であったかどうかを再度検討する。そして、必要があれば、モジュールの再分割やモジュールの再統合を行う。

モジュール分割の妥当性の検討については、Ⅳの６で詳しく述べる。

リ　モジュール詳細設計

モジュール構造図およびモジュール仕様書をもとに、モジュールの詳細な実現方式を設計する。この際、後に述べるドキュメント化手法に基づいて記述する。フローチャート、

ＨＩＰＯ、ＨＣＰ、ＰＡＤ、ＤＦＤなど幾つかの手法が提案されているので、システムに最も適合した手法を採用する。

モジュール詳細設計については、Ⅳの７で詳しく述べる。

ヌ　レビュー（ウォークスルー）

プログラム構造化設計の最終段階としてレビューを行う。

レビューは、既に行ったモジュール詳細設計において、実現方式の中に紛れ込んでいるバグをコーディングに入る前に発見し取り除くことを目的として実施する。

ウォークスルーはレビューの手法の一つであり、プログラム設計の終了時点で複数のメンバーが行うレビュー会のことである。レビュー会は、プログラム設計に関わったメンバーだけでなく、別の開発プロジェクトのメンバーも加えて実施するのが効果的である。それは、設計に関わったメンバーの間では当然であると思っていたことでも、それ以外のメンバーの違った視点からの指摘で、問題が浮き彫りになることが多いからである。

（3）プログラム仕様書作成

プログラム構造化設計を行った成果は、プログラム仕様書（図Ⅳ-１）として文書化する。プログラム仕様書は、以下の七つの内容から構成する。

イ　表紙

プログラム仕様書の先頭に置く表紙である。システムの名称、プログラム設計者名を記述する。また、プログラム仕様書を改訂した際の変更年月日、変更内容を記入する欄を設けておくと、改訂履歴として使用することができる。

ロ　概要

システムの概要について記述する。

ハ　ドキュメント化技法

ドキュメント化の技法名を記述する。

ニ　モジュール構造図

モジュール構造図を記述する。最上位モジュール、第一階層モジュール、第二階層モジュールと、モジュール番号の順番に記述し、最後に共通モジュールを記述する。

ホ　モジュール仕様書

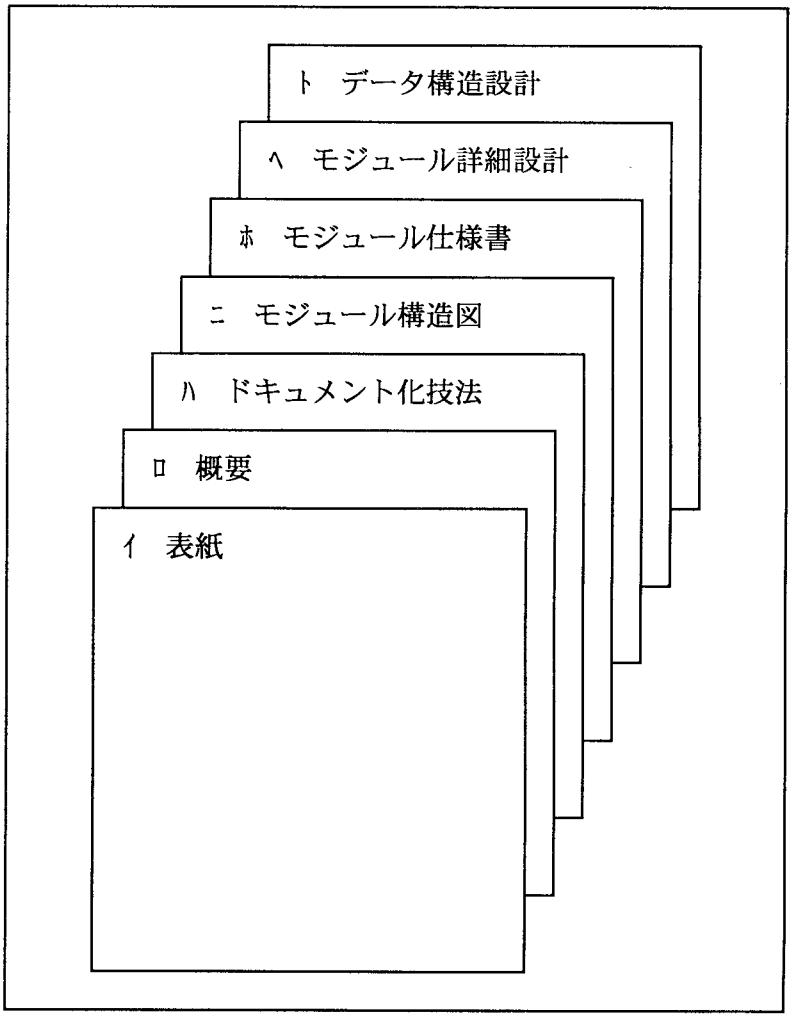
各モジュールごとのモジュール仕様を記述する。

ヘ　モジュール詳細設計

採用したドキュメント化技法を使用して、モジュールごとの詳細な実現方式を記述する。

ト　データ構造設計

共有データ構造について記述する。



図Ⅳ-1　プログラム仕様書の例

3　モジュール分割

（1）モジュール分割の方針

モジュール分割は、プログラム構造化設計にとって重要な手順である。分割を行う際に目標とすべき点は、以下の三点である。

イ　適切な量の分割

既に述べたように、一つのモジュールの大きさとして適切なのは２００～３００ステップ

であり、適切な量になるようにモジュール分割すべきである。このステップ数は、一つのモジュールを概観するのに最も適した量であるといえる。

ロ　独立性の高い分割

モジュール間の独立性がなるべく高くなるように、モジュール分割すべきである。分割の方法がよくないと、かえって分かりにくいプログラムになる可能性がある。モジュール間の関係がなるべく単純になるように分割すべきである。

ハ　階層構造を持つ分割

モジュールの呼び呼ばれ関係が、ピラミッド型の適切な階層構造になるようにモジュール分割を行うべきである。一般に、モジュール構造図の階層の深さは、２～５の範囲に収まるのが望ましい。また、上位モジュールが呼び出す下位モジュールの数が１～１０の範囲に収まるのが望ましい。

（２）モジュール分割手法

適切なモジュール分割を行うために、幾つかの手法が提案されている。ここでは、代表的な五つのモジュール分割手法について述べる。

イ　ジャクソン法

入出力データの構造に着目した分割手法の一つである。1970 年代前半に、M.ジャクソンによって提唱された手法であり、データ構造を分析することによって、プログラム構造が導出できるという考え方である。

ジャクソン法は、データ構造もプログラム構造も、「基本」、「連続」、「繰返し」、「選択」という基本的な四つの型で記述することができるという考え方に基づいている。そして、これらの型は図式論理という方法で記述される。

図Ⅳ-２に、ジャクソン法における四つの基本型を図式論理で示す。

(イ) 基本

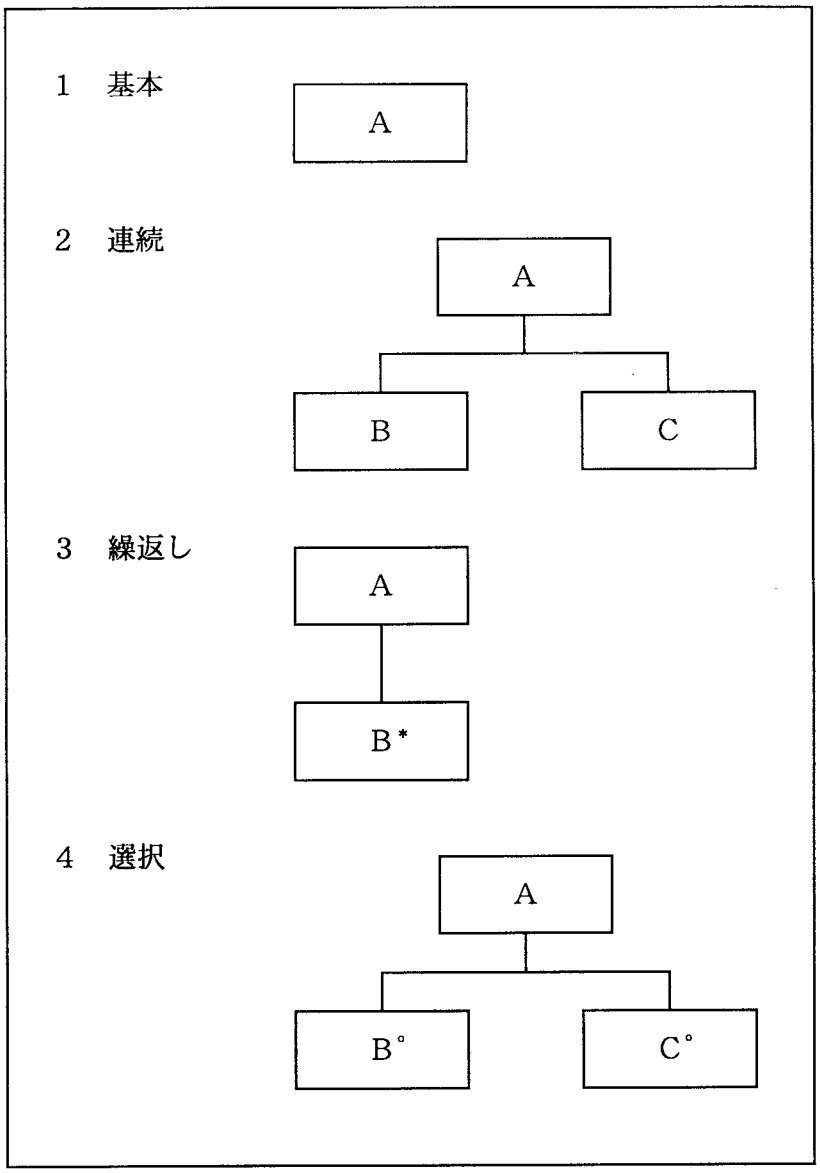
基本型は、図式論理での記述の核となる構成要素である。データ構造では一つのデータ項目、プログラム構造では一つの実行文に相当する。

