

## 1.8 Problem Analysis Diagram (PAD)\*

Yoshihiko FUTAMURA\*\* and Toshio KAWAI\*\*\*

### *Abstract*

This paper presents a limited number of diagram types and their construction rules that are used for depicting program structures. The diagramming method presented is called the PAD (Problem Analysis Diagram) and aims at:

- (1) Visualizing program structures;
- (2) Establishing systematic rules for coding and testing programs; and
- (3) Describing data structures in the same manner as program structures.

This method is proposed as a functionally superior substitute for flowcharts. The fact that some 10 000 Japanese programmers have converted from conventional charts to PAD within the last four years is indicative of the broad potential utility of this notation.

### 1.8.1 Introduction

It was Goldstein and Neumann in the 1940's who said "Coding begins with the drawing of the flow diagrams".<sup>9)</sup> Since then, it has been the custom of most programmers to draw flowcharts before programming.

However, defects in flowcharts have developed with the advent of higher level programming languages and structured programming.<sup>20)</sup> Improvements in programming charts that have been proposed include NS chart,<sup>14)</sup> Ferstl chart<sup>4)</sup> and many others.<sup>2,5,17)</sup> These incorporate notions of structured programming and stepwise refinement.<sup>3,25)</sup> Methods of describing program logic directly by structured language (as with PASCAL) or by pseudo-code (as with PDL) without using flowcharts have also been proposed.<sup>1)</sup> However, these methods are not as widely used as flowcharting. This paper presents a new diagramming method, the PAD (Problem Analysis Diagram), that aims at:

- (1) Visualizing program structures;
- (2) Establishing systematic rules for coding and testing programs; and
- (3) Describing data structures in the same manner as program structures.

The last two features, (2) and (3), were original to PAD, to the best of our knowledge, when it was first published in 1979 (in Japanese)<sup>6)</sup> and in 1981 (in English).<sup>7)</sup> PAD has evolved as an improvement of the Warnier Diagram.<sup>24)</sup> The resulting diagram coincides with a diagram expressing the structured language. PAD might also be interpreted as PAscal Diagram, since

---

\* This paper is a revised and extended version of the paper presented at the 5th International Conference on Software Engineering (1981).

\*\* Central Research Laboratory, Hitachi Ltd., Kokubunji, Tokyo 185.

\*\*\* Faculty of Science and Engineering, Keio University, Yokohama 223.

the original control structures for PAD were defined based on PASCAL.<sup>12)</sup>

PAD has been applied to work in developing various programs (OS, application programs, etc.) for machines ranging from programmable desk calculators to large computers. It has also been evaluated as being superior to conventional charts<sup>7)</sup> and PDL.<sup>15)</sup>

### 1.8.2 Comparison of Diagramming Methods

It is a well known theorem in computer science that any computation can be described in three basic structures: sequencing, repetition and selection. These are shown both in PAD and flowchart in Fig. 1.8.1. Figure 1.8.2 shows an example PAD and the equivalent flowchart. Based upon the theorem, it is reasonable to define a program structure as the combination of the basic structures. Therefore, when we say a program structure is visible, it means that relationships between basic structures are easily visible.

Flowcharts are readable when they describe such very simple structures as are shown in Figs.1.8.1 and 1.8.2. Let us look at the more complex structures shown in Fig. 1.8.3. Although they represent identical computations, two flowcharts in Fig. 1.8.3 look different. The diamond 3 in the upper chart looks repetitive, while the one in the lower chart looks selective.

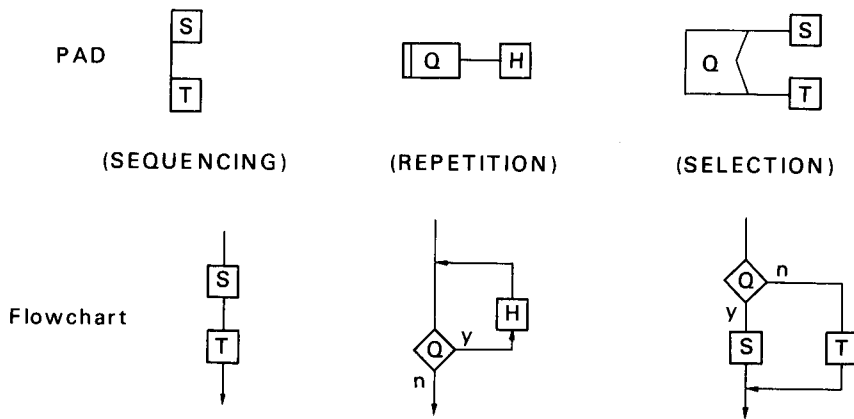


Fig. 1.8.1 Three basic program structures.

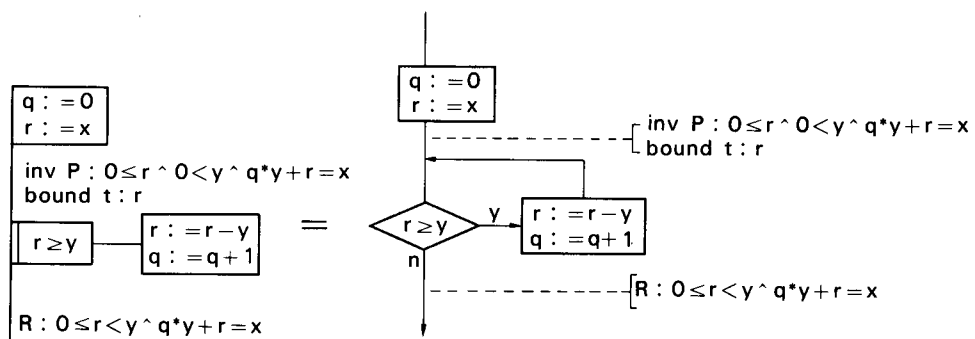


Fig. 1.8.2 Example PAD and equivalent flowchart. These both compute quotient  $q$  and remainder  $r$  for  $x/y$  (Sentences outside PAD are comments).

