解　説

プログラミング用ドキュメンテーション†

佐　藤　匡　正††

1. はじめに

プログラムの開発におけるプログラミング用のドキュメントの役割は重要である．要求条件や仕様の記述などの設計用だけでなく，進捗管理や保守のひきつぎ，教育などに用いられる．従来は，フローチャートを中心としたドキュメントが用いられてきた．フローチャートは処理の流れが明確にできるためプログラム動作の規定は可能であるが，理解するのに必要な処理目的や処理方式が明快には記述できないという問題点がある．

ソフトウェアの高度化，大規模化にともないその効率的な開発および保守が強く要求されている．ソフトウェア開発の全サイクルを効率よく進めるためには，その出発点となる要求分析からプログラム設計までの過程での誤りの混入をなくせばよい．この観点から要求定義や設計について，技法の考案，改善が図られている．その根源は，つかみどころのないソフトウェアをいかに明快かつ正確に表現するかに尽きよう．その一環として，従来のフローチャートにおける理解のしにくさの問題点を，制御構造やデータ構造の整理によって解決しようとする技法が出現してきている．

本稿では，プログラム記述言語の高水準化にもかかわらず，フローチャートの記述能力がプリミティブなアセンブラ時代から発展していないことにある事を指摘し，これまで発表された技法について，フローチャート高水準化の観点にもとづいて体系化すると共に，代表的技法について概説する．

2. ドキュメンテーションの概念

（1）　ドキュメンテーション改善のねらい

ドキュメンテーションの改善効果をドキュメント作成工数の削減のみと捉えると，図-1 に示すようにドキュメント作成にかかる工数は 10% に満たないので効果は薄い．しかし，改善効果をプログラムの論理的な整理に置くとすると，ドキュメントはプログラムと表裏一体の関係にあるので，開発工程全体にわたって，①設計のガイドラインが与えられ，設計のスムーズな進行，誤りの混入防止などが図れる，②記述結果の見直し（レビュ）によってバグの抑止（デバッグ工数の削減）がはかれる，③保守や改造が容易になる，など設計，デバッグを中心としたアクティビティが対象となり，開発工数の 70% が改善の対象となる．

（2）　記述すべき内容

プログラミングのためのドキュメントとして記述すべき内容はドキュメントの用途によって決まる．設計では処理方式を詳細化して具体化をはかるのに対して，レビュや保守では記述された内容を理解した上で，吟味，修正を行う．設計では処理をどのようにして実現するか（'how'），つまり実現手段の記述をすればよい．一方，レビュや保守では記述内容の理解を助けるために処理の目的や実現方式など何をするか（'what'）が記述されている必要がある．さらに，レビュは記述全体にわたって吟味するのに対して，保守ではバグの吸収，仕様変更の反映など部分的に修正することが多い．したがって保守では what に加え，点在する関連処理の索引が必要である（図-2）．

以上の観点から，ドキュメントに記述すべき内容は

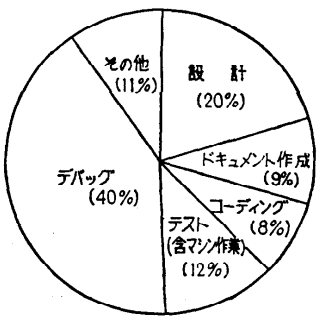


図-1　プログラム作成工数の構成例

† Documentation Methods for Programming by Tadamasa SATOH (Yokosuka Electrical Communication Laboratory, N. T. T.).

