はじめに

プログラム設計は，設計フェーズの最終工程である．

従来は，プログラミングを行いながら（または並行して），プログラム設計書を書き上げるというようなことも行われてきた．しかし，システム開発の規模拡大や生産性向上，部品化などにより，現在ではプログラミングのための補助的な作業ではなく，システム開発工程の一つとして扱われている．

プログラム設計のメインとなる作業はモジュール分割である．内部設計でサブシステムから機能単位に分割されたプログラムを，構造化設計技法によりコンパイルできる最小単位であるモジュールに分割する．このモジュールへの分割によって，プログラムはよりわかりやすく，かつメンテナンスしやすいものになる．このとき，モジュールの部品化のためにも論理的なモジュール分割が必要となる．

プログラム設計では内部設計書に書かれている内容を十分理解したうえで，目標・手順を設定し，作業を進めなければならない．

4.1 プログラム設計の目標と手順

プログラム設計は，ウォータフォールモデルのなかでは4番目の作業にあたり，内部設計書に基づいてプログラムの設計を行うフェーズである．

4.1.1 プログラム設計の目標

プログラム設計の目的は，プログラムの内部構造を設計することである．そのために，プログラム設計では**構造化設計技法**を用いて，プログラムを**モジュール**に分割する．プログラムをモジュールに分割し，モジュール相互の関係を明確にすることで，プログラムの構造が明らかになり，メンテナンスしやすくなる．

これまでの開発工程も含めて，システム開発における作業対象について見てみると，図表4-1-1のようになる．

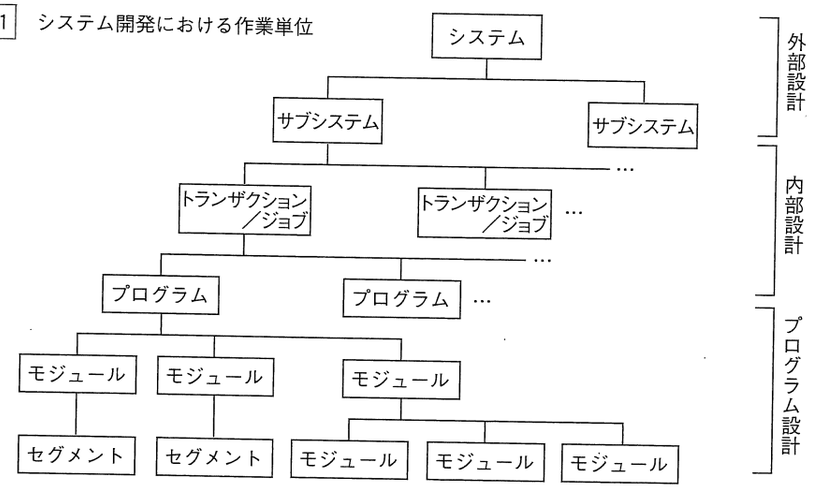
システム開発の作業対象は，開発規模に応じて

　システム・サブシステム・トランザクション/ジョブ・プログラム・モジュール・セグメント（関数）・命令

というように細分化されていく．

ここでは，そのなかのモジュールについて，実際にどのようなものを指すのか列挙してみる．

図表 4-1-1　システム開発における作業単位



・高水準言語におけるコンパイルの単位
・プログラムの機能項目
・メニュー項目の単位
・ソースプログラムが10～300ステップ程度のもの
・タスク（プロセス）管理におけるタスク（プロセス）
・ロードモジュールの単位
・オブジェクト指向型開発におけるオブジェクト
・GUIのイベント単位　など

