奈良産業大学『産業と経済』第3巻第4号(1989年3月)39 － 62

会計システムにおける構造化プログラミング技法

西 口 清 治

Ⅰ はじめに

1960年代後期にソフトウェア危機が叫ばれ，1970年代にはソフトウェアの信頼性と生産性向上のための技法が多く開発された。それらはソフトウェア工学という分野を形成するに至った。ことに，1960年代において，オランダの数学者のEdsger W Dijkstra から，分かりやすいプログラムのための技法として構造化プログラミング論（ストラクチャード・プログラミング，structured programing）が提唱され，その後，ソフトウェアの開発現場に受け入れられたことは，そのソフトウェア開発環境を取りまく時代・状況を大きく特徴づけるものであった。当時においては，プログラム手続きの流れにおいて，“GO TO 文”を用いることにより，流れを変更することが，一般的であった。つまり，処理の流れを厳密に規定していなくても，安易にこれを用いて論理の流れを制御することができ，その結果論理の流れを混乱化することになり，プログラムが複雑となる傾向であった。その解決策として，構造化プログラミング技法が導入され，発展してきたわけである。会計処理のシステム化においても，当然それらの影響を受け，ＧＯＴＯを避けてプログラミングすることになる。そして，“よいプログラミング”のための一つの哲学として構造化プログラミングによるシステム設計あるいは処理のプログラミング技法が定着してきたのである。

現実に，プログラミングの際，どのような構造化プログラミング手法が用いられているかについて，日経コンピュータ誌の調査(図表１)によると，従来からのフローチャートによる技法が大部分を占めているが，国産コンピュータメーカーが開発した技法も一部利用されていることがわかる。

図表1　流れ図・木構造チャートの採用（複数解答）[1]

採用の割合(%)	0 10 20 30 40 50 60
フローチャート（JIS規格）	69.7 (70.7)
疑似コード（IBM）	5.9 (6.5)
NSチャート	3.3 (3.7)
HCPチャート（NTT）	0.7 (0.5)
PAD（日立）	11.8 (5.5)
YACⅡ（富士通）	12.5 (7.9)
SPD（日本電気）	10.3 (5.0)
その他	9.2 (12.3)

注：( )内の数字は前回調査

一方，プログラミングに用いられている言語については，同様に日経コンピュータ誌の調査（図表2）によると，事務処理においては，従来からのCOBOLの利用が現在でも大きい比重を占めている。

図表2　ホスト・コンピュータ別のバッチ処理/オンライン・アプリケーションに主として使用する言語[2]

ユーザー種別（%）	バッチ処理				オンライン・アプリケーション			
	富士通	日本IBM	日立	日本電気	富士通	日本IBM	日立	日本電気
COBOL系	91.5	45.3	83.3	97.5	90.8	38.6	73.5	94.9
FORTRAN系	1.1	—	1.9	—	—	—	2.0	—
PL/I系	4.3	32.0	13.0	2.5	3.4	20.0	10.2	—
RPG系	1.1	10.7	—	—	1.1	7.1	—	—
BASIC系	—	—	—	—	—	—	—	—
アセンブラ系	—	6.7	—	—	1.1	11.4	6.1	2.6
データベース用アクセス言語	—	—	—	—	1.1	1.4	2.0	2.6
第4世代言語	—	2.7	—	—	1.1	12.9	2.0	—
その他	2.1	2.7	1.9	—	1.1	8.6	4.1	—
回答者数	94	75	54	40	87	70	49	39

(1) 上村孝樹，田中淳稿，『第3回バックログ関連実態調査 急増するバックログ』，「日経コンピュータ」1989. 1．16号，p. 77。
疑似チャート　(pseudo language，プログラム記述言語Program Description Languageとも言う。日常言語を用いて処理の構造化を表現する方法)
NSチャート　(Nassi-Shneidermannチャート，構造化フローチャート，構造化ダイヤグラムとも言う。長方形の枠内に一定の構造パターンを用いて記述する方法)
以下は木構造にて表現する方法である。
HCP (Hierarchical and ComPact description chart，開発者:NTT)
PAD (Problem Analysis Diagram: 日立)
YACⅡ (Yet Another Control chart Ⅱ：富士通)
SPD (Structured Programming Diagram: 日本電気)

(2) 上村孝樹，田中淳稿，前掲書，p. 78を一部加工した。

そこで，本稿ではかかる現状とソフトウェア工学の成果を踏まえつつ，現実に利用されている技法・プログラミング言語から，会計システムの設計における“わかりやすいプログラム”を開発するための構造化プログラミング技法に関する概念及び方法について，主としてソフトウェア・システムの開発サイクル，プログラムの制御構造，構造化プログラミングの技法等の項目についての，概要を述べ今後の方向性を考察するものである。

Ⅱ　ソフトウェア開発のサイクル

現在，コンピュータの利用分野の中で会計処理はいわば定型化した一般的な業務として定着している。そして，会計処理とその会計システムおよびソフトウェアの開発技法の特質は会計固有のものではない。そこで，ソフトウェア開発に関して，まずソフトウェアの開発サイクルについて検討する。

一般に，ソフトウェア開発サイクルについては，ソフトウェアが開発されて，もはや使用されなくなるまでの循環過程をソフトウェアのライフサイクルと呼び次の段階に分類する。

(1)　要求定義

開発システムについての課題，目的，処理内容等の要求を明確にする段階である。通常，システム要求仕様書あるいは機能仕様書が作成される。この記述にデータ・フロー・ダイヤグラム（ＤＦＤ）等のデータの流れ図が用いられる。システム・エンジニア（ＳＥ：system engineer）あるいはシステム・アナリスト（system analyst）が担当する。

(2)　システム設計

外部設計とも言われ，要求定義の段階で作成された利用者の要求を，満足するシステムを設計する段階である。ここでは，システム設計仕様書あるいは外部設計仕様書が作成される。通常，ＳＥが担当する。

(3)　プログラム設計

内部設計あるいは詳細設計とも言われ，システム機能内部の細分化を設計する段階である。ここで，プログラム仕様書あるいはプログラム設計仕様書が作成される。この段階で，流れ図・ＰＡＤ・ＨＩＰＯ等の構造化プログラミング技法を用いた図式的な表現で処理手順が記述される。システム設計をより具体的な処理の流れに分解して設計する。プログラマ(programmer)が担当する。

(4)　コーディング

プログラム開発とも言われ，プログラムのコーディングの段階である。プログラム仕様書に従って，ＣＯＢＯＬ等のプログラミング言語を用いてソースプログラム（source program：原始プログラム）にコード化（プログラミング）する。

(5) システム・テスト

要求定義の要件を満足するシステムかどうかのテストの段階である。個々のプログラムのテスト，システム全体を対象とするテストを行う。つまり，プログラムのコンパイル（コンピュータを使って，ソース・プログラムを機械語であるオブジェクト・プログラムに翻訳する）等の作業を行い，プログラムが完了すると，個々のモジュール（プログラム）上のエラー（バグ：bug，虫）を取る作業（デバッグ：debug, 虫取り）を行う。

