研究発表

事務処理プログラム作成の体系化とFAST 1

井田昌之*

企業体内では，業務の発展に伴って事務システムの改善が頻繁に行なわれ，それに応じてEDP部門では対応するソフトウェアの更新に絶えず忙殺されている．本論文では，各EDPプログラムをレイアウト（ファイル編成情報・データ構造の記述），ロジックユニット（入出力，分類・突合せなどを扱うためのプログラム骨組構造・技法の記述）および項目処理部（各プログラム固有の項目処理形態・手続きの記述）の3部分に分割して，プログラミング作業を合理化する方法とそのためのソフトウェアシステムの構成を示す．この思想で作成されたFAST 1システムは，富士銀行システム開発部で使われ，その結果上記3部分に対応した形での分業が行なわれて，プログラマの負担も著しく軽減されプログラムの組織的な生産・管理が可能となっている．

A Proposal for the Systemization of EDP Software Production and FAST1 System

Masayuki IDA

Frequent refinements of work systems make their EDP section busy to update the corresponding softwares. This paper presents a method for integrated programming and its implementation on computer, provided that each program consists of three components: layout (file and data structure), logic unit (program skeleton) and item handling (procedure or options to characterize each program). With this concept, the system named FAST1 was implemented for Fuji Bank and goes well through the division of labor which has been organized corresponding to FAST1-structure. It enables programmer's task much reduced. Thus, the systematic production and control of software are now made possible.

1. まえがき

企業体内の各種システムのコンピュータ・ソフトウェアは新規作成のみでなく，維持管理にかなりの比重がおかれている．コンピュータ白書（1973）でもソフトウェアに要する人件費のうち56%が維持に費やされていると報告されており，プログラムに対する仕様の変化ないしは追加は，その手工業的性格とからんでソフトウェア管理の問題をいっそう複雑なものにしている．

そこで，「各企業体のEDP部門では，類似プログラムの反復作成がみられがちである」という認識を糸口にして，記述の容易さとプログラミング環境の整備について研究を行ない成果が得られたのでここに報告する．

その過程のなかで，富士銀行業務管理部よりプログラミング部門の省力化対策が依頼され，そのためにFAST1（Fuji Aoyama Structual Translator 1）を作成した．FAST 1は，プログラマの負担を軽減するため，ファイル構造の定義および入出力ループを中心とする骨組構造を分離し，おのおの独立したスタッフを置きプログラミング機能の3部門への分化を実現させている．この体制はシステムのもつプログラム自動生成機能（その機構は文献〔1〕に報告した）により可能となっている．

2. ソフトウェアの類型化と機能分化

2.1 既存プログラムの調査——類型化

昭和47年夏に**表1**に示す2,240本のプログラムの類型化調査を行なった．

まず，**表2**の規則を設定し入出力装置の使用区分による集計を行ない**図1**を得た．13種のパターンで全体の95%を占め，なかでも上位5種のみで83.2%を占めている．

さらに，文献〔2〕，〔3〕を参考に12の内容的パターンを設定し，論理機能による分類を行なった．レポート印刷，データ変換，併合・照合，分類，マスター更新，編集，レコード抽出，データチェック，照合印刷，抽出印刷，演算，マスタ更新チェックの12種で88.4%を占めた（詳細省略）．しかし他は類型化不可能であり，さらにデータ変換・編集・演算などはこれをもとに分類するほどの強い基準を提供しえないことがいえ，これら二つの調査から次の2点が指摘された．

1) 論理構造の多様性：単能のプログラムは少なく，性質を一意に決定するのは無理がある．

2) 類型化可能な入出力処理：単純な入出力パター

* 青山学院大学 (Aoyama Gakuin University)
昭和50年度秋季研究発表会にて発表
受付：昭和52年3月1日

表1　プログラム調査対象（富士銀行提供）

種　類	サブシステム数	プログラム本数	比率（本数）
銀行業務	12	1,316	58.8%
付随業務	6	415	18.5%
行内事務	13	509	22.7%
計	31　システム	2,240	100.0%