This is evidence for the statement that the relationships between basic structures used in flowcharts are hardly visible. Another conclusion from this example is that identical computations can be described in completely different looking diagrams. Styles of diagrams tend to depend on the programmers who draw them.

In PAD on the contrary, relationships between basic diagrams are very clear and the computation of Fig. 1.8.3 can be uniquely described as Fig. 1.8.4. Therefore we may assert that:

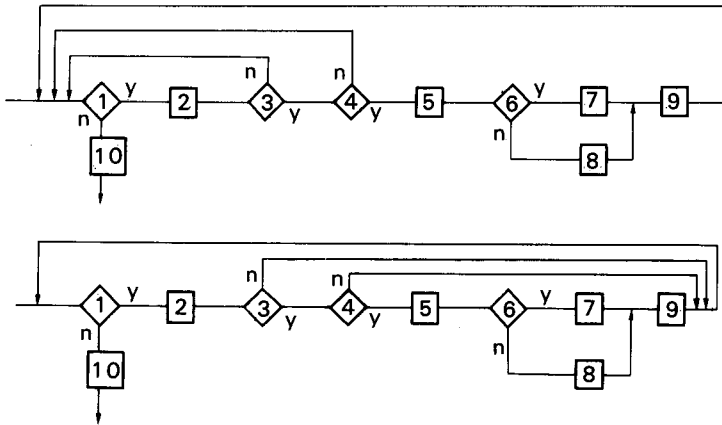


Fig. 1.8.3 Relationship between basic structures are not visible in flowcharts.

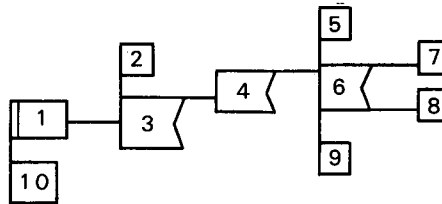


Fig. 1.8.4 Program structure is visible in PAD.

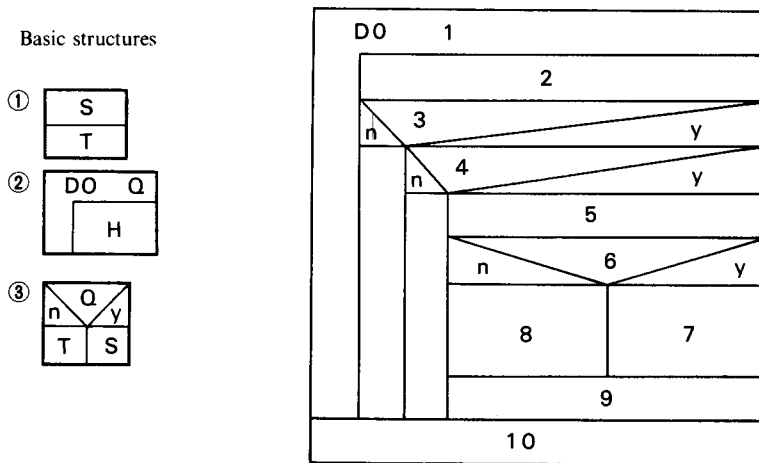


Fig. 1.8.5 Example NS chart.

- (1) Program structures are more visible in PAD than in flowcharts; and
- (2) Personal differences in diagramming style are less with PAD than with flowchart.

Figure 1.8.5 shows an NS chart equivalence of the PAD in Fig. 1.8.4. NS chart can claim the above two points against flowcharts as PAD. However, since NS chart is a nesting of rectangles, the correction of the chart is not very easy.

Figure 1.8.6 shows the PDL equivalence of the PAD in Fig. 1.8.4. Basic structures are less easily visible in PDL than in PAD. The fact that PAD is more readable than PDL was reported in.<sup>15)</sup>

```

DO WHILE 1
  2
  IF 3 THEN
    IF 4 THEN
      5
      IF 6 THEN 7
        ELSE 8
      ENDIF
    9
  ENDIF
ENDIF
ENDDO
10
    
```

Fig. 1.8.6 Example pseudo-code.

### 1.8.3 Basic PAD Diagrams

Using only three basic structures, any computation can, in principle, be described. However, it is more practical to add other structures so that programs can be more easily describable and readable. Additions to PAD include problem-oriented loops and parallel processing.

#### Process



A rectangle stands for a process. A process name or various statements can be written in the frame. This rectangle can be replaced by any PAD or ISO flowchart symbol so long as PAD symbols are not contradicted.

#### Sequencing



This diagram means that  $n$  ( $n > 1$ ) processes,  $S_1, S_2, \dots, S_n$ , are performed in the specified order.

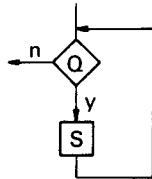
**Repetition**

There are two kinds of repetition diagrams.

