ソフトウェア工学　121－22
（1998. 11.　6）

ソフトウエア標準化のためのプログラムパターンの作成

大野　治　　降旗　由香理

株式会社　日立製作所　公共情報事業部
〒136-8632　東京都江東区新砂１丁目６番27号　新砂プラザ

あらまし　本論文は、業務ソフトウエア開発における部品化技術に関する研究とその実践結果の報告である。ソフトウエア部品は処理機能と処理パターン（スケルトン）に大別できる。処理機能は暦変換などのように一つの機能を持つ処理のまとまりである。処理パターンは、照合などのように処理方式であり、プログラムの構造を部品化したものになる。

本論文では、業務アプリケーション開発経験から標準化したプログラム部品を抽出した。そして、このプログラムパターンを実業務アプリケーションの開発に適用し、適用結果から、標準プログラムパターンが全プログラムのどの程度を網羅できているかを考察することによって、標準部品としての有用性を検証した。

キーワード　構造化設計　ソフトウエア標準化　プログラムパターン　部品　再利用

Creation of Program Patterns for Software Standardization

Osamu Ohno, Yukari Furuhata

Hitachi, Ltd Information Systems Group.
Government & Public Corporation Information Systems Division.
Shinsuna Plaza 6-27, Shinsuna 1-Chome, Koto-ku, Tokyo 136-8632, Japan

Abstract

This paper is a report on a research regarding partitioning technology in business software development and the results of the use of the technology. Software parts can be divided roughly into processing functions and processing patterns (skeletons). A processing function is a set of processes which perform a particular function, such as date conversion. A processing pattern is a processing method, such as collation, and is a program structure which has been partitioned.

For this paper, standardized program parts were extracted based on our experience in business application development. These program patterns were applied to actual business application development projects, and by studying the extent to which standard program patterns covered the total program, the utility of these patterns as standard parts was examined.

key words　Structured Design, Software Standardization, Program Patterns, Parts, Reuse

1．はじめに

企業による情報システムの開発では、大量のソフトウエアを、技術や知識レベルのばらつきが大きい多くの担当者が携わり、さらに、文化の異なる複数の協力会社との共同開発が多い。このような種々の人々から構成される大きな集団で、ソフトウエアの生産性向上を図るには、部品化・再利用技術が必要不可欠である。

部品化・再利用によるソフトウエア開発とは、共有化できるソースコードをある単位にまとめ、それを複数のプログラムで使うことである。部品化・再利用による開発が成功する鍵は、部品の利用率を高め、新たな作成部分を減らすことである。

部品には、処理機能と処理パターン(スケルトン)の２種類がある。

処理パターンとはプログラムの構造であり、この設計には Jackson や Warnier らによる構造化の技法がある。処理パターンの設計にあたり問題となるのは、構造化技法をプログラム構造として具現化することのみならず、再利用率を高めるために標準部品として、どのような種類のパターンをいくつ用意すれば最適かという点にある。

処理パターンの種類を多くするほど、仕様に適合する確率が高くなり、かつ、作り込む量が少なくなる反面、適切なパターンの検索や選択に労力を費やすことになり、再利用の効果が小さくなる。したがって、処理パターンの汎用性と種類の多様性のバランスをとることが肝要である。

本論文では、構造化技術に基づいた処理パターンの標準化方法について論じる。更に、作成した処理パターンが"標準"のパターンとして妥当であるかどうかを、開発ソフトウエアにおけるプログラムパターンのカバー率から検証する。

2．プログラム部品のとらえ方

はじめに、プログラムの構造から、プログラム部品の考え方を説明する。従来、プログラムの構造設計に関しては、さまざまな方法論や技法が提言されている[1]。この中の Jackson、Warnier、Myers 等の各設計方法に基づき、プログラムの制御構造をデータ構造から導き、プログラム構造を図２．１のようにとらえた。

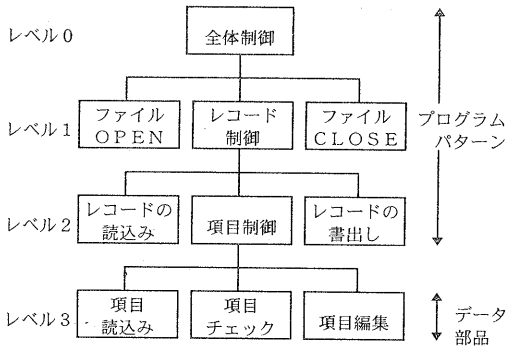


図２．１　プログラム構造の基本概念

ここで、プログラム部品を２つに分けて考える。一つは、レベル０～レベル２までをカバーしたプログラム構造を規定する部品であり、もう一つは、レベル３に相当する項目処理部品である。ここでは、前者をプログラムパターン、後者をデータ部品と呼ぶ。