以上のように，モジュールは「量的，あるいは論理的にまとまりのある単位」というように定義づけられる．ただし，これはあくまでも「基準」にすぎない．

4.1.2 プログラム設計の手順

プログラム設計は，

1．内部設計書の確認
2．モジュール分割
3．モジュール仕様の作成
4．プログラム設計書の作成
5．テスト仕様の作成
6．デザインレビュー

の手順で行われる．

（1）内部設計書の確認

システムの設計作業においては，基本設計・外部設計・内部設計・プログラム設計の各設

計作業はすべて一貫していなければならない．プログラム設計も，前工程である内部設計の内容を完全に反映させるために，内部設計において定義されたプログラムをモジュールに分割する前に，各プログラムごとに，

・機能の定義（何を行うのか）
・入力情報（何を入力するのか）
・処理（どんな処理を行うのか）
・出力（何を出力するのか）

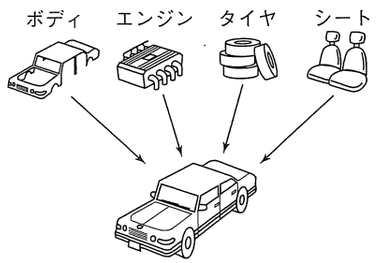
について検討しなければならない．

（2）モジュール分割

モジュール分割は，構造化設計技法に基づいて，プログラムの機能をコンパイル単位のモジュールに分割する作業であり，プログラム設計の中核部分となる．

たとえば，車を例にモジュール分割について考えてみる．

図表 4-1-2
車の分割



車はボディ，エンジン，タイヤ，シートなどを組み立てて作られる．このとき，車全体および各パーツの設計書や仕様書がしっかりしていれば，各パーツをあらかじめ作っておいて，それを組み立てたほうが製造しやすい．また，ブレーキが故障したとしても，その部分だけ修理（交換）すればよい．

システム開発も同じことがいえるが，システム開発の場合は「形」が見えないため，物理的あるいは論理的側面からパーツ（モジュール）に分割することになる．ただし，単純に大きさだけで分割すると，その後のプログラミングや保守作業がしにくくなるので，モジュールは原則として論理的な単位で分割する．そして，物理的な尺度は補助的に用いるようにする．

ここで，モジュール分割のメリットを挙げると，次のようになる．

- **モジュール独立性**の確保
  内部設計の処理内容を物理的に分割するのではなく，論理的にひとまとまりの単位に分割することにより，モジュールの独立性が確保される．
- 処理効率の向上
  他のモジュールとのかかわりを最小化することにより，処理効率が向上する．
- **部品化，再利用**

  プログラム間で共通に使用できるモジュールや，再利用できそうなモジュールなどを抽出して，部品化，再利用につなげることができる．
- 保守の効率化と信頼性の向上
  システムの機能変更などでは関連するモジュールのみを変更すればよく，保守の効率化と信頼性の向上を図ることができる．

なお，モジュール分割の手順など，構造化設計の詳細については後述する．

（3）モジュール仕様の作成

分割されたモジュールの，具体的な処理内容を定義する作業である．コーディング時に機能の漏れが発生しないように，細部にわたって配慮する必要がある．

（4）プログラム設計書の作成

（1）～（3）の結果を**プログラム設計書**としてまとめる．プログラム設計書はコーディング作業の指針となるものであり，作成基準に従って，次のような内容を記載する．

〈プログラム設計書の内容〉

・プログラム設計方針
・プログラム概要
・プログラム構造図
・処理内容
・テストケース
・項目説明

（5）テスト仕様の作成

各テストの目的に応じて，テストに関する仕様を作成する．プログラムテストには単体テストと結合テストがある．

（6）デザインレビュー

デザインレビューの目的は，

・確認（validation）：ユーザの要求を満足しているか
・検証（verification）：内部設計との一貫性が図られているか，またプログラミング作業に移行しても大丈夫か

の二つである．この二つのテーマを念頭におき，レビューを行う．どのフェーズにおいてもレビューは大切であるが，プログラム設計におけるレビューの成否はプログラミング作業に大きな影響を与える．

〈デザインレビューの留意事項〉

・プログラム設計書を確認する．
・内部設計書と比較して，欠落している機能および不具合な点について指摘する．
・モジュールに関する機能の漏れがないか．

・モジュール分割は適切か.
・モジュール間インタフェースの整合性や欠落を確認する.

