Vol. 22 No. 1　　情報処理学会論文誌　　Jan. 1981

コンパクト・チャートを用いたプログラム設計法†

花田収悦††　佐藤匡正††
松本匡通††　長野宏宣††

本論文は，プログラムに与えられた命題を情報とその操作とからなる機能に基づいて論理的な構造を確定した上で処理手順を規定する，二段階から成るプログラム設計方法および，各設計段階に適合するように工夫された新たなドキュメンテーション方法（機能分析図，コンパクト・チャート）を提唱する．
本設計方法は処理効率の向上を，処理手順の規定の段階で，主に，いくつかの機能で共通な情報へのアクセスを削減するとの観点からはかり，大規模ソフトウェアの適用性を高めた．
本設計方法の適用例では，従来手法のプログラム開発データに比べ，（1）開発工数を 30% 削減できた，（2）信頼性の尺度であるバグの発生率が，開発段階では約 50%，商用開始後は約 30～40% 減少した，（3）ドキュメント量が 1/3 に削減できた——などの効果がみられた．

1. まえがき

プログラム設計方法に関する多様な提案を大別すると，モジュラ・プログラミング (MP) 型とデータ分析型とに分類できる[1]．

前者の代表的設計方法には，トップダウン開発[3]，構造化設計[4),5]などがある．これらの方法の特徴は，プログラム化しようとする命題（以下，問題記述）を満たすプログラムを想定し，その構成要素であるモジュールに機能を割りあてることにある（モジュール化）．したがって MP 型の設計方法ではモジュール化の基準が重要である．この基準として，コントロール（メイン）・モジュールや共通モジュールなどの制御の移行関係を前提として，たとえばコーディング量[3]，処理過程のモデル化[4]などが提唱されてきた．しかし，これらの基準を用いてモジュール化をしたとしても，モジュールの概念自体がプログラム寄りであるため，本来は論理的な整理をすべき設計当初の段階から処理の都合や処理系に対する意識が強くなり，処理手順的な分解を行いがちである．この結果，問題記述での論理的な構造とモジュール構造との間の対応関係が不明確になってしまい，処理の欠落やインタフェースの複雑化などの不都合を生じ，生産性および保守性などの点で問題を引き起しやすい．

後者の代表的な設計方法としてはワーニェ法[6]，ジャクソン法[7]などが著名である．これらの方法では以下の手順に従って問題記述に基づいた詳細化を進める．最初にプログラムの扱う入・出力情報の内容と構造を決定する．次に，確定した情報とこれに対応するプログラム論理とを組合せてプログラムの構成要素とする．この構成要素によりプログラムを構築する．したがって問題記述をプログラムの処理に展開させる過程は MP 型の方法より明確であると言える．しかし，処理の都合上必要となるインタフェースに対する措置が十分でないため，われわれの試行例では，入出力情報の相互関係のために多段階の処理過程が必要なプログラム，たとえば，コンパイラや制御プログラムなどのシステム・プログラムには適用しにくいという欠点があった．

以上のように，現在，提案されている各種の設計方法はそれぞれに勝れた特徴はあるが，必ずしもすべての分野で効果的な方法であるとは言えない．

本論文で提案する設計方法は，① 入出力情報に複雑な相互関係があり，多段の処理過程をもつ，② 性能に対する厳しい要求がある——のようなプログラムを対象とするものである．背景にある理念は「設計をていねいに」行うことにあり，これまでは熟練した設計者が自己流に採用していた方法を体系化し，発展させたものである．「ていねいな設計」を着実に実施するために，問題記述を論理的な木構造に分解する機能分析過程と，プログラム処理を作成する過程との二段階のフェーズを設け，それぞれに対応した設計ドキュメント；機能分析図およびコンパクト・チャートによって設計内容を規定することとした．本設計方法の特徴

† A Program Design Method Using Compact Chart by SHUETSU HANATA, TADAMASA SATOH, MASAMICHI MATSUMOTO and HIRONOBU NAGANO (Processing Programs Section, Yokosuka Electrical Communication Laboratory, NTT).

†† 日本電信電話公社横須賀電気通信研究所処理プログラム研究室

を従来の MP 型設計方法と比較して**図1**に示す.

