PAD (Problem Analysis Diagram) によるプログラムの設計および作成†

二村良彦†† 川合敏雄††
堀越彌†† 堤正義††

「コーディングは流れ図の作成から始まる (Coding begins with the drawing of the flow diagrams.)」と言ったのは，1940年代の Goldstein と Neumann だった．それ以来今日まで，プログラムを書く前に流れ図，すなわちフローチャートを書くことが，多くのプログラマの習慣になってきた．ところが，高級言語が発達したり，構造化プログラミング技法が普及するにつれ，フローチャートの欠点が目立つようになった．

フローチャートに構造化プログラミングやプログラムの段階的改良 (Stepwise Refinement) の考えを取入れた図式としては NS チャートや Ferstl チャート等が提案された．また，フローチャートを使わずに，直接，PASCAL 等の構造化プログラム言語や PDL 等のシュードコードでプログラムの論理を記述することも提案されている．しかし，これ等はフローチャートほどには広く使われていない．

われわれは，PAD (Problem Analysis Diagram, すなわち問題分析図) と呼ぶ2次元木構造をした図面によりプログラムの論理を記述する方法を提案してきた．そして，多くの機種（プログラマブル電卓から大型計算機まで）に対する各種（OS, アプリケーション等）のプログラムの開発を使用してきた．

PAD は，ワーニエ図の問題点を改良するために，

（1） 制御構造を強化し，

（2） 図式から直接コーディングできるようにし，

さらに，

（3） ハードウェアの図面のような体裁を持つように図式を改良したものである．

結果的には，PAD は，構造化プログラムを2次元的に展開した図式になった．特に PAD が標準的に備えている制御構造は PASCAL に基づいて定めてあるので，PAD は PASCAL プログラムを2次元的に展開したような図式であり，PASCAL Diagram と言うこともできる．

1. まえがき

「コーディングは流れ図の作成から始まる (Coding begins with the drawing of the flow diagrams.)」と言ったのは，1940年代の Goldstein と Neumann[1] だった．それ以来今日まで，プログラムを書く前に流れ図，すなわちフローチャートを書くことが，多くのプログラマの習慣になってきた．ところが，高級言語が発達したり，構造化プログラミング技法が普及するにつれ，フローチャートの欠点が目立つようになった[2),3)]．

フローチャートに構造化プログラミング[4)]やプログラムの段階的改良 (Stepwise Refinement)[5)] の考えを取入れた図式としては NS チャート[8),9)]や Ferstl チャート[10)]等が提案された．また，フローチャートを使わずに，直接，PASCAL 等の構造化プログラム言語や PDL 等のシュードコードでプログラムの論理を記述することも提案されている．しかし，これ等はフローチャートほどには広く使われていない．

われわれは，PAD (Problem Analysis Diagram, すなわち問題分析図) と呼ぶ2次元木構造をした図面によりプログラムの論理を記述する方法を提案してきた[15)]．そして，多くの機種（プログラマブル電卓から大型計算機まで）に対する各種 (OS, アプリケーション等) のプログラムの開発に PAD を使用してきた．

PAD は，ワーニエ図[11)]の問題点を改善する研究から生まれた[13)]．ワーニエ図はプログラムの論理を，不完全ではあるが，記述する2次元木構造である．しかし，ワーニエ図を用いて構造化プログラムを作成する場合には，次のような問題がある[13)]．

（1） 反復処理は **repeat** S **until** P 型しか書けず，**while** P **do** S, **for** $i:=m$ **to** n **do** S 等，構造化プログラム用言語でポピュラーな他の制御構造が使えない．同様に，選択処理は **if** P **then** S_1 **else** S_2 型しか書けず，**if** P **then** S, **case** 等が使えない．

（2） ワーニエ図から直接コーディングできず，フローチャートを経由せざるをえない．したがって，フローチャートに代わる論理図とは言えない．

† Disign and Implementation of Programs by Problem Analysis Diagram (PAD) by YOSHIHIKO FUTAMURA, TOSHIO KAWAI, HISASHI HORIKOSHI and MASAYOSHI TSUTSUMI (Central Research Laboratory, Hitachi Ltd.).

