問題分析法 PAM によるプログラムの設計と審査†

二村 良彦††

本文では，まず，モジュールを構成する新しい概念としてスレッド (thread) を導入する．スレッドは，いわば，モジュールを織りなす糸である．次に，モジュールの機能仕様をスレッドに分割 (partition) し，そのスレッドを統合 (merge) して所望の PAD (problem analysis diagram) を設計するための手順 PAM (partition and merge) を提案する．PAM では，分割と統合の手掛りとして，入出力データ構造を利用する．すなわち，PAM はいわゆるトップダウン作成法をより具体化し，かついわゆるデータ分析法の利点をも取り入れたプログラム設計法である．PAM の特長は下記のとおりである．(1)プログラムの設計審査をほぼ完全に行える．(2)プログラマの質を向上させる．(3)あらゆる種類のプログラム開発に適用できる．本文では，プログラム設計法の概説，PAM の説明，PAM の効果と問題点等について述べる．

1. まえがき

プログラムの論理が本質的に木構造として記述できる点に着目し，この木を目に見える形に書いた論理図面が，われわれが昭和54年に提案した PAD(Problem Analysis Diagram)[1]であった．フローチャート等，在来のプログラム論理図と比べ，PAD はプログラム論理を(1)簡単に表現できる，(2)容易に目で捉えられる等の特長があるため，現在までに日本国内だけでも，推定数千人のプログラマに使用されるようになった．そして，「PAD が書けてしまえば，それは読みやすいし，それからのコーディング，テスト等は系統的にできることもわかった．今度は，与えられた問題(プログラムの仕様)から PAD を作成するための系統的な方法を教えてほしい」という多数のプログラマの声が湧き起こってきた．たしかにわれわれは，プログラムの論理を記述するための表記法を提供したが，それを利用して論理を設計するための思考法としては，具体的なものを提供しなかった．プログラマが PAD を使うことにより，ダイクストラやヴィルト等のように，トップダウン設計[3]や段階的改良[4]を自然に行えるようになることをわれわれは期待していた．

しかし，多数のプログラマを短期間で養成する必要のある今日において，「自然にまかせる」ことは非現実的である．また，そうしているとデカルト[5]流に言うならば，「無秩序な作業や曖昧な思考によって，自然の光がくもらされ知能が盲目にされてしまう．そして，そのように暗闇のなかをさまようことに慣れたプログラマは視力をそこない，明るみの光に堪えられなくなってしまう」という事態がおこりかねない．そこで，われわれ自身の思考法（設計法）をプログラマに知っていただければ，多少なりとも参考になるのではないかと考えた．自分自身がプログラムを設計するときに，おそらく無意識に用いたと思われる規則や方法を言葉ではっきりと書き記してみた．すると，当然のことながら，われわれの座右の書であるデカルトの「知能指導の規則」[5]やポリアの「いかにして問題をとくか」[6]に述べられている問題分析法とよく似たものとなった．

この方法で PAD を書くと，PAD の木の枝や葉が順に形をなしてゆく様子が，あたかも蛋白質が分解消化され，既存の組織に統合されるように見えた．そして，この方法に PAM (Partition And Merge, すなわち，分割統合) 法と名づけた．しかし，PAD (Problem Analysis Diagram) との関係では，Problem Analysis Method（問題分析法）と呼ぶ．

大規模ソフトウェアの設計過程を，システム設計（モジュールの定義）とプログラム設計（モジュールの設計）とに分けてみると，PAM 法は後者のモジュール設計法に関するものである．すなわち，PAM 法は，Yourdon[7]らの構造化設計の終了後のフェーズ，あるいは花田ら[12]のプログラム設計法における処理手順の作成フェーズへの適用を目的としている．

本文では，まず，プログラム設計法の概説を行い，次に PAM 法の説明，PAM 法の効果と問題点等について報告する．以下では，PAM 法という代りに，法を除去し，単に PAM と呼ぶ．

† Program Design and Review through Problem Analysis Method PAM by YOSHIHIKO FUTAMURA (Central Research Laboratory, Hitachi, Ltd.).