4.2 プログラムの構造化設計

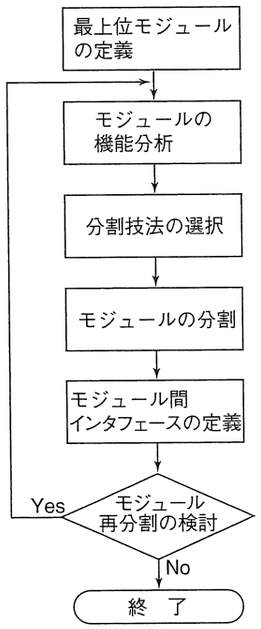
システムが大規模化し，ユーザの要求する機能が複雑になってくると，それを処理するプログラムも複雑さを増してくる．その結果，潜在するバグの数も多くなり，修正時間もコストも増大する.

以前のような，ハードウェアが高価であった時代では，高度な機能を実現するための複雑なロジックで構成されるプログラムは，プログラマの腕の見せどころであった．しかし，現在ではシステム資源も充実してきているので，品質の高い，わかりやすいプログラムを作成することが重要である.

4.2.1 構造化設計の手順

ここでは，プログラムをモジュール単位に分割する構造化設計の手順について説明する.
構造化設計のポイントは，モジュールの独立性を高めるような分割を行うことである.

図表 4-2-1
構造化設計の手順



各作業の詳細は以下のとおりである.

(1) 最上位モジュールの定義

最上位モジュールとは，プログラムが起動すると，まず最初に呼び出されるモジュールのことで，次のような機能をすべてもつ.

・プログラム全体（各モジュール）の制御
・データ項目（カウンタなど）に対する初期値の設定
・ファイルのオープン/クローズ　など

なお，最上位モジュールでプログラム全体の制御のみを行い，データ項目の初期の設定やファイルのオープン/クローズなどは別のモジュール（下位のモジュール）で行うケースもある.

(2) モジュールの機能分析

プログラムとして必要不可欠な機能をすべて洗い出し，必要であれば分割や統合を行い，最適な単位に分割する.

〈モジュールの機能〉
・ファイルの読込み機能
・読み込んだデータのエラーチェック機能
・データに対する加工（演算など）機能
・データの出力機能
・エラー処理　など

(3) 分割技法の選択

以下のような視点で，最適なモジュール分割技法を選ぶ．なお，各分割技法については，次の4.2.2項で詳しく説明する.

● データの流れに着目した分割技法
主に，トランザクション処理を行うオンラインシステムに適した方法である.
〈分割技法〉
・STS 分割法
・TR 分割法
・共通機能分割法

● データ構造に着目した分割技法
主に，ファイル中心のバッチ処理に適した方法である.
〈分割技法〉
・ジャクソン法
・ワーニエ法

(4) モジュールの分割

上記（3）において選択したモジュール分割技法を用い，分割基準に従ってプログラムをモジュール単位に分割する.なお,呼び出されるモジュールのことを**従属モジュール**という.

〈分割手順〉

1．最上位モジュールの定義.

2．最上位モジュールから呼び出されるモジュールの定義.

3．上記の手順2から呼び出されるモジュールの定義.

4．以下，順次下位のレベルへブレイクダウンしていく.

なお，分割のおおよその目安は次のようになる.

〈モジュール分割の目安〉

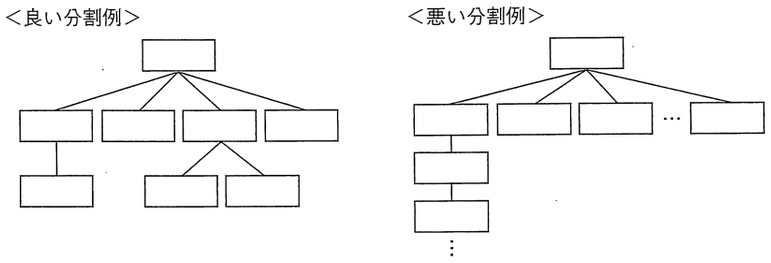
・1プログラムは，一般的には10～300程度のモジュールで構成されているとわかりやすい.