†† (株)日立製作所中央研究所

ワーニエ図の上述のような問題点を除去するために，(1)制御構造を強化し，(2)図式から直接コーディングできるようにし，さらに(3)ハードウェアの図面のような体裁を持つように図式を改良したものがPAD である．

ワーニエ図における制御構造の強化は Orr[12] も行っているが，上記(2)，(3)の改良を行っている図式を我々は知らない．

結果的には，PAD は，構造化プログラムを2次元的に展開した図式になった．特に PAD が標準的に備えている制御構造は PASCAL[7] に基づいて定めてあるので，PAD は PASCAL を2次元的に展開したような図式であり，PASCAL Diagram と言うこともできる．

2. PAD によるプログラム開発の原理

PAD によるプログラム開発手法の核心は，頭の中でモヤモヤしている問題解決手続を，計算機にかかるように明確化する次のような手順である(**図1**)．

(1) モヤモヤの部分の時系列的な前後関係
(2) 中心的な繰返し手続の開始条件，終了条件等
(3) 中心的な条件判定の選択条件

等を明確にすることにより，1つの大きなモヤモヤを細分化されたいくつかのモヤモヤに分割する．そして，モヤモヤが無くなるまでこの分割を繰返す．

上記の(1)～(3)の明確化による分割を各々，下記のように呼ぶ．

(1) 連接による細分化
(2) 反復による細分化

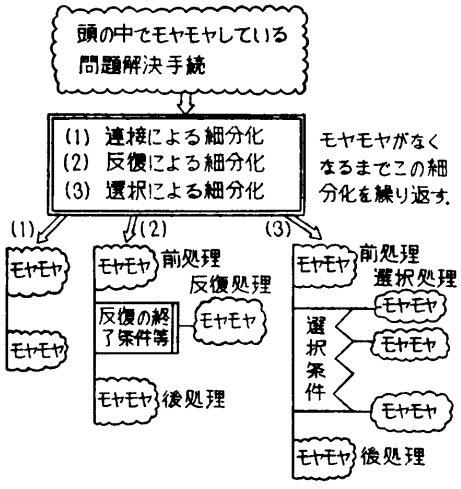


図 1 問題分析図 (PAD) によるプログラム開発の原理
Fig. 1 PAD program design principles.

(3) 選択による細分化

この原理は，topdown programming[4] あるいは stepwise refinement[5] の考え方をより明確に述べたものである．

並列プログラムを開発する場合には，上記の3つ以外に，

(4) モヤモヤの部分の並列性による細分化

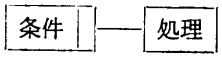
を加える必要があるが，この点については本文の後の部分で述べる．

2.1 PAD の書式

反復および選択のための条件の書き方は，特に制限していない．解こうとしている問題またはコーディングするためのプログラミング言語によって任意に定めてかまわない．しかし，標準的に，われわれは，条件の書き方を以下のように定めた．

(1) 反 復 図 式

図1の「反復による細分化」のために使うのは次の図形である．この図形を反復図式と呼ぶ．



条件の部分には，反復処理の開始条件，終了条件等を書くが，PASCAL, FORTRAN, PL/I, COBOL 等でコーディングする場合に書ける条件を**表1**および**表2**の前半のように4つ定めている．4つの形を各々 UNTIL 形，WHILE 形，DO 形および DOWNTO 形と呼ぶ．

(2) 選 択 図 式

図1の「選択による細分化」のために使うのは次の図形である(**図2**)．条件＜ の条件の部分には，処理1,…，処理 n の中の1つを選ぶための選択条件を書く．PASCAL, FORTRAN, PL/I, COBOL 等のプログラマにとって便利な4種類の選択条件を，表1および表2の後半に示した．それ等の型を，IF-THEN-ELSE 形，IF-THEN 形，算術 IF 形，および計算 GOTO 形と呼ぶ．

PAD の書き方の基本規則は上の2つだけである．

次に，例題を用いて PAD の記述例を示す．

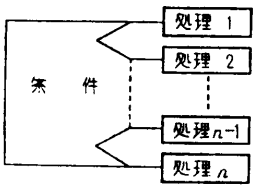


図 2 選択図式
Fig. 2 Selection diagram.

表 1 PAD の標準図式 (PASCAL 用).
Table 1 PAD standard diagrams (PASCAL).

		PADの標準図式	PASCAL	フローチャート
反復図式	UNTIL形	反復処理 until Q H	repeat H until Q;	H Q̄ Q
	WHILE形	反復処理 while Q H	while Q do H;	Q̄ Q H
	DO形	反復処理 i:=m to n H	for i:=m to n do H;	i:=m; i>n / i≦n; H; i:=i+1
	DOWN TO形	反復処理 i:=m downto n H	for i:=m downto n do H;	i:=m; i<n / i≧n; H; i:=i-1
選択図式	IF THEN ELSE形	選択処理1 H1; Q 選択処理2 H2	if Q then H1 else H2;	Q̄ Q; H2 H1
	IF THEN形	選択処理 H; Q	if Q then H;	Q̄ Q; H
	算術IF形	選択処理1 H1; e<0 選択処理2 H2; e=0 選択処理3 H3; e>0	if e<0 then H1 else if e=0 then H2 else H3;	e<0 / e≧0; e=0 / e>0; H1 H2 H3
	計算GOTO形	選択処理1 H1; i= L1 L2 … Ln 選択処理n Hn	case i of L1: H1; … Ln: Hn end;	i=L1 / i=Ln; L1 … Ln; H1 … Hn

図 3 ICHI-NI 問題の PAD
Fig. 3 PAD for the first-second problem.