†† (株)日立製作所中央研究所第八部

2. プログラム設計法概説

本章では，ソフトウェア開発工程におけるプログラム設計の位置づけ，および在来のプログラム設計法の問題点等について議論する．

2.1 システム設計法とプログラム設計法との違い

与えられたシステムの要求仕様書に基づいてプログラムを開発する際には，まずシステムをいくつかのモジュール（すなわち，サブプログラム，サブルーチン，コルーチン等）に分割し，モジュール間の関連と仕様を明確にする．このとき，システムを分割するために用いる方法がシステム設計法であり，Yourdon ら[7]の構造的システム設計法 (Structured Systems Design) 等がよく知られている．一般にシステム設計法では，分割の際の基準として，変更の容易さ (modifiability)，分かりやすさ (understandability)，モジュールの流用可能性等を最優先とする．すなわち，モジュール間で変数やコードを共有することを避けようと主張する．そして，モジュールとしての機能のまとまりや能率が多少低下しても目をつぶることにしている．これは大規模ソフトウェアを長期間保守していくために必要な，当然の基準である．このような基準に従ってシステムをモジュールへ，モジュールをさらに小さなモジュールへと分割していく．そして，モジュールが適当に小さく分割され，モジュールの独立性 (coupling) と機能としてのまとまり (cohesion) のバランスがとれたところでシステム設計は終了する．したがって，システム設計法はモジュールの機能仕様を決めるだけで，「いかにしてモジュールをプログラム化するか」には言及しない[7]．システム設計のあとをうけて，モジュールをプログラム化するための指針を与えるものが，本文でいうプログラム設計法である．

システム設計の基準を満たすようにモジュールの仕様が決まった後では，モジュールのプログラム化に際してシステム設計と同じ手法をとるのは，多くの場合に誤りである．そのようにするとプログラムのサイズは大きくなり，実行速度も遅くなり，結局「構造的設計法で開発されたプログラムの能率は悪い」ということになってしまう．

システムから見ればモジュールは部品であり，システムの保守に際しては部品の交換または追加，除去等ですませるようにシステム設計がされている．部品にとって必要な条件は，正しく動作すること，すぐに調達できること，不必要に大きくなく動作も鈍くないこと等である．モジュールにとっても保守性はもちろん重要であるが，システムとモジュールの保守性を同様に考えてはならない．システムとモジュールの設計法の相違は，PAM の説明後に明確にする．

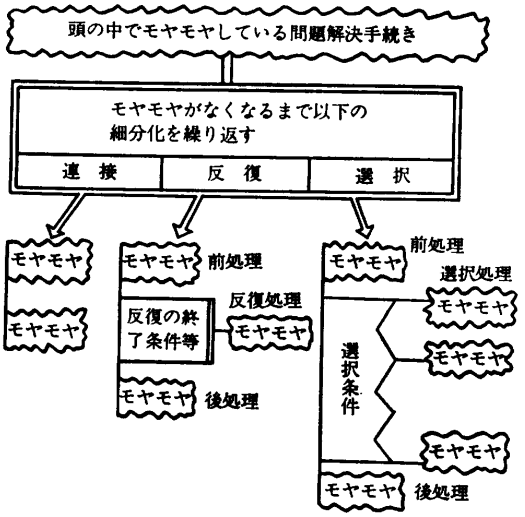


図 1 トップダウン作成法によるプログラム開発の原理
Fig. 1 Principle of top-down programming.

2.2 在来のプログラム設計法の問題点

在来のプログラム設計法の代表は，以下に述べるトップダウン作成法とデータ分析法であった[8]．

2.2.1 トップダウン作成法

「問題を解く際にトップダウンに考えるのがよい」と最初に言明したのは，ギリシャの哲人プラトンだと言われているが[6]，「プログラム作成もトップダウン方式でやるのがよい」と最初に言明したのは，オランダのダイクストラで，1969 年のことである．これが構造化プログラミングの先駆けとなった．これにより，プログラム作成法に関してもトップダウン作成法とか，段階的改良法等の指針が与えられた．その指針に従えば，頭の中でモヤモヤしている問題解決手続を，**図 1** のようにしてプログラム化することができる．しか

表 1 在来のプログラム設計法の問題点
Table 1 Shortcomings of program design methods.