プログラムのデバッグが完了すると，個々のプログラムをテスト・データでテストし，システムの各プログラムの統合テストを行う。

(6) 導入と保守

業務への移行・導入，その後のシステムの保守の段階である。本番稼働後のエラーの修正，業務変更によるシステム変更等のシステムの運用に付随して発生するプログラムの変更を行う。そして，プログラムが機能を果たさなくなるまで，システムの維持・保全に努めることになる。

このような段階をソフトウェアは辿るものと考えられている。本稿で扱うのは主として，自社開発，自社利用のプログラムの開発サイクルであり，上記のプログラム設計の段階を中心に考察している。

次に，ソフトウェア開発で，“わかりやすいソフトウェア”を開発する上で考慮しなければならないことの一つは，ソフトウェアの良さを測定する評価基準についての問題である。ソフトウェアの良さは，ある面ではプログラムの品質とも考えられる。プログラムの品質を考慮すべき基準として考えられるものに次の項目がある。

プログラムの品質（software quality characteristics）[3]

(1) 機能性（functionality）

システムで要求される機能を満足するか，どうかの基準である。性能として，まず機能が考慮される。

(2) 信頼性（reliability）

エラーの少ないプログラムか，どうか。

(3) 使いやすさ（usability, easy to use）

操作性がよいプログラムか，どうか。

(4) 効率性（efficiency）

効率は，時間経済性（time economy）と資源経済性（resource economy）の面で検討されることになる。

---

(3) 東基衛，菅忠義，松原友夫稿，『ソフトウェア工学における標準化動向　今後の課題』，「情報処理」，情報処理学会，Vol. 28 No. 9，Sep. 1987，p. 1176。

(5) 保守性（maintainability）

保守容易性とも言われ，容易にプログラムの保守ができるかどうかという点に関するものである。

(6) 移植性（portability）

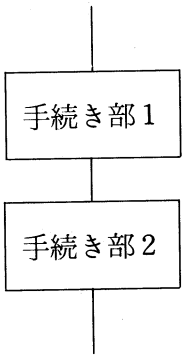
ある機種用に作成したソフトウェアを他機種に容易に利用することができるかどうかという基準であり，ソフトウェアの拡張性とも，汎用性とも考えられる項目である。

等が考えられている。しかしながら，これらの基準を全て満足するものが必ずしも良いソフトウェアとは判定できないところに，ソフトウェアの品質評価の難しさがある。例えば，使いやすさを強調すると，効率性が幾分犠牲になることがある。あくまで業種，業務等のシステム化要求要件によって異なる。

Ⅲ　構造化プログラミングにおけるプログラムの制御構造

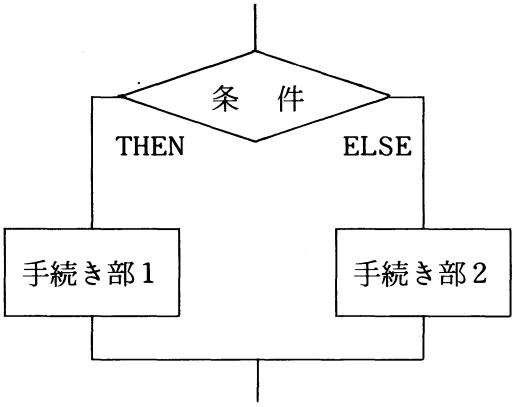
構造化プログラミングの基本的な考え方は制御構造（処理の流れの型）を次図のように，順次・選択・繰返しの３種類の基本的な制御構造だけから構成することができるというものである。これを用いることにより，プログラムの流れを単純化することが可能であり，プログラムを読み易く，また変更・保守のために有効な方法と考えられている。

構造化プログラミングで規準となる定理として，ベーム（Böhm）とヤコピニ（Jacopini）によるストラクチャード定理がある。これは，「１つの入口と１つの出口をもつようにプログラム（適正プログラム）を設計すれば，この３つの型で，どんな論理でも記述できるというものである。」[(4)] この３つの基本単位とは下図の順次（SEQUENCE），選択（IFTHENELSE），繰返し（DOWHILE）である。

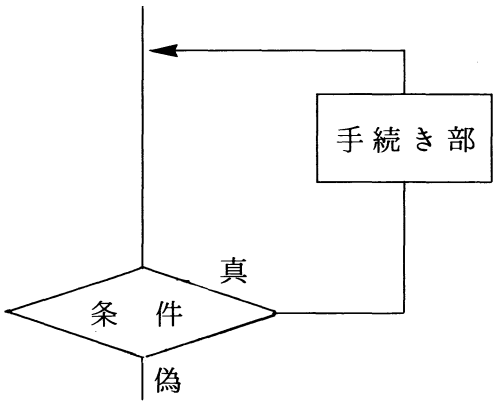


(1) 順次（sequence）

(4) 國友義久著，「効果的プログラム開発技法（第３版）」，近代科学社，1988，p. 137。



(2) 選択（IF THEN ELSE型）



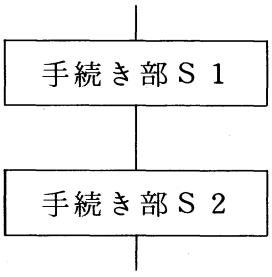
(3) 繰返し（DO WHILE型）

構造化プログラミングはこれらの制御構造を用いたプログラム作成においては，基本的に“ＧＯ ＴＯ文”を意識的に避けるが，しかし，論理の構造上“ＧＯ ＴＯ”を用いることが論理の上で容易になるなら，“ＧＯ ＴＯ”を用いる方が，理解しやすいと言える。これは，モジュール（パラグラフ）の構造，大きさによって判断しなければならないことでもある。

また，これらの基本構造だけでプログラムの制御を表現するには，実際上無理があるため通常以下の制御構造を用いる。この構造を一般的なプログラム文とＣＯＢＯＬのプログラムの記述を交えて表すとそれぞれ以下のようになる。

(1) 順次（sequence）

連続，連結（concatenation）とも呼ぶ，手続きを記述順に実行する型である。



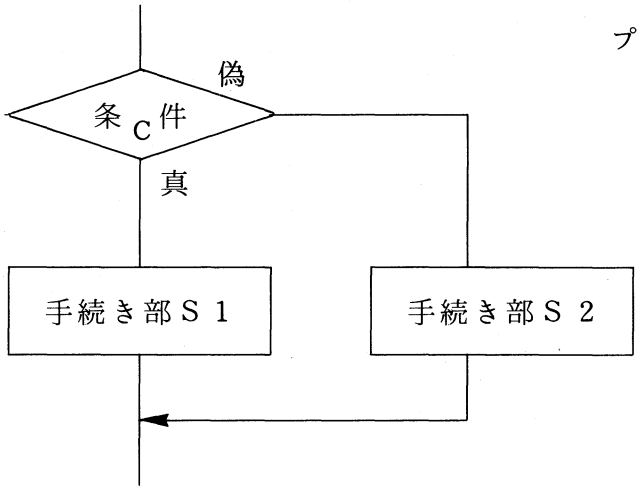
プログラムは次のように表現される。

Ｓ１

Ｓ２

(2) 選択 (IFTHENELSE)