- (1) Pre-judgement loop:



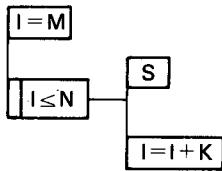
This diagram stands for the following repetition process:



Here,  $Q$  stands for a Boolean expression and  $S$  stands for an arbitrary process.

- (2) Problem-oriented loop:

There are many occasions when using only pre-judgement loops are not convenient for describing a procedure. For example, the below PAD illustrates a procedure for repeating a process,  $S$ , with variable  $I$  varying from  $M$  to  $N$  by  $K$ .



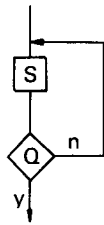
A problem-oriented loop however, permits the writing of the below PAD for the same procedure as the above.



That is, the diagram permits description of any starting and ending condition for a loop, and the value of a control variable in frame . However, when only a Boolean expression (say  $Q$ ) is written in a frame, such that

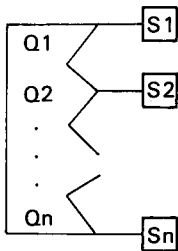


this signifies a post-judgement loop:



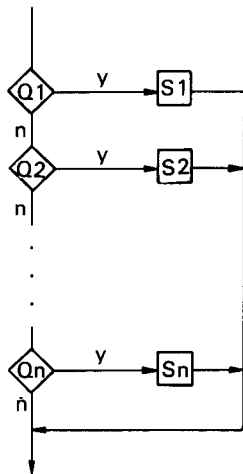
Remark) Note that we restrict the use of frame  $\square$  for a problem-oriented loop. It does not provide any definition of the expressions inside a frame except for a Boolean expression, i.e.,  $\square Q \square S$ . Therefore, whenever a new problem-oriented loop is used, its meaning must be defined explicitly, e.g. using Tables 1.8.1 and 1.8.2 shown in Section 1.8.4.

**Selection**



This diagram represents the selection of one out of  $n$  ( $n > 1$ ) processes,  $S_1, S_2, \dots, S_n$ , in accordance with  $n$  conditions  $Q_1, Q_2, \dots, Q_n$ . When  $Q_n = (Q_1 \vee Q_2 \vee \dots \vee Q_{n-1})$ , i.e.  $Q_n$  means “otherwise”, this  $Q_n$  may be omitted (here, “ $\vee$ ” stands for “logical or” and “ $\neg$ ” stands for negation).

Remark 1) If  $Q_i \wedge Q_j = \text{false}$  for all  $i, j$  ( $1 \leq i, j \leq n, i \neq j$ ), i.e. if the conditions are exclusive, this diagram is equivalent to the following flowchart (Here, “ $\wedge$ ” stands for “logical and”).



Remark 2) If  $Q_i \wedge Q_j = \text{true}$  for some  $i, j$  ( $1 \leq i, j \leq n, i \neq j$ ), i.e. if conditions are inclusive, then the selection is non-deterministic, i.e. the selection of either  $S_i$  or  $S_j$  is not determined.

Remark 3) The following two diagrams are equivalent:



<PAD>

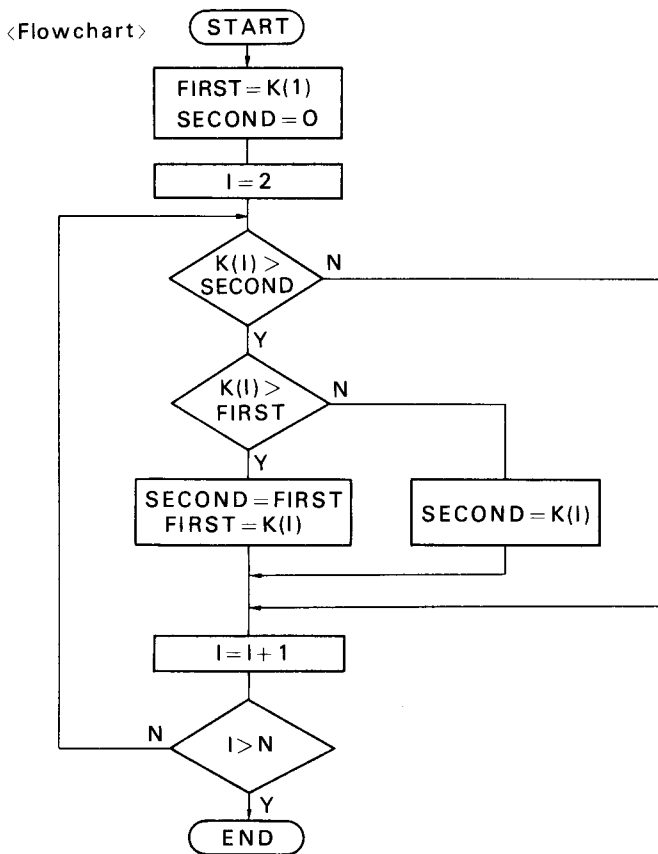
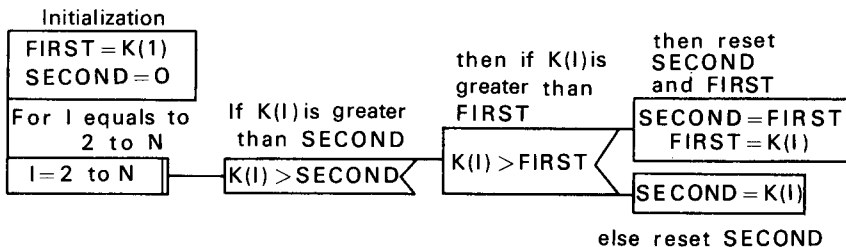
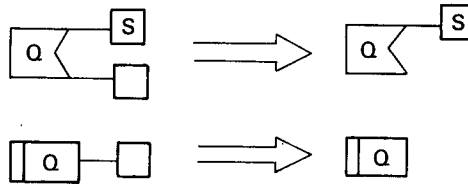


Fig. 1.8.7 Greatest integers problem. [after Y. Futamura et al.<sup>7)</sup>

Remark 4) Elimination rule: When a process frame is empty, the frame and its branch (left side horizontal line) may be eliminated. For example:



Remark 5) Commenting rule: Sentences outside PAD frames may be considered comments.