方法名	与えられた指針または手順	主作業	問 題 点
トップダウン作成法	トップダウンにプログラムの構造を設計せよ	制御構造の設計	トップダウン設計の方法自体は与えられていない
データ分析法	(1)データ構造を記述する (2)データ構造に基づき制御構造を決定する (3)プログラムにとって必要な操作を列挙する (4)列挙した操作を(2)で決めた制御構造の中に割り付ける	入出力データ構造の設計	データ構造の設計法自体は与えられていない

し，「いかにしてモヤモヤを分割するか」という問題に対しては解答が与えられておらず，そのやり方は個々のプログラマに委ねられていた．すなわち，トップダウン作成法は，プログラム設計法と呼べるほどの具体的設計手順を示さなかった（表1）．

2.2.2 データ分析法

一方，入出力データの構造が，それを処理するプログラムの制御構造を決める上での重要な手掛りとなる点に着目したワーニエ[9]やジャクソン[10]らは，いわゆるデータ分析法と呼ばれるプログラム設計手順を提案した．その原理は，下記のとおりである．

（1） データ構造を記述する．

（2） データ構造に基づき，プログラムの制御構造を決定する．

（3） プログラムにとって必要な操作を列挙する．

（4） 列挙した操作を，(2)で決めた制御構造の中に割り付ける．

与えられた入力データの構造をその処理手順を無視して決めても意味がない．ところが，データ分析法では，処理手順を考える前にデータ構造を与えることを要求する．したがって，この方法でうまくいくのは，データの構造がプログラムの仕様のなかで明確に述べられており，その構造とプログラムの制御構造が一致する場合である．その他の場合には，プログラム構造に合致したデータ構造の設計はやはり容易な作業ではない[11]．しかしここでも，トップダウン作成法と同様，「プログラム構造に合致したデータ構造をいかにして設計するか」という問に対する解答は与えられなかった（表1）．

3. PAM の説明

上記の問題点を改善する目的で提案するプログラム設計手順が PAM である．

本章ではまずモジュールに対してスレッド (thread) という新概念を提出する．スレッドは，いわば，モジュールを織りなす糸である．与えられたモジュールの仕様の中で，渾然一体となって絡まり合っているスレッドを一本ずつほぐし，整然としたモジュールに織りあげる方法が PAM である．

PAM は，ポリアの問題解決手順[6]と同様に次の4種類の方法の規定からなる．

（1） 問題の理解（入出力データの条件および対応の明確化）

（2） 分割（問題の小問題への分割）

（3） 統合（小問題の解答の組立てなおし）

（4） 検証（解答の正しさの確認）

PAM において一番重要な点は，プログラムを作成せよという問題，すなわち，「プログラム作成問題」が与えられたら，その問題を次のように理解することである．

プログラム作成問題を解くこと

問題の裏に隠された PAD を見つけ出しそれを白日のもとにさらすこと

すなわち，問題が与えられたら，自分には見えないが，答の PAD もいっしょに与えられたと考える．そしてその答の PAD を白日のもとにさらす精神の作用が分割と統合である（図2）．

PAM におけるプログラム作成手順は次のとおりである．

〔手順1〕 未知のもの（求めるデータ）を明確にする．

（1） 求めるデータ（出力データ）の構造がわかっているときには，その PAD を書く．

（2） 出力インタフェースを書く．

〔手順2〕 与えられているもの（入力データ）を明確にする．

（1） 入力データの構造を PAD で書く

（2） 入力インタフェース（入力パラメタ）を書く

〔手順3〕 求めるプログラムの機能を部分機能に分割 (partition) する．この分割された部分機能をスレッド (thread) と呼ぶ．スレッドは，原則として，それを実現する PAD がプログラマの頭に浮かぶ（すなわち，直観的にわかる）か，またはプログラマがすでにその存在を知っているプログラム（または PAD）があるような機能である．ただし，一つのスレッドをさらに小さなスレッドに分割するつもりなら，その時

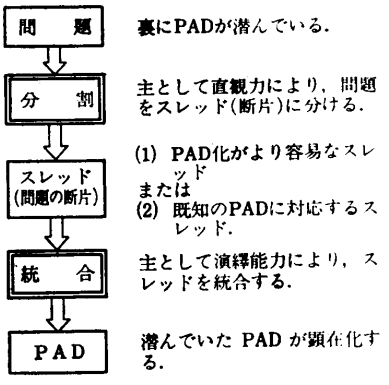


図 2 PAM における分割と統合

Fig. 2 Principle of partitioning and merging in PAM.