・一つのレベル（階層）は10モジュール以内が妥当である.

・階層の深さは4階層以内が妥当である.

モジュール分割の良い例と悪い例を，図表4－2－2に示す.

図表 4-2-2　良い分割例と悪い分割例

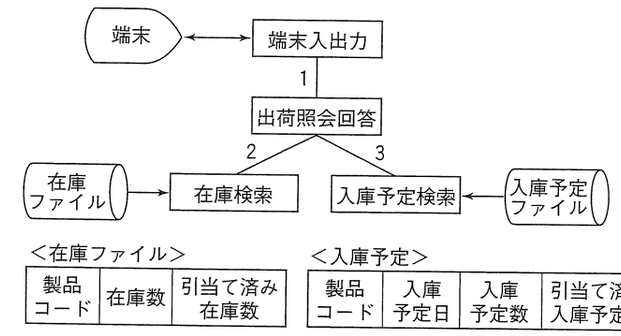


（5）モジュール間インタフェースの定義

ここでは，モジュール間で必要な情報をやり取りする場合の必要なデータ，および条件について定める（図表4－2－3）.

モジュール間インタフェースの定義はモジュール分割と同様，重要な作業である.「A」を入力すれば「B」が返ってくるという，モジュールインタフェースに対する信頼性があるからこそ，モジュールに分割してもプログラム自体の信頼性が確保されるのである．1回にやり取りする情報の数は，七つ以内を目安とする.

図表 4-2-3　モジュール間インタフェース

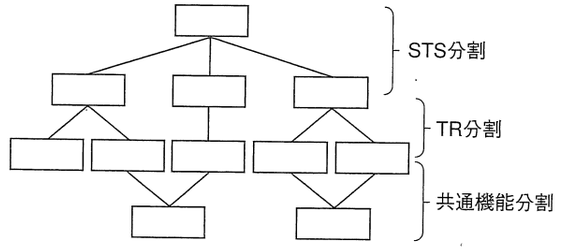


連結線番号	上位モジュールからの入力	上位モジュールへの出力
1	製品コード，希望数，出荷希望日	ステータス，メッセージA，メッセージB
2	製品コード	未引当て在庫数
3	製品コード	入庫予定日，未引当て入庫予定数

（6）モジュール再分割の検討

分割基準に従ってモジュールを分割し，その評価が悪かった場合には，再度前記の（3）で「分割技法」を選択し，トップダウン的に分割していく．その際，必要に応じて分割を併用することも行われている.

図表 4-2-4
分割技法の併用



〈順　序〉

1.　上位モジュールはSTS分割法で行う.

2.　中位モジュールはTR分割法で行う.

3.　詳細化された下位モジュールに対し，共通機能分割法で統合化を図る.