[Example 1] There are  $N$  different positive integers  $K(1), K(2), \dots, K(N)$ , where  $N > 1$ . Find the largest (FIRST) and second largest (SECOND) of the given integers (Fig. 1.8.7).

**Parallel** (Figs. 1.8.8 and 1.8.9.)

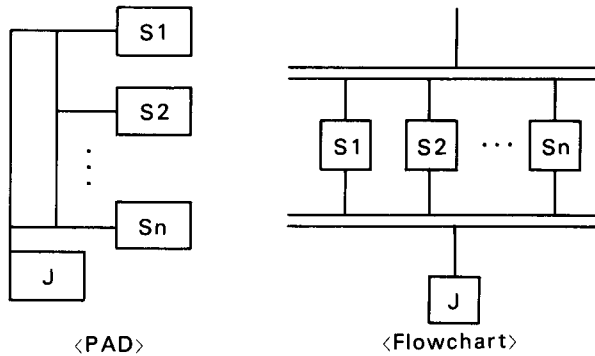


Fig. 1.8.8 Parallel diagram. These diagrams mean that processes  $S_1, S_2, \dots, S_n$  run parallel, and that after all are completed, process  $J$  starts to run.

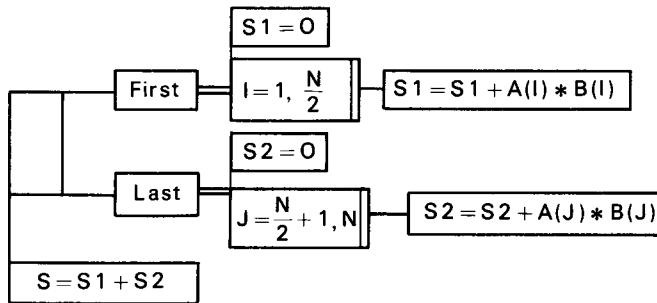


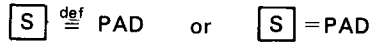
Fig. 1.8.9 Computation of inner product of  $A$  and  $B$  using two processors. [after Y. Futamura et al.<sup>7)</sup>

**Definition**

def or =



These symbols are used when a name (say S) is given to a PAD:



(See Figs. 1.8.10 and 1.8.11).

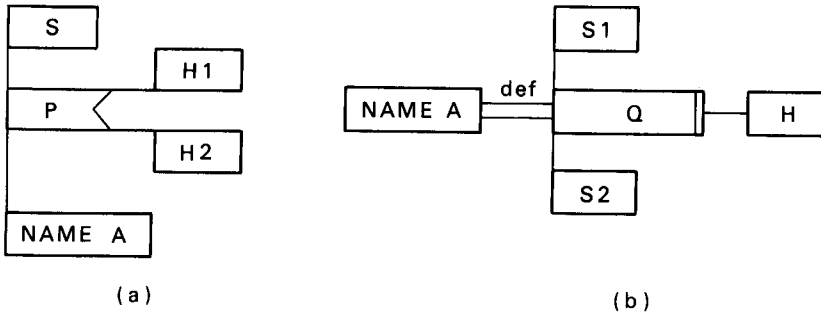


Fig. 1.8.10 Example for  $\stackrel{\text{def}}{=}$ . [after Y. Futamura et al. 7)]

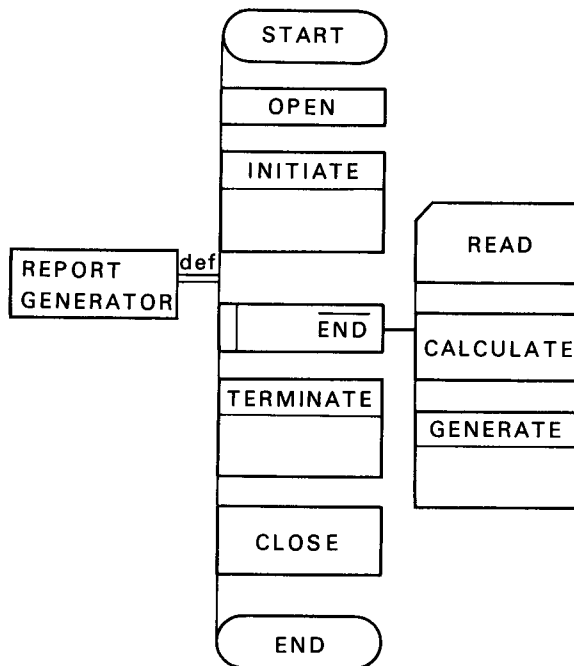


Fig. 1.8.11 PAD using flowchart symbols.