点ではそのスレッドの実現法を知っている必要はない．また，もとの機能自身も，その細部を取り除いて見たときには，部分機能と考えることができる．分割の際にはスレッドの機能だけに着目し，他のスレッドとの関係（変数やコードの共有等）については考える必要はない．分割の方法は一つには限らぬが，たとえば何が必要な機能または量かを考え，所要機能または量の鎖をすべて挙げるまで続ける．

〔手順4〕 スレッド間の依存関係を示すスレッド関連図*を書き，スレッド間の関係を整理する．必要ならば関連図が書きやすくなるようにスレッドを書き直す．全てのスレッドは，一つの関連図上に現れていなければならない．しかし，一つのスレッドが関連図上で2カ所以上現れてもかまわない．

〔手順5〕 一つのスレッドを選び PAD 化する．これは，プログラムの中心的構造（例えば，一番外側のループ）を与えるものであり，メインスレッドと呼ぶ．

〔手順6〕 残りのスレッドを一つずつ，すでに作られている PAD に統合 (merge) する．スレッドを統合する前に PAD を最適化しておいたほうが都合のよいことがある．そのときには，PAD を最適化する．この操作を，残りのスレッドがなくなるまで繰り返す．統合の各ステップの正しさは，直観的に明らかか，そうでなければ，証明によって示さなければならない．

〔手順7〕 PAD が完成したら，テストケース表[1]を作成し，かつレビュ（統合の段階の検証）をする (PAD が完成すれば，プログラムは機械的に作成できる[1])．以下に簡単な問題に対する PAM の適用例を示す．統合の各ステップにおいて，新たに追加または変更された部分には＊印が付してある．

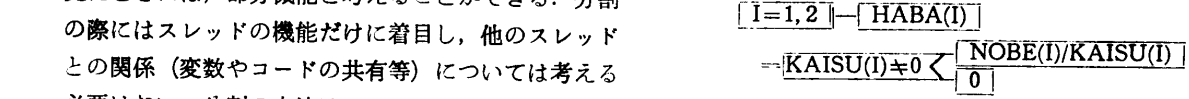
〔例題1〕 文字1と2から成る列が与えられたとする．このとき，同じ文字が連続して現われる部分の平均の幅をおのおの1および2について求める (HABA(1), HABA(2)) プログラムを作成せよ．ただし，与えられた文字列の最後には，文字9がストッパとして置かれているものとする．