†† 日本電信電話公社横須賀電気通信研究所

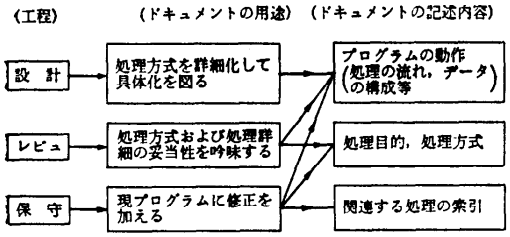


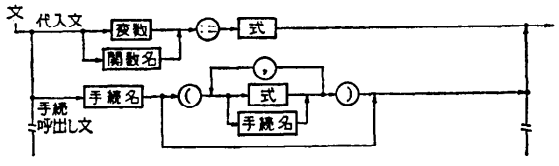
図-2　プログラミング用ドキュメントの用途ごとの記述すべき内容

用途に応じて大幅に異なるため，理想的には用途に応じて別個に作成することが望ましい．しかし，ドキュメント作成工数は 図-1 に示すように 9% 程度であるものの，これ以上の上積みは現実的とは言えない．設計ドキュメントでレビュまでは兼ね，保守ドキュメントで若干手直しをするという前提で，プログラミングのために必要となるドキュメントの記述内容は処理目的，処理方式，プログラム動作である．

(3)　表 現 方 式

設計ドキュメントの内容を表現する方法として，形式言語による方法とチャートによる方法とに分類される．形式言語によらないと誤解を招きやすいとの考え[31]もあるが，チャート記法は，言語記述よりも概して直感に訴える力が強く，主要な部分と付随的な部分とが明確に区別できる特徴がある．したがって，処理方式や全体的な構成などについてのアウトラインの理解を促進させるためには効果的である．たとえば，構文則の規定では，従来の **BNF** 型の構文記述から，最近ではシンタックス・チャートが採用されている点[27)~29)]からもチャートの有効性がうかがえる．**PASCAL** の構文則について BNF とシンタックス・チャートとの対比を 図-3 に示す．

```
〈文〉::=〈代入文〉|〈手続呼出し文〉|〈複合文〉|〈IF 文〉|〈CASE 文〉|
        〈WHILE 文〉|〈REPEAT 文〉|〈FOR 文〉|〈WITH 文〉|〈空文〉
  〈代入文〉::=〈代入先〉:=〈式〉
    〈代入先〉::=〈変数〉|〈関数名〉
  〈手続呼出し文〉::=〈手続名〉〈引数の指定〉
    〈引数の指定〉::=(〈引数の並び〉)|〈空〉
      〈引数の並び〉::=〈引数〉|〈引数の並び〉,〈引数〉
        〈引数〉::=〈式〉|〈手続名〉
        ……　　(1)BNF
```



(2)シンタックスチャート

図-3　BNF とシンタックスチャートとの比較 (PASCAL の構文)

チャート記法の普及上の阻害要因のひとつとして，計算機処理による支援系の不備があげられる．チャートの修正や編集はもっぱら人手に頼らざるを得なかったが，近年の図形処理技術の発達とともに，充実した支援系が近い将来に出現するものと予想される．

3. ドキュメンテーション技法の整理と評価

3.1　ドキュメンテーション技法の体系

フローチャートはプログラム論理を記述するために最初に使用されたチャート記法である．プログラムの大規模化にともない，詳細化の階層に応じて，全体をあらわすブロック・フローやモジュール構成図，主要な条件と処理の流れをあきらかにするジェネラル・フロー (**GF**)，コーディングと対応したディテール・フロー (**DF**) などの工夫がなされてきた．しかし，①各階層の対応が不明確になりがちである，②複数ページにまたがる流れの結びつきが把握しにくい，③処理の流れからだけでは処理方式や処理目的が理解しにくいなどの問題点がある．さらに，**DF** については，その情報がプログラムリストで十分代替でき，効果がないと報告されている[25)]．

従来のフローチャートが現状にそぐわなくなった理由は，次のように解釈できる．すなわち，フローチャートが有効であった時代のプログラムの記述は，アセンブラが中心であった．アセンブラには言語固有の概念はなく，メモリや処理スピードの制約に関心を置かざるを得なかったために制御の流れが最大の関心事であった．フローチャートは制御の流れに対しては抽象度の高い直視性の優れた記法であった．ところが，言語に豊富なデータ構造，整理された制御構造などの言語固有の概念が備わり始めると，これらの概念を十分に記述する能力に欠けるフローチャートは相対的に抽象レベルが下がってしまった．