4.2.2　代表的なモジュール分割技法

（1）データの流れに着目した分割技法

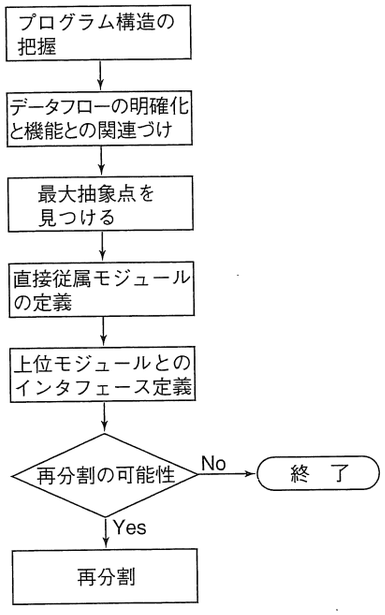
①STS分割

データは基本的に，入力/処理/出力という一連の流れに沿って処理されていく．**STS分割**はそこに着目して，プログラムを

・Source（源泉；入力処理機能）
・Transform（変換；データ処理機能）
・Sink（吸収；出力処理機能）

の三つに分割する技法である．この分割技法は入力/処理/出力機能がすべて含まれる上位モジュールへの適用に適している．

図表 4-2-5
STS 分割の手順

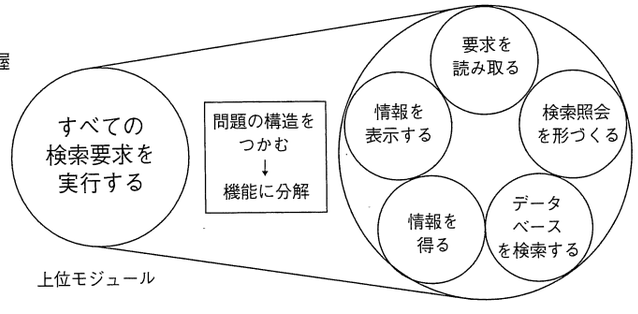


〈STS 分割の手順〉

1．プログラムの構造を把握する

図表 4－2－6 に示すように，プログラムの構造を機能中心にとらえ，3～10個にまとめる．

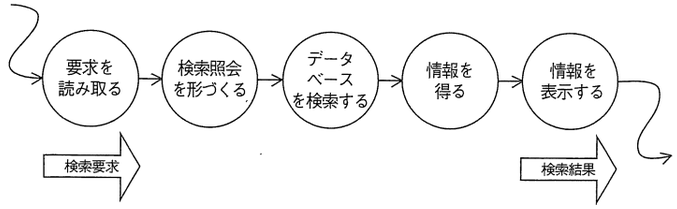
図表 4-2-6
プログラム構造の把握



2．入出力データの流れを明確にし，機能と関連づける

図表 4－2－7 に示すように，プログラムの中の，入力データと出力データの主要な流れを矢印で関連づける（バブルチャート）．

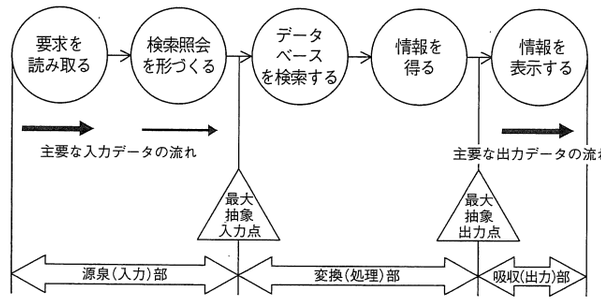
図表 4-2-7　データの流れの関連づけ



3．最大抽象点を見つける

図表 4－2－8 に示すように，入力といえない点まで抽象化された地点（**最大抽象入力点**）と，初めての出力データといえる形を表す点（**最大抽象出力点**）を見つける．

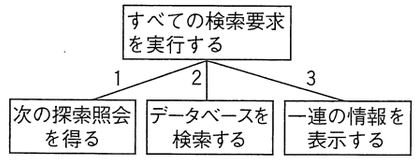
図表 4-2-8
最大抽象点



4．直接従属モジュールを定義する

図表 4－2－9 に示すように，上位モジュールから入力・処理・出力に3分割されたモジュールを構造化して，関連づける．

図表 4-2-9
モジュールの構造図



5．上位モジュールとのインタフェースを定義する

上位モジュールとの情報の受け渡し（モジュール間インタフェース）を定義する（図表 4－2－10）．

図表 4-2-10　モジュール間インタフェースの定義

	入力	出力
1	（なし）	・探索照会 ・要求文 ・端末アドレス ・エラーコード
2	・探索照会	・情報文 ・エラーコード
3	・情報文 ・要求文 ・端末アドレス	・エラーコード