本論文では，上記設計方法の概要を述べるとともに，実用システム（汎用コンパイラ）の開発に具体的に適用した実施例に基づいて評価した効果について述べる.

2. 設計方法

2.1 機能分析方法

本節では機能分析の主要な概念である「機能」および分析方法について述べる.

（1） 機能の概念

情報とそれに作用する操作とによって関係づけられた論理的な組合せを機能と定義する．具体的なイメージとしては，「入力情報にある操作を加えた結果，出力が得られる」ような文章を想定すればよい．この定義による機能は次の特徴があるため，問題記述を論理的に分析するのに適している.

（a） 一意性の高い思考

表現上，実現手段が隠ぺいできるので，処理の目的が明確になり，一意性の高い*思考が可能となる.

（b） 論理的整理

問題記述を機能の構成要素である情報や操作に着目して，後述の「機能関係」によって分解すれば，処理手順的な観点が薄められるために論理的に整理された分析が可能となる.

（c） 処理手順への関連づけ

処理手順は，原則として情報の入手順序によって規定されるので，(b)項で得られた論理構造体における機能の実行順序を与えることによって規定できる.

（2） 機能分析方法

次の（a）～(d）の手順に従って分析を進め，その結果を機能分析図にあらわす.

（a） 機能群の確定：問題記述を整理して，（情報，操作）を組とする機能群を確定する.

（b） 情報の明確化：機能の情報について構造をあきらかにする．情報の構造をモデル化して，階層化，くり返し要素，選択要素[1]などの観点によって捉える.

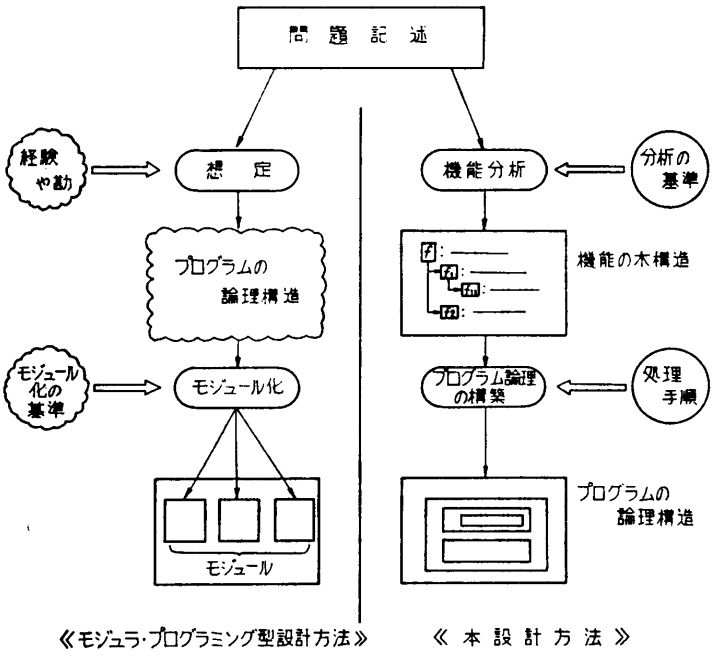


図1 設計方法の比較

Fig. 1 Comparison between the conventional modular programming method and this method.

（c） 機能の分解：次の（i）～(vi）項に述べる「機能関係」に着目して再帰的に分解してゆき，機能の階層構造体を作成する.

（i） **順序**：ある機能が情報の詳細化によって分解されるような関係（データフロー型）

（ii） **選択**：ある機能が条件によっていくつかの機能*に分解されるような関係

（iii） **同伴**：ある機能に伴って必要となる機能の関係（アルゴリズム型）

（iv） **独立**：ある階層内のいくつかの機能が互いに素なる関係

（v） **同族**：機能の論理的な分類によって同種の機能とみなされる関係

（vi） **同値**：同一内容を持つ機能の関係

（d） 分解の終末：以下の判断による

（i） 情報の要素が問題記述の規定に到達した.

（ii） 既出の機能である(同値).

（iii） 他機能の情報に到達した.

以上の方法によって分析した結果は問題記述と直接的な対応関係を持ち，機能を節点とし，機能関係を辺とする木構造をなす(**図4**参照).