図Ⅳ-２の１で、Ａとしてあらわれる。

(ロ) 連続

連続型は、複数の要素が、決まった順番に１回だけあらわれる基本型である。

図Ⅳ-２の２.で、ＡはＢとＣから構成され、Ｂ、Ｃの順番で一回ずつ連続してあらわれる。



図Ⅳ-2　ジャクソン法における構造の基本型

(ハ) 繰返し

繰返し型は、一つの要素が何回か繰返される基本型である。

図Ⅳ-2の3で、AはBが0回以上繰返してあらわれる。

(ニ) 選択

選択型は、複数の要素の中から、一時点でその中の一つだけが選択される基本型である。

図Ⅳ-2の4 で、AはBとCから構成され、BとCのいずれか一方が選択されてあらわれる。

ジャクソン法では、最初に対象となる処理の入出力データを分析し、そこから入力データ構造図と出力データ構造図を作成する。

次に、入力データ構造図と出力データ構造図から、入力と出力が１：１に対応する関係を見いだす。そして、これをもとにプログラム構造図を作成するのである。

実際のシステムでは、入出力が１：１に対応する関係を見いだせないことが多い。そのような場合は、入力から出力まで過程を詳しく検討し、必要ならば中間のデータ構造を作成し、１：１の対応を見つけださなければならないというところが、ジャクソン法の欠点である。

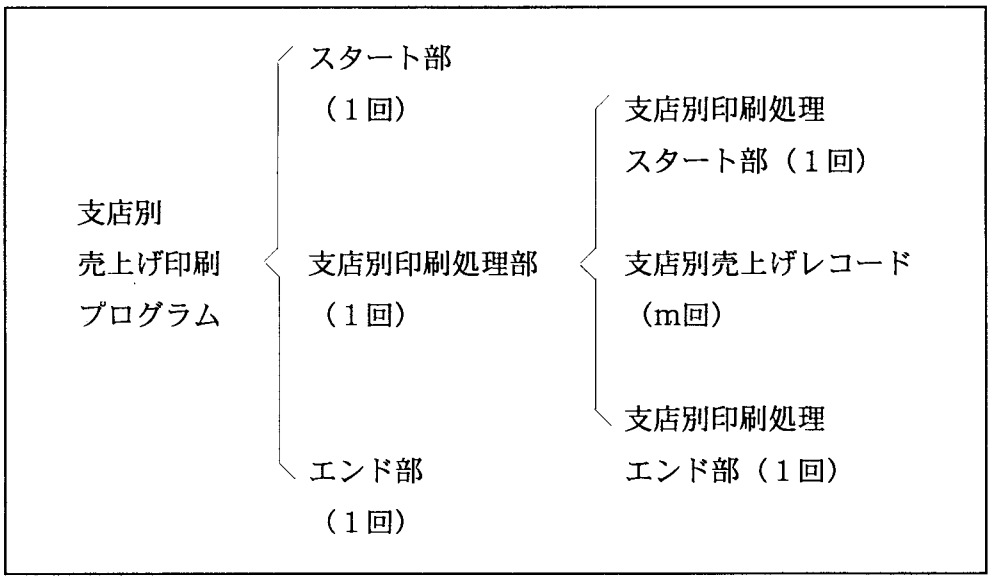
□　ワーニエ法

ジャクソン法と同じく、入出力データの構造に着目した分割手法の一つである。1970年代の初めに、Ｊ．Ｄ．ワーニエによって提唱された手法であり、システム設計とプログラム構造の作成に集合論を適用するという特徴をもつ。

ワーニエ法では、以下の三段階を経て最終段階に達する。

第一に、集合論に基づいてプログラムの論理を分解する。対象となるプログラムを一つの集合として取り扱い、複数の部分集合に分解する。そして、これをワーニエ図という形で記述する。

図Ⅳ-３にワーニエ図の例を示す。



図Ⅳ-３　ワーニエ図の例

第二に、データ構造の分析を行う。この場合も、対象となるファイルを一つの集合として取扱い、複数の部分集合に分解する。そして、これもワーニエ図で記述する。

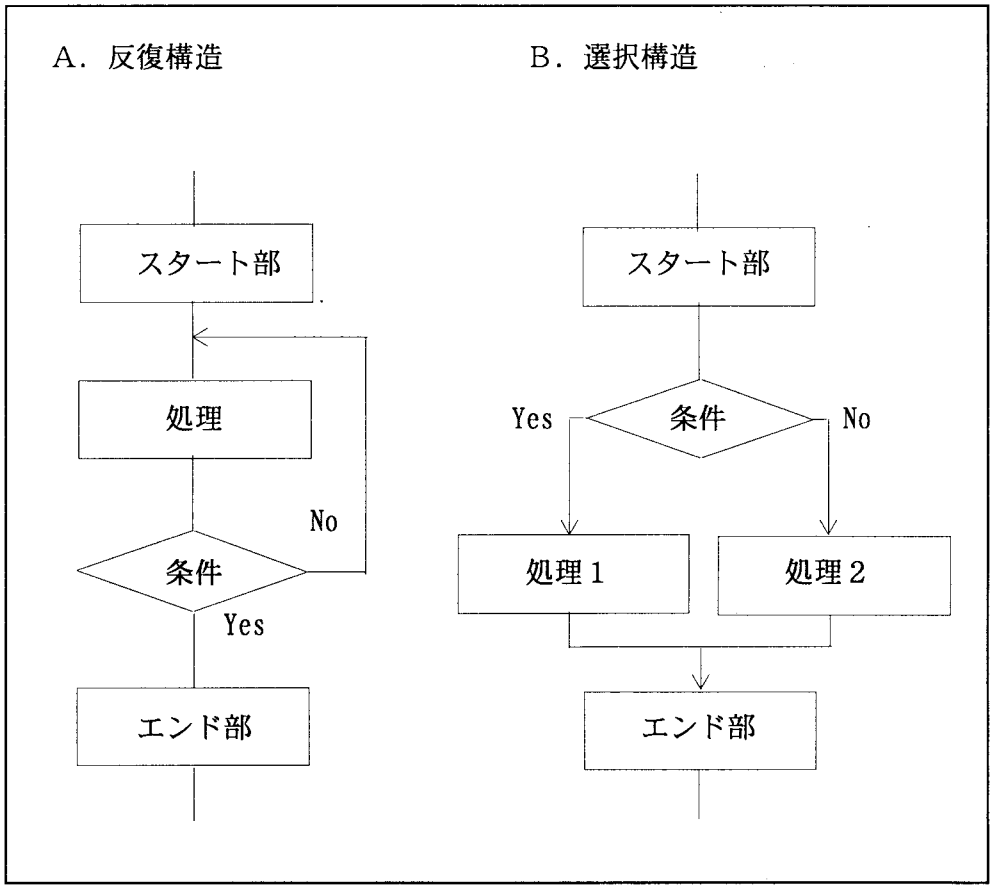
第三に、データ構造からプログラム構造への変換を行う。その場合、

(イ) 繰り返し構造

(ロ) 選択構造

という二つの制御構造でプログラム流れ図を作成する。

図Ⅳ-４に、ワーニエ法における流れ図の基本構造を示す。



図Ⅳ-４　ワーニエ法における流れ図の基本構造

ワーニエ法では、ワーニエ図を作ったからといってすぐにコーディングを始めることはできず、一旦、ワーニエ法における流れ図に変換してから、コーディングしなければならない。