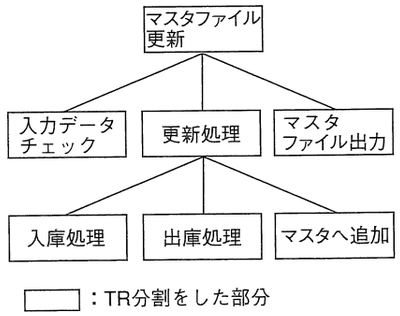
6．再分割の可能性をチェックする

さらに分割すべきモジュールがあるかどうかをチェックし，あればモジュール分割を繰り返す．

②TR 分割（トランザクション分割）

TR 分割は，データの種類によってトランザクションの種類が決まってしまう場合に，トランザクションの種類ごとにモジュール化する分割法で，データ依存型の分割法である．

図表 4-2-11
TR 分割の例

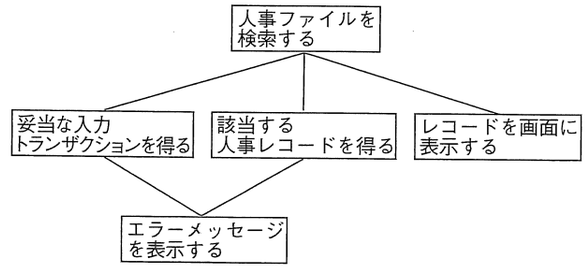


図表 4－2－11の例では，入力データの性質（入庫処理・出庫処理・マスタへ追加）によってトランザクションの処理を分割している．分割されたトランザクションは，選択処理となる．

③共通機能分割

共通機能分割とは，いくつかのモジュールに共通の機能があるとき，それらの機能を取り出し，別のモジュールとして定義する方法である（図表 4－2－12）．

図表 4-2-12
共通機能分割



（2）データ構造に着目した分割技法

モジュールの構造を入出力データの構造に対応させて，モジュール設計を行う方法である．

①ジャクソン法

ジャクソン法は，プログラムの構造を入力データと出力データの構造に対応させて分割する技法である．

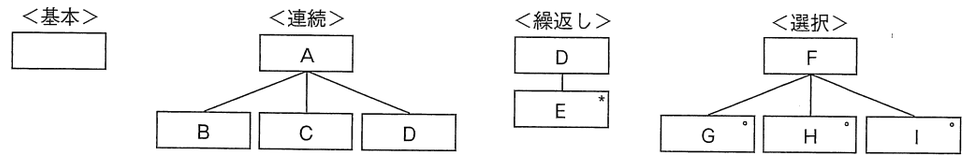
〈ジャクソン法の分割手順〉

1．入力と出力のデータ構造を定義する．
2．入力・出力データ構造の構成要素間の１対１の対応関係を見つける．なお，見つからないときは，中間データ構造を作成する．
3．出力データ構造をもとに，プログラム構造を作成する．
4．入力データ構造でプログラム構造の検証を行う．

