59—R 0 0 4

ソフトウェア開発・運用の高度化・効率化方法に関する調査研究報告書(II)—開発—

昭 和 60 年 3 月

JIPDEC

財団法人 日本情報処理開発協会

この報告書は，日本自転車振興会から競輪収入の一部である機械工業振興資金の補助を受けて昭和５９年度に実施した「ソフトウェア開発・運用の高度化・効率化方法に関する調査研究」の成果をとりまとめたものです。

序

近年における情報処理および通信技術の急速な進歩に伴い，現在では社会のさまざまな分野，局面において情報化の進展がみられる。しかし，この中核をなしているコンピュータ・システムそれ自身の問題を考えてみるとき，コンピュータのハードウェアについては，最近の著しい技術革新と生産性の向上により，コストパフォーマンスの向上がみられるものの，ソフトウェアについては，今なお旧態然とした方法で開発されていること，また，システムの大規模化・複雑化・高度化，さらにはニーズの多様化などによって，ソフトウェアコストのシステム全体に占める割合が一段と高まってきている。

このため，信頼性が高く，保守のしやすいソフトウェアをいかに効率的に，コストをかけずに開発するかが，コンピュータ・ユーザの重要課題となっている。

しかしながら，企業などの情報処理部門の現状は，既存システムの適用・保守に多くの負荷がかかり，多くのシステムがバックログとして放置されることから，エンドユーザの情報処理部門に対する期待感が薄れ，パソコンを初めとするＯＡ機器の導入によってユーザ自身が問題解決を図っていくなど，情報処理部門は,内外からさまざまなインパクトが強まってきている。従って，この様な状況から，ソフトウェア開発・運用の効率化をどのようにして確立するかが急務であると考えられている。

本調査研究事業は，この様な観点からソフトウェア開発経費・工数の削減（生産性の向上），保守費用の削減，ソフトウェアの品質の向上を目的として①開発計画，②開発，③運用・保守の３つの面からアプローチを行い，それぞれの内容，適用できる手法・ツールおよび各種指標などについて調査研究を行うものである。

今年度は，昨年度実施したソフトウェアの開発計画段階の調査研究成果を踏まえ，ソフトウェアの開発段階を中心として，コンピュータ・ユーザの開発実

態に基づいてソフトウェア開発の基本作業，高度化・効率化の方策，主な技法・ツールおよび要員管理などについて調査研究を行った。

各企業におけるコンピュータ利用は，企業の実態に即して行われるべきものであり，ましてや最近の情報機器の進展と相いまって情報処理の形態も多様化しておることから，利用形態の如何によって一概にその是非を評価することはできない。しかし，ソフトウェアの開発・運用にかかる諸問題は各企業が共通して抱えている課題の一つであろう。

本報告書が，この共通の課題を解決する上で，コンピュータ・ユーザに何らかの示唆を与えるものとなり，ひいては情報処理の発展に寄与することになれば幸いであると考える。

なお，調査研究に当ってご協力をいただいた委員ならびに関係各位に深く感謝するものである。

昭和６０年３月

目　　次

ソフトウェア開発・運用調査委員会

（五十音順敬称略）

委員長	道下忠行	東海大学工学部経営工学科教授
委員	今木寿康	富士写真フィルム㈱システム管理部 主任部員
〃	興津勝	㈶地方自治情報センター研究開発部 第一課課長
〃	菅野孝男	技術士（情報処理）
〃	妹尾稔	三井物産㈱情報システム管理部 企画グループ課長
〃	中島栄	㈳行政情報システム研究所総務部部長
〃	永渕寛幸	総務庁行政管理局副管理官
〃	広木正倫	日本電信電話公社データ通信本部 第4データ部調査役
〃	前川征弘	外務省大臣官房情報管理室
〃	山下広之	日立ソフトウェアエンジニアリング㈱ 企画室部長代理
〃	小嶋利文	㈶日本情報処理開発協会常務理事

ソフトウェア開発・運用調査専門委員会

（五十音順敬称略）

主　査	永渕寛幸	総務庁行政管理局副管理官
委　員	安倍史郎	ファコム・ハイタック㈱ファコム本部システム第1部第1課課長
〃	池田慶二	日本電気㈱情報処理官庁システム事業部第1システム部主任
〃	今木寿康	富士写真フィルム㈱システム管理部主任部員
〃	菅野孝男	技術士（情報処理）
〃	妹尾稔	三井物産㈱情報システム管理部企画グループ企画グループ課長
〃	広木正倫	日本電信電話公社データ通信本部第4データ部調査役
〃	山下広之	日立ソフトウェアエンジニアリング㈱企画室部長代理

概　　要

本調査研究報告書は，ソフトウェア開発・運用の高度化・効率化方法に関する調査研究事業の第２年度として，昨年度に実施したソフトウェア開発計画段階に引き続き，ソフトウェアの開発段階に焦点を当てて実施した結果をとりまとめたものである。

本報告書の構成は次のとおりである。

1. ソフトウェア開発の現状
2. ソフトウェア開発段階における基本作業
3. ソフトウェア開発・高度化・効率化のための方策
4. 主な開発技法と開発支援ツール
5. 要員管理のための具体的方法
6. ソフトウェア開発段階のチェックリスト

ソフトウェア開発の効率化に寄与しうるような各種の技法やツールが現在でも存在しているが，極論すれば，どんなにすばらしい技法やツールがあっても，それを自由に使いこなす人がいないことには，何の役にも立たないことになる。その意味で，ソフトウェア開発の効率化・高度化の基本は，やはり“人”の問題であるといえる。ソフトウェア開発要員の教育訓練や要員管理の重要性がいわれるゆえんもこの辺にあるものと思われる。従って，本報告書においては，要員問題については，特に配慮したところである。

本報告書の各章ごとの内容について要約すると，次のとおりである。

(1) ソフトウェア開発の現状 ― ソフトウェア開発に関し，その実態を把握するため，民間企業のコンピュータユーザ（６００社）に対し郵送調査（回答企業１６０社）した結果をもとに，ソフトウェア開発のための標準化，機械化，自動化，再利用および要員管理などについて，その現状をとりまとめたものである。

(2) ソフトウェア開発段階における基本作業 ― この章の概論として，工程

化概念の必要性を述べ，そのうえで，各工程とドキュメントの関係について述べている。次に，工程区分の現状について，コンピュータメーカが採用している開発標準の考え方について紹介している。さらに，各工程での作業項目とドキュメントについて述べている。

(3) ソフトウェア開発の高度化・効率化のための方策 ― ソフトウェアの高度化・効率化を図るための有効な方策として，まず標準化の問題をとりあげている。標準化に関しては，ソフトウェアの階層化，用語，開発作業方法，及び開発管理について標準化の内容や注意点について述べている。次に，機械化，自動化をあげ，ここでは，ドキュメント作成，プログラミング，テスト，および開発管理の各作業において，どのような機械化・自動化が考えられ，かつ実現されてきているかを説明している。また，ソフトウェアの再利用についても有効な手段としてあげており，ソフトウェアの部品化に焦点を当てて述べている。このほか，この章では，ソフトウェアのパッケージの活用，ソフトウェアの開発における分業の考え方，開発パワーの増大，第三者による品質保障などのための組織化，品質管理活動などについても説明している。

(4) 主な開発技法と開発支援ツール

開発技法として，要求定義技法としての面ももつＳＡＤＴ（Structured Analysis and Design Technique）とペトリネット・モデル，およびデータフローによる設計法であるマイヤーズの複合設計法，ヨードン，コンスタンティンの構造化設計法，またデータ構造による設計法であるジャクソン法，ワーニエ＝オア法について解説している。

また，設計記述技法としては，フローチャート，ＮＳチャート（Nassi-Shneiderman chart），ＰＡＤチャート（Problem Analysis Diagram），ＳＰＤチャート（Structured Programming Diagram），ＹＡＣⅡチャート（Yet Anather Control chart），ＨＩＰＯ（Hierarchy plus Input-Process-Output），ＨＣＰチャ

ート（Hierarchical Compact chart）などについて紹介している。開発支援ツールは，数多く開発されているが，ここでは，一つの範としてのＳＲＥＭ（Software Requirements Engineering Methodology），ＰＳＬ／ＰＳＡ（Problem Statenent Language／Problem Statement Analyser），およびＰＲＩＤＥのＡＳＤＭ，そしてＨＹＰＥＲ ＣＯＢＯＬ，ＪＡＳＰＯＬについて紹介している。テスト支援ツールも数多くあるが，ここでは，種類別の説明を加えるに止めている。

(5) 要員管理のための具体的方法 ― 要員管理のための具体的方法を説明するに当たって，まず，コンピュータ要員の情報処理教育がどのような状況にあるかを，アンケート調査をもとにその実態を論じたあと，最近増大しつつある情報処理分野における高齢化の問題をとりあげている。そして最後に，コンピュータ部門のモラールやモチベーションを向上させるための方策について論じている。

(6) ソフトウェア開発段階のチェックリスト ― ソフトウェアの開発段階において留意すべき事項をチェックリストとしてとりまとめている。

1. ソフトウェア開発の現状

1. ソフトウェア開発の現状

近年におけるソフトウェア開発は下記の点が特徴として上げられる。

① 開発対象分野が多部門にわたっている。

② 開発の対象が社会システム，特殊大型システムなどの分野まで拡大し大規模化している。

③ 多くの分野に，より高度のシステムが要求されるようになり，ソフトウェアも高度化・複雑化している。

④ ハードウェア技術の進歩に応じて，各種ソフトウェアの改良が必要となっている。

このように，システムが大型化，高度化し，社会生活の中で大きな位置を占めるようになり，そのおよぼす影響も計り知れないものになるにつれて，ソフトウェア開発の効率化，高度化，生産性の向上が大きな課題としてクローズアップされている。