本論文で提案するプログラム部品化では、ファイル操作、レコード操作を規定したプログラムパターンに、データ項目操作を部品化したデータ部品を取り込むことで、プログラムを作成する方法を採る。本方法は、プログラムパターンと、データ部品を自由に組み合わせることができるため、部品の再利用度を高め、開発領域の減少を図る。

プログラムパターンの設計では、バッチ系処理プログラムとトランザクション系処理プログラムとを分けて考えた。それは、バッチシステムとオンラインシステムでは、システムの状態管理方法が異なり、これによりプログラム構造が異なると考えたからである。トランザクション系処理は、ユーザとコンピュータとの対話により動的にシステムの状態が変化する。このため、システムの状態管理の設計が難しく、設計の要所となる。

一方、バッチ系処理は、ＪＣＬ(ジョブ制御言語)によりプログラム間の制御が行なえるため、動的な処理制御はほとんど必要としない。

3．バッチ系処理におけるプログラムパターンの設計

3．1　設計の観点

Jackson の構造化設計技法に基づくと、プログラムの処理ロジックとは、入力データを要求された出力データへ変換する加工工程と捉えられる。したがって、入力データと出力データの構造を各々設計し、互いの写像関係を見出すことによりプログラム構造を設計した。

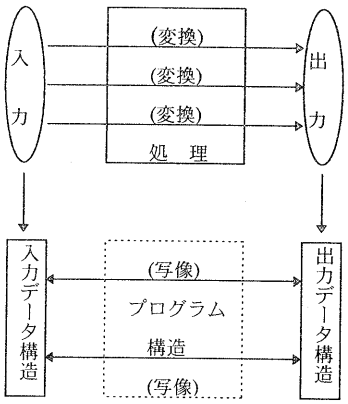


図３．１　Jackson 法に基づくプログラム構造の考え方

設計にあたり、プログラム構造の決定に影響する因子を考えるために、過去に作成した業務プログラムから、プログラム構造と入出力ファイルの関係を調べた。

その結果、２つの傾向を得た。

第１に、ファイルには、プログラム構造に決定的に影響を与えるファイルと、本質的な構造には影響しないファイルがあることである。例えば、注文伝票トランザクションファイルによる商品マスターファイルの更新処理では、参照するファイルは通常多数（５個や８個に達することもある）存在する。しかし、複数ある参照ファイルは、顧客マスターから名称や住所を参照する、あるいは販社マスタから情報を得るなど、特定処理で一時的に使用される存在であり、プログラムの構造の点からは、単なる付属ファイルとして見做すことができる。したがって、プログラム構造は、その構造に直接影響を与えるファイルにだけ着目すれば、一意に決定できる。

第２に、入力ファイルが更新対象である場合には、これが、ランダムアクセスファイルであるか、シーケンシャルアクセスファイルであるかによってプログラム構造が異なることである。

シーケンシャルファイルを読み込む場合には、プログラム中にループ構造ができる。ループ構造の存在により、データの状態が動的に変化し、処理対象のデータが一意に決定できず、プログラムが複雑になる。逆に考えると、データの状態がループの状態に依存しない構造にすることで、プログラム構造を単純にできる。この点に着目し、プログラマが複雑なループの制御を意識しなくてもすむように、ループ構造をプログラムパターンとして吸収することにした。そのため、プログラムパターンはシーケンシャルファイルを主軸に設計した。

以上のことを踏まえ、次の５つの因子により、プログラム構造を分類した。

①入力ファイルと出力ファイルの数
②入力ファイルのアクセスタイプ
③入力レコードの内部処理条件（レコードレベル）
④入力レコードの内部処理条件（項目レベル）
⑤プログラム構造に影響を与えない Other File の有無

３．２　プログラムパターンの分類

前節で決めた因子に基づいてプログラムパターンを分類する。分類は、図２．１で示したプログラム構造において、制御レベルが大きいファイル制御の因子で大別し、レコード制御、項目制御の各因子により詳細化した。

（１）　ファイル制御に関する因子による分類
—入出力ファイル数による分類—

プログラムの特性上、入出力ファイルは各々１個以上は必要である。また、プログラム構造に直接的に影響を与えるシーケンシャルファイルは、実際のプログラムを見る限りでは、５個以内である。