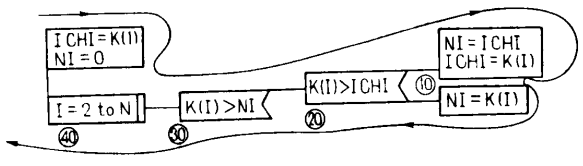


図 4 ツリーウォーク (tree walk) および文番号記入例
Fig. 4 Example of a tree walk and numbering.

〔例題 1〕 配列 K(1), K(2),…K(N) に N 個の相異なる正数が入っている．このうちの最大の数 (ICHI) と 2 番目に大きな数 (NI) を求める PAD を作成せよ．ただし N≧2 とする．

この例題に対する PAD を図 3 に示した．

2.2 コーディングの仕方

PAD に基づいてプログラムを書く方法を，FORTRAN を例として説明する．

PAD の標準図式には **GOTO** という考えは現れないが，FORTRAN に翻訳する場合には，**DO** 文，**GOTO** 文等のための文番号を PAD 中に付ける必要がある．文番号は，次の手順で付ける．

(1) PAD 完成後，表 2 に基づき，必要な箇所に○を記入する (図 3 に現れた PAD では，図 4 のように○を記入する)．

(2) PAD を横に生えた木と考え，例えば図 4 のように木をたどりながら，10 から 10 きざみに○の中に番号を付ける．木構造のこのようなたどり方をツリー・ウォーク (tree walk) と呼ぶ．

次に，やはりツリー・ウォークにより PAD をたどりながら，次の規則によって FORTRAN プログラムを作る．

(1) 処理の部 □ に出会ったら，その中身を書く．

(2) 文番号，例えば，100，に出会ったら，

100 CONTINUE

を書く．

(3) 反復図式または選択図式に出会ったら，表 2 にしたがって **DO** 文，**IF** 文，**GOTO** 文等を書き，ツリー・ウォークを続ける．

以上の方法でコーディングしたプログラムは，図 5 のように，文番号が上から小さい順に並べられる．

PASCAL, PL/I, COBOL 等でコーディン

```
      ICHI=K (1)
      NI    =0
      DO  40  I=2, N
      IF    (K(I). LE. NI) GOTO  30
      IF    (K(I). LE. ICHI) GOTO  10
      NI    =ICHI
      ICHI=K(I)
      GOTO  20
   10 CONTINUE
      NI    =K(I)
   20  CONTINUE
   30  CONTINUE
   40  CONTINUE
```

図 5 図 4 の PAD に対する FORTRAN プログラム
Fig. 5 FORTRAN program for the first-second problem.

グする場合は，上述のツリー・ウォークと表1または2に基づき，FORTRAN の場合より容易にできる．アセンブラ等の低級言語の場合は，表2と同様の PAD 対アセンブラの対応表を作り，FORTRANの 場合と同様，文番号を PAD に記入してからコーディングをする．

2.3 初期テストの仕方

PAD の木のすべての葉（先端）を通るテストをすれば，プログラムのすべてのステートメントは少なくとも一度はテストされる．このテストは「全ステートメント (all statements)」と呼ばれるテスト基準を満足する．「全ステートメント」を満足したからといって，プログラムの正しさが保証されるわけではない[16]．しかし，これは最低限やるべきテストである．以下に，例題によりそのテストを体系的に行う方法を示す．

〔例題2〕 A(1), A(2), ···, A(L) に入っているデータを，与えられた k の値に従って，小さい順 (K=1 のとき) または大きい順 (K≠1のとき) に並べ替えるソートのためのテストデータの表（テストケース表：**表3参照**）を作成せよ．

ソートのプログラムの PAD を**図6**に示した．PAD のすべての葉の部分に 10 から 10 番おきに番号がふってある．ステートメントのない葉の部分 (20, 40, 70, 90) にも番号がふってあることに注意されたい．

この PAD にふられたすべての番号に対応するステートメントブロックをテストする入力データを見つけるために，表3のような表を作成する．まず，なるべく多くのブロックを通る入力データを決め，表の項番1の入力データの欄に記入する．次に，その入力に対

表 2 PAD の標準図式
Table 2 PAD standard diagrams (FORTRAN, PL/I, COBOL).