ソフトウェア開発の効率化，生産性の向上に対する取り組みは，官民の別なく積極的に行われているが，ここでは，民間部門がこの問題にどのように取り組んでいるかについて，アンケート調査結果に基づき概観してみたい。

1.1 アンケート調査

(1) 目　　的

ソフトウェア開発の効率化，生産性の向上を計るための基礎資料を収集することにある。

(2) 調査対象

大手コンピュータ・ユーザを中心に，会社四季報から無作為抽出法により６００社を抽出した。

(3) 調査方法

郵送による自計方式

(4) 調査時期

昭和５９年９月

(5) 回収状況

１６０社（回答率２６％）

1.2 標準化

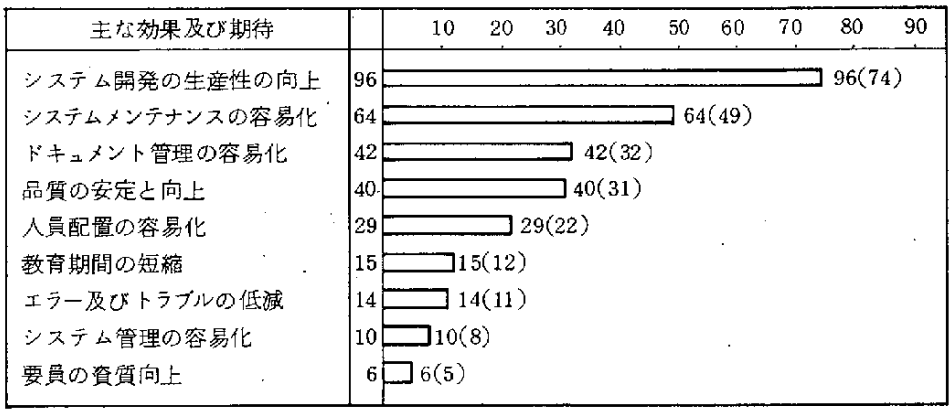
標準化は，一般に生産活動や事務作業における生産性の向上や互換機能を高めるために用いられる能率原理の一つであるが，ソフトウェア開発の場合においても，開発段階における種々の局面において標準化を行うことによって，ソフトウェア開発の効率化が期待できるものと考えられる。

今回の調査においては，標準化に関しては，①開発工程，②用語，③フォームシート，および④コーディングについて質問しているが，開発工程においては，プログラミング段階（７５％），プログラム設計段階（７３％），ドキュメントの作成段階（６８％）において標準化は高い割合を占めているが，基本設計やテスト段階では，それほど標準化は行われていない。

また，用語の標準化については，自社内で独自に決めたり（５９％），メーカなどの用語に統一したり（３８％）しており，ＪＩＳに合せたりしている企業はきわめて少ない結果となっている。フォームシートについては，独自に決めているものが殆んどであり，メーカツールなどの利用は少ない。さらに，コーディングについては，名前の付け方や，プログラムの構造，コーディングフォームについての標準化が目立っている。

そして，このような標準化の推進によって，得た効果については，図表１－１のとおり，システム開発の生産性の向上をあげるものが，７４％と最も多く，次いで，「システムメンテナンスの省力化」４９パーセント，「ドキュメント管理の省力化」３２パーセントの順となっている。また，このような標準化に関しては，ガイドブックまたは手引書として用意されているところが多い（８０％）。しかも，その内容の遵守の方法は，強制されることもなく（５１％），標準化の適用は各自にまかせている（２８％）など比較的

図表1－1　標準化の効果



N＝130

にゆるやかであり，標準化されていないものは認めないというのは２割弱（１９％）にすぎない。

しかし，標準化のための推進組織については「ある」とするものが４２パーセント，「ない」が５８パーセントとなっており，推進組織の整備の点では今一歩といった状況にあるといえる。

1.3　機械化・自動化

人間が行いうる能力には限りがあるので，人手でやっているものを機械に行わせることにより，人間が行う以上によりよい目的を達することが可能となる。ソフトウェアの開発は，従来から，いわば"職人芸"として，手作業による個人プレー的な考え方に基づく開発が支配的であったが，情報化の進展に伴って，ソフトウェアのニーズの増大に対処するため，ソフトウェア開発の効率化対策として，開発工程への機械化，自動化の導入が真剣に検討されるようになってきている。

ソフトウェア開発工程における機械化，自動化の実施状況，検討状況は図表１－２のとおりである。

これによると，プログラミング段階での機械化，自動化が約４０パーセントと割合は少ないが実施されている。ただ，この調査では，「少しでも機械化・自動化を実施している」という質問の仕方をしており，また，機械化なのか自動化なのかも区別していないが，自動化の割合いは現段階では少ないものと考えられる。

開発工程における管理面においては，「実績管理」が３４パーセントの企業において実施されている。概して，機械化，自動化については，「実施中」が少なく現在「検討中」であるとするものが２０～３０パーセントあり，これから実施しようとしている段階にあるのではないかと思われる。

図表１－２　開発工程における機械化・自動化の実施・検討状況

	実施中		検討中		未検討	
・設計段階	6	4％	38	24％	113	72％
・プログラミング	61	39	43	27	53	34
・ドキュメンテーション	20	13	60	38	77	49
・テスト	33	21	44	28	80	51
・予算管理	24	15	32	20	101	64
・実績管理	53	34	35	22	69	44
・進捗管理	31	20	40	25	86	55
・品質管理	11	7	27	17	118	75

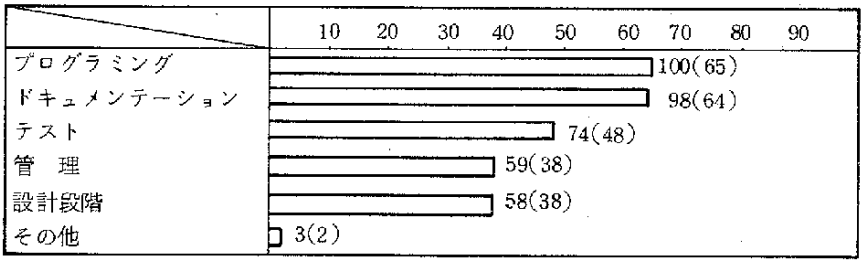
N＝157

ところで，この図表では，設計段階と予算管理，品質管理については，「未検討」とする割合が，それぞれ７２，６４，７５パーセントと高いが，これらの段階は自動化の余地がないということを意味しているのであろうか。しかし，一方で，「実施中」とするものが設計段階４パーセント，予算管理15パーセント，品質管理７パーセントと割合は低いが存在しているということ

は，これら実施中の企業がどのような自動化，機械化を行っているのか，その内容を知ることは非常に参考になるものと思われる。

次に，開発工程で，現在，どの工程を機械化，自動化したいと考えているかを図表１－３に示す。

図表１－３　機械化・自動化したい工程



N＝157

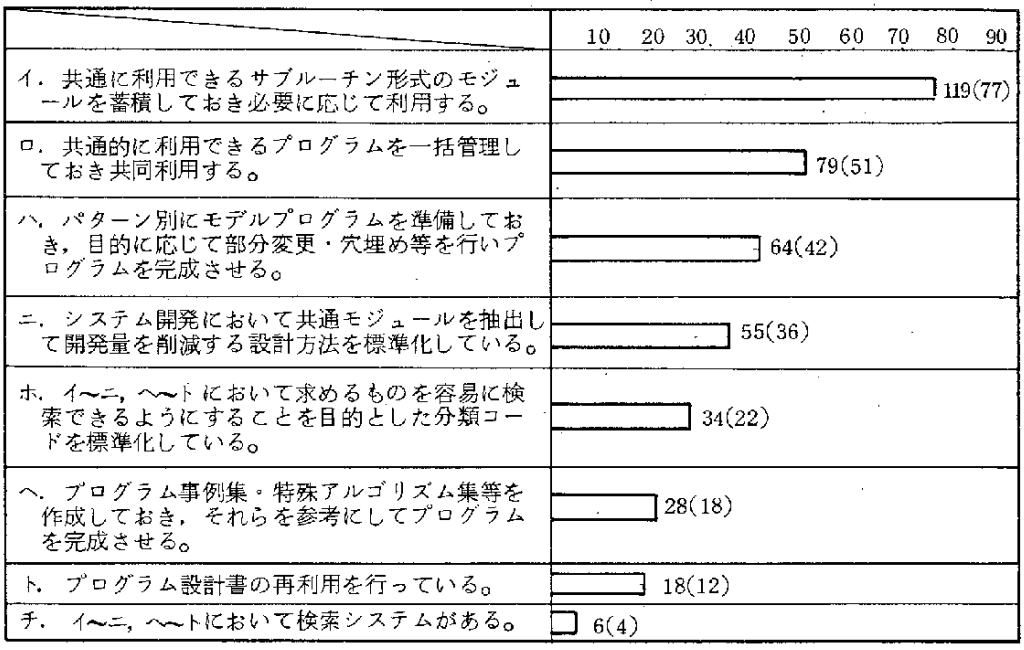
これによると，プログラミング，ドキュメントの段階の機械化，自動化を考えている企業が多い。このことは，プログラミング，ドキュメントの作成に最も工数がかかることから，何とかならないかという効率化，生産性の向上に対するニーズが大きいことがうかがえる。