---

* たとえば，「ある実数を4捨5入して整数をうる」機能を考える．機能としては唯一にもかかわらず，処理方法には2通りの方法；小数部の値によって条件分けをする方法と実数に0.5を加え，小数部を切捨てる方法，とがある.

* 機能の実体をもたない空の機能を含む.

2.2 処理手順の作成

機能の木構造に基づいてプログラム論理である処理手順を決定する．この結果を処理効率の観点から再構成する．プログラムとしてのモジュール構造は以下に示す処理手順を決定する過程で，① 処理の流れの区切り，② 走行環境および問題記述などの変更の可能性――などを勘案して行う．これらをコンパクト・チャートに記述する．

（1） 情報の詳細化：機能ごとに明確になっている情報の構造に応じたアクセス機構を考える．情報には処理対象の情報（処理情報）とその処理を実現または制御するために必要な情報*(制御情報）とがある．機能分析段階では処理情報の構造をあきらかにしたので，ここでは具体的な条件づけや要素のとり出し方などのアクセス機構を考える．このためにはほかの情報との結合方法，条件づけなどの与え方，くり返し要素の構成方法（リスト，ストリング等）とそれらを管理するための情報などの制御情報の明確化をはかる．

（2） 処理手順の決定：処理手順は対象とする機能のプログラムでの実行順序であるので，原則として機能の木構造の辺から順序関係の節点を連ねる**．同時に，前節であきらかになったアクセス機構を始めとする制御情報の取扱いを明確にする．

（3） 効率化：上の構造は機能相互間の独立性を保持したままで処理に変換するため，重複処理が多く処理効率上に問題がある．処理効率の向上施策として同一情報に重複してアクセスする回数を，アクセス機構の共通化によって行う．以下にその着眼点を示す．

（i） **前処理の設定**：同一情報へのアクセス機構を共通化し，処理情報をあらかじめ得ておく．

（ii） **アクセス状況の管理**：処理情報のアクセス状態を管理する制御情報を設ける．

このほか，アクセス特性に応じた機構（ハッシュなど）のアルゴリズムの工夫によって効率化をはかる．

2.3 設計方法の例題

モデル化した例によって，以上述べた方法の概念を示す．問題記述によって機能 f が与えられる．機能 f の構成要素である情報を入力 (I)，出力 (O) に分解し，更に I_1, I_2; O_0, O_1, O_2 に分解して機能分析図（**図2**）を作成する．この図における機能の順序関係に着目して，**図3**（1）に示すような木構造を作成する．ここで，情報の構造に着目する（I_1 は単一要素，I_2 はくり返し要素）と，プログラムの処理は図3(2)のようになる．f_{12} と f_2 とは同一の情報 I_2 を入力としているので，手順的に，D と O_2 をあらかじめ得ておくことによって，アクセス機構の共通化が実現でき，効率化がはかられる(図3(3))．

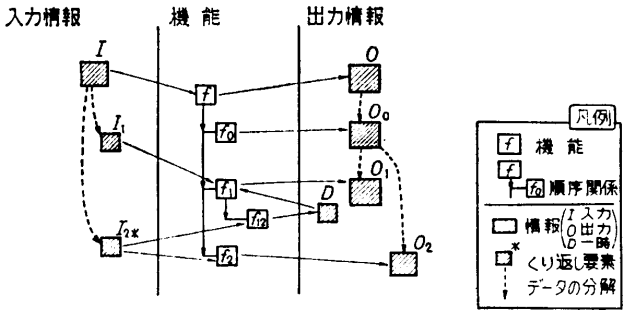


図 2 機能分析図

Fig. 2 The concept of function structure chart.

3. ドキュメンテーション

本設計方法において機能分析結果は，機能分析図に，処理手順の設計結果はコンパクト・チャートに記述する．

3.1 機能分析図

機能分析段階では問題記述の論理的な構成を機能に着目して分析できることが重要である．このため次のように工夫した．

（1） 機能の構成要素および関係を視覚的に表現する．

（2） 階層構造を番号体系によって簡明に表現する．

3.2 コンパクト・チャート[9]