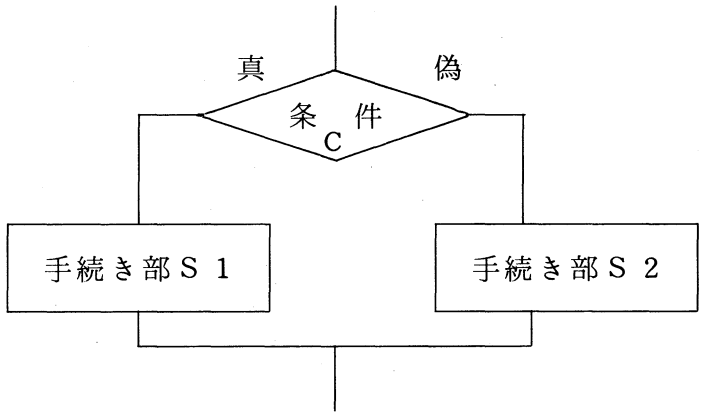
イフゼンエルス型とも呼ばれ，一定の条件によって，手続きを選択する型



プログラムは次のように表現される。

```
IF C
   THEN
        S1
   ELSE
        S2
END－IF
```

上図は下記の様にも表現することができる。



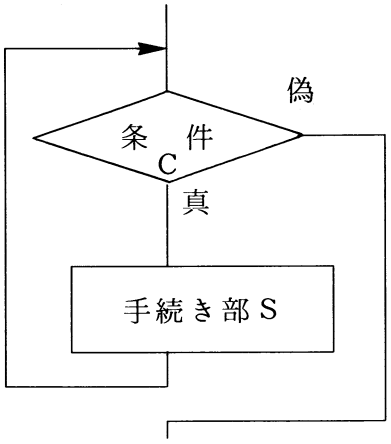
具体的なＣＯＢＯＬプログラム[5]では次のようになる。

```
IF A=B
THEN   MOVE 1 TO X
ELSE   READ M－FILE
       AT END
        IF T = HIGH－VALUE
        THEN MOVE 2 TO X
        FLSE MOVE 3 TO X
        END－IF
       END－READ
END－IF
```

(5) ここでのＣＯＢＯＬの例は，今城哲二稿，『COBOLの標準化の動向』，「情報処理」，情報処理学会，Vol. 24 No. 9，Sep. 1983，pp. 1062－1069のプログラム例を引用させていただいた。

(3) 繰返し (DOWHILE型)

繰返しは，反復，前判定反復，ドゥファイル型，WHILE型とも呼ばれる。条件が真である間手続きを実行し，条件が偽となると繰返しループからでる型である。



プログラムの基本型は次の様になる。

```
while  C  do
           S
```

COBOLプログラムの基本型は次のようになる。

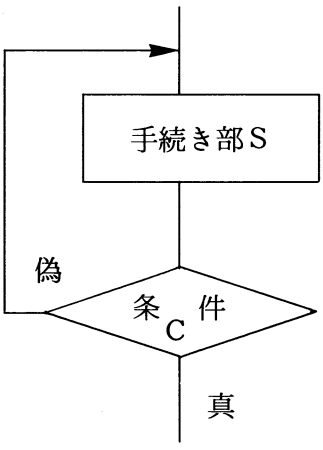
```
IF     C
       PERFORM  S
```

具体的なCOBOLプログラムでは次のようになる。

```
PERFORM
〔WITH  TEST  BEFORE〕
    UNTIL  条件
        手続き部，繰返し処理の命令群
    END−PERFORM
```

(4) 繰返し (REPEAT UNTIL型)

これは，後判定反復とも，リピートアンティル型とも呼ばれる制御構造である。手続きを実行し，条件が偽の間手続きを繰り返す。そして，条件が満たされれば繰返しのループから出る型である。



プログラムの基本型は次の様になる。

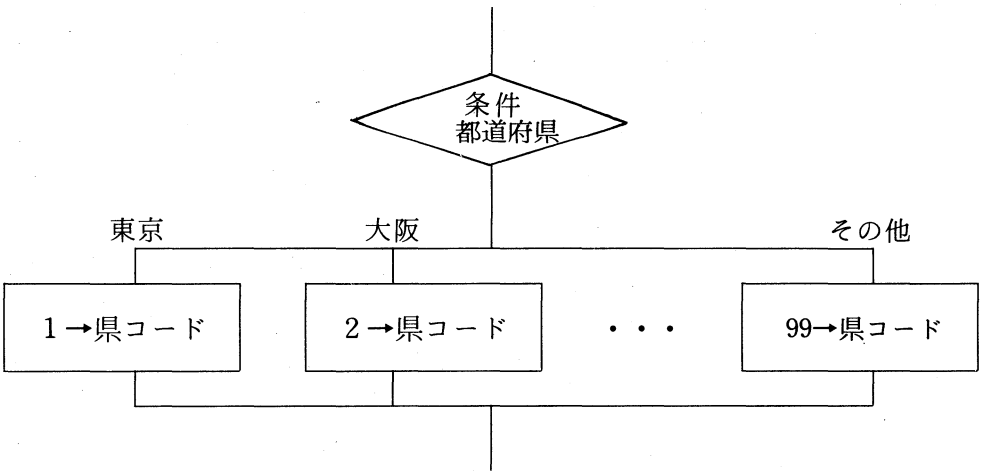
```
repeat
   S
until   C
```

COBOLプログラムの基本型は

```
PERFORM
WITH  TEST  AFTER
   UNTIL  条件
        手続き部，繰返し処理の命令群
END−PERFORM
```

(5) 多方向分岐型 (CASE型)