したがって、入出力ファイル数からは次の６つに分類した。

①入力ファイル：１　出力ファイル：１
②入力ファイル：１　出力ファイル：２
③入力ファイル：２　出力ファイル：１
④入力ファイル：２　出力ファイル：２
⑤入力ファイル：３　出力ファイル：１
⑥入力ファイル：３　出力ファイル：２

（２）レコード制御に関する因子による分類
—レコードの処理条件による分類—

レコード処理条件は、入力ファイルの数により異なる。入力レコードが複数ある場合は、レコード同士の突合処理で、ループの終了条件と、マスタファイルのアクセス方法によって分類できる。

ループの終了条件が、トランザクションファイルの終了時点の場合は照合処理、マスタファイルの終了時点の場合は更新処理に分類できる。また、マスタファイルのアクセス方法が、シーケンシャルの場合とダイレクトアクセスする場合で分類できる。さらに、更新処理の場合には、マスタファイルと一致しないトランザクションファイルのレコードをエラーデータとして扱うか、トランザクションファイルへの追加レコードとして扱うか、という点でも分類できる。

		出力ファイル数	
		1	2
入力ファイル数	1	レコード処理条件による分類はなし	
	2	突合処理—照合処理 └更新処理—シーケンシャルアクセス/ダイレクトアクセス └トランザクションレコードのエラー処理/追加処理	
	3		

（３）項目制御に関する因子による分類
—レコードの処理条件（ブレーク処理）による分類—

入力レコード中の特定項目により、プログラム構造がいくつかに分類できる。

一つは、ブレーク処理の有無である。ブレーク処理とは、特定項目でソートされたレコードをその値が変わるまで溜め込み、値が変わった（ブレークが生じた）時点で、溜め込んだレコードに対して集計や編集等の処理をすることである。この特定項目をコントロールキーと呼び、これはレベルにより複数あることもある。例えば、部署をコントロールキーとする場合は、レベルの大きい方から、部・課・係の３つのコントロールキーが存在する。コントロールキーの数に応じてループ構造が多重化するため、コントロールキーの数に応じたプログラムパターンが必要になる。実業務プログラムの作成では、コントロールキーが３個のプログラムまでサポートできれば概ねカバーできる。このため、コントロールキーの数が１～３個までの３種類のプログラムパターンを用意した。

この他、特定項目により、レコードを振り分ける（分配処理）、レコードを抽出する（抽出処理）などの処理から分類ができる。

（４）出力が帳票の場合

これまで、入出力媒体がファイルの場合についてのみ論じていたが、ユーザインターフェースを考慮した場合、出力媒体を帳票とするプログラムパターンが必要である。

		出力ファイル数	
		1	2
入力ファイル数	1	ファイルの変換 ファイルの抽出 レコードの集計 ブレーク処理 ├コントロールキーが１個 ├　〃　２個 └　〃　３個	ファイルの分配
	2 3	突合処理―照合処理 └更新処理―シーケンシャルアクセス/ダイレクトアクセス └トランザクションレコードのエラー処理/追加処理	

実業務プログラムを開発してきた経験から、ブレーク処理を行なうパターンとエラーデータを出力するパターンでは、出力媒体に帳票が必要と判断した。事務処理では明細表・一覧表・集計表の発行および、エラーデータの確認を帳票として見る必要があるためである。

帳票は「人間が見る」ことを目的とするため、人間にとって見やすい形式にする必要がある。

そのため、ヘッダ、トレイラの表示、集計の有無、ページ、ナンバリングや出力日付など、ファイルの属性とは関係のないものを出力したり、改ページの条件など帳票独自のロジックが必要となる。

このような処理をパターンで吸収できるように、出力媒体を帳票とするパターンを用意した。

以上のような観点で、表３．１のようにプログラムパターンを分類した。

３．３　プログラムパターンの設計

前節で分類したプログラムパターンに対して、以下の方針に基づいて内部設計を行なった。

（１）ループ構造の作り込み

プログラマの技量が要求される入出力レコードの制御構造を、入出力媒体（ファイル・帳票）の種類と数に合わせてパターンで吸収した。これにより、プログラマが凝った技巧的なロジックを作成できなくなる反面、プログラム作成がルーチンワークとなり、未熟なプログラマでも一定水準の品質のプログラムを作成できる。