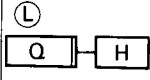
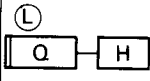
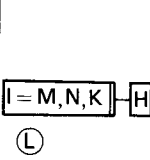
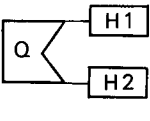
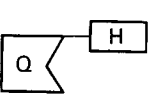
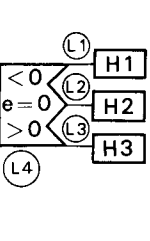
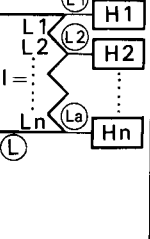
### 1.8.4 Coding Method Based on PAD

The previous chapter established PAD diagramming rules. However, without specifying the exact meanings of conditions and statements written in frames, the exact meaning of a procedure described by PAD cannot be specified either. That is, it is impossible to code programs from PAD until the semantics of the language to be used in the PAD frames is specified.

Table 1.8.1 Recommended PAD diagrams (PASCAL). [after Y. Futamura et al.<sup>7)</sup>

		PAD	PASCAL	Flowchart
REPETITION	UNTIL		repeat H until Q ;	
	WHILE		while Q do H ;	
	DO		for i := m to n do H ;	
	DOWN TO		for i := m downto n do H ;	
SELECTION	IF THEN ELSE		If Q then H1 else H2 ;	
	IF THEN		If Q then H ;	
	COMPUTED GO TO		case i of L1 : H1 ; ... Ln : Hn end ;	

Table 1.8.2 Recommended PAD diagram (FORTRAN, PL/I, COBOL). [after Y. Futamura et al.<sup>7)</sup>

		PAD	FORTRAN	PL/I	COBOL
R E P E T I T I O N	P O S T		L CONTINUE H IF (Q) GOTO L	DO UNTIL Q; H; END;	PERFORM H PERFORM H UNTIL Q (H is a paragraph name)
	P R E		L CONTINUE IF (Q) THEN H GOTO L ENDIF	DO WHILE Q; H; END;	PERFORM H UNTIL (NOT Q)
	P R O H I B I T E D		DO L I=M,N,K H L CONTINUE	DO I=M TO N BY K; H; END;	PERFORM H VARY- ING I FROM M BY K UNTIL I>N.
I F T H E N	E L S E		IF (Q) THEN H1 ELSE H2 ENDIF	IF Q THEN H1; ELSE H2;	IF Q H1. ELSE H2.
	I F T H E N		IF (Q) THEN H ENDIF	IF Q THEN H;	IF Q THEN H.
S E L E C T I O N	A R I T H M E T I C		1F(e)L1,L2,L3 L1 CONTINUE H1 GO TO L4 L2 CONTINUE H2 GO TO L4 L3 CONTINUE H3 L4 CONTINUE		
	C O M P U T E D		GO TO(L1,L2, ...,Ln),I L1 CONTINUE H1 GO TO L Ln CONTINUE Hn L CONTINUE	SELECT (I) WHEN (1) H1; WHEN (2) H2; : : WHEN (n) Hn; END;	GO TO L1,L2,..., Ln DEPENDING ON I. L1. H1. GO TO L. Ln. Hn L.

\* Note :  $\bar{Q}$  stands for negation of Q.  $\circ$  is written in the coding phase if necessary.

Therefore, it is very convenient for practical use of PAD to prepare such tables as are shown in Tables 1.8.1 and 1.8.2, so as to clarify the correspondence between PAD and the programming languages.

Using the tables, coding from PAD can be systematically performed based on the “tree walk” rules described below,

- (1) As is shown in Fig. 1.8.12, ○ is added to the PAD at the necessary locations specified in Table 1.8.2.

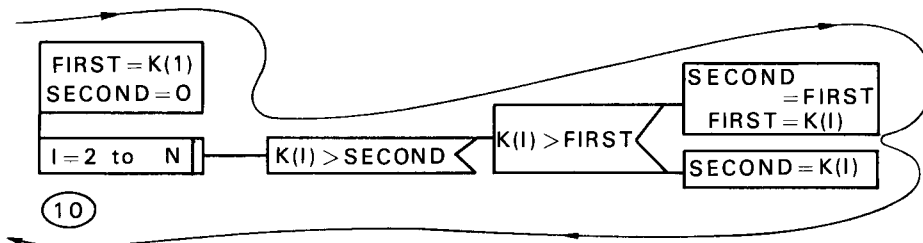


Fig. 1.8.12 Example of a tree-walk and numbering.

- (2) With PAD considered a tree, a “tree walk” is performed, as shown in Fig. 1.8.12. Successive multiples of ten are filled into ○, in the order encountered on the tree walk. Then statements are made one by one in the tree walk order, using the following rules:

- (1) When a processing “leaf”, H, is met, the statements are copied.
- (2) When a label like 100 is encountered, a statement number such as 100 CONTINUE for FORTRAN or 100 REM for BASIC is written; and
- (3) When  $\square$   $\square$  or  $\square$  is encountered, IF, DO or GOTO are written according to patterns specified in Table 1.8.2.

The program is completed when the tree walk ends. The resulting program in, for example, FORTRAN, has statement numbers arranged in order (Fig. 1.8.13).