多岐選択型，DOCASE型とも呼ばれる型であり，条件の値により多方向の手続き部に分岐する制御構造である。具体的な型をCOBOLプログラムを示せば次のようになる。



プログラム例

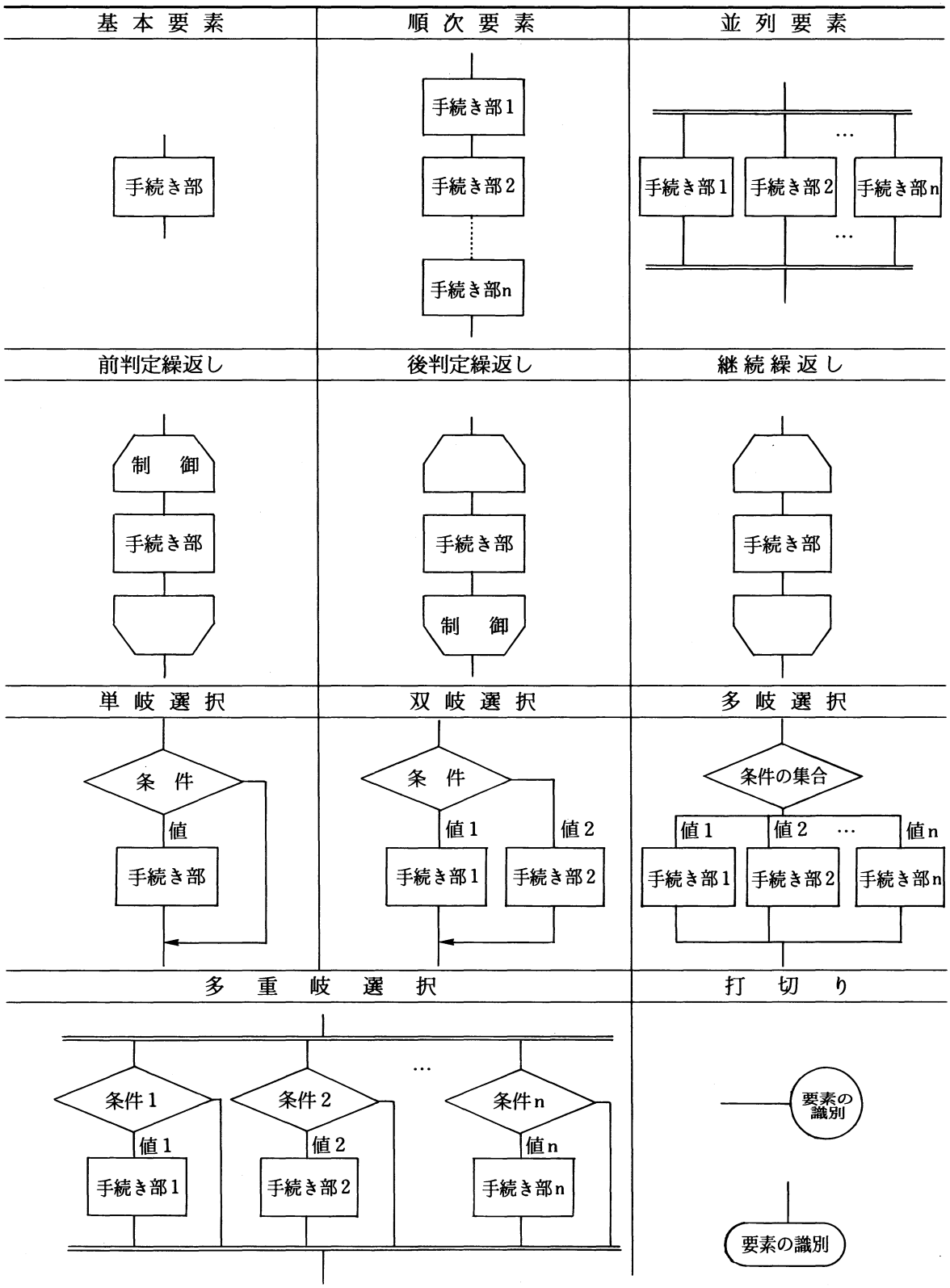
```
EVALUATE   TO−DO−FU−KEN
    WHEN   "TOKYO"   MOVE   1   TO   KEN−CODE
    WHEN   "OSAKA"   MOVE   2   TO   KEN−CODE
                .
                .
                .
    WHEN    OTHER    MOVE   99  TO   KEN−CODE
END−EVALUATE
```

さらに，これらの制御構造は日本工業規格にて次のように分類されている。また，その表記法も規定されており，最も汎用的は技法として，標準的に利用することが可能となっている。

プログラムの構成要素およびプログラム流れ図

1．基本要素（inperative construct）
2．順次要素（serial construct）
3．並列要素（parallel construct）
4．繰返し要素（iterative construct）
(1) 前判定繰返し（pre-tested iteration）
(2) 後判定繰返し（post-tested iteration）
(3) 継続繰返し（continuous iteration）
5．選択要素（selective choice construct）
(1) 単岐選択（monadic selective）
(2) 双岐選択（dyadic selective）
(3) 多岐選択（multiple exclusive selective）
(4) 多重岐選択（multiple inclusive selective）
6．打切り（termination）

図表３　プログラム構成要素及びその表記法（ＪＩＳ　Ｘ0121による表現）(6)



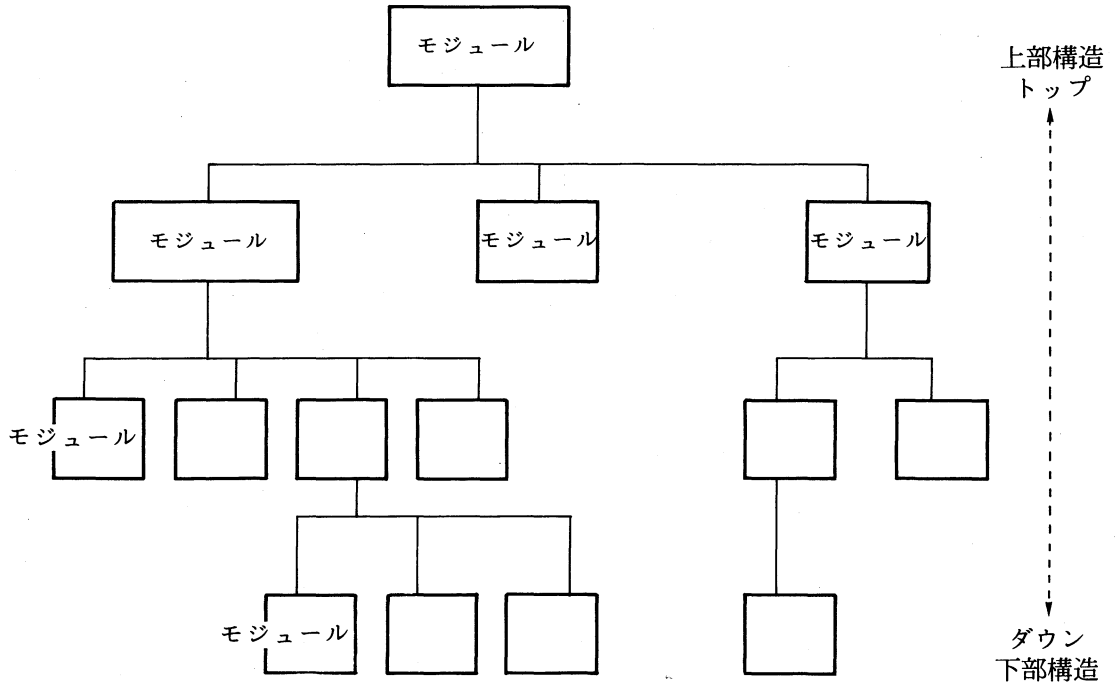
(6) 長野宏宣，東基衛稿，『プログラム構成要素及びその表記法』，「情報処理」，情報処理学会，Vol.28 No.9，Sep. 1987，p. 1169

Ⅳ　構造化プログラミング技法

会計処理のコンピュータ化において，プログラムを作成する際に考慮しなければならないことの一つは，“わかりやすいプログラム”になるようプログラムを作成するということである。ハードウェア・コストがコンピュータ・コストの大部分を占めていた時代では，総費用を縮小するため，あるいは処理速度を高めるために名人芸的なプログラムが作られた。しかしながら，ソフトウェア要員需要の増大，バックログ（開発待ちソフトウェア）の増大等から，現在ではソフトウェア開発の生産性向上が強く意識される状況に至っている。1970年代に開発された構造化プログラミングの考え方はその一つの解決策であった。