（２）ユーザコーディング部分の入出力レコードの明示化。

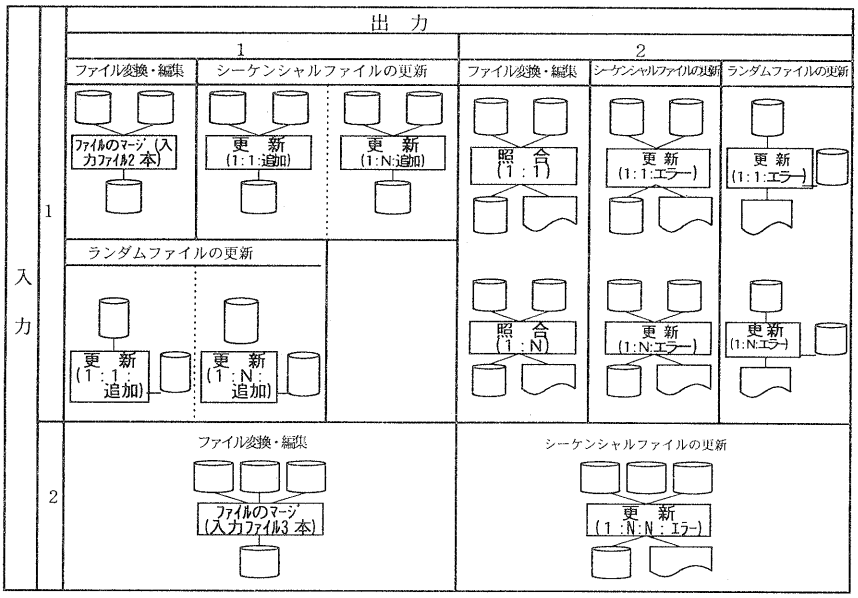
各セクションで取り扱う入力レコードと期待される出力レコードが何であるかを、予め明示的に示す構造にした。これにより、プログラムを複雑にしている要因と言われるフラグ、スイッチ等を一斉使用せずにすむ構造にした。

そして、当該セクションで取り扱うべき情報までを規定して、コーディングすべき処理ロジックを明示しユーザのコーディングを誘導した。

（３）ユーザコーディング部分の極所化

ユーザコーディング部分を特定セクションに集中させた。これより、ユーザは特定セクションさえ意識すれば、コーディングでき、不良の作り込みを防ぐことができる。ユーザコーディングを要するセクションが４～５箇所のパターンが最も多く、最大の場合でも８箇所程度である。

表３．１　プログラムの分類



照合・更新のパターン；（１：ｎ：エラー/追加）

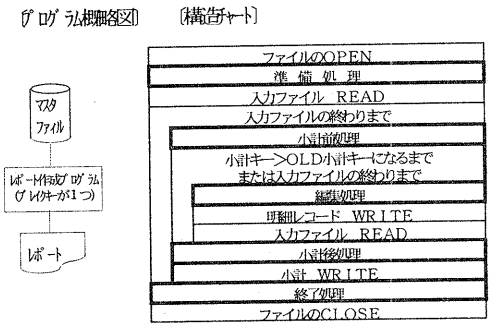
マスターファイルに該当データが無い

マスターレコードに対するトランザクションの数

（4）構造化プログラミングによるわかりやすく、
保守しやすいプログラム構造

ＧＯ　ＴＯ文を使用せず、階層化を図った構造としたため、他人の書いたプログラムでも読みやすく、保守が容易になる。

図３．２に、コントロールキーが１つのレポート作成プログラムパターンの構造を示す。



〔処理基準〕

入力	コーディングすべき処理		出力
マスタファイル 1レコード	小計前処理	SYOKEI-BEFORE-PROC	作業領域の小計キー 小計の集計項目
	1. 小計キーを作業領域にセットする。 2. 小計の集計項目を０クリアする。 3. 小計の集計に必要な前処理。		
ワークファイル 1レコード	編集処理	REPORT-EDIT-PROC	出力用エリア 小計の集計項目
	1. 出力用エリア（作業領域）のスペースクリア 2. 明細項目の編集を行う 3. 編集済みの明細項目を出力用エリアにセットする。 4. 小計の編集処理を行う。		
トランザクションファイル 1レコード	小計後処理	SYOKEI-AFTER-PROC	出力用エリア
	1. 出力用エリア（作業領域）をスペースクリア 2. 集計項目の編集を行う。 3. 編集済みの明細項目を出力用エリアにセットする。		