〔例〕 111221222222111119 に対しては，

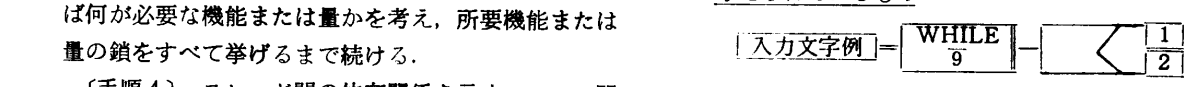
HABA(1)=NOBE(1)/KAISU(1)=9/3=3

HABA(2)=NOBE(2)/KAISU(2)=8/2=4

求めるもの



与えられているもの



スレッド

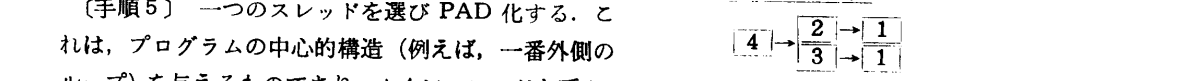
（1） データを終りまで読む

（2） NOBE(I) を数える

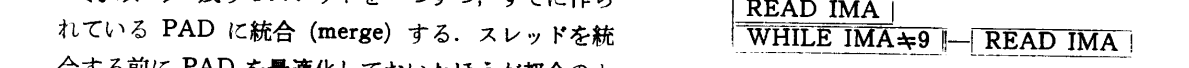
（3） KAISU(I) を数える

（4） HABA(I) を計算する．

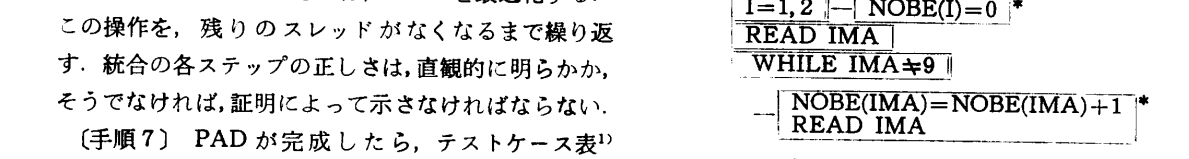
スレッド関連図



ステップ1 スレッド1の PAD 化



ステップ2 スレッド2のマージ



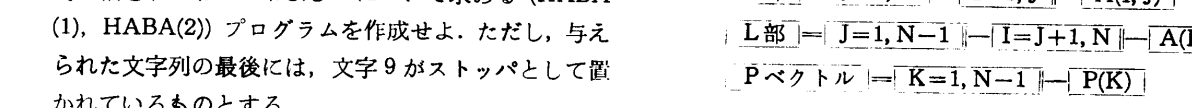
ステップ3 スレッド3のマージ（**図3**）

ステップ4 スレッド4のマージ（**図4**）

〔例題2〕 GAUSS の消去法による行列の三角分解

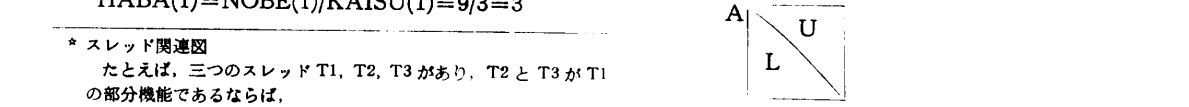
FORTAN 流に，縦（すなわち，列）方向に番地付けされたN次の正方行列Aを，部分軸選択を行う GAUSS の消去法により三角分解する PAD を作成せよ．

求めるもの



1～Nの整数

すなわち，



対角要素はUに含まれる

与えられているもの

スレッド

---

* スレッド関連図

たとえば，三つのスレッド T1, T2, T3 があり，T2 と T3 が T1 の部分機能であるならば，

と書く．これは，Yourdon, Constantine らの構造図[7]を横にしたようなものである．

「TにはSが必要である」と読む．

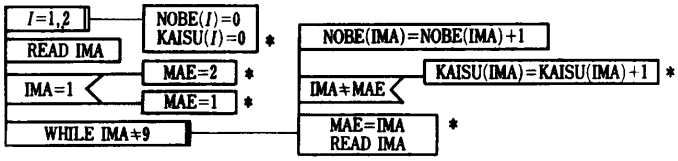


図 3　スレッド3のマージ
Fig. 3　Merging of thread 3.

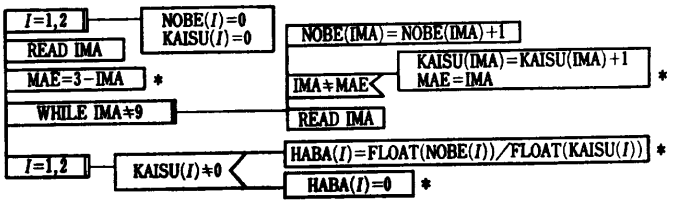


図 4　スレッド4のマージ
Fig. 4　Merging of thread 4.

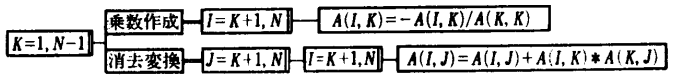


図 5　スレッド1の PAD 化
Fig. 5　PAD for thread 1.