構造化プログラミングを実施する際の基本的な思考は，プログラムをトップダウンで作成し，プログラムをモジュール化することが可能であると言うことである。

図表4　トップダウンとモジュール構造



トップダウン，つまりトップダウン・アプローチ，あるいは，「トップ・ダウン・プログラミングは，トップ・ダウン設計で開発したシステム構造（あるいはプログラム構造）を，コーディングしたモジュールの形に具現化していく１つの過程である。より具体的に表現すれば，階層構造化したモジュールを最上位レベルからより下位レベルへ，順次にコーディングし，テストしていく過程である。」[(7)]すなわち，上部の階層の下に下部の階層，あるいは総括記述の下に

(7)　國友義久著，前掲書，p. 193。

詳細記述を行うような，階層状のモジュール構造化が前提である。つまり，制御が上位モジュールから下位モジュールに移り，そしてその下位モジュールの処理の完了により上位モジュールに制御が戻ってくる機能間の従属関係を持つ構造が必要となる。

ここで，モジュール（module：構成部分）とは「手続きやデータの宣言から成る言語構成要素であって，他の同様の構成要素と相互に作用しうるもの」[8]と定義される。したがって，プログラムは「一つ以上の互いに関連するモジュールの論理的集まり」[9]と規定され得る。なお，モジュールあるいはプログラムの分割単位について考慮すべき点は次の事項[10]であると言われている。

(1) 個々のモジュールの大きさがプログラマにとって扱いやすいものである。すなわち作成したり，読んで内容を理解したりするのに適当なサイズである。

(2) モジュールが仕様（モジュールの果たすべき抽象的機能）通り機能するか否かは，そのモジュールが使用される文脈には依存しない。

(3) モジュールに対し，その仕様を満たすどのような具体的実現を与えても，そのモジュールを使用する他のモジュールの機能には影響しない。

これらのトップダウンとモジュール化の考えは，構造化プログラミングにおいて重要と思われる。

次に，構造化プログラミングを実施する際に利用される構造化プログラミング技法について検討してみよう。プログラム開発において，プログラム，ソフトウェアの生産性向上と品質改善のために，次のような「効果的プログラム開発技法（IPT：Improved programing Technologies)」が用いられる。

1．データ・フロー・ダイヤグラム（DFD：Data Flow Diagram)

データの流れを中心にシステムを図化するもので，データの源泉（出所）/吸収（データの到達先)，データ・フロー，処理（機能)，データ記憶を基本要素として表すものである。

2．HIPO図（ハイポ図：Hierarchy plus Input－Process－Output)

プログラムを階層構造で構成し，その構造のそれぞれを入力，処理，出力機能として記述する方法である。

大きくは次の目的図，概略図，詳細図から構成される。

目的図　システムの諸機能を記述する目次となる

---

(8) JIS X0015－1987 情報処理用語（プログラム言語）15.01.09。

(9) JIS前掲書，15.01.11。

(10) 湯浅太一，小島啓二，中島玲二稿，『モジュラー・プログラミングのための支援環境』，「情報処理」，情報処理学会，Vol.23 No.5 May 1982, p.433。

概略図　機能を構成する入力，出力，処理作業の概略の記述
　　拡張記述シート
詳細図　特定の機能を図、注釈で補足して記述する

3．プログラム記述言語（PDL：Program Description Language）

疑似言語，疑似コードとも言い，日常語に近い表現でプログラム手順を記述する方法であり，これを特定のプログラム言語を用いてプログラムに書き直すだけでプログラミングが完了する。

4．NSチャート（Nassi－Shneidermann Chart）

構造化フローチャート，構造化ダイヤグラム，Chapinチャートとも言う。長方形の枠内に一定の構造パターンを用いて記述する方法であり，制御構造は一つの入口と一つの出口を持つことになり，構造化された階層構造を表現するには適した技法である。

5．複合設計（Composite Design）

構造化設計，機能設計とも呼ばれ，設計段階でわかりやすいプログラムを開発する方法である。モジュールの機能，論理，インターフェイス等の属性に注意を払って設計する。

まず，問題の記述から，分析手法を用いてプログラム全体の構造を設計する。すなわち，プログラムを独立したモジュールごとに細分化する。完了した構造設計の客観的評価としてモジュール強度（個々のモジュールの良さ）を最大に，モジュール結合度（モジュール間の相互の連結・関連性）を最小にする（独立を保つ）よう試みる。そして設計を再度検討し，構造の最適化を図る。[11]

6．ジャクソン法

流れ図を避けて，階層的に構造化する。基本，連接，繰返し，選択の４つのコンポネント（プログラムの一部）を用いて，次の手順で設計する技法である。

(1) 問題の状況を考え，処理されるべきデータの構造を定義することによって理解したことを記録する。

(2) データ構造に基づいてプログラム構造を作る。

(3) 可能な基本演算により実行されるべき仕事を定義する。次に演算のそれぞれをプログラム構造の適当なコンポネントに割り当てる。[12]。

---

(11) Glenfold J. Myers, *Reliable Software through Composite Design,* Mason/Charter Publishers, Inc., 1975.
久保未沙・國友義久訳，「高信頼性ソフトウエア 複合設計」，近代科学社，1976。

(12) Michael A. Jackson, *Principles of Program Design,* Academic press, Inc. (London) LTD., 1975.
鳥居宏次訳，「構造的プログラム設計の原理」，日本コンピュータ協会，1980。

7．ワーニエ法

この方法は，プログラム構造設計の土台に，一定の細分法則に従って作成する階層型のデータ構造を使用し，構造図作成の手段として，集合論の考え方をデータと業務の分析に取り入れた。まず，論理設計のステップで，階層構造による処理のモデル化，流れの把握を行う。次に，詳細設計のステップで，その業務に必要な処理命令と，それらの命令が使用される場所（モジュール）との関連づけを行い。最後のステップで，プログラム言語によるコーディングする。[13]

8．PAD（問題分析図：Problem Analysis Diagram）

この技法は日立製作所で開発され，一企業の技法と言うよりも，広く一般化しつつあり，参考書などの資料が多く発表され，今後の普及が期待される問題分析図である。この PADの 特徴は，[14]

(1). プログラムの構造を見えるようにする。

(2). プログラムの製造（コーディング）と検査を系統的にできるようにする。

(3). プログラムで処理するデータの構造を，プログラムの構造と同じ記法表現できる。

等である。そしてPADで作成されれば，次のプログラミングはいわば自動的に作成可能である。

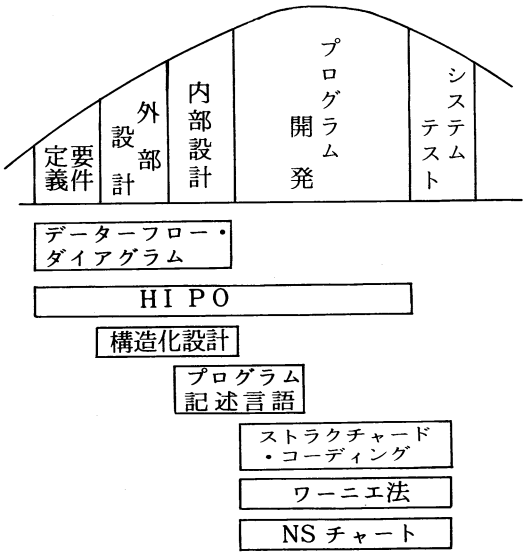
9．ウオークスルー

(Walk-Through：下稽古)

構造化ウオークスルーとも言われ，本番稼働前に行うのテストである。プログラムを実際に稼働させる前に各担当者が机上でリハーサルを行い，欠陥やエラー等を検討する技法である。

なお，これらの技法のいずれが最も望ましいか，効率が良いか等については，未だ結論は出ていない。多方面からの分析が必要である。例えば，ある分析では，ソフトウエア開発サイクルにおいて，プログラム開発技法のエラー防止技術では，次図の様に対応できると言われている。

図表5　おもなエラー防止技術[15]



---

(13) 鈴木君子稿，『構造化プログラミングワーニエ・メソド』，「情報処理」，情報処理学会，Vol. 25 No. 9，Sep. 1984，p. 946.