		PADの標準図式	FORTRAN	PL/I	COBOL
反復図式	UNTIL形	⑩ until Q H ⑪	L CONTINUE H IF(Q̄) GOTO L	L: H; IF ¬Q GO TO L;	PERFORM H. PERFORM H UNTIL Q. (Hはパラグラフ名，以下同様)
	WHILE形	⑩ while Q H ⑫	L1 CONTINUE IF(Q) GO TO L2 H GO TO L1 L2 CONTINUE	DO WHILE Q; H; END;	PERFORM H UNTIL (NOT Q)
	DO形	I:=M to N H ⑩	DO L I=M,N H L CONTINUE	DO I=M TO N BY 1; H; END;	PERFORM H VARYING I FROM M BY1 UNTIL I >N.
	DOWNTO形	I:=M down to N H ⑩	DO L II=N, M I=N+M-II H L CONTINUE	DO I=M TO N BY -1; H; END;	PERFORM H VARYING I FROM M BY -1 UNTIL I<N.
選択図式	IF-THEN-ELSE形	Q ⑩ H1 H2 ⑫	IF(Q̄) GO TO L1 H1 GO TO L2 L1 CONTINUE H2 L2 CONTINUE	IF Q THEN H1; ELSE H2;	IF Q H1 ELSE H2.
	IF-THEN形	Q H ⑩	IF(Q̄) GO TO L H L CONTINUE	IF Q THEN H;	IF Q THEN H.
	算術IF形	⑩ <0 H1 e=0 ⑫ H2 >0 ⑬ H3 ⑭	IF(e) L1, L2, L3 L1 CONTINUE H1 GO TO L4 L2 CONTINUE H2 GO TO L4 L3 CONTINUE H3 L4 CONTINUE		
	計算GOTO形	⑩ L1 L2 I= Ln ⑫ H1 H2 ⑬ Hn ⑩	GO TO(L1, L2,···,Ln), I L1 CONTINUE H1 GO TO L Ln CONTINUE Hn L CONTINUE	GO TO I; L1: H1; GO TO L; Ln:Hn; L:	GO TO L1, L2, ···, Ln DEPENDING ON I. L1. H1. GO TO L. Ln. En L.

(注) Q̄はQの否定を表わす．◯はコーディングの際に必要に応じて記入する．

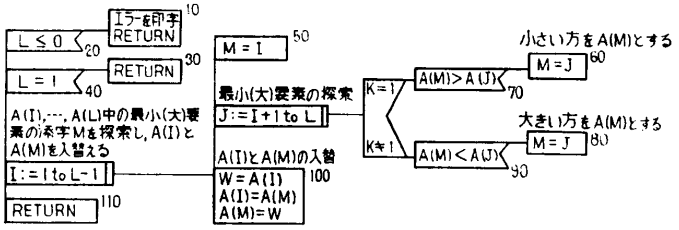


図 6 ソートプログラムの PAD
Fig. 6 PAD for sorting A(1),..., A(L).

表 3 テストケース表
Table 3 Test case table.

項番	テストケース		検査されるブロック	未検査ブロック	検査完了日	
	入力データ	期待される出力データ			机上	マシン上
1	K=1; L=3; A(1)=7; A(2)=9; A(3)=5	A(1)=5; A(2)=7; A(3)=9	20, 40, 50, 60, 70, 100, 110	10, 30, 80, 90	54, 06, 25	
2	K=2, L=3, A(1)～A(3) は同上	A(1)=9; A(2)=7; A(3)=5	20, 40, 50, 80, 90, 100, 110	10, 30	54, 06, 25	
3	K=1, L=1, A(1)=7	A(1)=7	20, 30	10	54, 06, 25	
4	K=1, L=0	エラーメッセージ	10	なし	54, 06, 25	

して期待される出力データを表に記入する．それから，入力データに対してPADをトレースし（すなわち，机上チェックをし），通ったブロックの番号を「検査されるブロック」欄に記入する．残りのブロックの番号は「未検査ブロック欄」に記入する．次には，なるべく多くの未検査ブロックを通るような入力データを見つけ，同様の作業を行い，未検査ブロックが無くなるまでこの作業を繰返す．上例の場合には，4つの入力データについてテストをすれば，「全ステートメント」のテストができることが表3よりわかる．

正確には上述のテストは，全ステートメントのみならず，全判定条件のチェック (individual predicate)[16] にもなっている．したがって，例えば上例のソートプログラムの1番外側のループの繰返し条件 I: =1 **to** L-1 を誤って I:=0 **to** L-1 とすると，項番1または2のテストケースで，出力結果が期待したものとちがうものとなる．そこで，バグの存在が明らかになる．

ここで述べたテスト法はあくまでも初期テストであるので，プログラムに要求される信頼性にしたがって，追加のテストをする必要がある．

2.4 PADの書き方に関する注意

ある処理（例えばA）をPADに追加しようとしたときに，その処理が同一の用紙（ページ）に収まりそうもないときは，その処理Aを的確に表わす短い説明または名前(例えば，NAME A) を付けて，そのページには**図7**(a)のように NAME A と書いておく．そして他のページに図7(b)のように二重線 def (defは省略しても良い) を使って，NAME A の詳細を書けば良い．

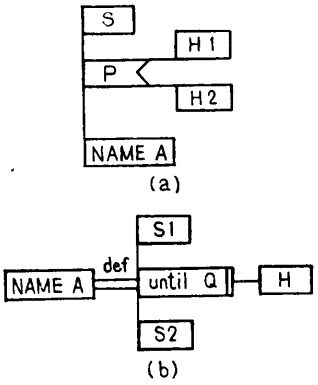


図 7 def の使用例1
Fig. 7 Example 1 for def.