最近は，低下した記述レベルの高水準化を図った技法が提唱されている．各技法について高水準化の着眼点に従って分類整理し，体系化したものを 図-4 に示す．高水準化の着眼点は，①処理自体の整理，②処理の流れの整理，③データの表現，に大別できる．技法の動向としては，②③を統合した技法に集約する傾向にあり，①との関係についてはそれほど関心が向けられていない状況にある．以下，主要な技法について概説する．

3.1.1　処理細部の隠ぺい

(1)　状態遷移図

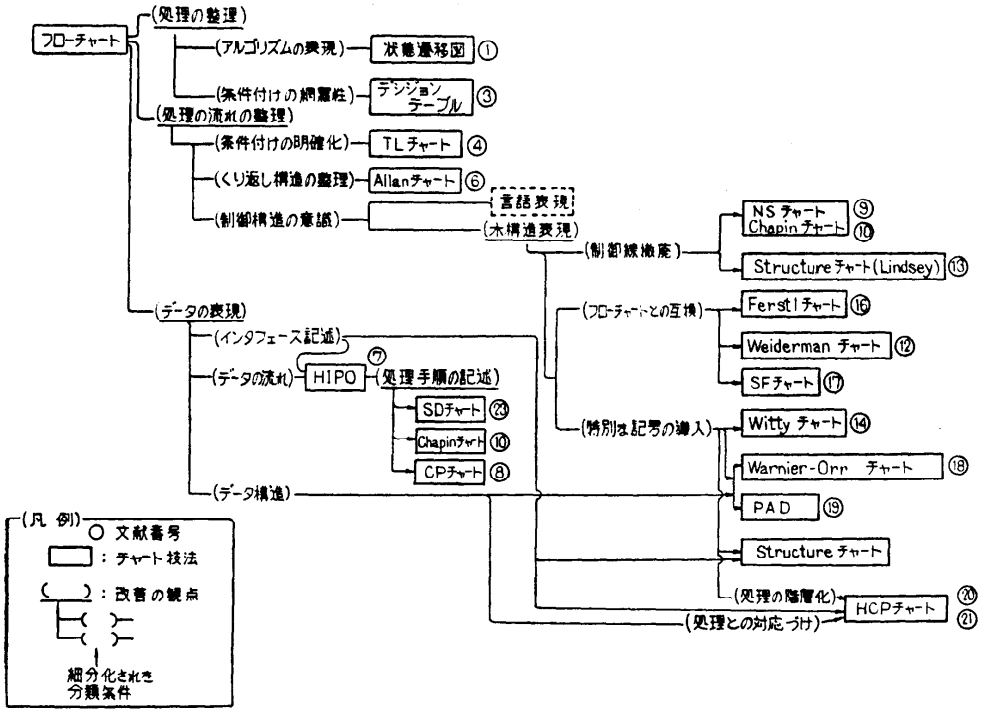


図-4 プログラミング用チャートの体系

状態遷移図は系全体の動作を，とりうる状態とその遷移条件とによって規定するものである．状態遷移図による記述は，プログラムとして実現するための諸々の処理や情報が隠ぺいできるため，全体的な見通しがよく，処理方式の記述に向いている．しかし，状態について明確な定義がしにくいことから，遷移線とフローチャートの流れ線とが混同されやすく，プログラミングのためのドキュメントとしては十分に活用されるまでには至っていない．交換機用プログラムでは「状態」がハードウェアの状態と一致させて整理できるために標準化され効果的に使用されている[1]が，一般のソフトウェアでは通信制御，タスク管理，分散処理，コンパイラ等の構文解析などの分野で使用されているにすぎない．図-7 に記述例を示す．