また、流れ図の基本構造は、反復構造と選択構造という２種類しか存在しない。そのため、プログラムの記述力に多少問題がある。

ハ　ＳＴＳ分割法

データの流れに着目した分割手法の一つである。1970 年代前半に G.マイヤーによって提唱された手法であり、データの流れを分析することによってプログラム構造が導出でき

るという考え方である。

ＳＴＳとは、Source ／ Transform ／ Sink の頭文字をとったものである。ＳＴＳ分割法は、複合設計と呼ばれることもある。

ＳＴＳ分割法では、プログラムを以下の三つの従属部分に分割し、それぞれをモジュールと定義して記述する。

(イ) 源泉（Source）
　入力データを処理する部分
(ロ) 変換（Transform）
　入力データを出力データに変換する部分
(ハ) 吸収（Sink）
　出力データを処理する部分

ＳＴＳ分割法では、以下の手順でモジュール分割を行う。

まず、第一に、データの流れに沿って処理機能を分析し、幾つかの機能をまとめて構造というものを考える。

第二に、データの流れを分析し明確化する。

第三に、データの流れの中で、入力データが明らかにその形を変え、もはや入力データといえなくなる点を探す。この点は、最大抽象入力点と呼ばれる。

第四に、データの流れの中で、最大抽象入力点以降、データが明らかに出力データにその形を変える点を探す。この点は、最大抽象出力点と呼ばれる。

第五に、プログラムの構造を、源泉(S)、変換(T)、吸収(S)の三つに分ける。

第六に、モジュール間の独立性が高まるようにモジュール分割を行う。

ところで、ＳＴＳ分割法は、Ⅳの４で述べるモジュール構造図の作成における最上位モジュールと第一階層モジュールまでには有効であるが、それ以下については対応していない。そこで、ＳＴＳ分割法と他の分割法を併用することが必要となる。

ニ　共通機能分割法

モジュール分割を行う上で、同じような機能のモジュールが二ヵ所以上現れる場合がある。共通機能分割法は、それらの共通な機能を一つのモジュールにまとめ、抽出する手法である。このモジュールを共通モジュールと呼ぶ。

共通モジュールは、プログラム構造化設計の途中で抽出することができる。また、共通モジュールだけを集め、それに一般のモジュールと区別できる番号や記号をモジュール番号として割り当てることが行われる。これによって、一般の階層モジュールと明確に区別することができる。

ホ　トランザクション分割法

データの内容に着目した分割手法の一つである。入力トランザクションに応じて、異なる処理を行う場合に有効な分割法である。

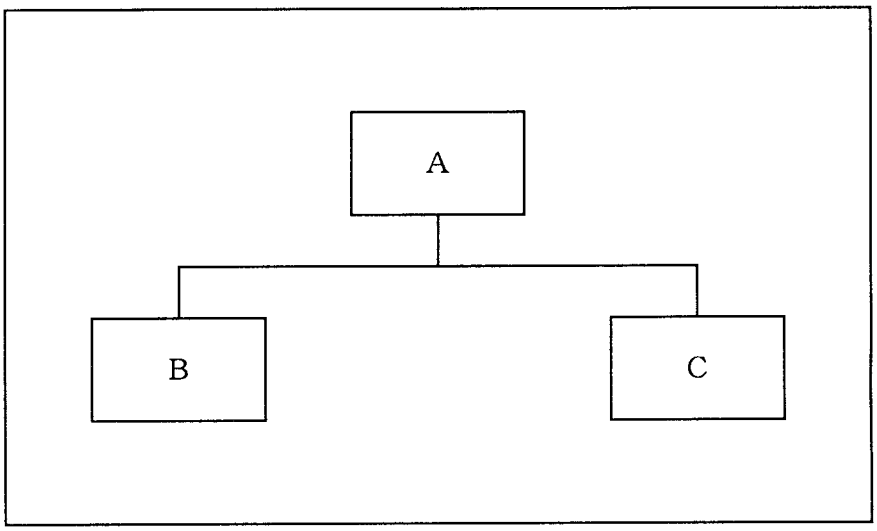
トランザクション分割法は、ＴＲ分割法と呼ばれることもある。

一般に、トランザクション処理においては、入力トランザクションの種類によって処理内容が異なる。そこで、入力トランザクションに１：１で対応する形で処理モジュールを作成する。これによって、各モジュール間の独立性が高まることになる。

（注）　新しいシステム開発環境では、入力メッセージに応じて異なる処理を行うようになっていることが多い。これは、メッセージ駆動と呼ばれる考え方である。入力メッセージに１：１で対応する形で処理モジュール（コールバックルーチン）を作成する。このように、トランザクション処理とメッセージ駆動は、きわめて類似している。

４　モジュール構造図作成

モジュールを分割する場合は、それらのモジュールの呼び呼ばれ関係を決める。例えば、Ａ、Ｂ、Ｃという三つのモジュールがあったとき、ＡはＢとＣを呼び、ＢはＡから呼ばれ、ＣもＡから呼ばれるといった関係である。この例を図Ⅳ-５に示す。



図Ⅳ-５　モジュールの呼び呼ばれ関係の例

この呼び呼ばれ関係を、全てのモジュールに対して記述した図のことをモジュール構造図という。

モジュール構造図は、各モジュールの上下関係を示していることになる。各モジュールは、ピラミッドのような三角形の階層構造を持つ。一般に、モジュール構成図は、一枚で記述するのではなく複数枚で記述する。そして、これら複数枚を関連づけるために、モジュール構造図の各モジュールにはモジュール番号が割り当てられる。

以下、モジュールをその階層構造内の位置で分類して説明する。

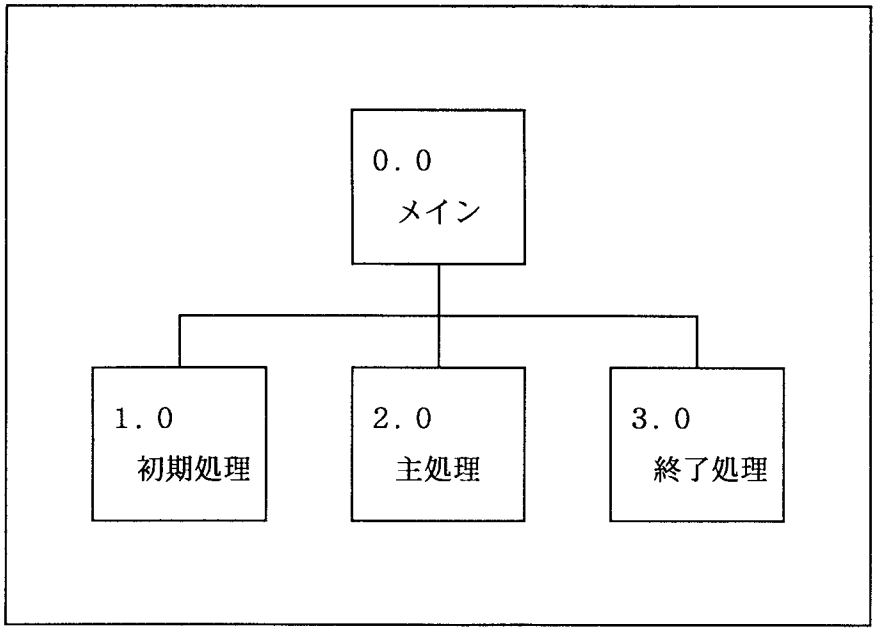
（1）最上位モジュール

まず、階層構造の最も上に位置するモジュールには、0.0という番号を割り当てる。これは、システムの入り口のモジュールとなる特別なモジュールであり、最上位モジュール、あるいはメインモジュールと呼ばれている。最上位モジュールは、システム全体でただ一つしか存在しない。

（2）第1階層モジュール

次に、階層構造で最上位モジュールのすぐ下に位置するモジュール群には、1から順番に整数のモジュール番号を割り当てる。これらは、第1階層のモジュールと呼ばれている。

一般に、第1階層のモジュールは、初期処理、主処理、終了処理という三つのモジュールに分け、それぞれ1.0、2.0、3.0というモジュール番号を割り当てることが行われる。これは、プログラムの本体となる主処理の前後には、必ず初期処理、終了処理が必要であるという考え方に由来している。この例を図Ⅳ-6に示す。