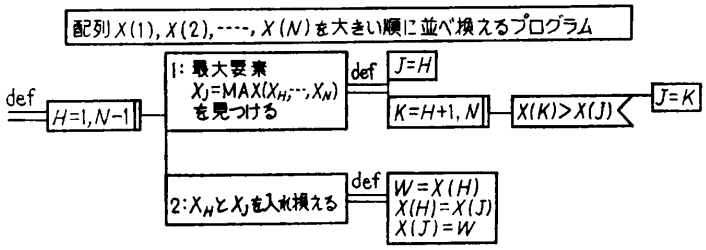


図 8 def の使用例2
Fig. 8 Example 2 for def.

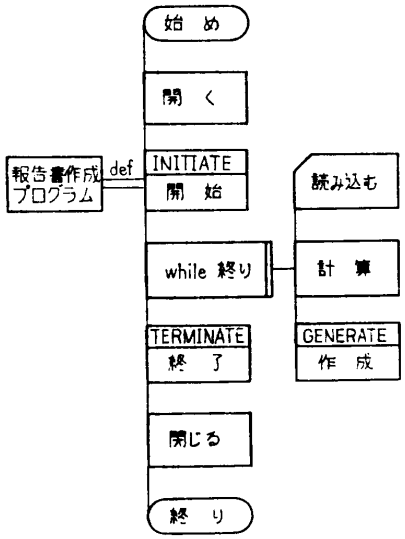


図 9 JIS 記号を使った PAD
Fig. 9 PAD using flowchart symbols.

def の用法としては，はみ出し対策だけではなく，もっと積極的な次の2つがある．

(1) PADをトップダウンに設計した過程が理解し易いように，段階的改良の各ステップを明確に示す(**図8**)．

(2) PADのサイズが大きくなり，理解しにくくならないように，適当な大きさで押える．

PADでは，繰返し，判定，処理，および定義を示す4つの記号，すなわち，□，◁，▭，def の他は構造を示す縦横の線しか規定していない．したがって，以上の記法および後で述べる N+1/2 ループと並列処理の記法に矛盾しなければ，どんな記法でもPADに書いてよい．例えば，JISの流れ図記号に慣れているプログラマは，判断，流れ線，結合子および並行処理以外のJIS流れ図記号を使用してかまわない(**図9**)．

3. PADによるデータ構造の記述

PADでは，プログラムの制御構造を

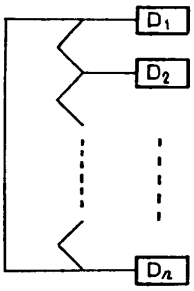


図 10 選択形データ構造
Fig. 10 Data structure of selection type.

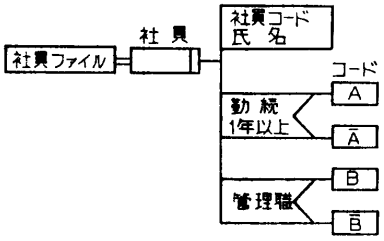


図 11 データ構造の例
Fig. 11 Example of data structure.

記述するのと同じ原則により，反復形と選択形の図式により，データの論理構造を記述することができる．

（1） 反復形の基本図式

例えば，Dという型のデータが複数個（N個）ある場合には | N | — | D | と書く．Nが不明の場合には省略してもかまわない．

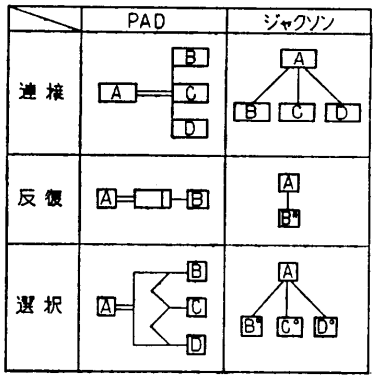
（2） 選択形の基本図式

例えば，$D_1, D_2, \cdots, D_n (2 \leq N)$ の型のデータのうちいずれか1つがある場合には**図 10** のように書く．

PAD によるデータ構造の記述例を**図 11** に示した．

問題分析に際して，処理しようとしているデータの論理構造の解明が重要な手掛りとなることが知られている[6]．PAD を用いれば，ワーニエやジャクソン[17]

表 4 データ構造の図式表現の比較
Table 4 Comparison of PAD and Jackson's tree.