図３．２　レポート作成処理プログラムの構造

4. トランザクション系処理におけるプログラムパターンの設計

４．１　トランザクションプログラムの構成

本論文では、トランザクション処理を「入力画面より入力されたデータに処理を加えて出力画面を決定し、作成する処理」ととらえる。

トランザクション系処理におけるプログラムをバッチ系処理プログラムと同様に考えると、図４．１に示すように、入力画面から入力されたデータを出力画面に出力するための変換処理ととらえることができる。したがって、トランザクション処理プログラムは、入力画面と出力画面にまたがる。

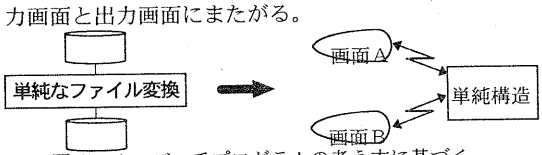


図４．１　バッチプログラムの考え方に基づく
トランザクションプログラムの設計

この構造では、プログラム内に画面制御（次画面を決定する処理）と、入出力データ（メッセージ）処理が混在し、かつ、同一画面に対する処理が複数のプログラムに分散する。そのため、プログラムの複雑化と、画面変更が複数プログラムに影響することによる保守性の低下という問題がある。

そこで、入力データ処理部と出力画面作成部を分離することで、メッセージ処理部を画面遷移制御から独立させる方法を考えた。この考えでは、プログラムの単位は、図４．２に示すように、同一画面の出力処理と入力処理となる。これにより、１画面に１つのメッセージ処理プログラムが対応する構造となる。

このプログラム構造を実現するために、図４．３に示すような、状態遷移の制御系処理部とデータ処理部とを切り離し、制御処理部がデータ処理プログラムを制御するようなプログラム構成を採った。

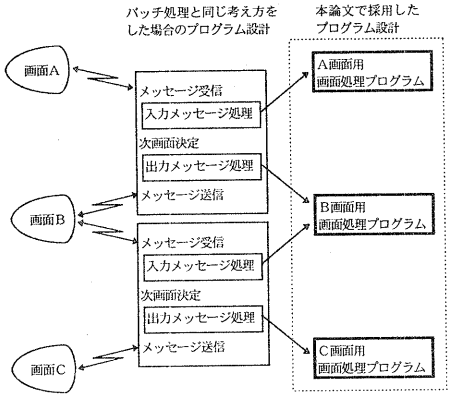


図４．２　プログラム単位の考え方

データ処理部には、画面だけではなく、ＤＢやテーブルなど種々のオブジェクト単位にデータ処理プログラムを持たせる。制御系処理部は、処理の順序を管理し、次に起動するプログラムを呼び出す。

制御系処理部は、データ処理プログラムの構造を工夫することで、バッチ系システムのＪＣＬに相当する柔軟な制御を実現した。以降、これについて説明する。

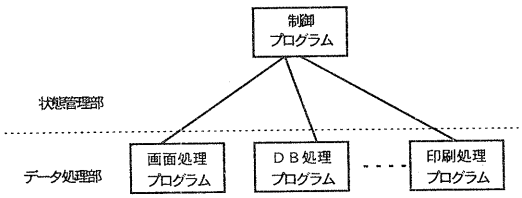


図４．３　トランザクション系のプログラム構造

４．２　データ処理プログラムの分割方法

状態遷移制御の仕組みを説明する前に、これと関係があるデータ処理プログラムの分割方法について説明する。

バッチ系処理プログラムの設計では、プログラム構造を複雑にする最大の要因をループ構造であるととらえ、これをパターンで吸収するように設計した。トランザクション系処理のメッセージ処理プログラムは、処理の単位が単一の当該トランザクションであるため、プログラム内にループ構造が存在せず、単純な構造をとる。このプログラムは、画面を対象とする場合は、出力データの編集と入力データのチェック・編集のみを、帳票を対象とする場合は出力データの編集のみを行う。

一方、制御系処理部には、画面遷移に伴うループ構造が存在し、システム設計およびプログラム構造を複雑にする。そこで、制御系処理部を完成部品として修正・追加不用とした。

メッセージ処理部分のプログラム分割方法は、バッチ系処理プログラムと同様に、主たる入出力媒体に着目した。画面をファイルや帳票に置き換えて考えるとわかりやすい。住民記録システムにおける住民情報検索の一連の処理の流れを例に説明する。