〈ジャクソン法の構成要素〉

ジャクソン法では，データ構造もプログラム構造も「基本」を最小単位とした，「連続」・「繰返し」・「選択」の三つの組合せで構成される．

図表 4-2-13　ジャクソン法の構成要素

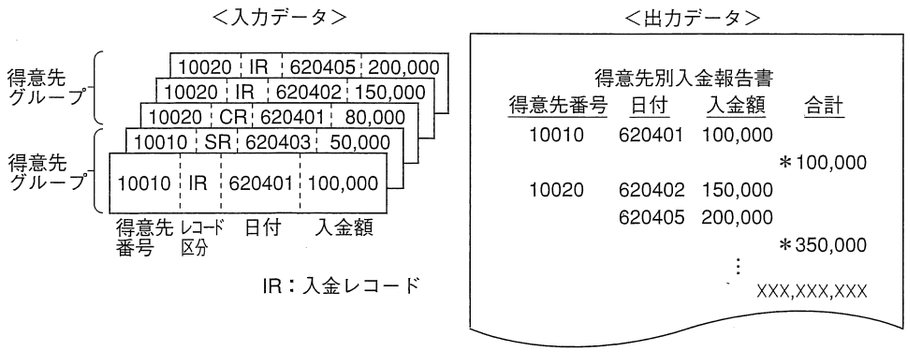


- 基本：核となる構成要素（これ以上分割できないもの）
  （例）データ項目，ステートメント　など
- 連続：複数の部分から成る構成要素で，それぞれの部分は１回だけ順に現れる．
  （例）レコード（複数データ項目），順次処理ルーチン　など
- 繰返し（*）：一つの部分が繰り返し現れる．
  （例）順次ファイル，PERFORM 文　など
- 選択（°）：複数の部分から成り，そのうちの一つを選択する．
  （例）借方・貸方，EVALUATE 文　など