等と同様に，データ構造に基づいたプログラムを作成することができる．ちなみに，ジャクソン流のデータ構造記述法と PAD との比較を**表 4** に示した．

4. PAD の拡張図式

より複雑なループや能率を重視したプログラムを書く場合に使う，N+1/2 ループや制限付き **GOTO**，および無限ループや並列処理の記述のためには，次の図式を用意してある．

4.1 N+1/2 ループ（表 5 参照）

4.2 制限付き GOTO

3つの制限付きの **GOTO** 文に，その性質を表わす名前 ENTER, EXIT, および MERGE を付け，PAD における使用を認める（**表 6**）．コーディングの際には，これ等はもちろん **GOTO** 文になる．これ等の使用例を**図 12** に示した．

4.3 無限ループと並列処理

PAD の標準図式を使って無限ループを書こうとすると | while true | — | H | のようにするしか手がなく，無限ループといった感じがでない．そこで，

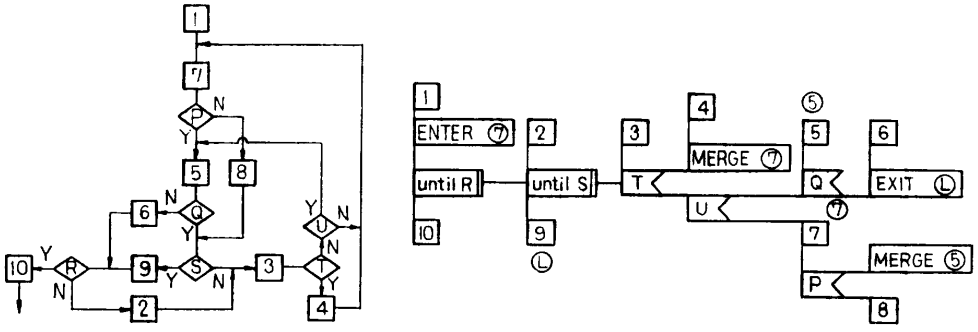
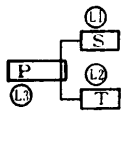
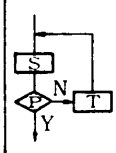


図 12 複雑なフローチャートの PAD への変換例
Fig. 12 Example of flowchart to PAD conversion.

表 5 N+1/2 ループの記述法
Table 5 N+1/2 loop.

PAD	フローチャート	PASCAL	FORTRAN

		*L1*:*S*;
		if *P* then
		begin
		T;
		goto *L1*
		end;
			L1 CONTINUE
			S
			IF (P) GOTO L3
			T
			GOTO L1
			L3 CONTINUE

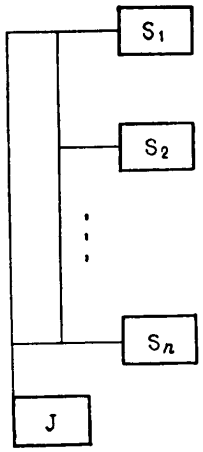


図 13 $S_1, S_2, \cdots, S_n$ を並列に処理し，それ等がすべて終了した後で J を処理する．
Fig. 13 Do $S_1, \cdots, S_n$ parallel, then do J.

無限ループは次のように書くことにする．

そして，これをコーディングするときには，

L : H ; **goto** L ;

とする．

並列処理はもう少し複雑である．$S_1, S_2, \cdots, S_n$ の処理が並列に行われ，すべての処理の終了後に処理 J を行うことを**図 13** の図式で表わす．ただし，並列に走る処理が多数あり，しかも処理内容がほとんど同じ場合には，**図 14**（a）の省略形を使うのが良い．

〔並列処理の例題 1〕 2つの N 次元ベクトル A と B の内積 $A(1)\times B(1)+\cdots+A\times B(N)$ を 2 台のプロセッサを用いて並列計算する．すなわち，一方のプロセッサでは $A(1)\times B(1)+\cdots+A(N/2)\times B(N/2)$ を計算し，同時に他方のプロセッサでは $A(N/2+1)\times B(N/2+1)+\cdots+A(N)\times B(N)$ を計算する．両者の計算が終了したら，両者の結果の和をとる．

このことを PAD では，**図 15** のように書く．

〔並列処理の例題 2〕 N 次元の正方行列 A, B の積

表 6 制限付き GOTO
Table 6 Restricted GOTO.