（2） デシジョンテーブル（決定表）

デシジョンテーブルは処理の条件を網羅的に整理するために，条件と処理との関係を分析し表示する記述方法である．条件づけに必要となる判断の手順や論理演算の方法などが隠ぺいできるため，①本来行うべき処理が明確になる，②条件の配慮もれがなくなる，などの効果がある．しかし，いわば case 文のみで論理を記述しなければならないため，条件と条件の間に処理が必要である場合やループを形成するような処理の表現には不向きであり，自己完結にできないことから補足的に使用されているにすぎない．

3.1.2 データの表現

一般に，フローチャートで規定される処理の流れは実行順序に関するものであり，直前に実行された処理と着目している処理とが関係を持っているとは限らない．例えば，A, B, C の3つの処理が連続して実行されるフローチャートにおいて，AとBとは独立で，Cの処理でAの結果とBの結果を使用するということはよくある．このような処理の因果関係を表現するには処理の流れとは別にデータの流れが必要となる．処理の因果関係を示す技法として HIPO[7] がある．HIPO ではデータを入力と出力とに分け，処理との対応を示すことによって処理の因果関係を表現している．しかし，データの流れはわかるものの，処理の流れが明確でないため，必ずしも因果関係は明確ではない．さらに，記述上，①処理記述が効率的でない，②記法が複雑で修正が難しい，などの問題点がある．このために，処理欄で処理の流れが記述できるほかの記法，擬似コード (PDL)[23]，Chapin チャート[10]，CP チャート[8]などと組合せたチャート技法を用いることが普及

している．擬似コードは直視性の点で，**Chapin**チャートは記述性およびデータとの対応がとりにくい点でそれぞれ問題がある．

3.1.3 処理の流れの整理

処理の流れを整理する観点としては，まず，条件付け[4]やくり返し[6]などの処理の表現改善が図られ，次にそれらを統合して制御構造を的確に表現しようとする技法に発展した．制御構造を整理する上での観点は制御移行方法をいわゆる構造化3構造（連接，くり返し，選択）に限定することにある．このために，チャート上は木構造表現が使用される．木構造の表現方式は，①制御線を撤廃する，②従来のフローチャート記号を利用し既存部分との親和性を保つ，③特別な記号を用いて効果的な概念を導入する，などに大別できる．

（1） 制御線の撤廃

フローチャートがわかりにくいのは制御線がスパゲッティのように入り組むためであるとの考えから制御線を撤廃して，原則的には箱だけで記述しようとするチャート記法である．Nassi らによる **NS** チャート[9]やその改良型の **Chapin** チャート[10]などがある．この記法は直感に訴えやすく，機械処理が容易であるなどの長所はあるものの，場合分けの多い処理では記述スペースが不足がちになる，修正の困難性，例外処理の記述が明確にならない，などの問題点がある．したがって設計段階での使用よりも，保守等で，プログラムの制御構造を機械処理によって分析した結果を出力するためのチャートとして向いている．記述例を図-5 に示す．

（2） 制御線付木構造チャート

制御線の交錯を防止するために，ループの区間や選択の区間などの区切りをつけることによって木構造を構成するためのチャート記法である．

制御線があるため，記述スペースの確保や修正に対しては NS チャートのような欠点はない．記法を大別すると，従来のフローチャートとの親和性を意識したものと，フローチャートの記法とは独立なものとがある．前者は制御構造の整理に主眼があるため，フローチャートの問題点に十分には対処できていない．後者もデータやインタフェースに対する意識はあるものの，制御構造の整理に主眼が置かれている．記法の比較を **表-1** に示す．

表-1 制御線付 木構造チャート記法の比較

比較項目 / 記法	処理	連接	選択 二分岐	選択 多分岐	反復	木構造表記 開始点	木構造表記 終了点	支援系	発表年
Ferstlの木構造チャート								—	1978
Dimentional Flowchart (Witty)								あり	1977
Weidermanの(*)木構造チャート								—	1975
S C (DIPS)								—	—（原典はMyersの構造図）
P A D (日立中研)								—	1979
HCPチャート (通研)								開発中	1979