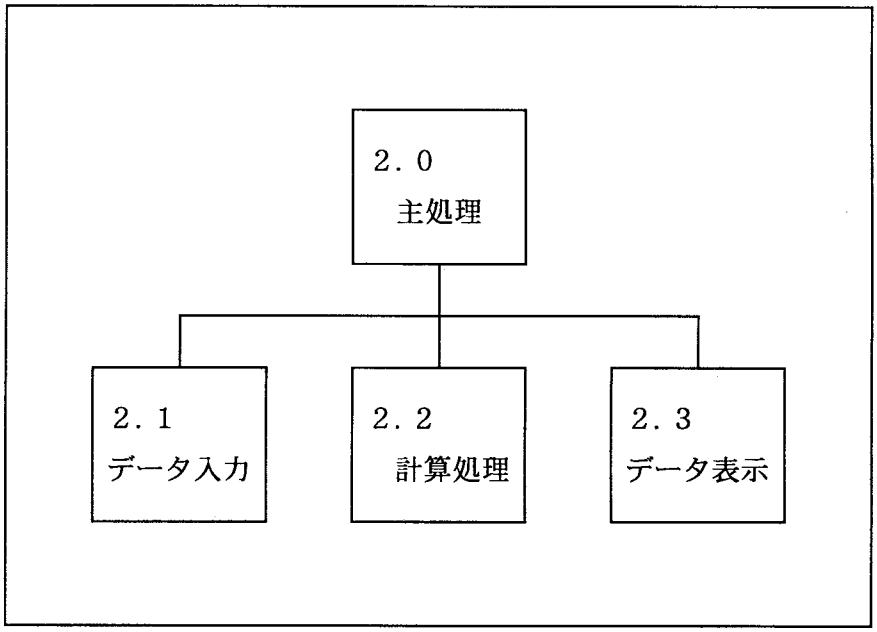


図Ⅳ-6　第1階層のモジュールの例

（3）第2階層モジュール

次に、階層構造で第1階層モジュールのすぐ下に位置するモジュール群には、第1階層モジュールに与えられた整数のモジュール番号の後に「.」を付け、その後に1からの通し番号を付ける。

例えば、第1階層モジュールの主処理（モジュール番号2.0）の下に、データ入力、計算処理、データ表示という三つのモジュールが存在する場合は、それぞれのモジュールに2.1、2.2、2.3という番号を割り当てる。この例を図Ⅳ-7に示す。



図Ⅳ-7　第2階層のモジュールの例

（4）第n階層モジュール（n＞＝3）

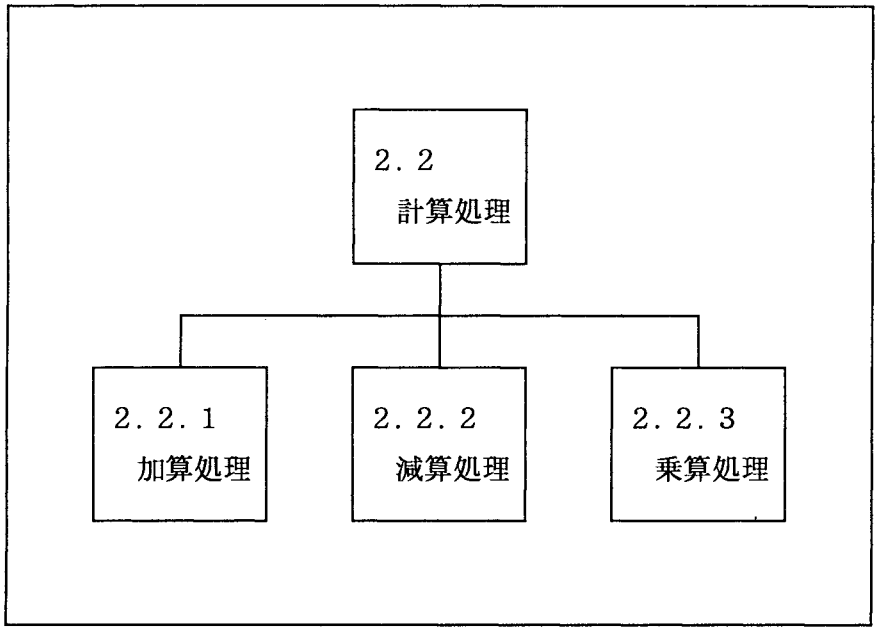
次に、階層構造で第 n−1 階層モジュールのすぐ下に位置するモジュール群には、第 n 階層モジュールに与えられた整数のモジュール番号の後に「.」を付け、その後に1からの通し番号を付ける。

例えば、第2階層モジュールの計算処理（モジュール番号2.2）の下に、加算処理、減算処理、乗算処理という三つのモジュールが存在する場合は、それぞれのモジュールに、2.2.1、2.2.2、2.2.3という番号を割り当てる。この例を図Ⅳ-8に示す。

（5）共通モジュール

階層構造の中で、全く同じ機能を持つモジュールが二ヵ所以上現れる場合がある。このような場合は、それらを一つのモジュールにまとめる。これを共通モジュールと呼ぶ。

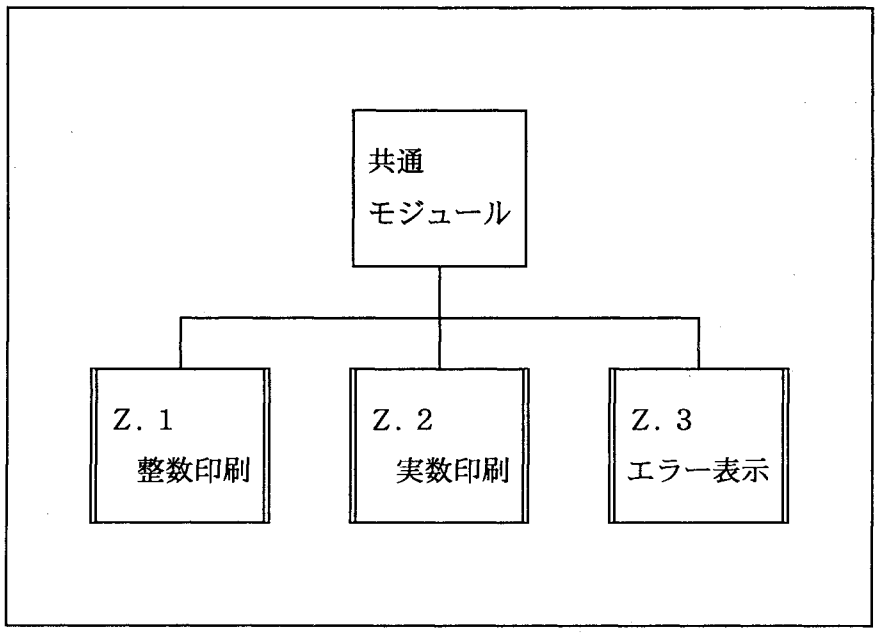
共通モジュールは、階層構造の特定の場所にはおさまらない。そこで、共通モジュールだ



図Ⅳ-８　第３階層のモジュールの例

けを集め、それに一般の階層モジュールと区別できる番号（記号）を付加する。例えば、Ｚ．１、Ｚ．２、Ｚ．３などといった番号（記号）を割り当てることによって、一般の階層モジュールと明確に区別することができる。

また、モジュール構成図上で、一般の階層モジュールと共通モジュールを区別するために、モジュールを囲む枠を変えるなどの方法をとることが必要である。共通モジュールの例を図Ⅳ-９に示す。



図Ⅳ-９　第３階層のモジュールの例

理論的には、モジュール構造図において、階層の深さも、一つのモジュールから呼び出すモジュールの数も無限に増やすことができる。しかし、一般的には、階層の深さは２～５、一つのモジュールから呼び出すモジュールの数は１～１０という範囲で設計されていることが多い。

５　モジュール機能設計

モジュール機能設計では、モジュール間のインタフェースを明確にすることが最大の目的となる。その結果はモジュール仕様書という形で文書化する。

モジュール仕様書では以下の７点を明確に記述する。

（１）モジュール番号

モジュール分割のときに割り当てた番号を記述する。

（２）モジュール名

モジュールをシステム全体で一意に識別することのできる名前を記述する。一般には、コーディング規約において、名前が重ならないような規則を定めておく。

（３）機能

モジュールの持つ機能を１行で簡潔に記述する。

（４）呼出しインタフェース

モジュールを呼び出すときの引数の順序やデータ型、あるいは戻り値のデータ型といったインタフェースに関することを記述する。他のモジュールから呼び出されるときに必須の情報である。

（５）引数

引数には、モジュールの呼び出し前に上位モジュールで設定し下位モジュールに伝達しなければならない入力引数、モジュールの処理結果が反映され呼び出し後に上位モジュールへ伝達される出力引数、そして、入出力の両方の機能を併せもつ入出力引数の３種類がある。