1.4　再　利　用

ソフトウェア開発の生産性を向上させるためには，既存プログラムなどの有効利用（再利用）も大きな効果があると考えられる。過去において開発，蓄積された多くのプログラムなどの中には，再利用可能なものが少なくないものと思われる。

今回調査において，ソフトウェアの再利用に関して調査した結果を示すと，図表１－４のとおりである。

図表１－４　再利用のための方策



（検索を目的とした分類コードの体系）

業務別（給与，人事，財務，販売など）	27	79%
機能別（テーブル処理，コード変換，エラーチェックなど）	27	79
処理パターン別（媒体変換，照合，更新，作表など）	23	68
その他	3	9

これによると，「共通に利用できるサブルーチン形式のモジュールを蓄積しておき，必要に応じて利用しているもの」が119企業（77%）と最も多く，次が「共通的に利用できるプログラムを一括管理しておき共同利用するもの」が79企業（51%），「パターン別にモデルプログラムを準備しておき，目的に応じ部分変更，穴埋めなどを行いプログラムを完成させるとするもの」が64企業（42%）の順となっている。このように，再利用に

関しては，プログラムのモジュール化が積極的に進められていることがうかがえるが，共同利用のためにプログラムの一括管理が行われている企業がかなり見受けられることは，ソフトウェアの開発が同一企業内であっても異なったセクションで別個に行なわれているケースが多いことから，このような一括管理方式（プログラム登録方式）により重複開発（投資）の回避という点からも生産性ということと併せて効果があることを示しているものといえる。

1.5 組 織 化

ソフトウェア開発体制という点から，組織の面を中心にみてみたい。ソフトウェア開発を限られた人数（定員）でより多くの開発要求に対応していくためには，当該組織内における開発要員の能力（経験）を効率よく配分，活用して最大の効果を確保できるようなマネジメントが必要である。

今回調査においては，組織の問題については，分業体制とその問題点，品質検査組織，要員不足をカバーするための対策などについて質問を行った。

その結果，ソフトウェア開発において，どのような分業体制をとっているかについては，回答企業（１４７）の８０パーセント（１１７）の企業が，「２～３グループに分けている」としている。そして，グループの分け方は，（設計），（プログラミング，テスト）又は（設計，テスト），（プログラミング）といった分業体制をとっているものが多い。（図表１－５参照）

図表１－５　ソフトウェア開発における分業体制

(1)

グループ数	件　数	%
1	18	12
2	51	35
3	66	45
4	11	7
5以上	1	1

N＝147

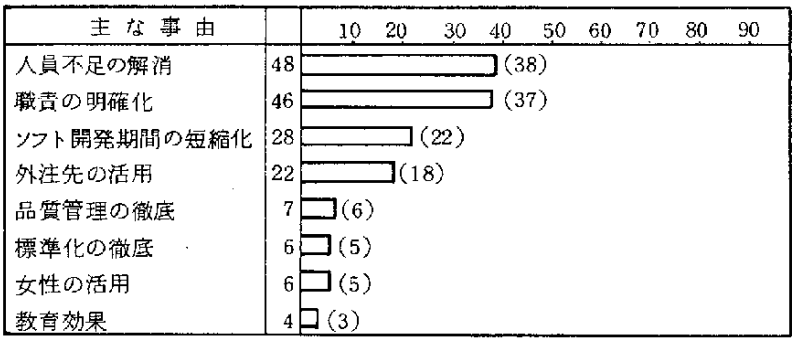
(2)

作業の分担	件　数	%
(設計)・(プログラム)・(テスト)	12	8
(設計・プログラム)・(テスト)	0	0
(設計)・(プログラム・テスト)	26	18
(設計・テスト)・(プログラム)	69	47
その他	20	14
分業しない	20	14

N＝147

このような分業体制をとる理由としては，図表１－６のとおり，①人員不足の解消（３８％），②職責の明確化（３７％），③開発期間の短縮（２２％）などをあげている。

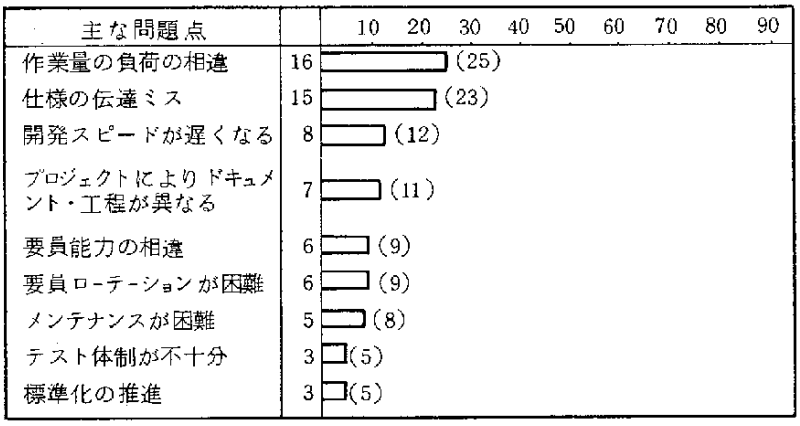
図表①－６　分業体制をとる理由



N＝１２５

分業体制をとった場合の問題点としては，図表１－７のとおり，①作業量の負荷の相違（２５%），仕様の伝達ミス（２３%），③開発スピードが遅くなる（１２%）などをあげているが，ソフトウェア開発期間短縮のために企業体制をとったのに逆に開発スピードが遅くなるという皮肉な結果を招いている例があることを示している。

図表１－７　分業体制の問題点



N＝６５

次に，品質保証や検査のための組織については，９４パーセントの企業が「ない」としており，この面での整備が非常に遅れていることがわかる。残りの６パーセントの企業（１０企業）について，取り組んでいる業務内容をみてみると，検収業務，システム監査，品質向上，運用効率，標準化などとなっている。こういった地道な業務に係る体制の充実がソフトウェアの質的な向上をもたらし，結果的には全体の生産性の向上に結びつくものであることを認識する必要があるのではないかと思う。量への対応に神経を集中するあまり，質の面からのアプローチをおろそかにしてはならないだろう。

さて，ソフトウェア開発の要員不足のためにどのような対策をとっているかをみてみよう。開発要員不足のための対策としては，外注利用，エンドユーザの活用等が考えられるが，今回調査では，外注利用については約３０パーセントの企業が「外注利用していない」としている。また，約５０パーセントの企業が総作業工数のうちの３０パーセント未満を外注（派遣外注，作業請負外注）に依存しているとしている。（図表１－８参照）

図表１－８　外注利用の状況

	利用してない	～10％	～20％	～30％	～40％	～50％	～100％
派遣外注	38件	23	24	19	7	7	7
作業請負外注	40件	38	9	9	7	4	11

ところで最近いわれている在宅勤務については，制度として確立している企業は３パーセント（４企業）にすぎず，ほとんどの企業は，現在在宅制度は存在していない。

この制度を将来採用するかについては，９パーセントの企業が「計画している」としているのみで，将来ともに考えていない企業がほとんどである。在宅勤務において考えられている業務内容としては，主に退職社員や契約社

員によって，コーディングやプログラム設計をやってもらう。その場合，ＴＳＳ端末を利用したり，パソコンにより業務処理をしてもらうことが考えられている。

プログラム開発は，その結果がエンドユーザのニーズに十分応え得るものでなければならないが，その意味で，エンドユーザ自身にプログラム開発をしてもらうこと，または開発の支援をしてもらうことも，要員不足をカバーする有効な手段であると考えられる。今回調査においても，５７パーセントの企業（９０企業）が，エンドユーザによるプログラム開発が社内における推進テーマとなっているとしており，エンドユーザの活用が，今後の要員を確保するための源泉となりうるものと期待される。

なお，女子プログラマは，情報処理部門やソフトウェア・ハウスなどでは，ソフトウェア開発の新たな戦力として注目されているが，今回調査においては，現在，ソフトウェア開発に携わっている女性は，コンピュータ部門の30％未満とする企業が７０パーセント（１０８企業）となっており，今後，女性の採用を推進していく考えのある企業も６６パーセント（１０３企業）となっている（図表１－９）。

図表１－９　コンピュータ部門における女性プログラマの状況

区分	0	～10	～20	～30	～40	～50	～60	～70	～80	～90	～100
件数	28	48	37	23	7	5	2	3	0	1	1

N＝155

1.6　品質管理

近年，企業内におけるＱＣサークル活動などの小集団活動が活発になりつつあるが，今回調査においても７８パーセント（１２４企業）の企業が全社的な品質管理活動を行っているとしている。このうち，コンピュータ室がＱＣ活動を行っている割合は，５０パーセント（７８企業）と 全社的な割合（７８％）よりも低く，コンピュータ部門のこの面での取り組みは，他部門

に比べて低調のように思われる（図表１－10，１－11）。

図表１－10　企業における品質管理活動の状況

はい	124件	78%	
いいえ	36	22	N＝160

図表１－11　コンピュータ室における品質管理活動の状況

はい	78件	50%	
いいえ	79	50	N＝159

コンピュータ部門の品質管理活動のためのサークル数は，「１～３」とする企業が合せて45パーセントとなっている。サークル数が「10ある」とするものが１０パーセント，「１６以上ある」とするものが１８パーセントもみられる。