表2　集計規則

No.	内　　容
1	入出力の表記方法として$\alpha \rightarrow \beta$と記す．ここでαは入力デバイス，βは出力デバイスを示す．
2	廃棄予定の近いシステムを考慮対象外とする
3	パラメータカードの有無は無視する
4	MT，ドラム，カードファイルの同一視
5	希少パターンの一般化による統合 （**図1**中の Conversion は規則4の対象とならない周辺装置間の1入力1出力型処理を統合）
6	プリント出力有無の同一視

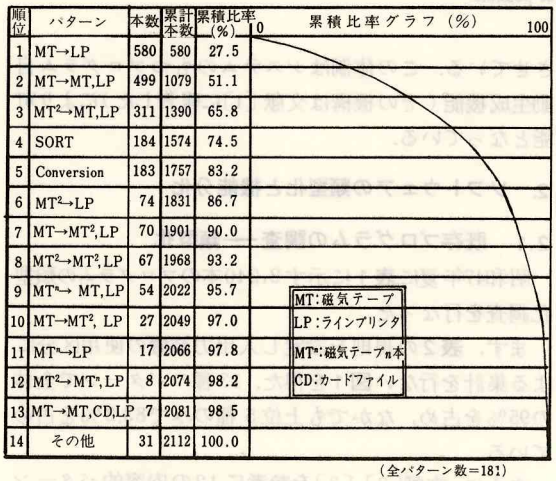

図1　入出力パターン調査結果

ンのプログラムが多い．

こうした性質をもつ手工業的生産形態による既存プログラムは短いコーディングで効率のよい仕事ができる特徴がある．これを生かして省力化し，ソフトウェア全体を再編成するための一案として次の2点が指摘できる．

1) 入出力パターンを複数個に標準化する（ただし，入出力パターンの拡張性に対する補助手段は提供する）．

2) 各プログラムのもつ複数の機能に対しては，標準化でなく，生産性・管理性をあげるための補助手段を提供する．

2.2　プログラムの3要素への分割と人的資源の有効構成

ドキュメントの整備をはじめとして，ソフトウェアの管理・作成体制が多くの企業で実施されているが，デザイン・管理・コード化という機能分化のもとでは，プログラマの負担は依然として大きい．この原因として次の二つが指摘できる．

1) 一つのプログラムを複数の人間がコード化することは事実上不可能で，プログラムの良否は担当者の個人的な能力に依存する．

2) 頻繁に要求されるプログラムの修正・改良を遂行するため担当者は内容の異なる複数本のプログラムを並行して作成・変更する事態がしばしば生じ，その過程は個人の力量に依存する．

この「超職業的プログラマによるEDP」〔5〕を打開するためいろいろな試みがなされてきたが〔6〕〜〔9〕，これらはソフトウェア生産体系を計測可能とする目的ももっている．本研究も同様の立場をとり，個々のプログラムの柔軟性を残しつつ全体の生産体系を高効率化するような方策として，各プログラムの機能の分化と対応する作業者の分担範囲の限定化を行なった．すなわち，プログラムの動的構造を入出力ループを主とする骨組（ロジックユニットとよぶ）と細部項目処理部（生成パラメータ群）の二つに分け，かつ静的側面としてのデータ構造（レイアウトとよぶ）をそれらから分離する．おのおのは独立して管理・維持され，プログラムの作成はこれらの結合作業を意味する．この形態による個々のプログラムの構造とEDP部門の組織構造は**図2**に示すように一致する．

3.　体系的プログラミング

3.1　FAST 1 システム構成

FAST1〔10〕はいくつかのサブシステムにより構成されているが（**図3**），その核となるのはプログラムジェネレータで，これは生成指示情報および細部処理用の言語形式によるコーディングを読み取り，これを変換して指示されたロジックユニットの展開のなかに組み込み，目的プログラムを生成する（**表3,4**）（付録2）．

データ構造・ファイル編成情報等の登録・修正などの管理操作はレイアウトハンドラを使って行なわれる．また，ロジックユニットの登録・修正などはロジックユニットハンドラを使って行なわれる（付録1）．