(14) 二村良彦著，「プログラム技法 PADによる構造化プログラミング」，オーム社，1984，p. 43。

(15) 國友義久著，前掲書，p. 228。

また，PADによると，プログラムの生産性が1.3～1.8倍に向上したと感じているという調査結果[16]も出ている。

これらの技法が普及することが望まれるが，わかりやすいプログラムや構造化プログラミング法によるプログラムは開発においては、現実には論理構造を充分に検討しておかなければならず，意外にこれらの技法で作成すると負担になる場合がある。但し，修正・変更の際には効率よく作業ができる。

V むすび

以上，種々の構造化プログラミングの技法を検討したが，どの技法が最も良いものであるかどうかは現状では評価する規準あるいは方法がない。したがって，どれを採用するかは，その技法を習得する環境，すなわち，それらの技法の資料や文献を入手あるいは，教育の機会があるかどうかによることになるであろう。

特に，わが国で利用されている構造化プログラミング技法は主にコンピュータメーカーの開発によるものが多く，従って，資料の多くは市販されているものが少なく，資料の入手が難しいものが多い。また，これらの手法に関する講習会の参加もユーザー又は潜在的ユーザーに限定されることも多い。

わが国の利用者が採用するプログラミング技法は，導入したコンピュータメーカーが提供するものを利用することが多い[17]現状であるが，情報化社会への移行に伴うソフトウェア技術者不足から，これらの技法が広く社会へ普及するよう努力することも急務である。

今後とも“わかりやすいプログラム”のための研究は行われていくであろうが，その技法が“名人芸”であってはならないし，標準化された方法であらねばならない。例えば，ソフトウェアの部品を再利用可能なプログラム化し，設計仕様をライブラリ化し，利用する方法もある。それによって生産性と信頼性の向上に貢献するかもしれない。また，プログラムの中に処理図を記載し，そこに処理名，処理システム，プログラム番号，作成者，等の概要書を記載し，プログラムを見る者が仕様書がなくても，プログラムを理解できるようにすることも一つの方法である。

また，コンピュータの用語の常であるが，概念や用語はメーカー，論者によって意味が違う。コンピュータのような新しい分野においては，一つの概念が誕生すると利用する立場の相違により微妙に違う可能性があり，注意が必要である。そこで，オーソライズする機関，ＪＩＳの役割に期待することになる。

---

(16) 二村良彦稿，『構造化プログラム図式』，「コンピュータソフトウェア」，日本ソフトウェア科学会，Vol. 1 No.1 Apr. 1984 p. 71。

(17) 上村孝樹，田中淳掲書，p. 77。

現在，特にバックログ（開発待ちソフトウェア）が増大して，深刻化しつつある。今後とも，コンピュータ化への要求とその生産能力との差は拡大していくことが予想され，現在のCOBOL，PL/I 等の手続型言語の限界が来るかもしれない，近年の第四世代言語の使用が期待される。

本稿では，構造化プログラミングの技法について，基礎となる所を検討した。各技法の分析は今後に述べたい。

資 料

日本工業規格（JIS）による情報処理用流れ図の概要を次に示す。

出典：情報処理用流れ図・プログラム網図・システム資源図記号
日本工業規格（JIS）X0121—1986

この規格は次の図の中で用いられる記号（symbol：データ記号，処理記号，線記号及び特殊記号）及び用法を規定したものであるが，これらの図に限定するものではない。

1．流れ図（flowchart）

(1) データ流れ図（data flowchart）

データの経路を表し，データ媒体と処理手順を定義するもの。

(2) プログラム流れ図（program flowchart）

プログラム中における一連の演算，制御の流れを表す。これをプログラミングしたものが，ソース・プログラムである。

(3) システム流れ図（system flowchart）

システムの演算の制御及びデータの流れを表す。

(4) プログラム流れ図（program network chart）

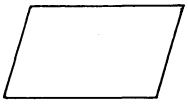
プログラムを起動する経路及び関係するデータとの相互作用を表す。

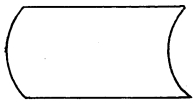
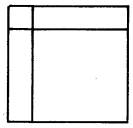
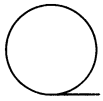
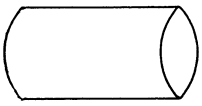
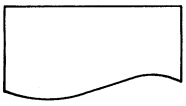
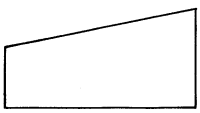
(5) システム資源図（system resource chart）

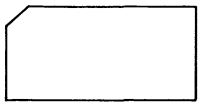
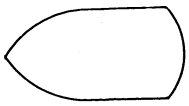
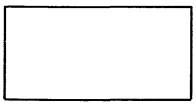
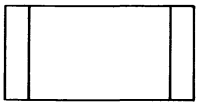
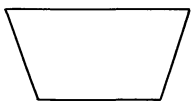
問題を解くのに適したデータや処理に関する機能単位の構成を表す。入力装置，出力装置，記憶装置，処理装置等が記載される。

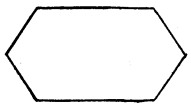
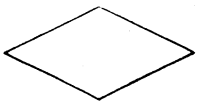
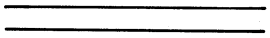
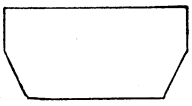
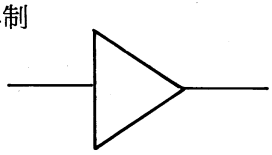
2．記号（symbol）

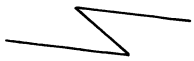
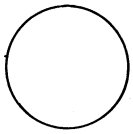
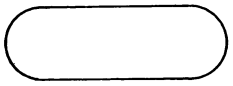
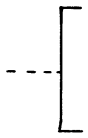
＊印はその図で用いられる記号

記号の名称	データ流れ図	プログラム流れ図	システム流れ図	プログラム網図	システム資源図
1．データ記号（data symbol）					
1.1 基本データ記号（basic data symbol）					
(1) データ（data）：媒体を指定しないデータ。 	＊	＊	＊	＊	＊