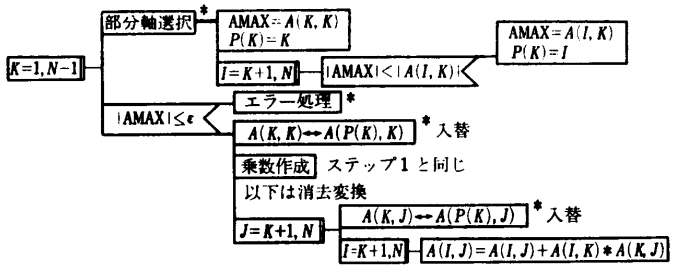


図 6　スレッド2のマージ
Fig. 6　Merging of thread 2.

(1)　$K=1, \cdots\cdots, N-1$ について，$A(K,K)$ を軸 (pivot) として，乗数

| I=K+1, N ||— | A(I, K) |=| −A(I, K)/A(K, K) |

を作り，その乗数を用いて

| J=K+1, N ||— | I=K+1, N ||— | A(I, J) |

に消去変換を施す．すなわち，その $A(I,J)$ に対して，

| A(I, J) |=| A(I, J)+A(I, K)*A(K, J) |

とする．

(2)　部分軸選択と行入替え

(2.1)　| I=K, N ||— | A(I, K) | の中から絶対値最大の要素を選び，その行番号を $P(K)$ に格納する．

(2.2)　| J=K, N ||— | A(P(K), J) | と | J=K, N ||— | A(K, J) | を入れ替える．

ステップ1　スレッド1のPAD化（図5）

ステップ2　スレッド2のマージ（図6）

以上の例題からわかるとおり，PAM ではスレッドへの分割（スレッドの直観による把握と既知の PAD への対応）および統合 (PAD の演繹的構成) 等の精神的作用そのものをいかに行うかというところまで規定していない．なぜなら，これらの作用それ自体は，デカルト流[5]に，何にもまして単純なそして本質的なものと考えるからである．この作用に細かい規定を設けると，直観が働きにくくなったり，演繹がしにくくなったりし，かえって人間の問題解決能力を阻害するおそれが生ずる．たとえば，限られた範囲でのみうまく適用できるような強い制限をつけてしまうと，その範囲から少しでもはみ出した問題に対して適用した場合に，その手法を全く用いないでやるよりもプログラム作成が困難になる．

したがって，直観力や演繹能力を用いることをプログラマの知性が心得ているのでなければ，PAM を理解し，実践することは困難である．逆に，問題解決の達人は，分割統合を自然のうちに体得しており，PAM を日常茶飯事と見よう．分割と統合こそプログラマにとって必須の素養であり，プログラマたるものこれを訓練しておかなければならない．プログラマの直観力や演繹能力を働かせるための手掛りとして，PAM では，入出力データ構造の記述を義務づけている．しかし，実際には，手掛りになるものであればデータフローでも状態遷移図でも，何でも用いたほうがよい．

以上の説明で，プログラム設計法もシステム設計法と同様に，そのかなめが分割にあることがご理解いただけたと考える．そこで，二つの方法の相違を表2にまとめた．

PAM では，「スレッドを統合する」という操作を明確に表面に出しているために，「PAM で作ったプログラムは理解しにくく，かつ保守しにくい」という誤解を与えることがある．本文の例題1と2で得られたプログラムは，完全な構造的プログラムであり，プログラミングの教科書の模範解答になりうるものである．プログラム作成においては，それが構造化プログラミング技法によるものであっても，本質的にスレッ

表 2 システム設計法とプログラム設計法
Table 2 Comparison of system and program design methods.

	システム設計法	プログラム設計法
分割の対象とその単位	システムをモジュールへ	モジュールをスレッドへ
対象の特徴	システムは大規模で作り直しが困難なため (1)長期間使われる (2)多数の人間の共同作業で作られる	モジュールは小さく（数十～数百ステップ），ハードウェアの変化の影響を受けやすいため (1)部分的に変更するよりは，作り直すほうが一般的には容易 (2)1人のプログラマにより作られる
分割の主要基準	(1)システムの保守性 (2)モジュール流用可能性 (3)モジュールの機能のわかりやすさ	(1)モジュールの作り直しやすさ (2)正しさの確認のしやすさ (3)実行速度，プログラムサイズ等の効率
最終的な産物	(1)モジュールの間の関係を表す木構造 (2)各モジュールの機能仕様書 (たとえば，HIPO)	スレッドの絡み合ったプログラム