修得の容易さのために**表8**に示す仕様はマクロ定義

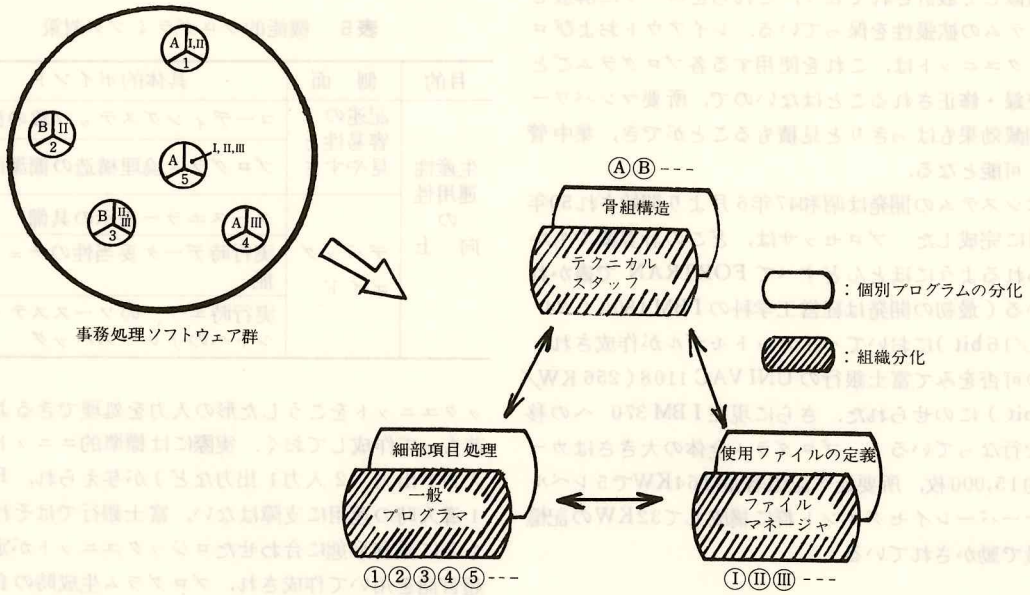

図2　機能の3分割の対応

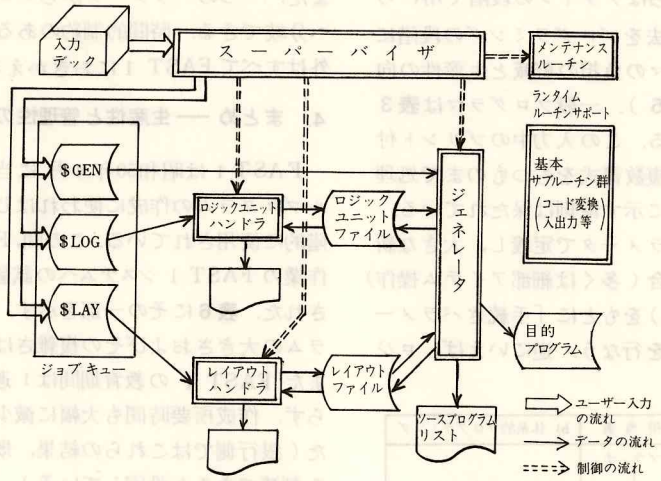

図3　FAST 1 システム構成図

表3　ジェネレータ入力

順序	内　　容　（*印は付加的）
1	プログラム名
2	使用するロジックユニットの指定
3	レイアウトおよび外部ファイル名の指定
4	プリント付加機能に対する記述 *
5	記号パラメータの指定 *〔キーワードパラメータに相当〕
6	手続きパラメータ（＊PROC）*〔複数可〕
7	＊PROC用サブルーチン（＊SUBPROC）*

表4　手続きパラメータの構成

書　式	フリーフォーマット，セミコロンで区切りステートメントを記述する．
ステートメント	15種類（IF文，GO文など）．不要な入出力文以外は完備．
変　数	レイアウトに登録された名前・配列名，プリント付加機能で与えた名前，システム変数（ユニットとパラメータ間の共通変数），ワーク変数・配列

に類似して設計されており，これらをユーザに解放しシステムの拡張性を保っている．レイアウトおよびロジックユニットは，これを使用する各プログラムごとに登録・修正されることはないので，所要マンパワーの削減効果もはっきりと見積もることができ，集中管理も可能となる．

本システムの開発は昭和47年6月より開始され50年4月に完成した．プロセッサは，どこの計算機でもかけられるようにほとんどすべてFORTRANで書かれている（最初の開発は経営工学科のIBM 1800（24 KW／16 bit）においてパイロットモデルが作成され，その可否をみて富士銀行のUNIVAC 1108（256 KW／36 bit）にのせられた．さらに現在IBM 370への移行を行なっている）．プログラム全体の大きさはカード約15,000枚，所要延べ記憶容量は64KWで5レベルのオーバーレイセグメント群に構成して32KWの記憶容量で動かされている．