(2) 記憶データ（stored data）：媒体を指定しない記憶データ。 	*		*	*	*
1.2 個別データ記号（specific data symbol）					
(1) 内部記憶（internal storage）：内部記憶を媒体とするデータ。 	*		*	*	*
(2) 順次アクセス記憶（sequential access storage）：磁気テープ等。 	*		*	*	*
(3) 直接アクセス記憶（direct access storage）：磁気ディスク，磁気ドラム，フレキシブルディスク（フロッピーディスク）等。 	*		*	*	*
(4) 書類（document）：人間の読める媒体上のデータ。媒体はプリンタ，光学的文字読取り装置又は磁気インク読取り装置の書類，マイクロフィルム，計算記録，帳票等。 	*		*	*	*
(5) 手操作入力（manual input）：手操作で入力するデータ。オンラインけん盤，スイッチ，押しボタン，ライトペン，バーコード等。 	*		*	*	*

(6) カード（card）：磁気カード，マーク読取りカード等。	*		*	*	*
					
(7) せん孔テープ（punched tape）：せん孔テープを媒体とするデータ。	*		*	*	*
					
(8) 表示（display）：情報を表示する媒体上のデータ。表示装置の画面，オンラインインディケータ等。	*		*	*	*
					
2 . 処理記号（process symbol）					
2.1 基本処理記号（basic process symbol）					
(1) 処理（process）：任意の種類の処理機能を表す。	*	*	*	*	*
					
2.2 個別処理記号（specific process symbol）					
(1) 定義済み処理（predefined process）：サブルーチンやモジュール等，別の場所で定義された処理。		*	*	*	
					
(2) 手作業 (manual operation) 人手による任意の処理。	*		*		
					

(3) 準備（preparation）：スイッチの設定，指標レジスタの変更，ルーチンの初期設定等，後の動作に影響を与えるための命令・命令群の修飾を表す。	*	*	*	*	
					
(4) 判断（decision）：一つの入口と一つ以上の出口を持ち，記号で定義された判断の条件によって，出口を選択することを表す。		*	*	*	
					
(5) 並列処理（parallel mode）：二つ以上の並行した処理を同期させることを表す。		*	*	*	
					
(6) ループ端（loop limit）：繰返し（反復）処理を表す。テスト命令の位置によって，ループ始端又はループ終端の記号中に，初期化，増分，終了条件を記載する。		*	*		
					
					
3．線記号（line symbol）					
3.1 基本線記号（basic line symbol）					
(1)線（line）：データ又は制御の流れを表す。必要なら矢先を付ける。	*	*	*	*	*
3.2 個別線記号（specific line symbol）					
(1) 制御移行（control transfer）：一つの処理から他の処理へ制御が即時に移行することを表す。				*	
					

(2) 通信（communication link）：通信線によってデータを転送することを表す。 	*		*	*	*
(3) 破線（dashed-line）：二つ以上の記号の間の択一的な関係を表す。また，注釈の対象範囲を囲むのにも用いる。 － － － － － －	*	*	*	*	*
4．特殊記号（special symbol） (1) 結合子（connector）：線を中断し他の場所に続けたりするのに用いる。 	*	*	*	*	*
(2) 端子（terminator）：外部環境への出口，外部環境からの入口を表す。 	*	*	*		
(3) 注釈（annotation）：説明又は注を付加するのに用いる。 	*	*	*	*	
(4) 省略（ellipsis）：三つの点で，線記号の省略を表す。 — ・・・ —　　| ・・・ |	*	*	*	*	*

参考文献（主として，入手が容易なもの）

1．総合書

1. 菅野文友著，「ソフトウェア・エンジニアリング」，日科技連出版社，1979
2. 國友義久著，「プログラム開発管理」，オーム社，1982
3. 宮本勲著，「ソフトウェア・エンジニアリング　現状と展望」，TBS出版，1982
4. 菅野文友編，「ソフトウェアの品質管理」，日科技連出版社，1986
5. 菅忠義著，「ソフトウェア開発の実際－標準とその活用－」，日本規格協会，1988
6. 有沢　誠著，「ソフトウェア工学」，岩波書店，1988
7. General Electric, Software Engineering Handbook, General Electric Company, 1988
   井上義祐・岡野寿夫・荒川淳三訳，「ソフトウェア工学ハンドブック」，マグロウヒルブック，1985
8. 山下英男監修，「共立 総合コンピュータ辞典第２版」，共立出版，1982
9. 江村潤朗編，「図解 コンピュータ百科事典」，オーム社，1986
10. 日本経済新聞社編，「情報処理ハンドブック」，日本経済新聞社，1988
11. 日本情報処理開発協会編，「情報化白書1988」，コンピュータ・エイジ社，1988
12. 情報サービス産業協会編，「情報サービス産業白書1988」，コンピュータ・エイジ社，1988，
13. 日本規格協会編，「ＪＩＳハンドブック 情報処理 ソフトウェア編」，日本規格協会，1988
14. 日本規格協会編，「ＪＩＳハンドブック 情報処理 用語・コード編」，日本規格協会，1988
15. 日本規格協会編，「ＪＩＳ テキスト情報処理1988」，日本規格協会，1988

2．構造化ＣＯＢＯＬ等

1. R. M. Armstrong, Modular Programing in COBOL, John wiley & Sons, Inc. New York, 1973
   大日方真訳，「COBOLによるモジュラー・プログラミング」，TBS出版会，1975
2. CODASYL, Structured Programming in COBOL －Future and Present
   日本経営協会訳，「COBOLによるチトラクチャード・プログラミングその現状と将来」，日本経営出版会，1977
3. D. D. Mc Cracken, A Simplified Guide to Structured COBOL Programming, John Wiley & Sons, Inc., 1976
   加藤満，村原貞夫訳，「構造化COBOL入門」，テクノ，1979
4. ㈱日立製作所コンピュータ事業本部教育センタ部，「コンピュータ教材　構造化COBOLプログラミングⅠ」，日刊工業新聞社，1983
5. ㈱日立製作所コンピュータ事業本部教育センタ部，「コンピュータ教材 構造化COBOL プログラミングⅡ」，日刊工業新聞社，1983
6. ㈱日立製作所コンピュータ事業本部教育センタ部，「コンピュータ教材 構造化COBOL VSAMプログラミング」，日刊工業新聞社，1983
7. 今城哲二稿，『COBOLの標準化の動向』，「情報処理」，情報処理学会，Vol.24 No.9, Sep.1983, pp. 1062－1069
8. 遠山暁，海老澤榮一著，「ストラクチャードプログラミングCOBOL」，実教出版，1983

9. 澤田晃著,「アルゴリズムの作り方」, 共立出版, 1985
10. A. S. Philippakis, L. J. Kazmier, Program Design Concepts with Applications in COBOL, McGraw-Hill, Inc., 1983
石井正躬, 鑰山徹, 戒田秀二訳,「COBOLプログラム設計入門」, 共立出版, 1986
11. 小島崇弘, 内野明, 町田欣弥著,「例解構造化COBOL」, 実教出版, 1986
12. 木立義也, 務台哲郎, 穂積和子, 太田純, 須藤恵子著,「ジャクソン法による構造化COBOLプログラミング, 啓学出版, 1986
13. 小野博敏著,「実習構造化COBOL」, オーム社, 1986
14. 加藤昭著,「COBOL—85構造化プログラミング入門」, 技術評論社, 1986
15. 木ノ下勝郎著,「COBOL85実践事務処理プログラムの考え方」, 啓学出版, 1987
16. 高橋守清,「COBOL 85 プログラミング」, オーム社, 1988
17. 中山二夫, 太田宗雄著,「JIS準拠COBOL文法」, 共立出版, 1989
18. 中山二夫, 中原幹雄, 川崎聡著,「構造化COBOLプログラミング」, 共立出版, 1989
19. 河西朝雄著,「構造化BASIC」, 技術評論社, 1985
20. 篠原靖市著,「構造化プログラミング入門」, 一橋出版, 1986