処理を表現するドキュメントに対する要求条件を整理すると次のようになる．

（1） 機能との対応が明確になる．

（2） 処理記述が明解で，コーディングしやすい．

（i） 制御構造を整理して記述できる

（ii） 情報構造と制御構造との対応が明解である

（3） 記述性がよく，修正が容易である．

（i） スペース効率がよい

（ii） 記法が単純（専用テンプレートによらず手書きすることも可能である）

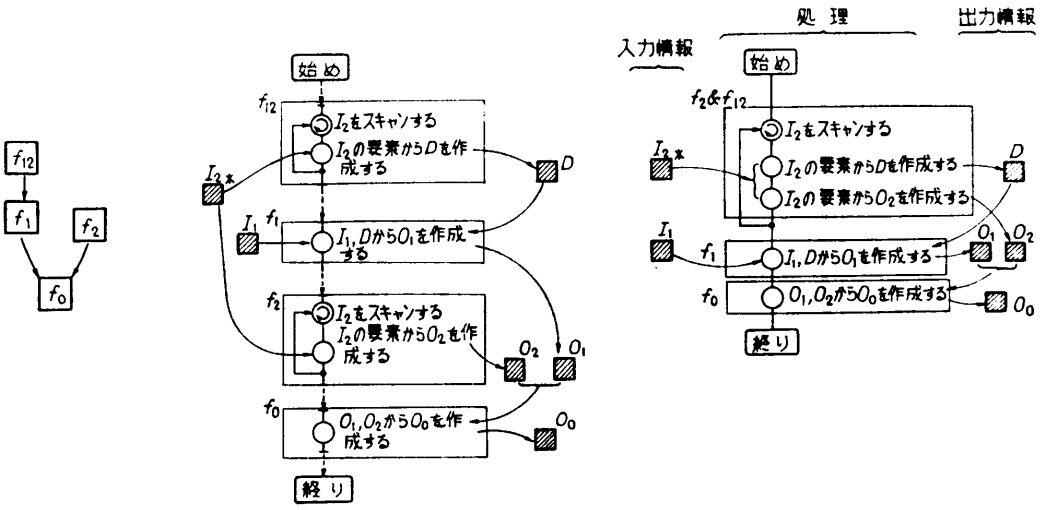
（iii） 直感的に理解しやすい

以上の条件を指向したドキュメンテーション技法と

---

* 代表例としてフラグやポインタなどがある．

** 2.1 節の例では，f_{12}-f_1-f_2-f_0, f_2-f_{12}-f_1-f_0 などのように連なりは幾条もあるが，例外処理等の条件を見通してひとつを選択する．



(1) 機能の木構造
(1) A function structure.

(2) 効率化以前の処理
(2) Process of the function.

(3) 効率化した処理
(3) Improved process.

図 3 プログラムの処理の作成

Fig. 3 An efficiency improving process by compact chart.

して「コンパクト・チャート」を考案した．この記述例を**図 5** に，記述上の規約を**表 1** に示す．

4. 効果等の考察

従来のモジュール化型設計方法と，問題記述から処理の作成までを機能に着目してていねいに設計する本設計方法とを用いて，それぞれ開発した同種の汎用コンパイラの実測値を比較することにより本設計方法の有用性を評価する．なお，比較対象は新規に開発した約 100 k Step の汎用コンパイラである．

4.1 開発工数

2 つのコンパイラ開発の工程別累積工数を**図 6** に示す．本設計方法の適用により，機能分析フェーズの新設に伴う工数増加，処理手順を作成する詳細設計工程およびテスト工程での工数削減などの結果，全体工数の 30% を削減できた．その要因について工程別に考察する．