3.2 トップダウンプログラミングと機能の分担

FAST 1におけるプログラミングは**図4**（b）に示す手順で行なわれる．それらはデザインの段階で用いられてきたトップダウン手法をプログラミングの段階にとり入れ，一般プログラマの負担の軽減と生産性の向上を目的としている（**表5**）．一般プログラマは**表3**に示す入力をコード化する．この入力中のプリント付加機能は，1ページ中に複数書式をもつものまで処理することができ，**表2**(6)に示す原則は保たれている．

選択的な手順は記号パラメータで定義し，大きな新しい概念を付加したい場合（多くは細部アイテム操作）は設定された言語（**表4**）をもとに「手続きパラメータ」としてコーディングを行なう．逆にいえば，ロジックユニットをこうした形の入力を処理できるように前もって作成しておく．実際には標準的ユニット（1入力1出力，2入力1出力など）が与えられ，FAST 1導入時の運用に支障はない．富士銀行ではそれらをもとに業務形態に合わせたロジックユニットが定義用超言語を用いて作成され，プログラム生成時の負担のいっそうの軽減と，集中化の強化が行なわれた．

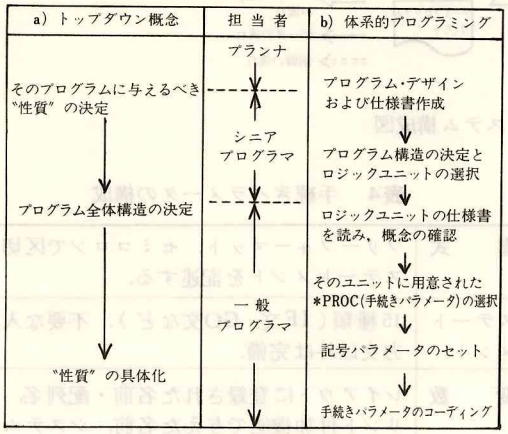

図4 FAST 1でのプログラミング

表5 機能的プログラミング対策

目的	側面	具体的ポイント
生産性運用性の向上	記述の容易性と見やすさ	コーディングステップ数の減少化
		プログラム論理構造の簡潔化
	デバッグエイド	ソースエラー表示の具備
		実行時データ妥当性のチェック機能
		実行時エラーのソースステートメントへのトレースバック

手続きパラメータの入力しうる箇所は複数個許され，また，一つのパラメータからユニット中の任意の場所へ分岐できる．時間的制約のある実時間プログラム以外はすべてFAST 1におきかえることができる．

4. まとめ ── 生産性と管理性の向上

FAST 1は昭和50年4月に当該係によってサンプルプログラムの作成に使われはじめ，50年下期より段階的に使用されている．この間，FORTRANによる現行作業のFAST 1システムへの試験移行と比較がくり返された．**表6**にその一部を示す．分業化によりプログラムの大きさおよびその複雑さは明らかに減少する．またFAST 1の教育期間は1週間であるにもかかわらず，作成所要時間も大幅に減少することが認められた（銀行側ではこれらの結果，既存体制の3割に工数を削減できると設定している）．

通常の高級言語においても見られるように手続きパラメータのコンパイルにはランタイムルーチンのサポートを利用している（**図3**）．これはデータ処理実行時の各種のチェックを目的としているが，反面実行時間およびメモリ数が増加する．

たとえばMOVE文（二つのエリア間の転送）は単純な転送の場合はこうしたルーチンを用いないコードを生成しているが，それにもかかわらず61.1%がランタイムルーチンを用いている[1]．このことは逆に，i）コード変換を含む操作，ii）テーブル操作（配列データ），が多いことを示している．