引数の部分には、引数名、データ型、データ長、入力／出力／入出力の区別、引数の内容を記述する。

（６）モジュール処理概要

モジュールを呼び出すと主に実行される処理の概要を記述する。内部的に行われる処理手順を記述するのではなく、上位モジュールに対して何をしてくれるかを記述する。

（7）特記事項

このモジュールを呼び出すと、副作用が発生するといったコーディング上留意すべき事項について記述する。

表Ⅳ-２にモジュール仕様書の例を示す。

表Ⅳ-２　モジュール記述書の例

ﾓｼﾞｭｰﾙ番号	２．２	ﾓｼﾞｭｰﾙ名	ｘＣａｌｃ
機能	加算、減算、乗算を行う。		
呼出しｲﾝﾀﾌｪｰｽ	ｘＣａｌｃ（type, data1, data2, result）；		

	引数名	ﾃﾞｰﾀ型	長さ	入出力	引数の内容
引数	type	char	1	入力	計算の種類 １：加算　２：減算　３：乗算
	data1	int	4	入力	１番目のデータ
	data2	int	4	入力	２番目のデータ
	result	int	4	出力	計算結果

モジュール処理概要	引数で指定された計算の種類にしたがって、１番目のデータと２番目のデータとの間で計算を行う。 その結果は、引数の計算結果を用いて上位モジュールに伝える。
特記事項	計算の種類に１～３以外の数字を設定すると、計算結果は不正となる。

6　モジュール分割の妥当性の検討

最初に行ったモジュール分割が、果たして妥当であったかどうかを再度検討する。そして、必要があれば、モジュールの再分割やモジュールの再統合を行う。

再検討の判断の基準は、以下の５点である。

（１）モジュールのステップ数が大きい。

もし、一つのモジュールが３００ステップを超えると予想されるときは、そのモジュールは大きすぎると判断し、さらに小さいモジュールに分割できないかどうかを検討する。

その理由は、大きいモジュールを作ってしまうと、モジュール全体を見渡すことが困難になり、バグが紛れ込む可能性が高くなるからである。そこで、大きいモジュールの機能を再検討し、その中に二つ以上の機能が含まれているようであれば、さらに小さいモジュールに分割する。

（２）モジュール内の機能が多すぎる。

もし、一つのモジュールの中に二つ以上の機能のまとまりがあることが判明したときは、分割によってモジュールがあまり小さくならない範囲で、さらに小さいモジュールに分割できないかどうかを検討する。

その理由は、モジュールの中の機能が多すぎると、モジュールの見通しが悪くなり、バグが紛れ込む可能性が高くなるからである。また、保守性も低下する。

（３）共通化できる処理がある。

もし、複数のモジュール間、あるいは一つのモジュールの中で共通の処理が存在する場合は、共通モジュールにできないかどうかを検討する。

共通モジュールにすることができれば、後にプログラムに修正を加える場合も、共通の一ヵ所だけ修正すればよいことになり、生産性が向上する。

（４）モジュール構造図の階層が深い。

モジュール構造図の階層の深さが、２～５の範囲に収まっているかを検討してみる。もし、これよりも階層が深い場合には、その部分のモジュールの再統合を考える。

（５）一つのモジュールから呼び出すモジュールの数が多すぎる。

モジュール構造図の上で、上位モジュールが呼び出す下位モジュールの数が、１～１０の範囲に収まっているかを検討してみる。もし、これよりも下位モジュール数が多い場合は、下位モジュール同士を再統合できないか、あるいは上位モジュールを複数のモジュールに分割できないかどうかを検討する。

7　モジュール詳細設計

（1）モジュール設計の目的

モジュール構造図およびモジュール仕様書をもとに、モジュールの詳細な実現方式を設計する。これをモジュール設計と呼ぶ。

以下にモジュール設計の目的について述べる。

イ　論理の構造化

モジュール設計を行うときは、論理の構造化を念頭に置いて行う。その理由は、プログラムの複雑さを減らし、分かりやすくするためである。

構造化プログラミング以前のプログラミングは、goto 文を多用し、プログラムのあちこちに制御が移動するため、プログラムが非常に複雑になってしまう傾向があった。このようなプログラミングの方法は、論理がスパゲティーのように複雑にからまりもつれあうことから、スパゲティープログラミングと呼ばれていた。

特に、BASIC言語で作成されたプログラムの場合、言語の仕様上、論理の構造化が困難であったために、生産性・保守性の低いBASICプログラムが氾濫する結果となった。

また、フローチャートといったドキュメント化手法では、矢印を引っぱるだけで容易にプログラムのあちこちに制御を移動させることができる。そのため、生産性を高めるために採用されたはずのドキュメント化手法が、goto 文を多用するプログラムを支え、生産性の足を引っぱるという皮肉な結果を招いていた。

これに対して、論理の構造化を行うことによって、生産性・保守性を高めることができる。

ロ　トップダウンの考え方

論理を構造化するときに複雑さを排除するためには、トップダウンの考え方を用いることが有効である。

構造化プログラミング以前のプログラミングは、システムのどの部分から始めてもよかった。そのため、いきなり詳細部のコーディングを始めることもしばしばであった。これはボトムアップの考え方に相当する。

トップダウンの考え方では、まず全体の論理を組立て、次第にそれを詳細化していくという手順をとる。すなわち、段階的詳細化を行うのである。これによって、一度に考えなければならない範囲を狭めることができ、複雑さを排除した形で論理の組立てができるのである。

ハ　構造化定理

構造化定理は、ボエムとジャコピニが提唱した次のような定理である。

> 構造化定理：
> 一対の入口と出口を持ち、無限ループもなく、どのような条件下でも実行されないような命令が存在しないといった適正プログラムは、三つの基本制御構造である「順次」、「選択」、「繰返し」だけで、その論理を記述することができる。

この定理に基づく構造化プログラミング手法は、1980 年代に、ソフトウェアの生産性と保守性の低さを嘆いていたプログラマに熱狂的に受け入れられ、世界中に普及することになった。

また、Ⅳの８で述べるドキュメント手法のうち、構造化プログラミングに適しているといわれる手法の中には、ＨＣＰのように goto 文を記述できなくしたものも登場した。これは、ドキュメント化手法の立場から、論理の構造化を支援するものであった。

しかし、それに伴い、『構造化プログラミングにおいては、三つの基本制御構造である「順次」、「選択」、「繰返し」だけで論理を組み立てなければならず、goto 文を絶対に使ってはならない』というという誤解も生じ、ループからの脱出やエラー処理に goto 文を一切使わないプログラミングを実践したプログラマも数多くいた。

しかし、goto 文を全く使わない場合、ループからの脱出にフラグを使用しないといけないなど、かえってプログラムを複雑で分かりにくくする結果になることもあった。

ここで大切なことは、構造化プログラミングにおいて、むやみに goto 文を使ってはいけないが、goto 文を使った方がプログラムがむしろ簡潔で分かりやすくなるときは、goto 文を使ってもよい。