書式	機能	使用制限
ENTER ⓛ	ループの入口を指定する．すなわち，ループに入るときには，ラベルⓛのついた場所から計算を開始する	自分より下にあるループにしか入れない（図 12 参照）
EXIT ⓛ	ループの出口を指定する．すなわち，ループから出て，ラベルⓛのついた場所から計算を開始する	自分より下にある場所にしか出られない（図 12 参照）
MERGE ⓛ	1つの枝から，ラベルのついた枝に制限を移す	自分を含む 1 番小さいループ内でしか制御を移せない（図 12）．しかも，これによりループを構成してはならない

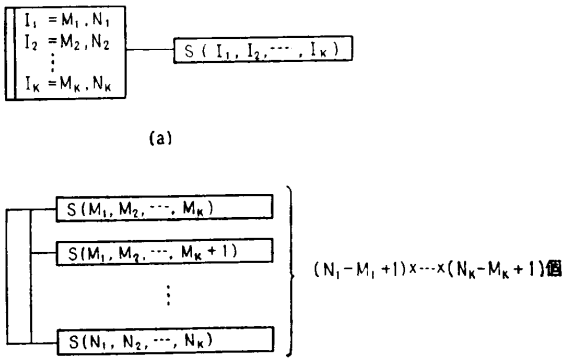


図 14 （a）は（b）の省略形（図 16 参照）
Fig. 14 (a) is an abbriviation for (b) (See Fig. 16).

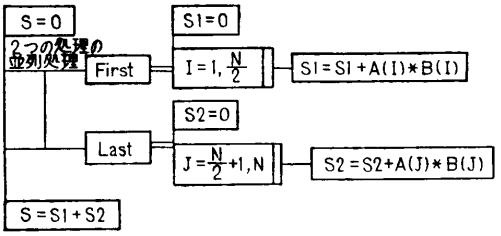


図 15 N 次元ベクトル A, B の内積を，2 台のプロセッサを用いて並列計算する PAD
Fig. 15 Compute inner product of A and B using two processors.

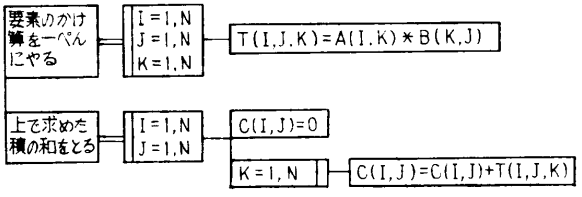


図 16 N 次元の行列 A, B の積 C=AB を，N^3 台のプロセッサを同時に使って計算するプログラム
Fig. 16 Compute C=AB using N^3 processors.

を計算し，Cとする処理を N^3 台のプロセッサを使って行う PAD を**図16**に示した．

5. PAD の有効性

プログラムの生産技術の効果の測定には時間がかかるものである．例えば，Dijkstra の "Notes on Structured Programming" が世の中に出てから，構造化プログラミングの効果を多くの人に納得させる定量的データ[22]が公表されるまでには，5年以上の歳月を要している．PAD の有効性の定量的評価は今後の課題であるが，われわれはとりあえず，次のような主観的な評価を行った．

新しい物が世に受け入れられるためにはその物が持つ「フィーリングの良さ」が大切である．そこで，まず，「PAD を使ってプログラムを作成すると，それ以前と比べてどれだけプログラムの生産性が向上したと思うか」という「PAD によるプログラム生産性向上感」を調べた．実務で使用するプログラムを PAD を用いて開発している6チーム（12名）を選び，2週間に一度「生産性向上感」を報告していただいた．作成したプログラムの種類は，OS のスケジューラ，テキストエディタ，図形処理プログラム，数値計算ライブラリ等の一部である．（使用言語は，PL/I, FORTRAN, アセンブラ等）．**表7**には，プログラム開発の各フェーズにおける生産性向上感をまとめた（3カ月分）．表7中，1というのは，在来手法（HIPO，フローチャート等を用いる構造化プログラム技法）とほとんど同じという意味である．在来手法と比べて PAD の方が生産性が下がったと思えば1より小さく，良くなったと思えば1より大きな数を回答していただいた．したがって，「3」という数字は「今までより3倍ぐらい生産性が向上したと感じた」ということを意味する．

表7 PAD による生産性向上感の評価
Table 7 Evaluation of PAD.

開発フェーズ	評価		
	平均	最高	最低
機能設計	1.5	2	1
論理設計	1.6	3	1
テストケース分析	1.9	2.5	1.1
コーディング	2.0	3	1.3
デバッグ	2.6	3	1
修正・変更	1.4	2	1

表7の数値は，あくまでも PAD ユーザの「フィーリング」を示すものである．実際に生産性がそれだけ向上したことは意味しないが，最低の評価でも「在来手法と同じ程度」ということは，PAD の質の良さを示すものと考える．表7のデータ量は多くないが，PAD 講習会の受講者やその他の PAD ユーザの感想は，大体この数値と合っている．

上の評価で，「修正・変更」が在来手法より良くなっているのは，次の理由によると思われる．

（1） バグが減少したため，修正の量が減少した．

（2） PAD は，HIPO やフローチャートより図形として単純である（すなわち，線が少ない）．

（3） PAD は，根本（すなわち左）の方に判定・操作等を押入する際はフローチャートよりやっかいな場合があるが，葉（すなわち右）の方での修正はフローチャートより容易である．