１サークル当たりの構成員は５～７人とするものが全体の７割程度を占めており，活動年数については１年とするものが，４０パーセント，２年が約２０パーセント，３年が１５パーセントとなっており，１～３年の活動年数の合計で７５パーセントを占めている。また，取り組んでいるテーマ数は，「１～３件」とする企業で約８０パーセントを占めている。

このことから，コンピュータ部門における品質管理活動は，概ね１～３程度のサークルを，５～７人程度の規模でもち，サークル当たりのテーマも１～３件程度，それに活動期間を１～３年を目途としている企業が比較的に多いといえる（図表１－12，13，14，15）。

図表１－12　品質管理活動のサークル数

サークル数	件　数	%
1	16	12
2	9	13
3	7	10
4	4	6
5	5	7
6	3	4
7	2	3
8	2	3
9	0	0
10	7	10
11	1	1
12	1	1
13	0	0
14	1	1
15	1	1
16以上	13	18

N＝72

図表１－13　サークルの規模

構成員数	件　数	%
3名	4	6
4	3	4
5	14	19
6	25	35
7	12	17
8	6	8
9	1	1
10	6	8
12	1	1

N＝72

図表１－14　サークルの活動期間

活動年数	件　数	%
6カ月	12	17
1年	29	40
2年	14	19
3年	11	15
4年以上	6	8

N＝72

図表１－15　サークル当たりのテーマ数

テーマ数／サークル	件　数	%
1	25	35
2	24	33
3	11	15
4	4	6
5	4	6
5以上	4	6

N＝72

また，取り組んでいるテーマの内容をみてみると，図表１－１６のとおり，プログラム構造の標準化の推進，生産性の向上，トラブルの防止およびファイル管理などとなっている。

図表１－16　品質管理活動の主なテーマ

主なテーマ	件　数	%
プログラム構造の標準化推進	39	40
ソフトウェア生産性向上	29	30
トラブルの防止	29	30
ファイル管理	24	24
メンテナンス工程の標準化	22	22
品質管理及び向上	16	16
仕様書の標準化	16	16
品質管理・品質向上	15	15
ＯＡ化推進	14	14
検収・テスト	11	11
要員教育	8	8

N＝98

1.7　ソフトウェアパッケージの活用

ソフトウェア開発の効率化，生産性の向上を図るための効果的な方法の一つは，ソフトウェアパッケージを活用することである。ソフトウェアパッケージは，①自社開発よりもコストが安い，②開発期間がとれない，③自社開発の工数が不足する，④自社開発より機能・性能などがよい．といったような場合に導入が考えられる。

今回調査においては，回答した１５７企業のうち１０３企業（６６%）が市販パッケージを購入したことがあるとしている。購入したことのある企業

について購入動機理由を調べてみると，図表１－１７のとおり，「自社開発よりも機能・性能などがよいため」とするものが４０パーセントと最も多く，次いで，「自社開発よりもコストが安い」３２パーセント，「自社開発の工数不足」１５パーセントとなっている。

図表1－17　ソフトウェアパッケージの購入動機理由

購　入　動　機	件数	割合
・自社開発より機能・性能等がよい	122 件	40 %
・自社開発よりコストが安い	99	32
・自社開発の工数不足	46	15
・開発期間がない	40	13
計	307	100

（注）市販パッケージを購入したことのある１０３企業の複数回答の合計と比率である（図表１－18も同じ）

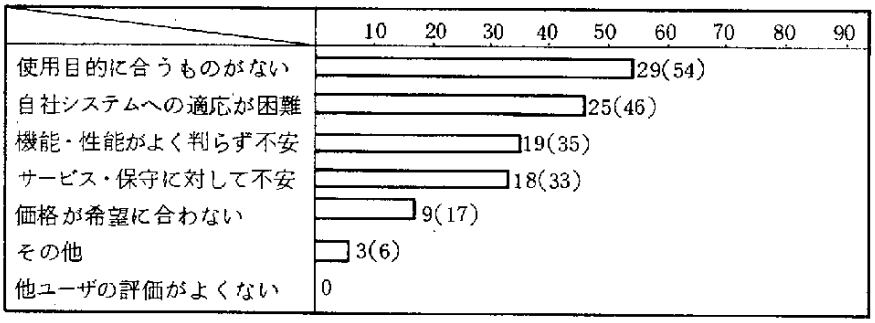
ソフトウェアパッケージを購入した場合の評価は，図表１－１８のとおり，「良い」とするもの４１パーセント，「大変良い」３０パーセント，「一応評価できる」２５パーセントの順となっている。

図表1－18　ソフトウェアパッケージの評価

・大変良い	86 件	30 %
・良　い	116	41
・一応評価できる	70	25
・期待はずれ	10	4
計	282	100

次に，ソフトウェアパッケージを購入したことのない企業に対して，購入に消極的だった理由を調査した結果は，図表１－１９のとおりである。

図表1 －19　購入に消極的な理由



N＝５４

これによると，ソフトウェアパッケージが，「使用目的に合うものがない」とするもの（５４％），「自社システムへの適応が困難」とするもの（４６％）が多い。しかし，「機能・性能がよく判らず不安」（１９％），「サービス・保守に対して不安」（１８％）とするものの割合が比較的少ないということは，必ずしも自社で使用するソフトウェアは，自社の手作りかオーダメイドでないと安心できないといった従来型の気質が根強く残っているということではないように思われる。

なお，今後のソフトウェアパッケージを利用する場合の考え方としては，使用目的に合うものや自社開発よりも有利なものがあれば積極的に購入したいとする企業が多い。購入に当たっての評価のポイントとしては，第一に機能・性能を，次に工期やアフターケアをあげている（図表１－２０）。

図表1－20　今後のパッケージ購入の方針

	件　数	%
多少の不便はあってもできるだけ市販パッケージを活用し自社開発量を減らしたい	15	7
使用目的に合う良いものがあれば積極的に購入したい	78	38
自社開発よりも有利なものがあれば購入したい。	95	46
自社開発を優先して考えたい	16	8
市販パッケージの購入はしない	2	1
その他	0	0
計	206	100

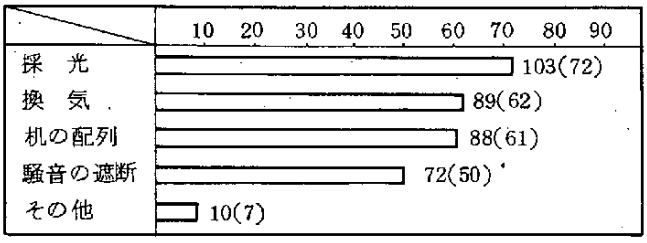
（注）回答企業の複数回答の比率である。　　N＝160

1.8　作業環境

作業環境という場合には，一般的には，物理的環境と社会的環境の二つに分けられている。物理的環境とは，職場において心身機能と関連し作業能率に影響を及ぼす物理的環境条件といい，一般的なものとして採光，換気，騒音，温度，湿度，色彩などがあげられる。また，社会的環境とは，生産活動が組織体または職場というような社会的環境内で行われていることから，生産性を向上させる観点からは，物理的環境の調整よりは，むしろ社会的・人間的諸関係の調整が重要な条件であるとし，職場における人間関係とモラールを重視することである。今回調査においては．物理的な環境について，どのような考慮をしているかを調査した。

調査結果によると，回答企業（１４４社）では，採光に最も気を配っており（７２％），次いで換気（６２％），机のレイアウト，騒音の順になっている（図表1－２１）。

図表①－21　作業環境の配慮状況



N＝144

また，実際に実施している作業環境の工夫・改善事例としては，①ディスプレイ面の反射防止対策，②騒音対策，③机及び椅子などをあげている企業が多い。

1.9　要員管理

ソフトウェア・クライシスの一つの要因として，要員確保の困難があげられているが，そのために，現に在職するコンピュータ部門の要員の資質の向上を図ることが必要であり，そのための要員管理体制の整備が重要な課題である。

今回のアンケート調査では，①要員の年齢構成，定年関係，②要員の教育訓練の体系と内容，③要員のモラール向上策について調査した。

1.9.1　要員の年齢構成

調査対象企業におけるコンピュータ要員の年齢構成は，図表1－22のとおりである。

これによると，職種により明確に年齢の分布状況に特徴がみられる。例えば，プログラマについては，一般に35歳定年がいわれているが，本調査結果においても，「35歳未満」で約90%を占めており，「35歳以上」のプログラマは10%程度となっている。

図表１－22　要員の年齢構成

（単位　%）

	29歳以下	30～34歳	35～39歳	40歳～定年	定年退職者の再雇用
プロジェクト管理者	0.2	4.3	21.3	73.7	0.5
プロジェクトリーダー	1.7	23.4	47.1	27.2	0.6
システム設計者	22.6	41.5	27.8	7.6	0.5
プログラミング担当者	73.1	16.6	6.4	3.4	0.5
教育担当	30.2	31.8	19.8	18.2	－
オペレータ	64.4	12.0	7.5	15.7	0.4

N＝154

また，回答企業における定年年齢は，「６０歳」とするものが６５パーセント，「その他」が３５パーセントとなっている。なお，プログラマについて年齢制限を設けている企業はない。

1.9.2　要員の教育訓練の体系

要員の教育訓練については，①一般的な情報処理技術，②ソフトウェアに関する専門技術，③情報処理技術者試験など対応，のための教育などに関して調査したが，その結果は次のとおりである。

①　一般的な情報処理技術の教育については，コンピュータの入門教育が最も多く（９７%），次いでプログラミング技術（９０%），システム開発手順全般にわたる標準的な技術（４９%）などとなっている。教育期間は，入門教育が平均２～３週間，プログラミング技術が平均５週間程度となっている（図表１－２３）。