所要時間の若干の増加は，通常のプログラムの場合

表6　プログラム調査結果

項目 / 指標 / プログラム	プログラムサイズおよび簡潔性		作成所要時間			実行速度・大きさ	
	ステップ数	分岐ステートメント比率	デザイン・フローチャート	コーディング	デバッグ	CPU時間	所要メモリ
A	86 (150)	9.8 % (18.5)	5時間 (—)	8時間 (—)	5時間 (—)	93秒 (90)	54 KW (49)
B	89 (319)	8.9 % (12.6)	1時間 (—)	7時間 (—)	20分 (—)	45.6秒 (31.6)	17.7 KW (16)
C	105 (496)	13.9 % (15.0)	4時間 (8時間)	11時間 (48時間)	20日* (30日*)	58.7秒 (37.4)	26 KW (17)
D	161 (444)	5.6 % (11.1)	5時間 (24時間)	10時間 (40時間)	10日* (20日*)	115秒 (61)	25 KW (18)

1. 上段はFAST 1での値．下段カッコ内にFORTRANでの値を示す．
2. *印はバッチ処理のための待ち時間を含んでいる．

表7　管理面への影響

	FAST 1の特徴	管理的効果
1	ロジックユニットによる類型化	プログラム構造の簡潔化 オペレーション手順の標準化
2	レイアウトの登録制	フィールド名の統一 ファイル形式変更の容易化
3	スーパーバイザによるジョブ管理	プログラマ管理情報・システム使用状況の記録

定量的にみて無視できる程度である（付録3）．また管理面では，アンケート調査等により**表7**のような定性的な効果が確認された．とくにレイアウトおよびロジックユニットの一元管理は，それらの変更・改良を容易にし，各プログラマは意識せずに最新のものを利用できる特質をもたらした．

さらに，これらの考察をもとに約200本のプログラムからなるシステムの全面変更が約10名のプログラマによりわずか3か月で実施され，システム開発・プログラミング日程が正確になり，かつ短縮された．

事務処理ソフトウェア作成の体系化はこのFAST1に見られるようにプログラムの3要素への分割（個々の構成の明確化）と作成部門の3グループへの分割（全体構成の明確化）との一致により達成されるが，この結果をふまえてIPT等の手段を中心とする局所的構造化の研究をとり入れ，プログラムをより簡潔な表現形態へ導びくことが次のテーマと考えられる．

FAST 1 システム作成の機会は富士銀行システム開発部部長代理白鳥秋彦氏によって与えられた．論文作成に際しては経営工学科間野浩太郎教授の指導を仰いだ．

FAST 1 の構想は木村公則氏（現協栄計算センタ勤務）の助言と支援によりまとめられた．レイアウトおよびそのハンドラは木村の設計による．プログラムの調査は井田昌之・木村公則・井田いく子の3名により行なわれた．FAST 1 システムは上記3名および岡田一氏（現日産コンピュータ勤務）を中心に作成されたが，このほか森銅真喜子ほか数名の経営工学科学生の助力を仰いだ．

FAST 1 の実施・運営にあたっては森建二調査役を中心とする富士銀行スタッフの方々の熱心な支援と経営工学科横山巽子講師の助言を受けた．ここに慎しんで感謝いたします．

参考文献

〔1〕 井田昌之，木村公則：“ソフトウェアマネージメントシステム FAST 1”，情報処理学会「構造化プログラミングシンポジウム」報告集，pp.122-128，(1975)

〔2〕 事務管理編集部：“計算センタ，ソフトウェア会社におけるアプリケーション・プログラム・パッケージ一覧表”，事務管理，pp.123-139，Vol.12，No.12，(1973)

〔3〕 森田　章：“汎用ファイル処理システムSIMPSの特徴とその機能”，情報処理，pp. 325-332，Vol. 12，No. 5，(1972)

〔4〕 森本謙一：“MARK IVシステム”，事務管理，pp.120-127，Vol.12，No.5，(1973)

〔5〕 情報処理学会：“ソフトウェアエンジニアリング特集号”，情報処理，Vol.16，No.10，(1975)

〔6〕 木村　泉（編）：“GO TO論争”，*bit*，pp.346-379，Vol.7，No.5，(1975)

〔7〕 Dijkstra, E. W.：Structured Programming, Academic Press,（1970）
〔8〕 Armstrong, R. M.：Modular Programming in COBOL, John Wiley & Sons,（1973）
〔9〕 藤枝純教：「大型情報処理体系」，共立出版,（1975）
〔10〕 富士銀行：「FAST 1 解説書」，（1975）