〈具体的な分割方法〉

図表4－2－14を用いて，ジャクソン法による具体的な分割方法について説明する．

図表 4-2-14　入力データと出力データ例



● 入力と出力のデータ構造を定義する

図表4－2－14を見ると，

・入力ファイル：同じ得意先グループの繰返し

・得意先グループ：同じ得意先レコードの繰返し

・得意先レコードの種別：入金レコードまたはそれ以外のレコード

から構成されている．同様に

・出力データ：得意先ごとの入金記録の繰返し

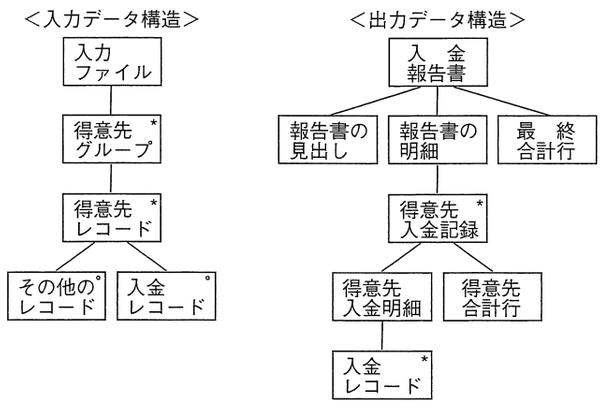
・入金記録：得意先入金明細と得意先合計行の連続

・得意先入金明細：入金レコードの繰返し

から構成されている．

以上のことから，入力と出力のデータ構造を定義すると，図表4－2－15のようになる．

図表 4-2-15
入力と出力のデータ構造



● 入力と出力のデータ構造の，構成要素間の1対1の対応関係を見つける（図表4－2－16）．対応関係がないときは，中間データ構造を作成する

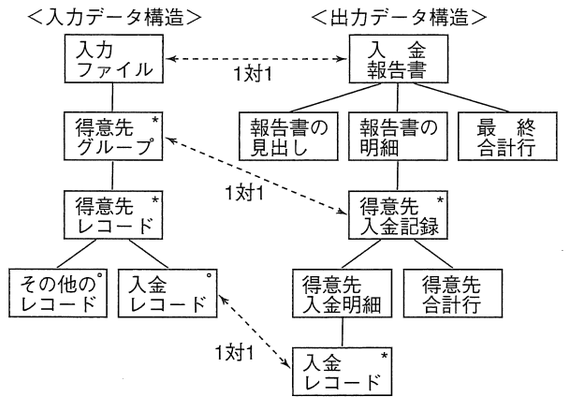
図表4－2－15では，次の三つの対応関係がある．

・入力ファイルと入金報告書

・得意先グループと得意先入金記録

・入金レコード同士

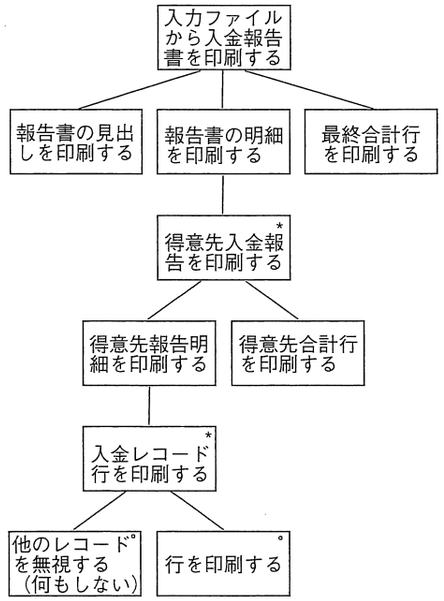
図表 4-2-16　構成要素間の1対1の対応関係



● 出力データ構造をもとにプログラム構造を作成する

構成要素間の1対1の対応関係をもとに，プログラム構造を作成する（図表4－2－17）．基本的に，プログラム構造は出力データに対応させ，入力データはプログラム構造の検証および一部補正のために使用される（最下位の「他のレコードを無視する」は一部を補正した結果である）．

図表 4-2-17
プログラム構造



②ワーニエ法

ワーニエ法は，集合論に基づく構造的モジュール設計技法であり，主にファイル処理を中心とする適用業務のモジュール分割に広く用いられている．

〈ワーニエ法の特徴〉

・データ分析が基本となる．

・「いつ・どこで・何回」を基本とし，トップダウン方式で分析を行う．

・ジャクソン法が主に出力データを基本として構造化を図ったのに対し，ワーニエ法は主に入力データを中心に構造化を図る．

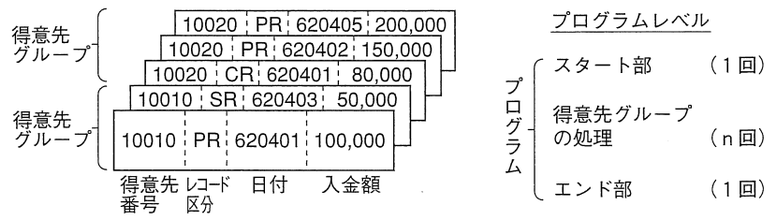
以下に，ワーニエ法の分割手順について説明する．なお，具体例は先述の図表4－2－14を用いる．

〈ワーニエ法の分割手順〉

1．入力データ構造とプログラム論理構造の対応関係を見つける

まず，入力データ構造とプログラムの論理構造を比較する．

図表 4-2-18　入力データ構造とプログラム論理構造の比較



入力データ構造は図表4－2－14を引用しており，また対応するプログラムの論理構造は

・スタート部

・得意先グループ部

・エンド部

の部分集合から成る．（n：繰返し回数，0と1：選択）

図表4－2－18では，ファイルが終了するまで得意先グループの処理を行うことを示している．

2．部分集合をトップダウン的にブレイクダウンする

得意先グループをさらにブレイクダウンすると，図表4－2－19のようになる．

図表 4-2-19　部分集合のブレイクダウン（その1）

得意先グループレベル

得意先グループ
- 得意先グループスタート部（1回）
- 得意先レコードの処理　（m回）
- 得意先グループエンド部　（1回）

つまり，得意先グループの論理構造は

・得意先グループスタート部

・得意先レコード部

・得意先グループエンド部

の部分集合から構成されている．

さらに，得意先レコード部をブレイクダウンすると，図表4－2－20のようになる．

図表 4-2-20　部分集合のブレイクダウン（その2）

得意先レコードレベル

得意先レコード
- 得意先レコードスタート部（1回）
- 入金レコードか否か　（0～1回）
- 得意先レコードエンド部　（1回）

つまり，得意先レコード部の論理構造は

・得意先レコードスタート部

・入金レコードか否か

・得意先レコードエンド部

の部分集合から構成されている．「入金レコードか否か」はこれ以上ブレイクダウンできないので，このレベルで終了する．