図表1－23　一般的な情報処理技術教育の状況

	件数	%	平均（週）	1週未満	1週間	2週間	3週間	4週間	～以上
コンピュータの入門教育	140	97	2.2	15	63	24	10	18	10
要求定義技術	28	19	2.6	4	17	5	1	0	1
システム設計技術	66	46	3.8	4	32	12	4	6	8
プログラミング技術	129	90	5.0	3	21	21	28	22	34
システム開発手順全般にわたる標準的な技術	70	49	2.2	7	37	13	4	5	4

N＝144

② ソフトウェアに関する技術の教育については，ＯＳなどの基本ソフトウェア技術（８１％），オンライン・データ通信技術（７３％），データベース技術（６８％）となっており，これらの教育に平均２～３週間をかけている（図表1－２４）。

図表１－24　ソフトウェア技術教育の状況

	件数	%	平均（週）	一週未満	1週間	2週間	3週間	4週間	～以上
ＯＳ等の基本ソフトウェア技術	86	81	2.3	14	47	12	1	5	7
オンライン・データ通信技術	77	73	1.7	10	41	17	2	4	3
データベース技術	72	68	2.0	12	33	16	4	4	3
マネージメントサイエンス技術	10	9	0.9	3	6	1	0	0	0
ＣＡＤ／ＣＡＭ技術	5	5	0.7	3	2	0	0	0	0

N＝106

③ 教育訓練の具体的な方法としては．ＯＳなどの基本ソフトウェア技術教育や一般的な情報処理技術の教育は，メーカのユーザ教育コースや社内（教育部やベテランＳＥ）による教育などが多く，マネジメントサイエンスなどの高度な技術に関しては，メーカや各種セミナーを活用している例が多い。情報処理技術者試験対策としては，非常に少ないが社内

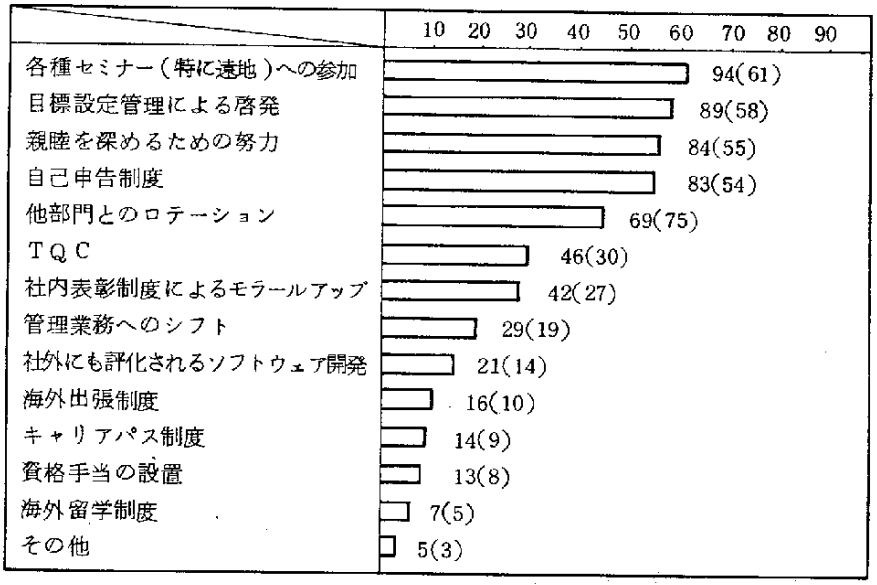
教育や通信教育などが活用されている。

また，これらの教育訓練のために投入された費用は，平均して年間１人当たり約１０万円，作業時間全体の平均６～７％をかけているとしている。

1.9.3　モラール向上対策

要員のモラール，モチベーションをあげるための対策としては，図表１－２５のとおり，「各種セミナーへの参加」，「目標設定管理による啓発」，「親睦を深めるための努力」，「自己申告制度」などが比較的多くとられている。数は少ないが，「海外出張制度」や「海外留学制度」をとっている企業もみられる。

図表１－25　モラール向上対策



Ｎ＝１５３

1.10　技法・ツール

ソフトウェア開発支援のために用いられている技法・ツールの状況について調査した結果は，図表１－２６のとおり，１１種類の技法・ツールが用いられている。これらの適用範囲は，プログラム設計→プログラミング→モジュールテスト→結合・運用テストの各工程において使われている度合が多い。とりわけ，プログラミングの工程では多く使われている。使用した技法・ツールに対する評価は，一応評価できるとするものが多い。

図表１－26　技法・ツールの使用状況と評価

技法・ツール名	使用延件数	適用範囲										使用した感想				
		要求定義	基本設計	詳細設計	プログラム設計	プログラミング	モジュールテスト	結合・運用テスト	ドキュメンテーション	保守	その他	大変良い	一応評価できる	機能の割に高い	使いにくい	今は使用していない
15	56	2	7	8	24	29	17	10	5	6	4	15	35	3	2	1

なお，使用した技法・ツール名は，次のとおりである。

①CORAL，②TRTR－C，③NS－チャート，④JASPOL，
⑤オンラインシミュレータ，⑥EASYTRIVE，⑦DYANA，
⑧インスペクション，⑨擬似コード，⑩HELP，⑪PRO／TEST，
⑫PAD，⑬EXCEED，⑭HYPER－COBOL，
⑮構造化ウォークスルー

1.1.1　今後の課題

システム開発段階において，企業が直面している問題点としては，要員不足，システム開発の生産性の向上及び効率化をあげる企業が多く，この辺が，最近いわれているソフトウェア・クライシスの大きな要因ともなっているものと考えられる。

今後の課題としては，要員教育，システム開発期間の短縮，標準化の促進などをあげている。

2. ソフトウェア開発段階における基本作業

2. ソフトウェア開発段階における基本作業

2.1 基本作業の考え方

2.1.1 工程化概念の必要性

ソフトウェアの世界での問題点としてよく云われる事項として「ソフトウェアは見ることができない。」であるが，これは物を生産する場合，生産する過程のなかで中間物や原材料を見ることが出来る世界と，全く生産過程の中で物として識別出来ないソフトウェアの世界とを比較して，実際的に物を見ることが出来ない為に管理が出来ないといった問題点を指摘している。

しかしこの問題は対象とするソフトウェアの規模が小さく，一人の要員の頭の中に十分取り入れることが出来る程度ならば，優秀な要員を手元に抱えることによって，管理が十分行なわれなくてもそれほど大きな問題とならない。

ところが，ソフトウェアの利用状況を眺めてみると，開発されてからかなり長期間にわたって利用される場合が多い。つまり新規のソフトウェアが開発されると，必ずそれにはそのソフトウェアを活用していく為の運用といった作業が発生し，しかもそれを使い易く，最大限の効用を発揮させ，環境変化に対応させる為に，ソフトウェアの修正作業が長期間にわたって発生してくる。つまりソフトウェアにも人間の一生と同じ様に，開発の開始といった誕生からそれが利用されなくなる廃棄までの一連のライフサイクルがあり，或る時点に発生した問題が，いろいろな局面に影響をおよぼしている場合が多い。そのためにソフトウェアの問題は或る一時点だけをとらえて議論出来ない場合が多い。

また，開発初期の段階で管理が不十分なために，品質上に問題を含んだまま開発工程を進めていくと，それがどんどん後工程に進むにつれて品質不良状態を内在したまま増殖されていってしまうといった特質を持ってい

る。

その上ソフトウェアの規模が大きくなってくると一人だけの要員では対処出来なくなり，多くの要員を同時に抱えて開発する必要があり，そのためにそこでの作業を円滑にするコミュニケーションのし易い環境の構築やしっかりした管理の導入といった側面の重要性が大きく叫ばれるようになってきた。

このような状況を踏まえソフトウェアの世界の中に，ソフトウェア工学といった，工程の概念がとりいれられソフトウェアのライフサイクル論が展開されてきた。しかし現実にソフトウェア開発を進めるにあたっては，要求仕様をより詳細化していく過程で，かなり繰り返し的な作業を実施する場合が多いことや，実世界の仕組を2進法のコンピュータの世界に組み込む過程で思考的には大きなギャップがあり，工程といった概念の普遍化を困難にしていた。

しかし最近の情報化の進展によってソフトウェア開発の量や規模において拡大傾向をたどっており，その結果ソフトウェア開発に，一時期に数十人から数百人が一つのプロジェクトに参加するケースが増加している。そこでこのような大規模プロジェクトの生産性や品質を向上させるために標準化の推進や管理面の強化が必要不可欠となってきている。この管理と標準化を推進する手段としてこの工程化の概念が広く普及してきた。これによってソフトウェア生産の世界に，ソフトウェアに関する科学的な知識を基礎にして，より工業的な生産をはかっていこうとする考え方が一般的となってきた。

2.1.2 工程とドキュメントとの関連

ソフトウェア開発への工程化概念と同様に重要な役割を担っているのがドキュメンテーションである。このドキュメントはソフトウェア開発における可視性の向上と管理の可能性を導き出した点では高く評価される手段