図４．４は、「住民情報を検索する」処理を、主要な入力と出力データの流れとして図示したものであり、MyersのＳＴＳ分割技法に準じた考えである。MyersのＳＴＳ分割により、分断点「世帯番号」「検索結果」を定める。これは、世帯情報ＤＢの検索条件「世帯番号」を受け取って内容をチェックし、「世帯番号」を一時的ファイル(対話処理で使用するテーブル類も入出力データと考える)へ出力するプログラムと、「世帯番号」データを入力してＤＢを検索し、検索結果テーブルへ出力するプログラム、検索結果テーブルを読み込んで画面に出力するプログラムの３本のプログラムに分割することに等しい。

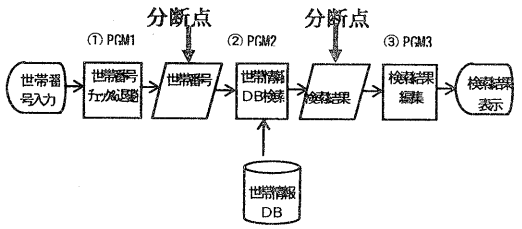


図４．４ プログラム分割の例

上記では、「住民情報を検索する」という上位モジュールが、図４．４に示すような下位モジュールの連鎖に置き換えられた。このように階層的な展開を繰り返しにより、プログラム分割を図った。

４．３ 状態遷移制御のしくみ

以下に、状態遷移制御の仕組みについて説明する。

柔軟な制御の実現手段として、状態遷移制御プログラムの実行順序をプログラム遷移表として定義する。図４．４の「住民情報の検索」処理のプログラム遷移表は表４．１のようになる。

表４．１ 「住民情報の検索」処理のプログラム遷移定義

ルート名称	プログラム名						
ルート1							
：							
ルートn	世帯番号入力 (PGM1)	世帯情報DB検索 (PGM2)	世帯情報表示 (PGM3)				
：							

＊ルート：画面遷移を構成する基本的な遷移の単位

図４．４では、検索式を入力後、すぐに検索結果が編集･表示されるように単純化した。しかし、通常は複数データがヒットするものである。したがって、検索結果の表示においては、まずヒットしたデータの一覧表を表示し、その中から操作者の選択した１データを編集して出力するのが一般的である。さらに、ヒットしたデータが１件ならば、一覧表示画面を出力することなく、ヒットした１件を直接出力する要求が強い。

ここで、図４．４の処理に対して、上記に示したような仕様変更をした場合を考えてみる。

変更作業は、図４．５に示すように「一覧表示」プログラムを１本を追加するだけで済む。「検索結果編集」プログラム(PGM3)は、常に１件のデータを対象としたプログラムでよい。

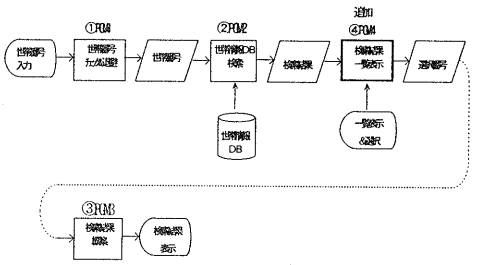


図４．５ 仕様変更後のプログラムの流れ

これだけの修正で仕様変更ができるのは、「一覧表示」プログラム自身が、自プログラムの置かれた状態から自発的に判断するようにしたからである。すなわち、ヒット件数が１件の場合には、「一覧表示」プログラム(PGM4)は、無条件にその１件が選択された状態であると判断し、検索結果一覧画面を表示せずに処理を終了すると同時に、状態遷移制御プログラムに画面の表示がないことを知らせる。この連絡を受けた状態遷移制御プログラムは「検索結果編集」プログラム(PGM3)を起動する。

つまり、静的に定義したプログラム遷移表でも、人の操作あるいはＤＢの状態などにしたがい、システムに動的な振る舞いをさせるようにした。この実現は、データ処理プログラムを、使用するデータの有無を判断し、自発的に起動するようにしたことに因る。このために、データ処理プログラムには、状態遷移制御を支援するための要素を持たせた。

この状態遷移制御の仕組みでは、システム変更で生じる画面、ＤＢ、帳票または状態等に対する処理の追加・削除・修正が、他の処理プログラムへ影響を及ぼすことなく行なえる。

4．4 データ処理プログラムパターンの分類

以下に、データ処理プログラムのパターンをどように分類したかを説明する。

プログラムの構造に影響を与える因子を、データ処理対象装置と状態遷移制御の処理の２つと考え、この観点からパターンの分類を行なった。

（1）データ処理対象装置による分類

４．１で述べたように、データ処理プログラムは、オブジェクトに対応したデータ処理を行なう。そのため、オブジェクト単位にパターンを用意する。

①処理対象装置：画面　　画面処理パターン
②処理対象装置：帳票　　帳票出力パターン
③処理対象装置：ＤＢ　　ＤＢ処理パターン
④処理対象装置：テーブル　テーブル処理パターン
⑤処理対象装置：他のプロセス
（他ＣＰＵ、他業務プログラム）
処理依頼パターン