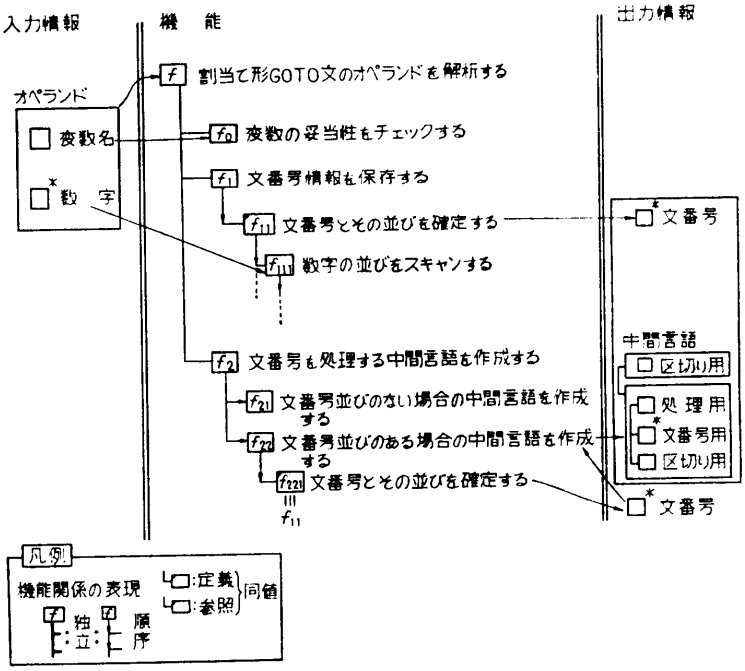


図 4 機能分析図の記述例 (FORTRAN のコンパイラの割当て形 GOTO 文の解析)

Fig. 4 An example of function structure chart (analysis function of assign Goto statement in a Fortran compiler).

(1) **機能設計**：本設計方法が約 2 倍の工数を要した．機能分析フェーズに伴う工数増加が主要因である．

(2) **詳細設計**：本設計方法の工数が半減している．主要な理由は，① コンパクト・チャートの記述性が高く，ドキュメント量が約 1/3 となった，② 仕様の欠落による設計の手戻りが少ない——などである．

(3) **コーディング**：おおむね同等である．

(4) **テスト**：設計時点でのバグの混入が防止できた結果，テスト段階でのトラブルが減少し工数削減が

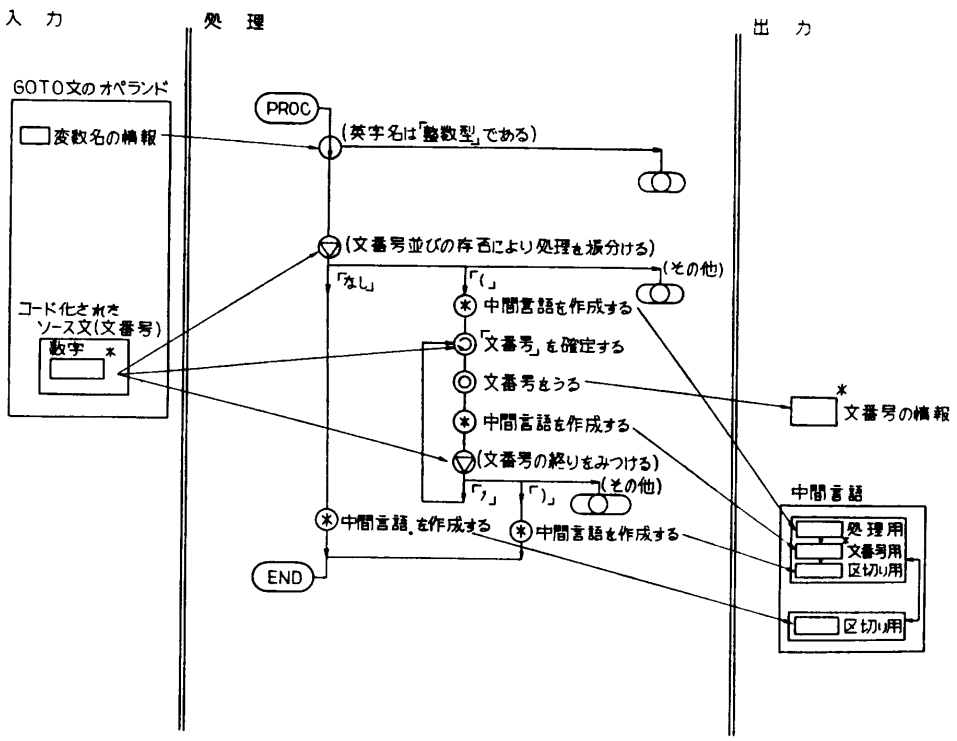


図 5 コンパクト・チャートの記述例 (FORTRAN コンパイラの割当て形 GOTO 文の解析)
Fig. 5 An example of compact chart corresponding to Fig. 4.