である。このドキュメントは大きく分けると次のような役割を帯びているドキュメントとして分類することが出来る。

① ソフトウェア開発の作業上に必要とするワークドキュメント

② 工程の終了段階において作成される各種のシステム・ドキュメント

③ 進捗管理や品質管理といった管理用に必要となる管理ドキュメント

この場合特に大切なことは，昔の個人レベルで実施していたソフトウェア開発と比較すると，かなり多量のドキュメントを作成しなければならず（大規模ソフトウェアの場合，コミュニケーション上，必須の生産物であるが）かなりの負担になっているのが事実である。そのために出来るだけ作成すべきドキュメント量をミニマイズすべくライフサイクル全体を見通しながら，情報の重複作成を排除するように心がけることが大切である。

また，ドキュメントと工程とは，裏，表のような関係にあり作業の進捗状況に応じてドキュメントが自然な流れの一環として作られるような仕組を考えておく必要がある。よく頻繁に発生するケースとして，ドキュメントの作成をどちらかというと不必要な作業とみて仕方なく後付けの作業として開発が終了した段階で，一気にドキュメントを作成するといった本末転倒的な作業が行なわれることがあり，ソフトウェアの品質面や開発効率の観点から眺めると十分注意をする必要があり，管理者が積極的に機能を発揮出来る局面でもある。

一般的には詳細レベルでの作業とドキュメントとが一対一の関係での対応がとれており，作業結果として作られるワークドキュメントの集大成がシステム用のドキュメントになるように工夫されている場合が多い。また，ドキュメントの作成の手間のミニマイズ化と品質向上のために，開発工程の進捗を通してドキュメントが自動的に作られていくことが，開発要員の作業負荷を軽減する意味においても望ましい。

最近では，ツールとしてソフトウェア開発に必要となるドキュメントが自動生成されるようなパッケージもかなり市販されてきているが，一貫性

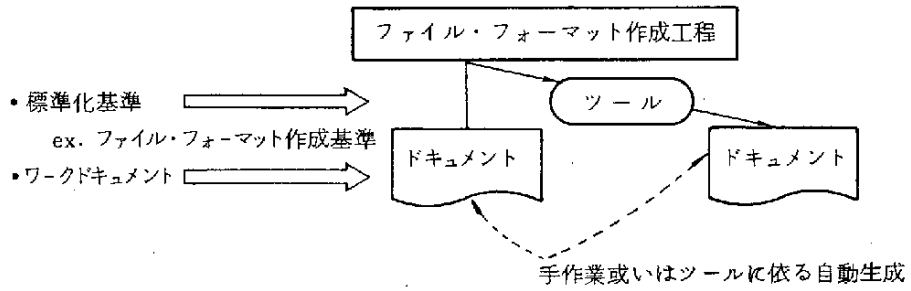
や日本語の問題を十分解決したツールは少ない。

作業工程とドキュメントが作成される関連を図表２－１に概念的に取りまとめた。例えば詳細設計段階での作業

・作業内容；ファイル・フォーマットを作成する作業

・作業工程；ファイル・フォーマット作成工程

図表２－１　工程とドキュメントとの対応概念図



2.1.3　ウォータフォール・モデルでの工程の考え方

ソフトウェア生産の世界に工学的な考え方が取り入れられていなかった時期では，メモリ効率や実行速度の向上に力点が置かれたプログラムが組まれている場合が多かった。しかし最近では2.1.1でも触れたようにソフトウェアの生産の場に工学的な考え方を取り入れて生産効率の向上やソフトウェア品質の向上をはかっていくソフトウェア工学の考え方がかなり普及してきた。

このウォータフォール・モデルでの考え方は，ソフトウェア生産の連続性の中にも，或る程度の仕事の切れ目があるとし，その切れ目の終了段階で，その工程での作業内容をチェックしたり，評価する方法である。そこでこのソフトウェア開発を実行する側であるソフトウェア生産者の立場に

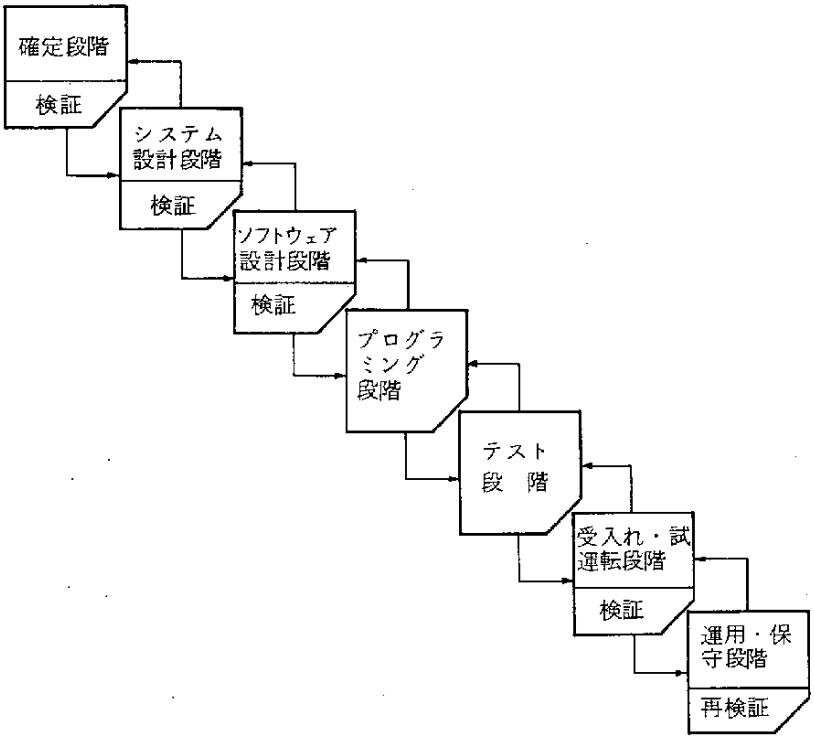
たって，開発工程を眺めてみると生産工程（フェイズ）が７工程に分割され，しかもそれぞれの工程での作業結果が次の工程へ，次の工程へと次々に落下する滝のようにパスされていく状態に似ているのでウォータフォール・モデルと俗称されている。

また，このモデルでの工程といった概念の導入が工程間で受渡すべき情報をドキュメント化することによって，スケジューリング，マンパワー計画，技能の差別化，機能分担による外注化といった工業生産的な手段を採用することが出来るようになった。

このモデルを提唱したのがＴＲＷ社のＢｏｌｈｍ である。彼はウォータフォール・モデルを次のように定義している。

つまりソフトウェアのライフサイクルをシステム要求に始まり，運用・保守といった工程で終わる一連のプロセスと定義しており，しかもそれぞれの工程にはテストや検証といった作業を伴ない，其の上このテストや検証作業の結果の如何によっては前の工程に戻る作業が発生するものとしている。この考え方のモデルを図的に表現すると次の図表２－２のようになる。

図表２－１　ウォータフォール・モデル図



次にこのモデルでの工程において如何なる作業が行なわれるかをソフトウェアのライフサイクルに沿って概説する。

①　システムの要求仕様確定段階

エンド・ユーザが意図しているシステム像を調査し確定する段階である。システム規模や対象とする領域が拡大してくるとシステムの要求仕様を確定する作業が難しくなってくる場合が多い。しかもこの工程での詰めが十分行なわれていないと，手戻り作業が頻繁に発生し，品質の悪化や生産効率を大巾に悪化させる原因になるので，非常に重要な工程である。

②　システム設計段階

エンド・ユーザが要求している内容を満足させる為に，構築すべきシステムが如何なる機能や性能を具備すべきかをシステム仕様として取り

まとめる段階である。この工程は，この他に機能設計段階とか論理設計段階とも呼ばれている。

③　ソフトウェア設計段階

前工程で作成されたシステム仕様に沿ってよりコンピュータ領域に近い形態に落とす段階である。アプリケーション・システムを開発する場合，実世界での仕組をコンピュータの世界に組み直す工程であるために，思考上での大きなギャップが発生する工程でもある。そのためにも，この工程を担当する要員には両方の世界を十分知っていることが望まれる。また，この工程は，この他にプログラム設計段階，物理設計段階，詳細設計段階とも呼ばれている。

④　プログラミング段階

前工程で作成されたプログラム仕様書に沿って，プログラム構造などを考えて，ターゲットとするコンピュータで稼動出来る言語でコーディングする段階である。この工程での作業には，上記のコーディング作業の他に，最終的に個々のプログラムが，記述された仕様書通りに正しく機能するか否かを確認する迄の一連の作業を含んでいる。

⑤　テスト段階

前工程で作成されたプログラムの総合的な検査を行う段階である。また，この工程では，システムで必要とする各種の機器とを結合させ，ソフトウェア全体にわたるシステムの検査を行う作業をも含んでいる。

⑥　受入れと試運転段階

出来上ったシステムを実際の現場に導入し，システムの利用者を含めてシステムが正常に稼動するか否かを確認する段階である。

⑦　運用・保守段階

システムのライフサイクルの最終工程に位置し，構築されたシステムを正常に運用していく工程である。また，開発段階で作り込まれたシステムの欠陥を修正したり，環境条件の変化に対応するためのシステム修

正を行なう工程である。システムのライフサイクルをより長くするために，この工程での作業は注意深く実施する必要があり，この工程での作業結果が，システム利用者に対して直接的に影響を与えるので非常に大切な工程である。