ドを統合することが必要であり，従来はそのことを無意識的に行っていた．PAM は従来無意識に行っていた作業を，意識して行わせるようにしたものであるから，結果として得られるプログラムが従来のものより複雑になることはない．

4. PAM の効果

PAD には，「見やすい」ということ以外にも「分割と統合がしやすい」という特長があった．そして，いままでも PAD が自然にプログラマの分割・統合能力を促進し，PAD プログラマの生産性を向上させてきた[1]．しかし，PAM により意識的に分割・統合を行わせれば，プログラマの生産性をさらに向上させることが期待できる．PAM に期待できる大きな効果は下記の三つである．

（1） プログラムの設計審査をほぼ完全に行える．

プログラムエラーの一番有効な発見法は，設計審査（デザインレビュ）であることが知られている[8]．しかし，従来の審査はできあがった大きなプログラム（または論理図）についてなされており，一度にそれを把握するのが困難であると同時に，それがいかにして作られたかという，作成の過程も理解困難であった．

PAM の分割フェーズで得られたスレッドとスレッド関連図は，プログラムを構成する機能と，その間の関係を理解する一助となる．また，統合フェーズの各ステップは，プログラムがいかにして作られたかという，プログラムの生成の過程を示す．したがって，プログラムの設計思想が明確となり，第三者にもプログラムの設計審査が可能となる．

（2） プログラマの質を上げることができる．

優秀なプログラマを観察してみると，その多くは無意識のうちに，PAM のような分割と統合，データ構造の利用等を行っている．したがって，普通の水準のプログラマでも，PAM を実践することにより，優秀なプログラマと同様の思考法を体得することができる．また，PAM を実践すれば，自分のプログラム設計思想を明記し，かつそれを見直すという習慣がつく．これにより，常に知識が増え，プログラム生産能力は向上する．

（3） あらゆる種類のプログラム開発に適用できる

PAM は問題に固有な性質にとくに依存していないので，プログラム開発一般に適用できる．

われわれ自身は約2年間 PAM を実践し，実感としては非常に有効であると考えている．しかし，PAM は思考法であるので，その効果の定量的評価には，長い年月を要しよう．「この羊羹はうまい」というのと同様に，「PAM は有効である」ということをわかっていただくためには，味わっていただくのが一番と考え，56年10月までに200名以上のプログラマに PAD/PAM を一体として教育してみた．プログラマへの受けがたいへんよく，56年末からは PAM に重点を置いた PAD/PAM の教育コースを日立京浜工業専門学院において発足させている．

5. PAM の問題点と対応策

現在までの使用および教育を通して発見された問題点および対策を次に述べる．

（1） 多くのプログラマは，データ構造の記述に慣れていない．そのため彼らは，PAM の最初のステップである入出力データ構造の記述でとまどってしまう．しかしデータ構造が，それを処理するプログラムの構造を決める上での重要な手掛りであることは明確である．したがって，それができるようにプログラマを教育する必要がある．そのための教材の開発を行っている[2]．

PAM におけるこの問題は，ジャクソン法[10]の問題点でもある．ジャクソン法でのデータ構造は，直接にプログラム構造を決めてしまうので，その記述にはプログラム構造も考慮する必要がある．しかし，PAM におけるデータ構造は，プログラムの入出力集合さえ記述すればよいので，プログラム構造を考えずに決めることができる．すなわち，PAM においては，入出

力データの構造はあくまでもプログラム構造を決める上での手掛りにすぎないが，ジャクソン法ではデータ構造がプログラム構造を完全に規定してしまう．したがって，PAM の方がデータ構造を気楽に書くことができる．しかも，PAM で用いている PAD のほうが，ジャクソンの木よりも，記述能力が強力である（付録参照）．