6. むすび

PAD のオリジンはワーニエ図と PASCAL であるが，結果だけ見ると PAD と類似の木図面は，日本では，1979年7月の情報処理全国大会やその他で発表されている[14),18)~20)]．また外国では Ferstl チャート[10)]や Warnier-Orr 図[12)]が知られている．これ等の木図面の比較は文献 15) で行った．

われわれは PAD を開発し，普及を図ってきた．これまでに，ソフト開発の辛苦を実際に体験された多くの方に，PAD の良さを深くご理解いただいている．プログラム作成法に関する新しい技術は文献 21) にあるように，なかなか受入れられないのが普通である．その割には，PAD は受入れられるスピードが速いように感じている．国の内外で多くの研究者が，プログラム論理図の木構造化を研究されているので，PAD のようなプログラムの木図面が広く普及する日は遠くないと思っている．

PAD の開発と普及には日立中研の津田順司主任研究員，小沢時典主任研究員，桑原裕計算センター長，高橋栄主任研究員および日立京浜工学専門学院の藤田勝彦助教授他，多くの方々が心から協力して下さった．これ等の方々のご理解とご協力がなければ，PAD のような新しいプログラム作成技法が日の目を見る機会はなかったと思う．

最後に日頃ご指導いただいている日立中研の村田健郎技師長および PAD 普及活動に協力を惜しまない日立中研の柴田さゆり氏に感謝する．

参考文献

1) Goldstein, H. H. and von Neumann : Planning and coding problems for an electronic computing instrument, part II, in von Neumann Collected Works Vol. V, McMillan, New York, pp. 80-151.
2) Shneiderman, B., Mayer, R., McKay, D. and Heller, P.: Experimental investigations of the utility of detailed flowcharts in programming, Commun. ACM, Vol. 20, No. 6, pp. 373-381 (1977)
3) Leer, V.: Top-down development using a program design language, IBM Syst. J. Vol. 2, pp. 155-170 (1976).
4) Dahl, O.-J., Dijkstra, E. W. and Hoare, C. A. R.: Structured Programming, Academic Press (1972).
5) Wirth, N.: Systematic Programming; An Introduction, Prentice-Hall (日本語訳：野下浩平, 筧 捷彦, 武市正人訳：系統的プログラミング／入門, 近代科学社) (1973).
6) Wirth, N.: Algorithms+Data Structures=Programs, Prentice-Hall (1976).
7) Jensen, K. and Wirth, N.: PASCAL User Manual and Report, Springer-Verlag (1974).
8) Chapin, N.: New format for flowcharts, Software: Practice and Experience Vol. 4, No. 4, pp. 341-357 (1974).
9) Nassi, I. and Shneiderman, B.: Flowchart Techniques for Structured Programming, SIGPLAN Notices Vol. 8, pp. 12-26 (Aug. 1973).
10) Ferstl, O.: Flowcharting by Stepwise Refinement, SIGPLAN Notices Vol. 13, No. 1, pp. 34-42 (Jan. 1978).
11) J. D. ワーニエ：ワーニエプログラミング法則集, 鈴木君子訳, 日本能率協会 (1975),
12) Verdegraal, P. A. and Goodman, A. S.: The Warnier-Orr Diagram, Digest of Papers, COMPCON 79, IEEE Catalog No. 79, CH 1393-8 C, pp. 301-306.
13) 二村：構造的プログラム作成のための一方法, 電子通信学会総合全国大会予稿 S 8-9, 昭和53年3月.
14) 二村, 川合, 堤：問題分析図によるプログラムの設計と作成, 情報処理学会第20回全国大会予稿 2 I-1 (54年7月).
15) 二村, 川合：PADによるプログラムの開発, bit, 3月, 4月および5月号, 55年共立出版.
16) Goodenough, J. B. and Gehart, S. L.: Toward a Theory of Test Data Selection, IEEE Trans. of Software Engineering, Vol. SE-1, No. 2 (June 1975).
17) Jackson, M. A.: Principles of Program Design, Academic Press (1975).
18) 夜久, 二木：フローチャートの木構造型記法, 電子通信学会 AL 研究会, AL 78-47 (1978).
19) 大原, 野島, 前田：フローチャートの木構造化の一提案, 電子通信学会総合全国大会予稿, 1499 (54年3月).
20) 佐藤, 浅見：フローチャートの階層的表現のための一提案, 情報処理学会第20回全国大会予稿 2 I-9 (54年7月).
21) 宮本：プログラム生産の場からソフトウェア工学に望む事柄, 電気四学会連合大会講演論文集 (54年10月).
22) Baker, F. T.: Structured programming in a production programming, IEEE Trans. of Software Engineering (June 1975).

(昭和54年6月29日受付)
(昭和55年4月22日採録)