付　録

1　ロジックユニットの作成

ロジックユニットは**表8**に従い作成する．例を**図5**に示す．**図5**は2入力1出力パターンを定義している．

表8　ロジックユニット記述用超言語

言語形式	マクロアセンブラにおけるマクロ定義に類似
ステートメント	生成ボディ（可変シンボルを用いた原形文）と生成制御文の組合せ
生成制御文	＊IF，＊GO，＊SETA，＊SETC，＊NOP，＊PREC（生成数字桁数の指定），＊PIF（パラメータの有無の条件分岐），＊ATTR（変数属性の参照），＊STSUBおよび＊EDSUB（ユニット内での**表4**によるコーディングセクションの開始および終了），＊END
可変シンボル	仮パラメータ名，内部生成用変数

```
$LOG,C 新規登録の指示（ほかに既有ユニットの一部修正,印刷など）
'NAME=PROG2TO1 ユニット名の指示
'PARM &KEY ジェネレート時に与えられるパラメータ名
'INPUT (&IN1),(&IN2) 入力に対する仮ファイル名
'OUTPUT (&OUT) 出力に対する仮ファイル名
'ATEND &9999 入出力ループを脱出した場所の文番号
'INITIAL 初期値設定部
      LOGICAL IN1REQ,IN2REQ,OUTREQ
      IN1REQ=.TRUE.
      IN2REQ=.TRUE.
'MAIN                    主ループ部
C     MAIN LOOP
&1000 IF(IN1REQ) CALL GET(&IN1)
      . . .
      IF(IN2REQ) CALL GET(&IN2)
      . . .
#1    *SETA 1
#2    *SETA 2
      IF(&KEY#1.GT.&KEY#2) GO TO &1
      IF(&KEY#1.LT.&KEY#2) GO TO &100
C     UPDATE
      *PROC1  ; KEY1 = KEY2 PROCEDURE IF DESIRED
&100  CONTINUE
      *PROC2  ; KEY1 < KEY2 PROCEDURE IF DESIRED
&1    CONTINUE
      IF(OUTREQ) CALL PUT(&OUT)
      *PROC3  ; LOOPING PROCEDURE IF DESIRED
      GO TO &1000
'FINAL                   後処理部
C     CLOSING
&9999 CALL CLOSE(&IN1,&IN2,&OUT)
      *PROC4  ; FINAL PROCEDURE IF DESIRED
      STOP
'END
```

図5　ロジックユニットの例

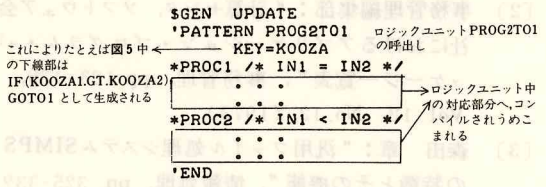

図6　ジェネレーション例

2　ジェネレーション過程

たとえばマスター更新プログラムは**図6**のようなコーディングだけでよい．＊PROCのなかには全部で100ステップ程度までの記述を与えればほとんどのプログラム要求を満たすことができる．

3　工数削減効果に関する補足

プログラミング工数を大幅に削減できるが，実行時間および所要メモリ数が増加する．メモリ数については問題がないが実行時間の増加は機械経費の増加を生じるため，この点について定量的に明らかにする．

同一のプログラムを作成するのに必要とされる所要工数 x_1（従来手法）および x_2（FAST 1）と所要CPU時間（デバッグおよび実行）y_1，y_2 の間に次の関係があれば人件費の削減効果が認められるとする．

$$a(x_1-x_2)>b(y_2-y_1),$$

a：単位人件費，b：単位機械経費，$x_1>x_2$，$y_2>y_1$

これを次のように変形する．

$$y_1/x_1<\alpha\ \gamma/\beta$$

$\alpha=(x_1-x_2)/x_1$：工数減少率

$\beta=(y_2-y_1)/y_1$：機械経費増加率

$\gamma=a/b$

$\alpha=0.7$，$\beta=0.5$，$\gamma=0.03$（$a=60$，$b=2{,}000$）とすれば上の式は次のようになる．

$$y_1/x_1<0.042$$

事務処理プログラムの実行CPU時間はたかだか3分／回くらいで，所要工数 x_1 は0.5～3人月程度である．このことより反復使用されるプログラムの場合においても y_1 はたかだか時間のオーダであり，十分与式を満足するといえる．