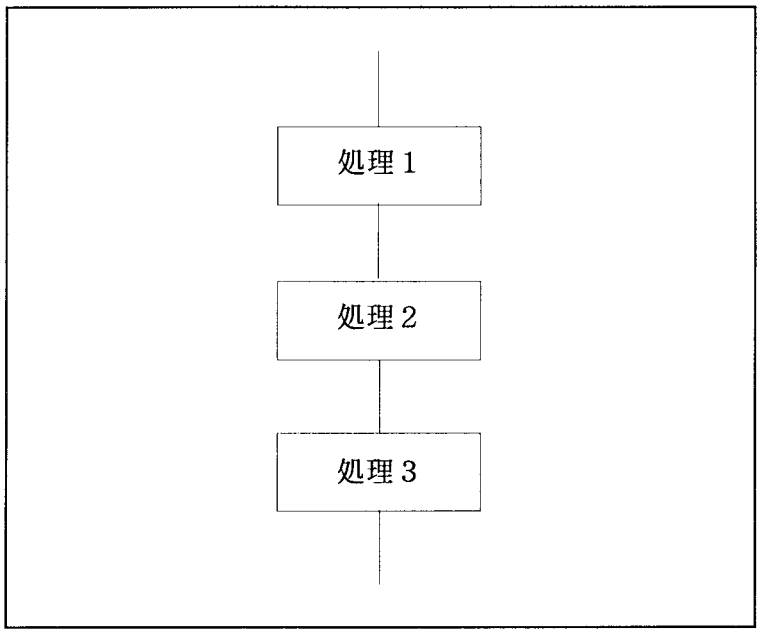
（２）基本制御構造

構造化定理に使われる三つの基本制御構造について述べる。

イ　順次型（ＳＥＱＵＥＮＣＥ型）

順次型は、論理が上から下へ順番に実行される文だけで構成される論理構造である。goto 文などで下から上へ実行順序が変更された場合は、順次型とは呼ばない。

図Ⅳ-10に順次型の論理構造を示す。ここでは、処理１、処理２、論理３が順番に実行される。

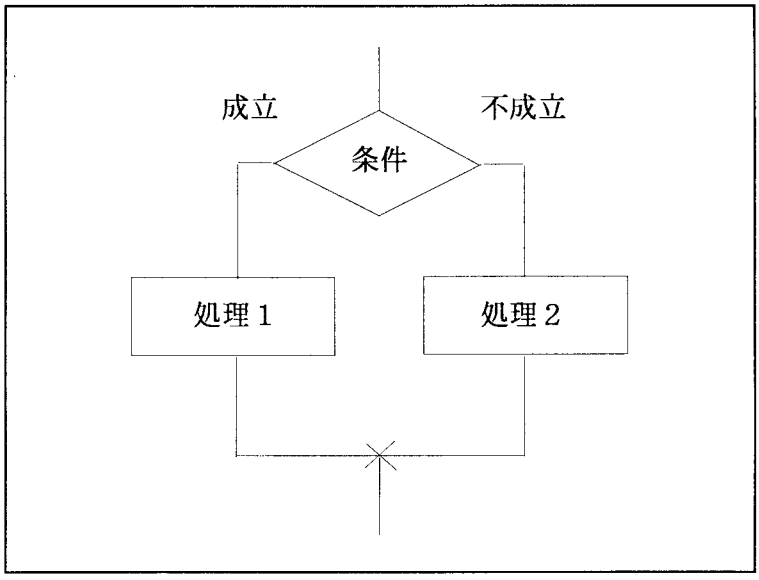


図Ⅳ-10　順次型（ＳＥＱＵＥＮＣＥ型）

□　選択型（ＩＦ　ＴＨＥＮ　ＥＬＳＥ型）

選択型は、条件が成立するか不成立かによって実行される文が異なるとき、すなわち、条件分岐に使用される論理構造である。

図Ⅳ-12に選択型の論理構造を示す。ここでは、条件が成立する場合は処理１、成立しない場合は処理２が実行される。なお、処理１が実行されたときは処理２は実行されない。また、処理２が実行されたときは処理１は実行されない。また、処理１、処理２の内容が空の場合もある。

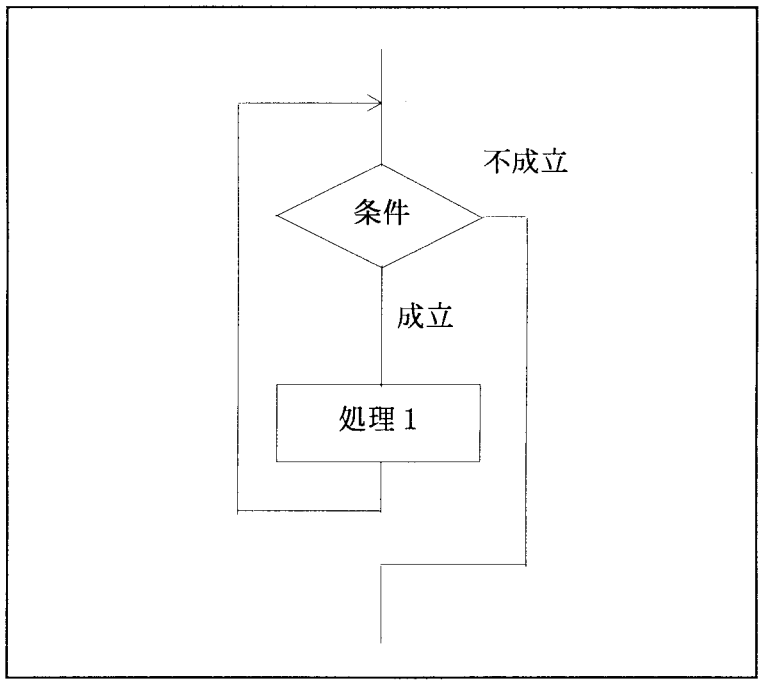


図Ⅳ-11　選択型（ＩＦ　ＴＨＥＮ　ＥＬＳＥ型）

ハ　繰返し型（ＤＯ　ＷＨＩＬＥ型）

繰返し型は、ある条件が判断され、その条件が成立すると、指定された処理が実行され再び条件判断に戻るという論理構造である。

図Ⅳ-12に繰返し型の論理構造を示す。ここで、条件が判断され、成立している間は処理１が繰り返され、条件が不成立になると次の処理へ進む。なお、最初から条件が不成立の場合は、処理１は１回も実行されないまま次の処理へ進む。また、処理１の内容が空の場合もある。



図Ⅳ-12　繰返し型（ＤＯ　ＷＨＩＬＥ型）

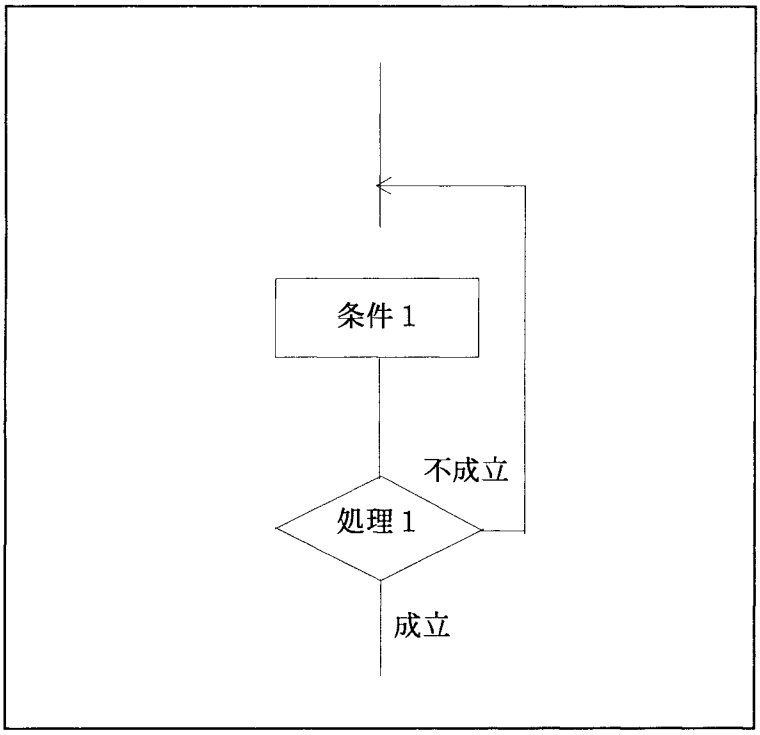
（３）応用制御構造

実際の論理設計では、構造化定理に使われる三つの基本制御構造以外に、さらに二つの制御構造を追加して使用することが多い。これは応用制御構造と呼ばれる。

イ　繰返し型（ＤＯ　ＵＮＴＩＬ型）

繰返し型は、ある条件が不成立の間、指定された処理を繰り返すという論理構造である。

図Ⅳ-13に繰返し型の論理構造を示す。ここでは、条件が判断され、不成立の間は処理１が繰り返され、条件が成立すると次の処理へ進む。なお、最初から条件が成立している場合でも、必ず１回は処理１が実行されるという特徴がある。また、処理１の内容が空の場合もある。

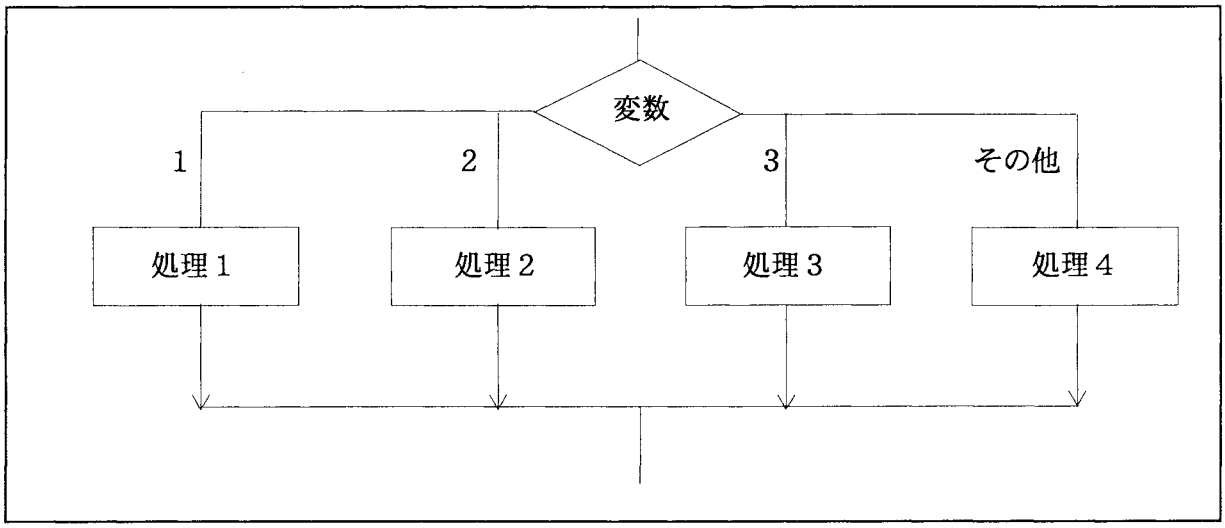


図Ⅳ-13　繰返し型（ＤＯ　ＵＮＴＩＬ型）

□　多岐選択型（ＣＡＳＥ型）

多岐選択型は、変数の値によって三つ以上の選択肢のいずれかが選択され、処理が実行されるという論理構造である。

図Ⅳ-14に多岐選択型の論理構造を示す。ここでは、例えば変数の値が１なら処理１を、変数の値が２なら処理２を、変数の値が３なら処理３を、変数の値が１、２、３以外なら処理４を実行する。