（2） PAM の統合のフェーズが長くなると，ドキュメント量が増える．また，一つ前の PAD に新しいスレッドを統合してできる新しい PAD の大部分は，前の PAD と同じなので，同一の PAD を何回も書く必要があり面倒である．しかし，PAM の大きな目的は，「PAD 生成の過程を明確にする」ことであるので，そのためにドキュメント量が増えるのはやむをえないと考える．それにしても，何回も同じ PAD を書くのは能率が悪いので，統合のステップを実行する計算機システム（PAD マージャ）の作成を検討している．PAD マージャは，強力な PAD エディタの一種である．これは，既存の PAD を VDT（ディスプレイ）上に表示しておき，プログラマの指示に従い，この PAD に新しいスレッドをマージするものである．PAD マージャは PAD 生成の過程も記憶しているので，設計審査の際にはその過程を VDT 上に再現することができる．これにより，ドキュメント量も PAD を書く手間も減らすことができる．

6. むすび

モジュールの機能仕様をスレッドに分割し，そのスレッドを統合して所望の PAD を織り上げるための手順 PAM を提案した．PAM は，いわゆるトップダウン作成法をより具体化したプログラム設計法である．そして，分割と統合のための手掛りとして，プログラムが処理するデータの構造を利用する．

PAM の効果の定量的評価はまだ行っていない．PAD の普及に際しても同様に感じたが，プログラム開発手法の定量的評価には長期間を要し，結果が出にくい．しかし，「その技法を自発的に受け入れたプログラマの数」は，測定が比較的容易であり，かつ技法の有効性を示す重要な指標の一つであると考える．PAD/PAM の教育コースに出席したシステムやプログラム開発担当の主任クラスは PAM を理解し，実務に適用する意欲を持って帰っているので，PAM も PAD と同様，多数のプログラマに使われるようになると考えている．

謝辞 PAM の普及活動に協力してくださった日立京浜工業専門学院の藤田勝彦教授に深謝します．

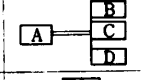
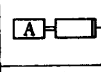
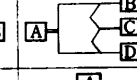
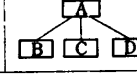
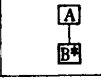
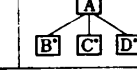
参考文献

1) 二村良彦，川合敏雄：PAD によるプログラムの開発，*bit*，3月，4月，5月号，共立出版(1980)．
2) 二村良彦：問題分析法 PAM マニュアル，日立製作所中央研究所 (1981. 9)．
3) Dahl, O.-J., Dijkstra, E. W. and Hoare, C. A. R.: Structured Programming, Academic Press, New York (1972).
4) Wirth, N.: *Systematic Programming: An Introduction*, Prentice-Hall, New Jersey (1973)（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，東京）．
5) R. デカルト：知能指導の規則，世界の大思想 21，河出書房新社，東京 (1974)．
6) G. ポリア(柿内賢信訳)：いかにして問題をとくか，丸善，東京 (1954)．
7) Page-Jones, M.: The Practical Guide to Structured Systems Design, Yourdon Press, New York (1980).
8) Jones, C.: A Survey of Programming Design and Specification Techniques, Trans. of the IEEE Symposium on SRS (1979).
9) J. D. ワーニエ(鈴木君子訳)：ワーニエプログラミング法則集，日本能率協会，東京 (1975)．
10) Jackson, M. A: Constructive Methods of Program Design, Proc. of the First Conference of the European Cooperation in Informatics (1976).
11) Peters, L.: *Software Design: Method & Techniques*, Yourdon Press, New York (1981).
12) 花田収悦，佐藤匡正，松本匡通，長野宏宣：コンパクト・チャートを用いたプログラム設計法，情報処理学会論文誌，Vol. 22, No. 1, pp. 44-49 (1981. 1)．

付 録 PAD とジャクソンの木との比較

PAD とジャクソンの木との比較を**表 3** に示した．PAD では，□ と ◁ の枠内に制御構造と同様の条件が書けるので，ジャクソンの木よりも記述能

表 3 PAD とジャクソンの木との比較
Table 3 Comparison of PAD and Jackson tree.

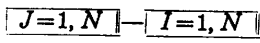
	連接	反復	選択
PAD			
Jackson			

力は強力である．たとえば，ジャクソンの木では配列は記述できない．

〔例〕 $N\times N$ の2次元配列

A(1, 1), A(2, 1), ⋯, A(N, 1), A(1, 2), ⋯, A(N, N)

のPADによる記述



(昭和56年12月8日受付)
(昭和57年2月16日採録)