①の処理対象装置が画面の場合のみ、入力に人間が関係するため、一旦コンピュータ制御が切れる。

つまりＯＬＴＰの制約上、別トランザクションとなる。このために、プログラム構造が特異になるが、他の処理対象装置のプログラム構造は同一となり、１つのパターンで賄うことができる。

（2）状態遷移制御の処理による分類

状態遷移の制御には２つの処理がある。

一つは、入力された情報や貯えられた情報により次に表示する画面を決定する処理である。例えば、メニュー画面から一つのメニューを選択するような場合が該当する。この処理のより、画面処理プログラムの構造は、画面遷移上、次に表示する画面が１画面しかないもの、複数画面が存在し、表示する画面が決定されるものに分けられる。本論文では、前者を単純構造、後者をインデックス構造と呼ぶ。

もう一つは、自分自身が起動された時に、前画面から渡された情報によって、自らの処理を実行するかしないか(画面を表示するかしないか)を判断する処理である。例えば、図４．５の検索一覧表示画面がこれに該当する。この処理をスキップ処理と呼ぶ。

上記(1)(2)の因子によって、データ処理プログラムは表４．２のように分類できる。

＊　表示した画面からの入力結果により画面の流れを制御するのではなく、それまでの操作の蓄積結果、あるいはＤＢ、テーブルの内容により制御する場合、分岐先の決定だけの処理パターンとして設けた。

表４．２　データ処理プログラムの分類

		状態遷移制御に関する因子			
		構造		スキップ	
		単純構造	インデックス	あり	なし
データ処理対象装置	画面	○		○	
		○			○
			○	○	
			○		○
			*○		
	帳票			○	
					○
	ＤＢ			○	
					○
	テーブル			○	
					○
	その他			○	
					○

4．5 データ処理プログラムの設計

前節で分類したプログラムパターンに対して、内部設計を行なった。

図４．６に画面処理プログラムの構造を示す。プログラムは、画面の出力編集処理、入力データのチェック＆編集処理、エラーメッセージの編集処理の３つから構成される。

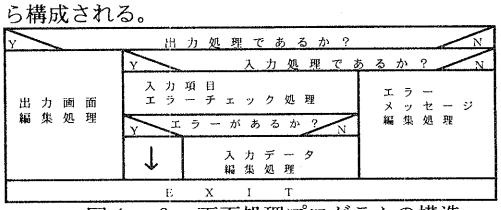
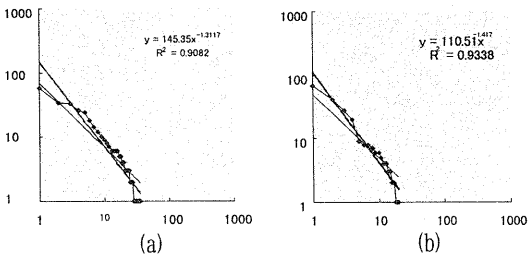
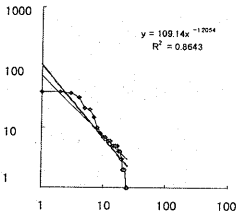


図４．６　画面処理プログラムの構造

5．プログラムパターン適用結果の考察

業務アプリケーション開発におけるバッチ系プログラムのプログラムパターン毎の使用頻度を調査した。結果の一部を図５．１(a)～(c)に示す。図５．１は、プログラムパターン毎の使用回数をカウントし、それを昇順に並べたときの順位を求め、縦軸を使用回数、横軸を順位とした両対数グラフ上にプロットしたものである。





(c)

図５．１　プログラムパターンの使用回数と順位の関係

図５．１より、２つのプロジェクトにおけるグラフの各曲線の近似式を求めたところ、(1)～(3)を得た。

(a)の近似式　$y = 145.35x^{-1.3}$　(相関係数：0.91)　(1)
(b)の近似式　$y = 110.51x^{-1.4}$　(相関係数：0.94)　(2)
(c)の近似式　$y = 109.14x^{-1.2}$　(相関係数：0.86)　(3)

これより、プログラムの使用状況は(4)式で表される分布に従うと考えた。

$y = a * x^b$　(a　b：定数)　(4)