図Ⅳ-14　多岐選択型（ＣＡＳＥ型）

8　ドキュメント化技法

（1）概要

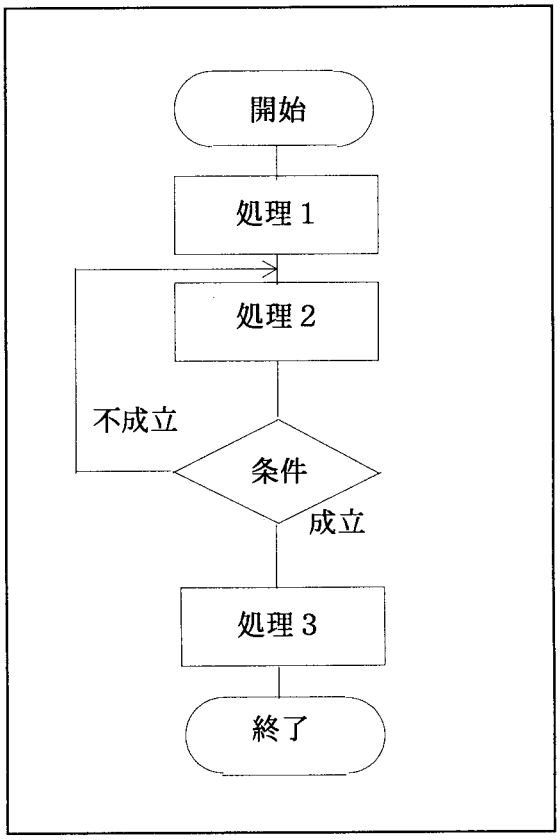
プログラム設計を行い、プログラム仕様書を作成するときには、文章だけを使用するのではなく、図や表を多用した方がより理解しやすくなる。

プログラム仕様書を作成する際に使用するドキュメント化技法は数種あり、それぞれ特徴があるので、開発するシステムに最も適合した技法を採用する必要がある。

（2）フローチャート

フローチャートは、流れ図とも呼ばれるドキュメント化技法である。処理、条件、開始、終了といった要素ごとにシンボルが決められており、それらを制御の流れを示す矢印で結んで完成させる。フローチャートでは、制御の流れは記述できるが、データの流れは記述できない。

図Ⅳ-15にフローチャートの記述例を示す。

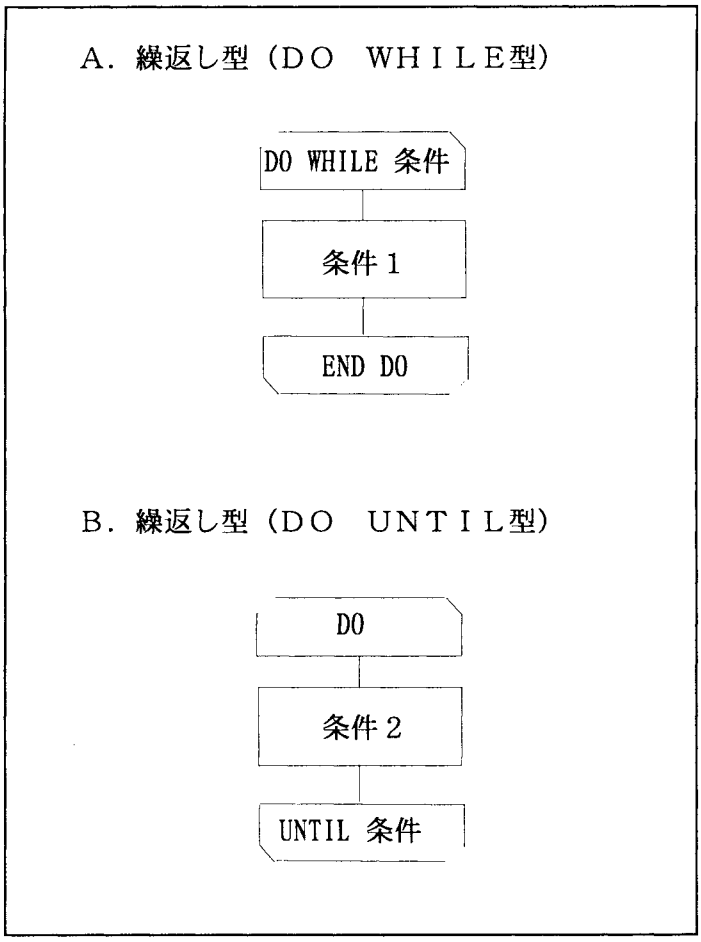


図Ⅳ-15　フローチャートの記述例

フローチャートは、プログラムの動作を簡単に記述できるため、その普及度は高い。しかし、もともと構造化プログラミング手法が普及する以前から存在した手法のため、矢印を引くだけで手軽に goto 文を使用することができたり、構造化定理に基づく五つの論理構造を簡単に示すシンボルが欠けているなど、構造化プログラミング手法には適合しない面が多かった。

その後、フローチャートでも、繰返し型の論理構造を示すシンボルを追加し、構造化プログラミング手法に適合させる改良が行われている。これは、構造化フローチャートと呼ばれている。

図Ⅳ-16に繰返し型のシンボルの使用例を示す。



図Ⅳ-16　繰返し型のシンボルの使用例

（3）ＨＩＰＯ

ＨＩＰＯ（Hierarchy Input Process Output）は、システムの機能を階層的に表現するためのドキュメント手法である。

ＨＩＰＯでは、モジュールの入力、処理、出力を抽象度の高い図で表す。フローチャート

と異なり、ＨＩＰＯでは、制御の流れもデータの流れも併せて記述することができる。図Ⅳ-17にＨＩＰＯの記述例を示す。

ＨＩＰＯ

		ページ	4
システム名	学籍管理システム	作成日	１９９６年１月２３日
サブシステム名	成績管理サブシステム	作成者	構造　設計
処理名	成績評価	備考	

入　力	処　理	出　力
欠席回数 →	１．学生の授業の欠席回数を受け取る。	
	２．欠席回数が規定以内かどうかを判定する。もし、欠席が多すぎたら「不可」を成績情報に書きこむ。	→ 成績情報
試験素点 →	３．学生の試験の素点を受け取る。	
	４．学生の素点が８０点以上なら「優」、７０点以上なら「良」、６０点以上なら「可」、それ未満なら「不可」を成績情報に書きこむ。	→ 成績情報

補足説明：欠席回数が３回以上の場合、成績情報を「不可」とする。

図Ⅳ-17　ＨＩＰＯの記述例

（4）ＨＣＰ

ＨＣＰ（Hierarchical and ComPact description chart）は、ＮＴＴ電気通信研究所が開発した構造化プログラミングの思想に基づいた設計手法である。

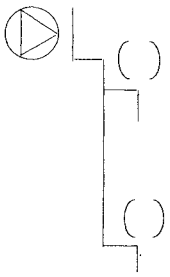
ＨＣＰでは、処理の種類ごとに○を基本にしたシンボルが用意されており、そのシンボルと字下げによる階層表現を組み合わせて制御の流れを記述する。