(*) これと似た記法として SF（東海大，1979年）がある．

(**) 処理の階層表現をするという点で他の記法とは異なる．（ の右には下位レベルを包括した処理名を記述する）

表-2 各チャート記法の比較

処理の記述項目 ＼ チャート記法		フロー・チャート	状態遷移図	HIPO	NS チャート	PAD	HCP チャート
処理の記述	階層化の記述 モジュール間	○	×	○	○*)	○	○
	階層化の記述 モジュール内	×	×	△	×	×	○
	処理方式の記述	×	○	△	×	×	△
処理手順の記述	データと処理との関連づけ記述	×	×	○	×*)	×	○
	制御構造の記述	×	○	×*)	○	○	○
データの記述	データ構造の記述	×	×	△	×	○	○
	データの流れの記述	×	×	○	×	×	○

(注) *) 記述しうる複合記法は存在する　○：明確に記述できる　△：記述できる　×：記述できない，記述不十分である

制御構造の整理だけでは処理目的や処理方式は明確にはできない．処理の論理的な構成が自明となるような記法が望ましい．この観点に立つ技法として **HCP** チャートがあげられる[20),21)]．**HCP** チャートの狙いは処理目的 (what) とその実現手段 (how) とを対にして階層的に処理の詳細を記述することにある．チャート上に記述すべき内容は，①データの記述 (論理的構造，インタフェースとワークの別，流れ)，②処理の記述 (論理的な処理の階層，制御構造) である．記述された内容から処理の論理的な階層性，データ構造と処理との対応関係，データの流れ，などが整理され，論理性が高まるために，設計者はもちろん，担当者以外も理解が容易となり，レビュや保守に効果的である．また，改造開発において既存プログラムの関連部分を論理的に整理するためにも効果的である[22)]．**HCP** チャートの記述例を 図-8 に示す．

3.2 ドキュメンテーション記法の評価

記法の評価は定量化しにくいため，特徴の比較，記述結果の比較によって簡単な評価を加えたい．

チャート記法の特徴を **表-2** に示す．状態遷移図は処理方式を明確に表現できるものの，処理目的やその実現手順は表現できない．HCP チャート以外の木構造チャートは処理の実現手順はあきらかでも，処理方式や目的が直接的には表現されていないため処理手順からこれらを読みとらねばならない．これに対して，**HCP** チャートは処理方式の明快さは欠けるものの，処理目的，実現手順が明快に表現されることがわかり，理解の容易なドキュメントであることがわかる．

「ファイル中の単語の数を数える」というプログラムの論理を，**NS** チャート，**PAD**，状態遷移図，**HCP** チャートで記述した結果を，**図-5～図-8** に示す．記述結果の比較から **NS** チャートや **PAD** では制御の移行はわかるものの処理目的がわかりにくく，プログラム論理を表わすドキュメントとしては **HCP** チャートが優れているといえる．