2.1.4 データベースアプローチを中心とした場合の工程の考え方

情報資源は，人，金，物に次ぐ第四の経営資源として企業のなかでますますその重要性が増大してきた。この情報資源を支えるのが企業内に構築されてきているデータベースであり，これの活用如何が今後の企業発展の重要な鍵を握っているものと思われる。そこで多くの企業では企業内に統一的なデータベースを構築し，活用していく方向にある。そこで或る程度データベースが整備された段階での開発工程がどうなっていくのかを考察してみる。この場合，従来型の開発方法とデータベース・アプローチによる開発方法との間には，工程という皮相的な面から眺めるとそれほど大きな開きはないと思われる。しかし工程の詳細レベルで眺めてみると大きな違いが存在している。特に自己完結型のシステム開発方式を採用している場合には，新たにデータベース・アプローチによる開発方法を採用すると非常に大きな影響を受けると思われる。つまり自己完結型のシステム開発を行う場合には，情報処理部門内にそのシステムの開発責任者が任命され，システム全般に対する責任を負うことになる。また，そのシステムで取り扱うデータ自身に関しては，その開発プロジェクト内の問題として，そのシステムのなかに取り込まれ，部分的な事としてデータに関する定義や構造化がなされていく。

一方，データベース・アプローチによる開発の場合には，開発組織，運用組織を含めて企業内に，新らしくデータを全社の共有資源として扱うべきデータ管理を専門に行う体制を確立する必要がある。

その上，全社レベルでのデータ定義，収集蓄積，それに附随する保守作

業や，その配布 といった諸々の作業に関する標準や手順などを確立する必要がある。

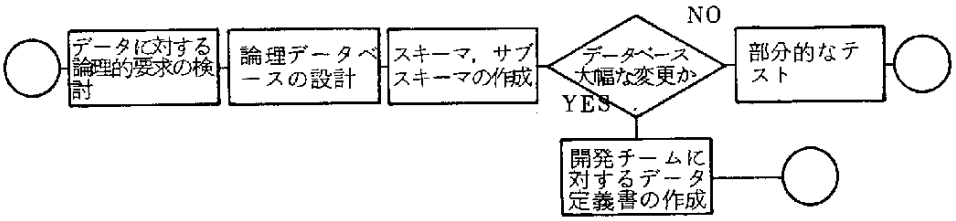
この様にデータベース・アプローチによる開発を推進していく場合には，開発工程内での詳細レベルでの作業内容が異なっており，しかも開発の実施に先だってデータベース管理者と一般的にいわれているデータ管理者の育成と体制の確立が重要な案件となっている。その意味において，開発方法や組織体制に大きな変更を要請されることになる。

また，データベース・アプローチを採用する場合には，データベースを専門に扱うデータベース設計チームを別組織として確立する必要があり，ここでの作業は開発を推進しているプロジェクト・チームからエンド・ユーザの要求仕様に沿ってデータ要求文書を提出させ，それを基礎にしてデータベース設計チームは，データベース・スキーマの変更の実施やデータ定義文書を作成し，開発チームにパスする作業を並行的に行なっている。

つまりこの事は，従来の開発方法に比べると，データベース・アプローチによる方法は開発全般にわたって，多くの作業が開発チームとデータベース設計チームとの広い範囲にわたる協力関係によって作り上げられていくものである。この点から考えるとシステム開発での分業化を促進すると同時にデータに関するしっかりとした管理体制を確立していないと効率的な開発が出来ないともいえる。

以上のような点を総合すると，従来での開発方法とは詳細レベルでの工程では，大巾に異なっており，しかもそこにはまだ確立された開発方法論が存在しているとはいい難い状況であると思われる。そこでデータベース・アプローチによる開発工程の中で，特に設計段階での概略の工程がどうなっているか眺めてみると，従来型の開発工程と並行して次の図表２－３のような作業工程が発生している。ただしこの工程は必ずしも確立したものでなく，あくまでもサンプル的な工程として例示してみる。

図表２－３　設計段階での概略の詳細工程図サンプル



2.1.5　プロトタイピング手法による工程の考え方

プロトタイピング手法の発想が生れてきた背景としては次のようなことが考えられる。

① 大規模のソフトウェアになると要求仕様を早い段階で十分確定したいと思っても，要求そのものが茫然としている場合が多く，なかなか確定するのが困難である。

② エンド・ユーザと要求仕様を十分煮つめても，開発の終了段階になって，コンピュータからアウトプットされてきたテスト用のアウトプット（例えばエラー・チェック・リスト或いは各種の報告資料など）を眺めて，エンド・ユーザが意図していた内容と異なっていることを発見する場合が多い。

③ エンド・ユーザの心理状態として開発が終了する迄，自分が要望した通りの出力が正しく得られるのか否かを不安な状態で待たなければならない。

④ エンド・ユーザが要求している内容が正しく，開発者側に伝達されているかが定かでない。

このために，出来るだけ早い時期に，エンド・ユーザ自身が意図している概要を実際的な形で示すことによって相互の認識上の不一致をなくすための手段として，このプロトタイピング手法の考え方が生れてきた。この

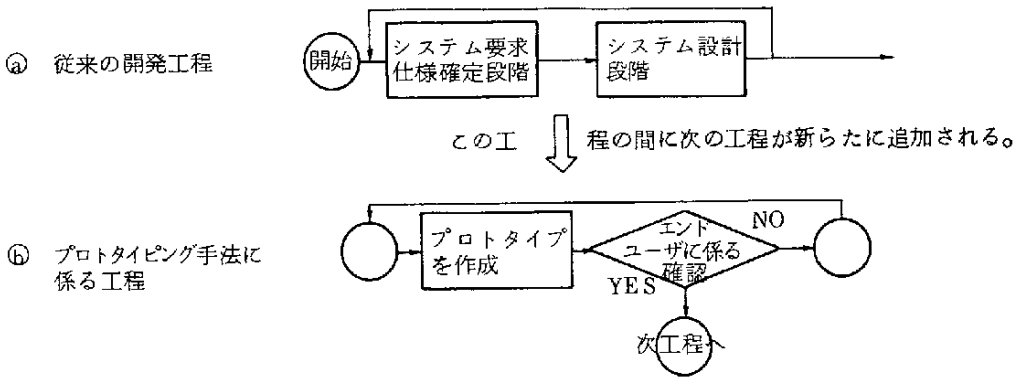
プロトタイピング手法には次の２通りの考え方がとられている。

① プロトタイピングの定義どおりに，システムの試作品を短期間に作成し，出力条件や内容などについてエンド・ユーザに確認してもらう方法で，基本的には作り出されたソフトウェアは使い捨てにされる場合が多い。

② 基本となる部分を早い時期に作り上げ，その処理内容や形式などについてエンド・ユーザに確認してもらい，さらに，それをベースによりきめの細かい部分に対しては，機能拡張としてシステムの内容を拡大していく方法である。基本的には積み上げ方式となっており，出来るだけ，プロトタイピングで開発したソフトウェアも活用していく方法である。

これらの方法は，工程的には従来の開発方式とは，作業内容の点において大きな相違がみられない。ただし，このプロトタイピング方式を採用する場合には，エンド・ユーザからの要求仕様を簡単にアウトプットとして生成出来るようなツールや環境条件が整備されている必要がある。この方法の利点は，出来るだけ早い時点においてエンド・ユーザと実際上のアウトプットなどを利用して確認出来る点である。プロトタイピングによる工程が従来の開発工程とどの様に位置づけにあるかを図表２－４として概念的に描いてみる。

図表２－４　プロトタイピングに依る工程図



2.1.6　ソフトウェアＣＡＤでの工程の考え方

ソフトウェアＣＡＤに対する明確な定義づけはないが，概念的にはソフトウェアのライフサイクルである，ソフトウェアの要求仕様を確定する段階からテスト段階といった広い領域を対象に，コンピュータの支援を得ながら開発する方法といえる。現時点では，開発工程の或る作業の効率化を狙って，コンピュータを利用した多くのツールが開発されている。そこでこれらのツールとソフトウェアＣＡＤとの相違が何処にあるかを眺めてみると，多くの従来型のツールは，単独のツールとして設計されており，しかも開発工程のある部分だけの作業の効率化を狙っている場合が多い，また，ソフトウェア開発の特性である連続性をあまり意識されないで作られている場合が多い。

そのために或る工程での或る作業の効率は非常に増大するが，全体としてはそれほど大きな効果を発揮しないといった悩みがあった。

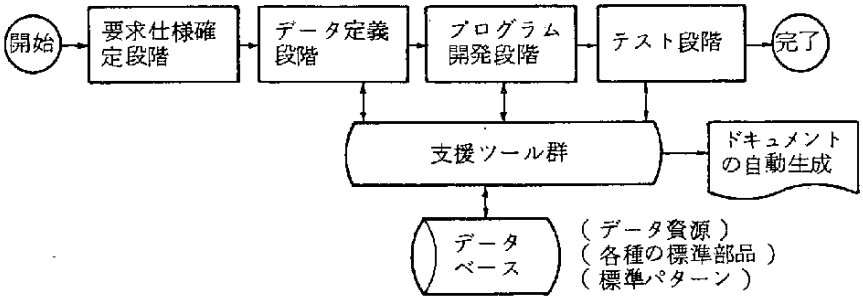
この様な問題を解決する方法としてソフトウェアのライフサイクル全体を通しての生産性向上を狙った統合化されたツールであるソフトウェアＣＡＤの手法が考えられてきた。

この場合，特に考慮に入れる点は，対象となるソフトウェアの規模や複雑性の問題である。比較的，小規模で，それほど複雑でないソフトウェア開発の場合には，ライフサイクル全体をとおしたソフトウェアＣＡＤ的なツールが市販されており（例えば日本電気のＳＥＡ／Ｉ，日立製作所の，ＥＡＧＬＥなど）それなりの効果を発揮している。