また、データは長方形で表し、データ構造が複雑なときは長方形の中に複数の長方形を配置することによって表現する。

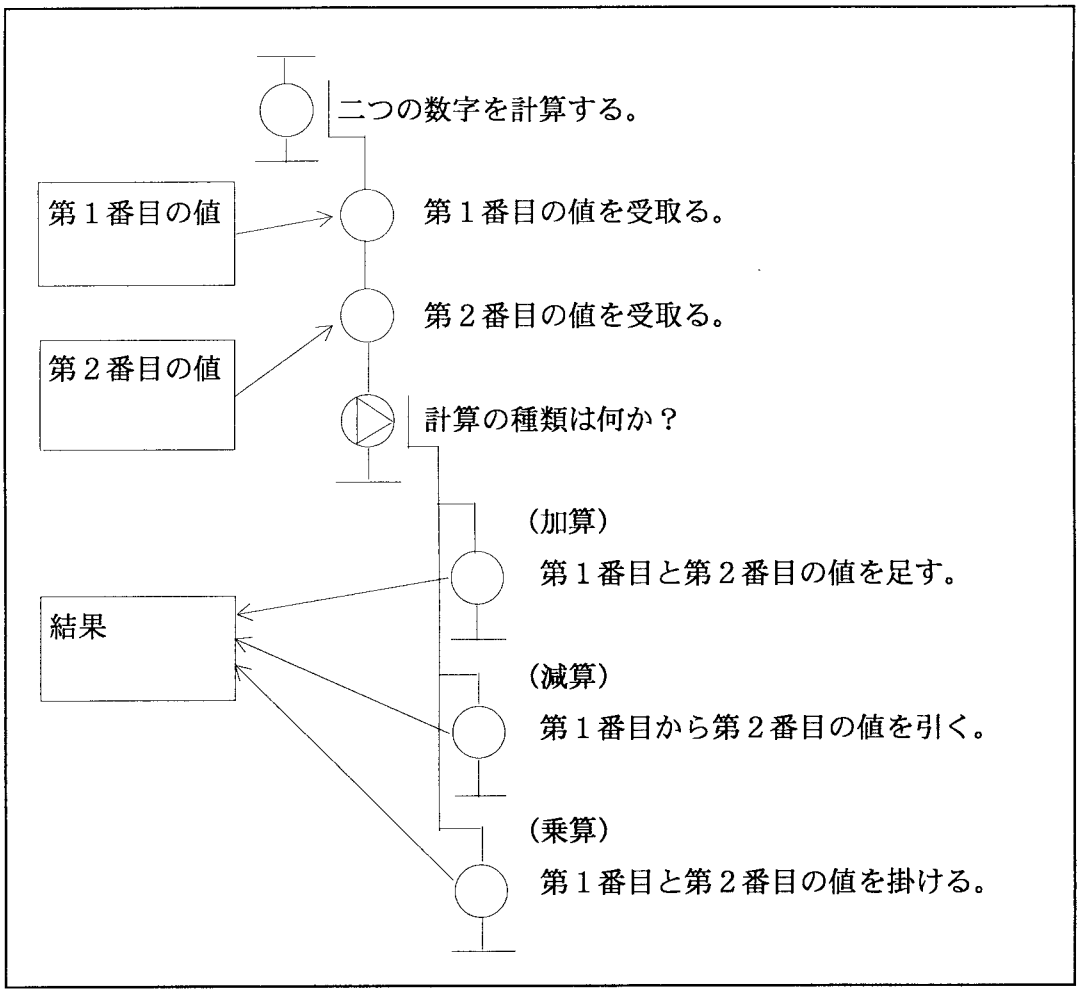
また、ＨＣＰチャートは、もともと定規を使わず、フリーハンドで記述することが推奨されている。それは、設計の改良を容易にし、保守性を向上させるためである。

表Ⅳ-３に、ＨＣＰに使われるシンボルの一部を示す。

表Ⅳ-３　ＨＣＰのシンボル

シンボル	意味	説明
	基本形処理	処理の単位を示す。
	繰返し処理	条件分岐による繰返し処理を示す。
	選択処理	選択条件に従って、分岐することを示す。（　）には、選択条件をそれぞれ記入する。
	非正常処理	異常が発生したときの処理を示す。
	プログラム呼出し	サブルーチンを呼ぶことを示す。
	マクロ	システムまたは自分のマクロを使うことを示す。
	別のページでの詳細記述	別のページで詳細に記述されていることを示す。

また、図Ⅳ-18にＨＣＰの記述例を示す。



図Ⅳ-18　ＨＣＰの記述例

（５）ＤＦＤ

ＤＦＤ（Data Flow Diagram：データフローダイアグラム）は、データ中心アプローチに基づくドキュメント化技法である。

ＤＦＤでは、対象となるシステムの中で、さまざまな行動とその行動の原因または結果として作成されるデータの流れとの関係を図式表現を用いて整理する。プロセスも考慮するが、中心はデータ構造の分析であるところが特徴である。

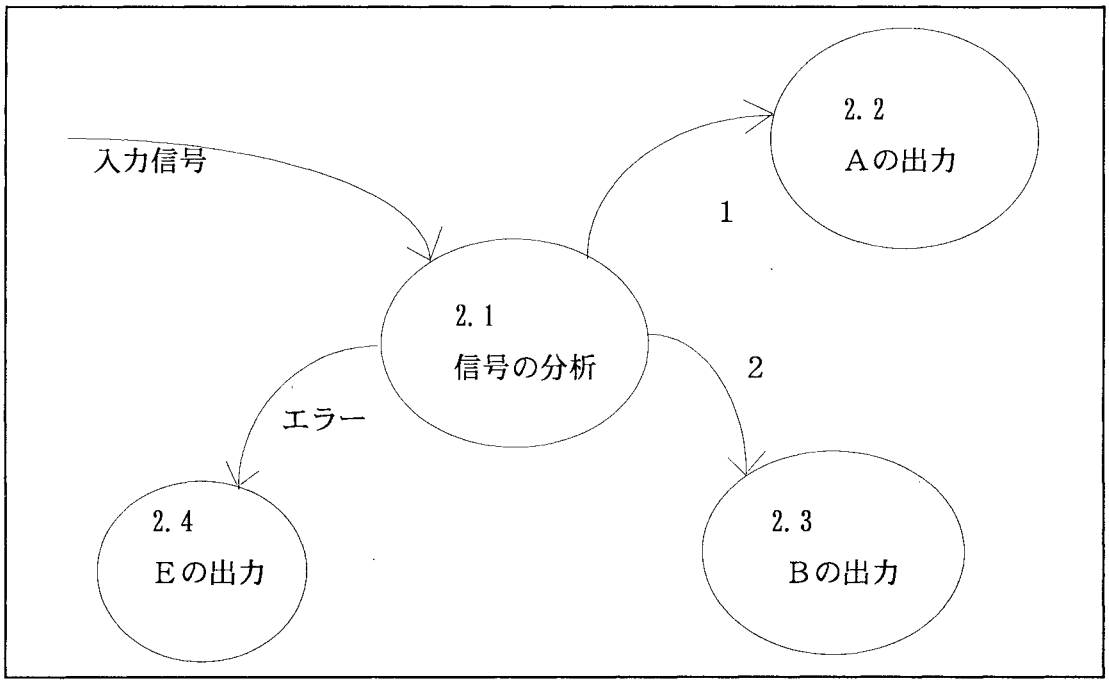
図式表現に用いる記号は、ソース／シンク、データフロー、プロセス、データストアの４種類である。

表Ⅳ-４にＤＦＤの図式表現に使用される記号を示す。

表Ⅳ-4　ＤＦＤの図式表現に使用される記号

記　　号	名　　前	説　　明
情報名 ⟶	データフロー	各処理間および処理とソース／シンク間、処理とデータストア間のデータの流れを示す。 情報の内容を矢印の上または下に記入する。
処理名 番号	プロセス （バブル）	入力データフローから出力データフローへの加工・変換処理（プロセス）を示す。 処理名は具体的に記述する。 バブルの中に番号を記入する。
データストア名	データストア	データが保存（蓄積）される場所を示す。 一般にはファイルを意味する。
ソース名または シンク名	ソース／ シンク	対象ＤＦＤの外部にあり、ソースはデータの発生源、シンクはデータの行先（吸収先）を示す。

図Ⅳ-19にＤＦＤの記述例を示す。



図Ⅳ-19　ＤＦＤの記入例

ＤＦＤを用いた分析はトップダウンで行い、また、ＤＦＤ自体が階層化される。

最上位のＤＦＤを、特にＤＣＤ（データコンテキストダイアグラム）と呼ぶ。これは、システム全体をただ１個のプロセスとして記述したものである

次のレベルのＤＦＤをレベル０ダイアグラム、さらに次のＤＦＤをレベル１ダイアグラム、さらにその次のレベルのＤＦＤをレベル２ダイアグラム等々と、順番に呼んでいく。このとき、番号体系も、ＤＣＤ、レベル０ダイアグラム、レベル１ダイアグラム、レベル２ダイアグラムで、それぞれ上位レベルのＤＦＤと下位レベルのＤＦＤが関連付けられるようにしていく。