Coding into PASCAL, PL/I or COBOL is done similarly, but more easily, using Tables 1.8.1 and 1.8.2. Coding into lower level languages, such as assembler, can be done using prepared conversion tables like Tables 1.8.1 and 1.8.2.

```

FIRST = K (1)
SECOND = 0
DO 10 I = 2, N
IF (K (I). GT. SECOND) THEN
IF (K (I). GT. FIRST) THEN
SECOND = FIRST
FIRST = K (I)
ELSE
SECOND = K (I)
ENDIF
ENDIF
10 CONTINUE
    
```

Fig. 1.8.13 FORTRAN program for greatest integers problem.

### 1.8.5 Testing Method Based on PAD

All-branch testing can be performed by passing through all leaves of a PAD tree. Program correctness is not necessarily guaranteed even if a program has passed this test. Still, it is a necessary, though not sufficient, test. PAD is a convenient instrument for systematically performing such tests as will be demonstrated below.

[Problem] The PAD in Fig. 1.8.14 shows a procedure for sorting  $A(1), A(2), \dots, A(L)$  in ascending or descending order for  $k = 1$  or  $k \neq 1$  respectively. The test data for an all-branch test must be prepared.

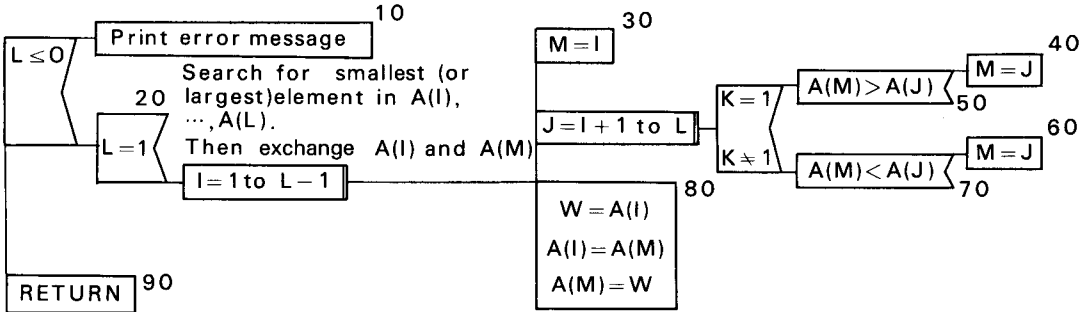


Fig. 1.8.14 PAD for sorting  $A(1), \dots, A(L)$ . [after Y. Futamura et al.<sup>7)</sup>

The PAD for the sort program is shown in Fig. 1.8.14. The leaves in the tree (9 in all) are numbered from 10 to 90. Note that the empty leaves (20, 50, 70) are also numbered.

Test input was prepared in the form of Table 1.8.3, and the procedure is as follows. First a set of data is chosen arbitrarily. The data set determines the control flow and leaves to be covered. The remaining uncovered blocks are marked in a separate column in the table. Then, another set of test data is devised to cover as many uncovered leaves as possible. This step is repeated until all leaves have been covered. In the example, 4 sets of input data were used for the all-branch testing.

Table 1.8.3 Test case table. [after Y. Futamura et al.<sup>7)</sup>

No.	Test cases		Checked blocks	Remaining blocks	Date of check	
	Input data	Correct output			on desk	on machine
1	$K=1, L=3,$ $A(1)=7$ $A(2)=9, A(3)=5$	$A(1)=5, A(2)=7,$ $A(3)=9$	30, 40, 50, 80, 90	10, 20, 60, 70	6/25/79	
2	$K=2, L=3,$ $A(1)=7$ $A(2)=9, A(3)=5$	$A(1)=9, A(2)=7,$ $A(3)=5$	30, 60, 70, 80, 90	10, 20	6/25/79	
3	$K=1, L=1,$ $A(1)=7$	$A(1)=7$	20, 90	10	6/25/79	
4	$L=0$	Error message	10	none	6/25/79	

### 1.8.6 Describing Data Structure in PAD

The data structure can be described by employing the same method used to describe control structures through taking advantage of sequence, repetition and selection.

(1) Basic diagram for repetition

N data of type D can be described as



N can be omitted if the number is unknown.

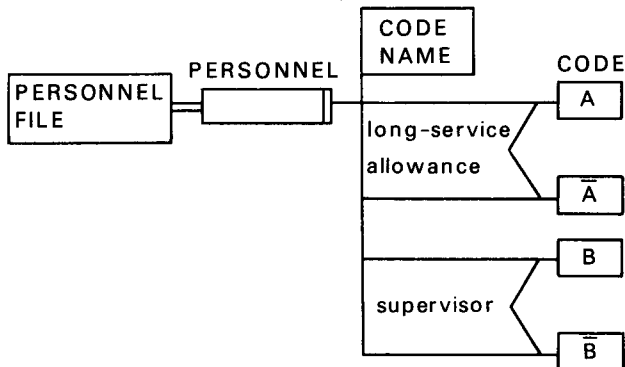


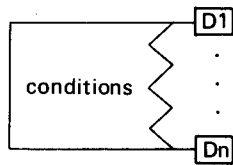
Fig. 1.8.15 Data structure example. [after Y. Futamura et al.<sup>7)</sup>

Table 1.8.4 Comparison of PAD and Jackson tree. [after Y. Futamura et al.<sup>7)</sup>

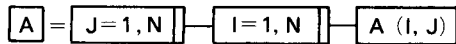
	PAD	Jackson
Sequencing		
Repetition		
Selection		

(2) Basic diagram for selection

A datum of type D1, D2,... or Dn can be described in the following manner,



An example of a data structure written in PAD is shown in Fig. 1.8.15. A two dimensional  $N \times N$  array  $A(I,J)$  can be described using problem-oriented loops as follows:



It is known that data structure can be useful as a guide to the program structure for processing that data. Use of PAD makes it possible to design programs based on data structure, just as with Warnier's or Jackson's methods.<sup>11)</sup> The correspondence between the PAD data representation method and Jackson's method is shown in Table 1.8.4

1.8.7 Extended Diagrams

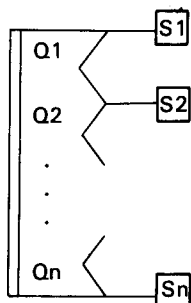
Four kinds of diagrams are introduced to concisely represent commonly used process patterns in program development. Note that they can in principle be described in basic PAD.