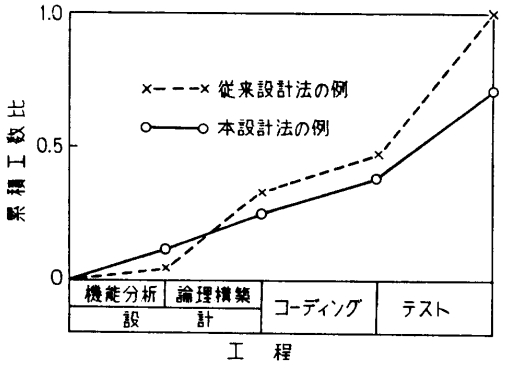


図 6 工程別の累積工数比較
Fig. 6 Development efforts reduction.

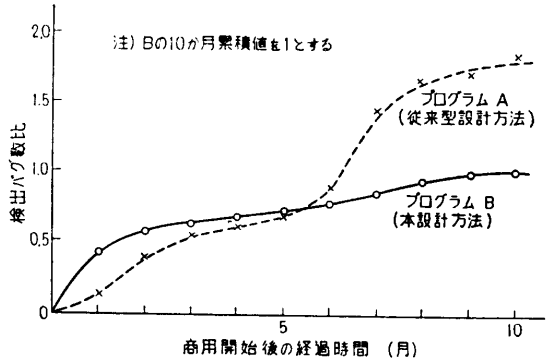


図 7 運用開始後の検出バグ数の時間的推移
Fig. 7 Reduction of the number of program errors in production run time.

可能となった.

4.2 信 頼 性

信頼性の目安として，運用開始後の検出バグ数に着目する．これは開発工程のテストが効果的であることおよび，設計段階からの混入バグ数が少なかったことを示している.

4.3 保守性および性能

保守性を推定する目安として，バグ1件あたりの平均修正モジュール数を用いる．**図 8, 9** に示すように約 1.3 モジュール/バグで，おおむね従来値と同等である．これは処理の作成が従来と同じく処理手順に基づいているためであり，バグ1件についての修正内容も従来と同等と考えられる．したがってバグ数の減少分だけ保守性は向上したといえる.

性能については，**2.2** 節(3)項の効率化によって，従来と同等以上であることが推測できるが，実測の結果では平均して約 10% 向上した．処理が整理された結果，グローバルな観点からの処理の共通化が実現できた効果であると推定できる.

表 1 コンパクト・チャートの記法

Table 1 Compact chart notating conventions.

分類		図記号	意味	用法
処理の記法	機能表現（実現方法）		一般的な機能を表す	〈　〉 機能内容 〈　〉 の説明
			手続きの呼出しを表す	〈機能の説明〉 [〈モジュール名〉]〈番号〉
			マクロの引用を表す	〈機能の説明〉 [〈マクロ名〉]
			インライン展開手続きを表す	〈機能の説明〉
	判断		二値判断 （矢印の方向がTHENの方向）	〈判断の目的〉 ELSE THEN
			多値判断	〈判断の目的〉 （〈条件〉） (条件1) (条件2) (ELSE) その他
	くり返し		ループ構造を表す	〈ループ内の対象データ〉 （ループ条件）
	終端		正常な入口出口を表す	PROC ENTRY END END
			例外時や異常時の出口を表す	〈あと処理の内容〉
データの記法			データを表す	階層構造を表す
		*	くり返し要素を表す	

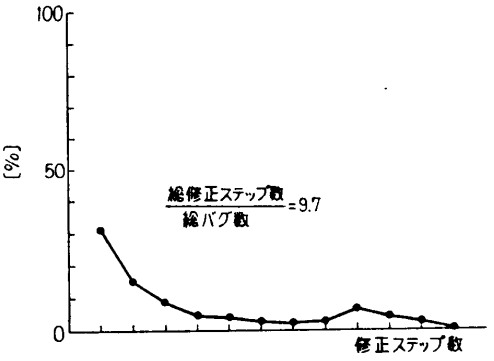


図 8 バグの影響範囲

Fig. 8 Distribution of maintained module number to a program error.