ここで、プログラムパターンの使用頻度が式(4)で表されるデータと仮定し、このデータについて、a=100として、縦軸を使用回数の累計の百分率、横軸をプログラムパターンの種類としたグラフ上にプロットする。その結果を図５．２に示す。

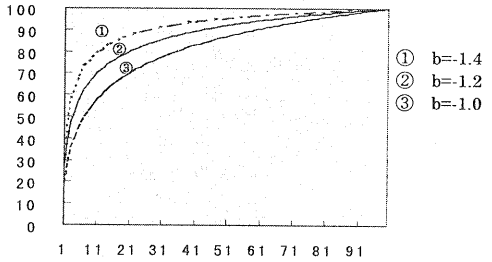


図５．２　プログラムパターンの使用回数累計と順位の関係

ａの値によって変わるのはｙ切片の値であり、ｂの値によって変わるのはグラフの傾きである。図５．１で示した３つのプロジェクトのｂの値が-1.2～-1.4の間にあったため、図５．１で示したプロジェクトのデータは、ｂの値が-1.2と-1.4の２つの曲線の間にあると考えられる。

プログラムパターンの種類を多くするほど、仕様に適合する確率が高くなり、かつ、作り込む量が少なくなる反面、適切なプログラムパターンの検索や選択に知識を要し、工数を費やす。したがって、プログラムパターンの汎用性と数のバランスにより、最適なプログラムパターンの数が決まる。少ないプログラムパターンで、要求仕様をカバーできることが理想である。

図５．２は、ｂの値による相違はあるものの、ｂの値が1.0以上であれば、使用頻度の高い20種類前後のプログラムパターンで全体の７割以上をカバーできることを示すものである。これは、我々の経験則に一致する。

一般に、我々が１プロジェクトで開発するバッチプログラム数は、200ks、300本程度である。この場合のプログラムパターンの使用状況においては、使用頻度の高い５～６種類のプログラムパターンが全プログラムの70～80％程度を占めているとの結果を経験則から得ている。業務内容の相違により、使用するプログラムパターンの種類は異なるが、約20種程度で大凡の業務のプログラム仕様はカバーできるとこのグラフからも言える。

以上を踏まえて図５．２のグラフを考えると、バッチプログラムパターンは実際には22種類しかないため、ｘの取りうる値は22までしか存在しないが、作成したプログラムパターンがx=22までに相当すると推測できると判断した。

したがって、今回作成したバッチ系プログラムパターン22種類は、バッチ系プログラムの90％程度をカバーしており、プログラムパターンの標準としては妥当と考えている。

６．むすび

本論文では、構造化技術に基づいて処理パターンを標準化し、プログラムパターンを作成する考え方を説明した。そして、作成したプログラムパターンの業務アプリケーション開発の適用結果から、プログラムパターンが"標準"のパターンとして妥当性を証明できたと考える。

７．参考文献

[1] Glenford　J.Myers　邦訳　国友,伊藤訳：ｿﾌﾄｳｴｱの複合/構造化設計, 近代科学社, 1991
[2] Ohno, O.：Development and Evaluation of Structured　Software Development System "EAGLE/P(CANDO)", COMPINT'85,1985
[3] Ohno, O.："EAGLE/P" A　Program Synthesizing　System　Using　Original Components, COMPSAC'87,1987,pp.306-310
[4] Ohno, O.：Development of CASE on the Basis of Program Design by Means of "Software Bus", JCSE'93,1993,pp.351-356
[5] Morioka, Y.：Productivity Analysis of Software Development　with　an　Integrated　CASE　Tool, ICSE14, 1992, pp. 49-58
[6] 小野, 大野, 天明：構造化ソフトウエア開発システム"ＣＡＮＤＯ", 日立評論 66, 3, 195-198, 1984
[7] 森岡, 降旗：データ中心アプローチを応用した標準データ項目辞書の開発とその活用方法, 日立評論, 75, 11, 29-34, 1993
[8] 大野, 松本, 小谷：ＥＡＧＬＥにおけるシステム設計支援システムの開発, 日立評論 68, 5, 39-42, 1986
[9] 大野, 森岡, 松本：EAGLE/P (CANDO) オリジナル部品を用いたプログラム自動合成, ソフトウエア工学分科会, 52-11, 1987
[10] 大野, 石田：データ中心型ソフトウエア開発技法, 電気学会論文誌 C, Vol. 114-C, pp. 636-644, 1994