Mid-judgement loop (Table 1.8.5)

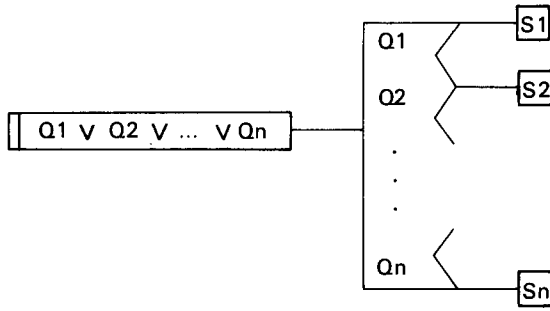
Table 1.8.5 Mid-judgement loop. [after Y. Futamura et al.<sup>7)</sup>]

PAD	Flowchart	PASCAL	FORTRAN
		<pre>L : S ;   if p̄ then   begin     T ;   goto L   end ;</pre>	<pre>L CONTINUE S IF (P) THEN T GOTO L ENDIF</pre>

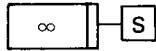
Selective loop



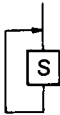
This diagram stands for the following process,



**Infinite loop**



This diagram stands for the following process,



Note that this is a type of problem-oriented loop.

**Restricted GOTO**

Three kinds of forward jumps are allowed in and out of a loop, and from one PAD branch to another without entering or leaving a loop. These are ENTER, EXIT and MERGE.

**Table 1.8.6** Restricted GOTO (forward jump). [after Y. Futamura et al.<sup>7)</sup>

Form	Function	Example
ENTER (L)	Control is transferred to (L) inside a loop from the outside.	
EXIT (L)	Control is transferred to (L) outside a loop from the inside.	
MERGE (L)	Control is transferred to a branch (L) without going in or out of any loops.	



Their meanings and example uses are shown in Table 1.8.6. They should be used carefully, because they tend to spoil PAD readability.

### 1.8.8 PAD Effectiveness

Though a quantitative evaluation of the PAD effectiveness is now being conducted, programmers have already been polled as to their evaluation of PAD with regard to productivity. It is important for a new notation to give a "good feeling" to its users in order for it to be accepted quickly. Thus PAD users were asked "How much do you feel PAD improves your program productivity?" two weeks for three months. Programmers responded by providing information about the development stage they were in and the productivity improvement ratio they felt. If they felt that their productivity was doubled, the ratio they responded with was 2. The programmers had been using HIPO and flowcharts before they used PAD, and the languages included PL/I, FORTRAN and assembly language.

Investigations were conducted at two different institutions. One was our own laboratory, where programmers could be helped in solving problems in writing PAD (there were 12 subjects from 6 projects). Results are shown in Table 1.8.7 (a). The other was one of our plants where there was no contact with the programmers (there were 36 subjects from 14 projects). The results are shown in Table 1.8.7 (b). The reason for such low scores as 0.5 and 0.7 in the table, is misuse of PAD. An example of this is the drawing and maintaining both PAD and flowcharts.

Table 1.8.7 Evaluation of PAD. [after Y. Futamura et al.<sup>7)</sup>  
(a) (b)

Development Phase	Productivity Improvement Ratio			Development Phase	Productivity Improvement Ratio		
	Average	High score	Low score		Average	High score	Low score
Functional Design	1.5	2	1	Functional Design	1.4	2	1
Logical Design	1.6	3	1	Logical Design	1.2	2	0.5
Test Case Analysis	1.9	2.5	1.1	Test Case Analysis	1.3	2	0.5
Coding	2	3	1.3	Coding	1.7	2	1.3
Test	2.6	3	1	Test	1.8	4	1
Correction/Change	1.4	2	1	Correction/Change	1.2	2	0.7

Since its announcement in August 1979, PAD has been accepted very rapidly among Japanese programmers. We estimate the number of present PAD programmers to be more than 10 000. This estimation is based on the fact that the total number of PAD templates sold exceeded 5 000 in May, 1983. The template has been sold at a price of about \$6.00 since 1980. Because of PAD simplicity, only a certain portion of PAD programmers use the templates. The other programmers get by with conventional templates or rulers. Therefore, our estimation is not an overestimation.

### 1.8.9 Activities Related to PAD

- (1) ISO is now discussing PAD for use as one of the substitutes for the ISO flowchart.<sup>10)</sup>
- (2) Many companies and research organizations, such as Hitachi,<sup>13)</sup> Fujitsu,<sup>21)</sup> University of Tokyo,<sup>19)</sup> Keio University,<sup>22)</sup> and so on, have implemented PAD tools. These include:
  - (i) A PAD compiler for translating PAD to a source language.
  - (ii) A PAD editor for inputting, correcting and writing PAD.
  - (iii) A PAD generator for assisting programmers in developing PAD (i.e. an interactive computer assisted program development system).