4. おわりに

本解説では最近話題となっている木構造チャートを中心に，フローチャートからの発展経緯，記法の比較などについて概説した．木構造チャートのほとんどは

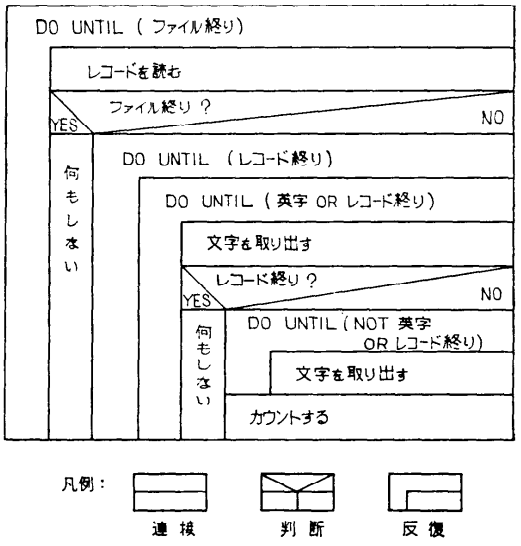


図-5 NS チャート

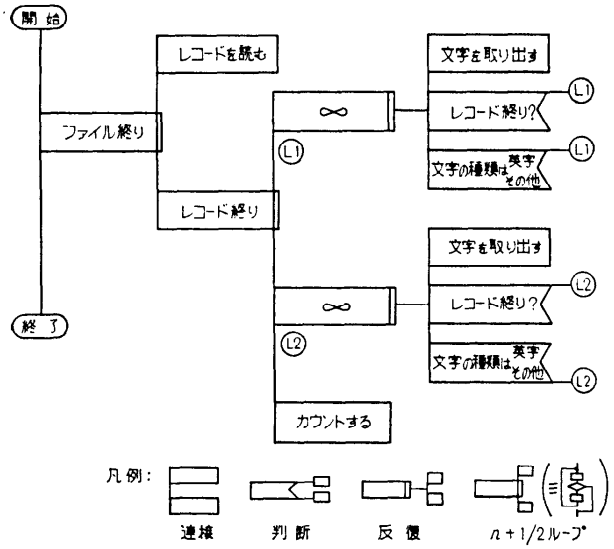


図-6 PAD

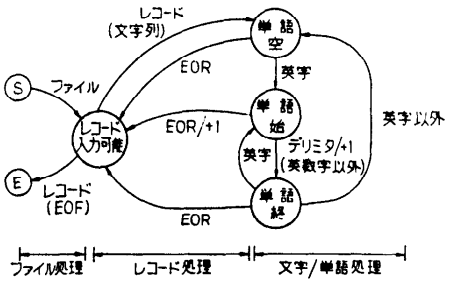


図-7 状態遷移図

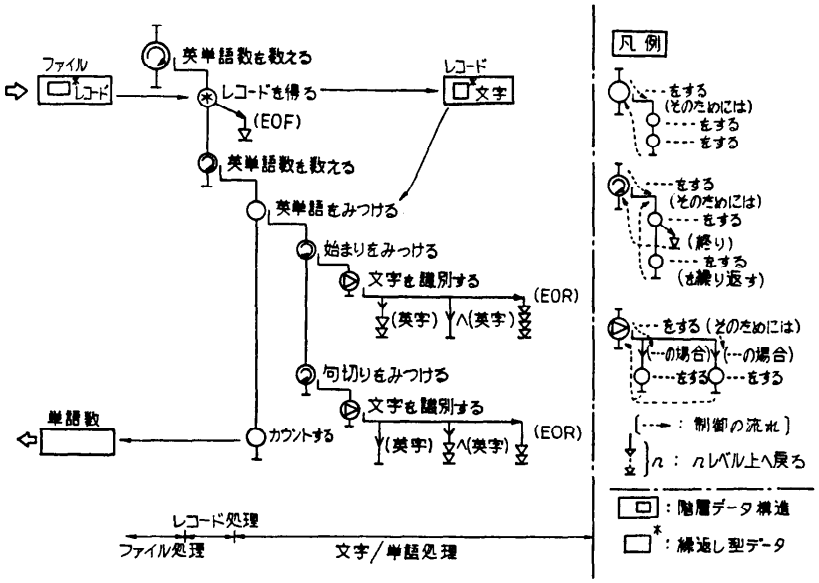


図-8 HCP チャート

制御構造の整理が中心であり，コーディングにおけるGOTO-less 化の域を脱していない状況である．プログラム論理の複雑さは制御移行上の問題もあるが，本質的な処理と例外処理との分離や処理目的による論理的な区切りなどをデータとどのように結びつけて整理するかが整理のポイントである．この点で HCP チャートのアプローチは優れていると言える．

今後のチャート技法の発展動向としては図形処理技術を応用した支援系の充実，さらにはチャートを直接入力してコードを生成するチャート・コンパイラなどの実用化が期待される．