3. 流れ図等の諸技法

1. 若山芳三郎, 吉川信之著,「新しいJISによるコンピュータのためのフローチャートの考え方・書き方」, 啓学出版, 1987
2. 花田収悦著,「プログラム設計図法」, 企画センター, 1983
3. 二村良彦著,「プログラム技法 PADによる構造化プログラミング」, オーム社, 1984
4. 二村良彦郎稿,『プログラム設計法 PAD/PAM』,「情報処理」, 情報処理学会, Vol. 25 No.11, Nov. 1984, pp. 1237—1246
5. Computer Today 別冊,「PAD—構造化プログラム開発技法」, 1985
6. 川合敏雄著,「PADプログラミング」, 岩波書店, 1985
7. 金敷準一著,「PAD入門」, サイエンス社, 1988
8. 河村一樹著,「PADによる構造化プログラミング」, 啓学出版, 1988
9. 岡本茂監修, 加藤木和夫著,「PADによるプログラム技法」, 啓学出版, 1988
10. Krishna K. Agarwal, Programming with Strunctured Flowcharts, Petrocelli Books, Inc. 1984
戸内順一著,「構造化フローチャートによるプログラミング入門」, 啓学出版, 1985
11. 國友義久著,「ストラクチャード・プログラミング入門 改訂2版」, オーム社, 1988
12. 國友義久著,「効果的プログラム開発技法 (第3版)」, 近代科学社, 1988
13. 中楯文雄著,「モジュラ・プログラミング」, 日刊工業新聞社, 1986

4. その他の技法

1. 鈴木君子稿,『構造化プログラミング ワーニエ・メソド』,「情報処理」, 情報処理学会, Vol. 25 No. 9, Sep. 1984, pp. 946—954

2. J. D. Warnier, B. M. Flanagan, Entrainement a la Programmation, Les Edition d' Organization, Paris, 1971, 1972
鈴木君子訳,「ワーニエ法によるプログラミング学習 上・下」, 日本能率協会, 1973
3. J. D. Warnier, Les Procedures de Taitement et leurs Donnees Les Editions d Organization, Paris, 1973
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, 1975
4. J. D. Warnier, Entrainement a la Programmation Tome 1, Les Editions d' Organization, Paris, Paris, 1984
鈴木君子訳,「ワーニエ法による構造化プログラムの作り方」, 啓学出版, 1988
5. Michael A. Jackson, Principles of Program Design, Academic press, Inc. (London) LTD 1975
鳥居宏次訳,「構造的プログラム設計の原理」, 日本コンピュータ協会, 1980
6. 峰尾欣二稿,『プログラミング法論 (ジャクソン法)』,「情報処理」, 情報処理学会, Vol.23No. 11, Nov. 1982, pp. 1063−1074
7. 大野侑郎稿,『ジャクソンシステム開発法』,「情報処理」, 情報処理学会, Vol. 25 No. 9, Sep. 1984, pp. 955−962
8. Enid Squire, Introduction ・・・System Design, Addison- Wesley Publishi ng Company, Inc. 1980
原田勝訳,「システム設計入門」, 啓学出版, 1983
9. Glenfold J. Myers, Reliable Software through Composite Design, Mason / Charter Publishers, Inc., 1975
久保未沙・國友義久訳,「高信頼性ソフトウェア 複合設計」, 近代科学社, 1976
10. Glenfold J. Myers, Composite / Structured Design, Van Nostrand Reinhold Company, 1978
國友義久・伊藤武夫訳,「ソフトウェアの複合 / 構造化設計」, 近代科学社, 1979
11. 久保未沙稿,『複合設計』,「情報処理」, 情報処理学会, Vol.25 No.9, Sep. 1984, pp.935−945
12. J. Martin, C. McClure, Diagramming Techniques for Analysts and Programmers, Prentice-Hall, Inc. 1985
國友義久・渡辺純一訳, 「ソフトウェア構造化技法 ダイアグラム法による」, 近代科学社, 1986
13. J. Martin, C. McClure, Action Diagrams Clearly Structured Program Design, Prentice-Hall, Inc. 1985
國友義久訳,「構造化プログラム設計：行動ダイアグラム」, 近代科学社, 1988
14. O.-J. Dahl, E. W. Dijkstra and C. A. R. Hoare, Structured Programming, Academic press, London, 1972
野下浩平, 川合彗心, 武市正人共訳,「構造化プログラミング」, サイエンス社, 1975
15. Brian W. Kernighan, P. J. Plauger, Software Tools, Bell Telephone Laboratories, Incorporated. 1976
木村泉,「ソフトウェア作法」, 共立出版, 1981
16. Brian W. Kernighan, P. J. Plauger, The Elements of Programming Style, Second edi-

tion, Bell Telephone Laboratories, Incorporated. 1978
木村泉，「プログラム書法第2版」，共立出版，1982

17. N. Wirth, Systematisches Programmieren. eine Einfuhrung, B.G. Teubner, 1982
野下浩平，筧捷彦，武市正人共訳，「系統的プログラミング／入門 第2版補訂」，近代科学社，1986

18. 鳥居宏次，二木厚吉，真野芳久稿，『プログラミング方法論の展望』，「情報処理」，情報処理学会，Vol. 20 No. 1，Jan. 1979，p. 22

19. 湯浅太一，小島啓二，中島玲二稿，『モジュラー・プログラミングのための支援環境』，「情報処理」，情報処理学会，Vol. 23 No. 5，May 1982，p. 433

20. 二村良彦稿，『構造化プログラム図式』，「コンピュータソフトウェア」，日本ソフトウェア科学会，Vol. 1 No. 1 Apr. 1984 pp. 64－77

21. 紫合治稿，『ソフトウェア設計法』，「コンピュータソフトウェア」，日本ソフトウェア科学会，Vol. 1 No. 2 July. 1984 pp. 55－68

22. 長野宏宣，東基衛稿，『プログラム構成要素及びその表記法』，「情報処理」，情報処理学会，Vol. 28 No. 9，Sep. 1987，1166－1173

23. 東基衛，菅忠義，松原友夫稿，『ソフトウェア工学における標準化動向 今後の課題』，「情報処理」，情報処理学会，Vol. 28 No. 9，Sep. 1987，pp. 1173－1180

24. 大内和幸稿，『発表相次ぐプログラム開発・保守支援ツール』，「日経コンピュータ」，1985. 9. 16号，pp. 71－86

25. エドワード・J・ジョイス，「生産性向上の現実的手法「ソフト再利用」に注目集まる』，「日経コンピュータ」，1988. 11. 21号，pp. 107－110

26. 上村孝樹，田中淳稿，『第3回バックログ関連実態調査 急増するバックログ』，「日経コンピュータ」，1989. 1. 16号，pp. 58－93