Some of these tools are still being improved. Therefore, we believe many PAD programmers will be benefitting from these tools soon.

- (3) We have developed a PAD design methodology called PAM (Problem Analysis Method).<sup>8)</sup> PAM has up to now been taught to more than 500 programmers, and its effectiveness and problems are currently being investigated.

### 1.8.10 Conclusion

Generally, new programming notations gain slow acceptance. However, PAD seems to be gaining in popularity very quickly. Since the announcement of PAD in August 1979, at least 10000 Japanese programmers have converted from conventional charts to PAD. Several institutions have also adopted PAD as their standard charts.

Although a quantitative evaluation of PAD has yet to be finished, it is felt that the speed of acceptance by programmers is indicative of PAD's effectiveness.

Many researchers throughout the world are conducting research on representing program structures by tree diagrams.<sup>4, 16~18)</sup> Programmers are finding usefulness in tree diagrams. Therefore, a program tree diagram like PAD should be widely accepted in the near future.

### References

- 1) Caine, S.H. and Gordon, E.K.: "PDL—a tool for software design", Proc. of the 1975 National Computer Conference, *Vol. 44* (Montvale, N.J., AFIPS Press, 1975) 271-276.
- 2) Chapin, N.: "New format for flowcharts, Software—Practice and Experience", *4, 4* (1974) 341-357.
- 3) Dahl, O., Dijkstra, E.W. and Hoare, C.A.R.: "Structured Programming", (Academic Press, 1972).
- 4) Ferstl, O.: "Flowcharting by stepwise refinement", SIGPLAN Notices, *13, 1* (January, 1978) 34-42.
- 5) Futamura, Y.: "An approach to structured programming", Proc. of National Conference of Inst. Electronics Comm. Engrs. Japan (March, 1978) [in Japanese].
- 6) Futamura, Y., Kawai, T. and Tsutsumi, M.: "Design and implementation of programs by Problem Analysis Diagram (PAD)", Proc. of National Conference of Information Processing Society of Japan (July, 1979) [in Japanese].
- 7) Futamura, Y., Kawai, T., Tsutsumi, M and Horikoshi, H.: "Development of computer programs by Problem Analysis Diagram (PAD)", Proc. of 5th International Conference on Software Engineering (New York, IEEE Computer Society, 1981)

- 325-332.
- 8) Futamura, Y.: "Program design and review through Problem Analysis Method (PAM)", *Journal of Information Processing*, 23, 4 (July, 1982) [in Japanese].
  - 9) 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) 80-151.
  - 10) ISO/TC 97/SC 7 N307 (1983).
  - 11) Jackson, M.A.: "Principles of Program Design" (Academic Press, 1975).
  - 12) Jensen, K. and Wirth, N.: "PASCAL User Manual and Report" (Springer-Verlag, 1974).
  - 13) Maezawa, H., Saito, K., Kobayashi, M. and Futamura, Y.: "Support system for structured program production — PDL/PAD", *Proc. of Feedback '82* (Ken Orr and Associates, Inc., Topeka, Kansas, 1982).
  - 14) Nassi, I. and Shneiderman, B.: "Flowchart techniques for structured programming", *SIGPLAN Notices*, 8 (August, 1974) 12-26.
  - 15) Odaki, F., Maezawa, H. and Kawasaki, Z.: "Readability evaluation: PDL and PAD", *IPSJ* (1983) [in Japanese].
  - 16) Oohara, S., Nojima, S. and Maeda, S.: "A proposal for tree-structuring of flowcharts", *Proc. of National Conference of Inst. Electronics Comm. Engrs. Japan* (March, 1979) [in Japanese].
  - 17) Peters, L.: "Software Design: Methods and Techniques" (Yourdon Press, 1981).
  - 18) Sato, T. and Asami, H.: "A proposal for a hierarchical representation of flowcharts", *Proc. of National Conference of IPSJ* (July, 1979)[in Japanese].
  - 19) Shimada, K., Fujita, A., Tanaka, H. and Moto-oka, T.: "Graphical program input system", *Proc. of National Conference of IPSJ* (October, 1981)[in Japanese].
  - 20) Shneiderman, B., Mayer, R., McKay, D. and Heller, P.: "Experimental investigations of the utility of detailed flowcharts in programming", *Commun. ACM*, 20, 6 (1977) 373-381.
  - 21) Sugimoto, M.: "Software diagram description", in this volume.
  - 22) Suzuki, R., Okada, K., Yokoyama, M. and Kitagawa, S.: "Multi-screen PAD terminal for a micro-computer", *Proc. of National Conference of IPSJ* (March, 1983) [in Japanese].
  - 23) Verdegraal, P.A. and Goodman, A.S.: "The Warnier-Orr diagram", *Digest of Papers, COMPCON 79, IEEE catalog, No. 79, CH1393-8c* (1979) 301-306.
  - 24) Warnier, J.D. and Flanagan, B.: "Entrainment de la Construction des Programmes", *d'Informatique, Vol. I and II* (Les Editions d'Organization, Paris, 1972).
  - 25) Wirth, N.: "Systematic Programming: An Introduction" (Prentice-Hall, 1973).
  - 26) Yaku, T. and Futatsugi, A.: "Tree-structured notations for flowcharting", *Inst. Electronics Comm. Engrs. Japan* (1978) [in Japanese].