参考文献

1) 高村，川島，中島：電子交換プログラム入門，電子通信学会 (1976).
2) 中島，大高：状態遷移表を用いた高信頼度ソフトウェア設計の一手法，昭和54年度電子通信学会情報・システム部門全国大会.
3) McDaniel, H.: An introduction to decision logic tables. 岸田訳，デシジョンテーブル入門，日本経営出版 (1970).
4) 鈴木：木構造によるプログラミング，ノエマ (1975).
5) TINKY：ループの話，bit Vol. 10, No. 14 (1978).
6) Allan Gottlieb: A FLOWCHARTING PROPOSAL, SIGPLAN Notices (Dec. 1976).
7) Stay, J. F.: HIPO and Integrated Program Design, IBM System Journal, Vol. 15, No. 2, pp. 143-154 (1976).
8) 佐藤，長野：HIPO 記述の一手法，情報処理学会第17回全国大会 (1976).
9) Nassi, I. and Shneiderman, B.: FLOWCHART TECHNIQUES FOR STRUCTURED PROGRAMMING, SIGPLAN Notices (Aug. 1973)
10) Ned Chapin, "New Format for Flowcharts", SOFTWARE-Practice and Experience Vol. 4. pp. 341-357 (1974).
11) John B. Holton and Bill Bryan: STRUC-, TURED TOP-DOWN FLOWCHARTING, Datamation (May 1975).
12) Weiderman, N. H. et al.: FLOWCHARTING LOOPS WITHOUT CYCLES, SIGPLAN Notices (Apr. 1975).
13) Lindsey, C. H.: Structure Charts, A Structured Alternative to Flowcharts, SIGPLAN Notices (Nov. 1977).
14) Robert, W. Witty: Dimensional Flowcharting, SOFTWARE-Practice and Experience Vol. 7, pp. 553-584 (1977).
15) 夜久他：フローチャートの木構造型記法，AL 研究会，AL 78-47 (1978).
16) Ferstl, O.: Flowcharting by Stepwise Refinement, SIGPLAN Notices Vol. 13, No. 1 (1978).
17) 大原他：フローチャートの木構造化の一提案，54年度電子通信学会総合全国大会 (1979).
18) Peter A Verdegraal et al.: The Warnier-Orr Diagram, Compcon '79 (spring) digest of paper pp. 301-306.
19) 二村他：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 (1979).
20) 佐藤，浅見：フローチャート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 (1979).
21) Hanata, S. and Satoh, T.: COMPACT CHART—A Program Logic Notation with Describability, SIGPLAN Notices Vol. 15, No. 9 (1980).
22) 渡辺，中村他：階層的フロー・チャート HCP を用いたプログラム改造法，情報処理学会第 21 回全国大会 (1980).
23) 桑原：構造的プログラミングの理論と応用，日経エレクトロニクス (1976. 6. 28).
24) 花田他：ソフトウェア・プロダクション・コントロールセミナ，設計(1)，設計法とドキュメンテーション日科技連，SP テキスト (1980).
25) Ben Shneiderman et al: Experimental Investigation of the Utility of Detailed Flowcharts in Programming, CACM, Vol. 20, No. 6 (1977).
26) Haskell, R. E.: Use of Structured Flowcharts in the Undergraduate Computer Science Curriculum, SIGCSE Bulletin, Vol. 8, No. 3 (Sep. 1976).
27) Programming Language FORTRAN, ANSI X 3. 9-1978.
28) Wirth: Report of the Programming Language Pascal (和田訳，プログラム言語 PASCAL の文法，bit Vol. 8, No. 4 (1976)).
29) Frank De Rember et al.: A Syntax Diagram for (Preliminary) Ada, SIGPLAN Notices Vol. 15, No. 7 & 8 (1980).
30) 流れ図，日本工業規格 (JIS-C 6270-1975).
31) Hoare: Data Reliablity, Proc. ICRS pp. 528-533 (1975).
32) CONVENTIONS FOR STRUCTURE DIAGRAMS (BSI), ISO/TC 97/SC 7 N 215 (1980).

(昭和56年2月18日受付)