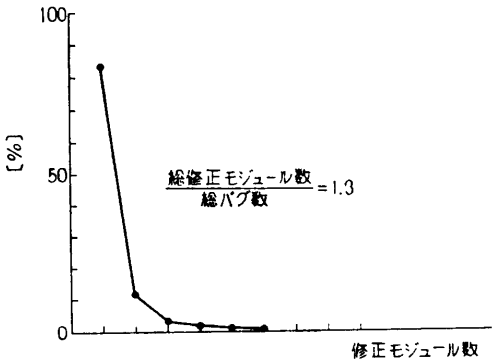


図 9 バグの影響度

Fig. 9 Distribution of maintained steps to a program error.

5. むすび

複雑な内部状態を有する大規模なシステム・プログラムを主たる適用分野とするプログラム設計方法およびドキュメンテーション方法を提案した．

本設計方法は，命題（問題記述）を機能に着目して詳細化する．そのために，機能を情報と操作によって定義し，再帰的に機能を分析して詳細化する機能分析フェーズと，それらの細分化された機能を手順的なプログラム論理として構築するフェーズとからなる2段階の設計方法である．設計ドキュメントには，各フェーズ対応に機能分析図とコンパクト・チャートを用いる．

この設計方法は，数回の改良を重ねて具体的にコンパイラや制御プログラムの開発に適用して効果をあげている．主要な効果を要約すると以下のようになる．

（1） 開発工数は約 30% 削減できる．

（2） 信頼性の尺度として，本格的使用（たとえば，商用）において検出された単位規模あたりのバグ数を用いて比較すると，約 30〜40% 少ない．

（3） 性能は向上する（約 10%）．

また，本設計方法で提案した2つの設計ドキュメントの定性的な特徴は以下のとおりである．上記の効果はこれらが総合された結果と想定される．

（1） 機能分析図

（i） 問題記述を機能に着目して網羅的に分析できるので仕様もれが少なく，設計，テスト工程における手戻りが減少する．

（ii） 機能が階層的に整理されるため，機能追加等が容易である．

（2） コンパクト・チャート

（i） フローチャートの図記号を小型，単純化することにより，ドキュメントの記述密度を向上(約3倍)させ，大部分のモジュールを一葉（A3版）のチャートとして記述できる．

（ii） 制御構造のほかに情報の構造に関する記述を設けたため，両者の対応関係が明確になり，処理アルゴリズムが理解しやすくなった．

（iii） 図記号の表記法を工夫し，良制御構造のコーディング・イメージを作りやすくした．

なお，コンパクト・チャートは日本からISOのSC 7（設計とドキュメンテーション）のWGにも紹介された（1977年）．

今後は具体的な使用実績を蓄積し，ツール化などの改良を図る予定である．

謝辞 最後に本研究を進めるにあたり有益なご指導を賜った元横須賀電気通信研究所調査役筑後道夫氏に深く感謝の意を表する．

また，本研究はDIPSプロジェクトの一環として実施したものであり，公社関係各位，ならびに共同研究各社（日本電気，日立製作所，富士通）の関係各位に感謝する．

参考文献

1) Parker, J.: A Comparison of Design Methodologies, Software Engineering Notices, Vol. 3, No. 4 (1978).
2) Mills, H.D.: Top Down Programming in Large Systems, Debugging Techniques in Large Systems, Courant Computer Science Symposium 1, NYU, Ed. Randell Rustin, pp. 41-55 (1971).
3) Peters, L. J. and Trippo, L. L.: Comparing Design Methodologies, Datamation (Nov. 1977).
4) Stevens, W.P., Myers, G.J. and Constantine, L.L.: Structured Design, IBM Syst. Vol. 13, No. 2 (1974).
5) Myers, G.J. 久保他訳：高信頼性ソフトウェア—複合設計，近代科学社 (1976).
6) Warnier, J.D. et al. 鈴木訳：ワーニェ方式によるプログラミング学習(上，下)，日本能率協会 (1973).
7) Jackson, M. A.: Principles of Program Design, ACADEMIC PRESS (1975).
8) Stay, J. F.: HIPO and Integrated Program Design, IBM Syst. Vol. 15, No. 2 (1976).
9) 佐藤匡正，長野宏宣：HIPO記述の一手法，第17回情報処理学会全国大会論文集 (1976).
10) 佐藤匡正，松本匡通，長野宏宣：機能的設計法における2，3の考察，第18回情報処理学会全国大会論文集 (1977).

（昭和55年7月4日受付）
（昭和55年9月18日採録）