この場合，工程といった概念をそれほど意識せずにコンピュータと対話しながら開発を進めることが出来るので，ソフトウェア工学での工程とは異なった新らしい工程の概念が確立されてくるものと思われる。

それに対して大規模で，複雑なソフトウェア開発を支援するソフトウェアＣＡＤは，現状ではまだ開発されていない。しかし今後はこの分野でのソフトウェアＣＡＤの出現が，ソフトウェア開発の品質の向上や，大巾な生産性向上のために大いに期待される分野でもある。ソフトウェアＣＡＤを利用した場合の工程の考え方がどうなるかを日本電気のＳＥＡ／Ｉをベースに眺めてみると概ね次の図表２－５のような工程になるものと思われる。

図表２－５　ソフトウェアＣＡＤによる工程概念図



2.2　工程区分の現状

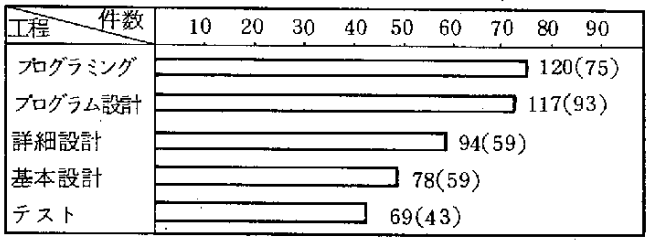
2.2.1　アンケート結果での現状

標準化と工程の概念の導入とは，非常に密接な関係にあるのでアンケート調査結果である標準化の状況から推測してみる。

標準化の推進組織の有無の設問では，回答企業１６０社のうち約4割の６８社に標準化の推進組織が存在しており，しかも標準化のなかで比較的実施のし易く，かつ基本をなしている用語の統一化やフォーム・シートの標準化については１６０社中の１５６社が実施しており，かなり高い割合を示していた。

さらに，開発工程での標準化状況について質問しており，その回答結果を次の図表２－６のように取りまとめた。

図表２－６　工程毎の標準化実施状況のアンケート結果

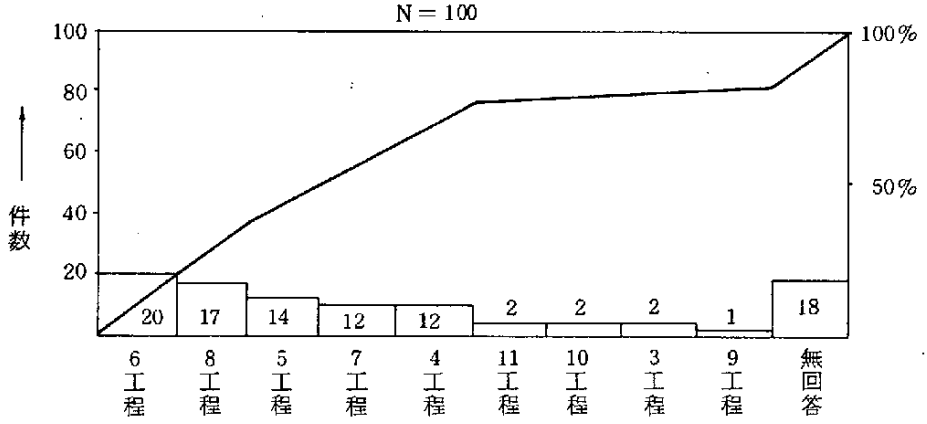


これらの結果から推測するに，開発方法の中に標準化の考え方や工程といった概念がかなり取り入れられ，各企業とも標準化の推進に腐心しているものと思われる。

このアンケートのほかに，日本システミックス社が行なったアンケートがあるので参考までに掲げる。ここでのアンケート内容は，情報処理全般にかかわるかなり広い範囲を対象としている。対象企業としては一部上場の１００社を対象としている。その設問の中の一つの項目として開発工程

をどのように設定しているかといった質問項目があるのでその回答結果をパレート図の形式にとりまとめてみると次の図表２－７の通りとなる。

図表２－７　導入している開発工程数のパレート図



このアンケート結果では．約半数の企業が，開発工程を６～８工程に分割していることが分る。又１０工程とか１１工程といったかなり多くの工程分けを実施している企業も５社あり，全般的には，開発を幾つかの工程に分けて実施している状態が分る。これらの事から，推測するに，かなり多くの企業において開発工程に対して工程分けの概念がかなり普及していることが分る。

そこで，これらの開発工程がどんな名称で呼ばれて区分されているかを眺めてみる。昭和５６年度の㈳日本情報センター協会が実施したアンケート結果（協会内の企業を対象とした）では．そこで使用されている名称にはかなりのバラツキがあり，各社の独自性が出ている。しかも同じ名称があっても作業内容が異なっていたりし，工程といった観点からは統一性があまり見られなかった。

一方，フレーム・メーカが設定している工程分けについて調査してみると，それほど大きな相違はみられないが，開発工程の上流部分での工程分けに幾分異なっている。この事は．各社の開発に対する思想的な相違から

くるものと思われる。又富士通がシステム規模の点に，ソフトウェア開発上の相違があるものとして，大規模，中規模，小規模といったシステム規模別の工程区分を設定している点が注目される。昭和５５年度の作業として実施された「各省庁電子計算機利用効率化共同研究会」でのレポートの中にフレーム・メーカがどんな工程分けを実施しているかの調査結果があるので，これを参考迄に図表２－８に掲載する。

図表２－８　各企業の採用している開発工程

開発標準名（企業名）	工程区分とその工程名称							
BEST（バロース）	調査・分析	基本方針	概要設計	詳細設計	プログラミング			
HIPACE（日立）	分析	システム計画	システム設計	プログラム設計	プログラム作成	組合せ総合テスト	移行／運用テスト	
IPT（日本IBM）	ユーザ要件の明確化	要求分析	システム設計	詳細設計	コーディング	テスト		
PRIDE（日本システミックス）	現状調査と検討評価	システム設計	サブ・システム設計	管理手続き設計			システム・テスト	
				コンピュータ手続き設計	プログラム設計	コンピュータ手続きテスト		
SDEM（大規模用）	計画		設計		プログラミング	テスト		
SDEM（中小規模用）	調査・立案	プロジェクト計画	システム設計	プログラム設計	プログラミング	結合テスト	システムテスト	運用テスト
SDEM（小）	調査・立案	プロジェクト計画	初期設計	論理設計	プログラム構造設計	モジュール設計	プログラミング	〃	〃	〃
STEPS（日本電気）	システム分析	概要設計	詳細設計	プログラミング	導入準備			

2.2.2 工程に対する基本的な考え方

前述したように，開発工程の区分においてそれぞれの企業毎にすこしずつ異なっている。そこで代表的な国産メーカ3社（日立，富士通，日電）をとりあげて，各社が開発工程や開発基準に対してどのような考え方をとっているか，各社の提供しているマニュアルをベースに眺めてみた。

(1) 各社の開発標準の特徴

① HIPACE－SPDS (High Productive Application Creation and Engineering－Standard Procedure to Develop System)

日立製作所で開発した，システムの計画，設計，開発及びこれらの作業に関する作業標準である。この開発基準は次のような特徴をもっている。

① コンピュータ部門主導型のシステム開発からエンド・ユーザ参画画型のシステム開発への指向；

これは開発作業をエンド・ユーザに肩代りさせるといった消極的な意味でなく，エンド・ユーザの真に役立つシステムを構築出来るのは，業務内容を一番熟知しているエンド・ユーザ自身であるという考え方から発生している。この開発方針の転換によって，コンピュータ部門の役割を高度なコンサルテーション業務や全体に対する計画業務の策定といった作業に中心を移していくことを狙っている。

② 計画段階の充実による手戻り作業のミニマイズ化；

開発作業の中で頻繁に発生する手戻り作業をミニマイズさせるために出来るだけ，開発の上流工程でエンド・ユーザ要求を十分に検討し確定する必要がある。その為に計画段階においてシステム案を多方面にわたって検討するようにしている。

③ 共有資源設計及び管理の一元化；

コンピュータやそれに関連するソフトウェアなどを共有資源とし

て位置づけ，それらの資源を個々の開発プロジェクトの作業範囲の中に取り込まずに，システム管理といった専門の職能にまかせて全体としての最適化がはかれるようにしている。

② SDEM（Software Development Engineering Methodology）

富士通が開発したソフトウェア開発及びプロジェクト管理のための開発作業標準であり，次のような特徴をもっている。

① 作業項目のカテゴリ化と定義；

各工程内で作業すべき項目をカテゴリとして分類しておき，そのカテゴリ毎にいつまでに，どんな種類の作業を実施しておかなければならないかが明確になっているので計画性を持ち込んだ作業が実施できるようになっている。カテゴリとして次の様な項目を対象としている。

㋑ システム環境（運用・移行，ハードウェア，設備，障害対策）

㋺ ソフトウェア開発（入出力，業務処理，ファイル，テスト，障害対策）

㋩ 管理（技術管理，教育・訓練，プロジェクト管理）

② 開発技法，方法論，ツールとの整合性に対する配慮；

ＨＩＰＯや構造化設計などといった新らしい開発技法などにも適応出来るように考えられており，しかもどの工程で，どんな開発技法やツールを適用するのが良いかを作業標準もその関連づけで明確化している。

③ レビュ工程の設定；

問題点を後の工程に持ち込まない為に，十分個々の作業でもレビューを行なうようにしているが，さらにこれを加えて中工程レベルでの工程の中にレビュー工程を明確に位置づけしていることである。このレビュー工程を設定することによって，大工程での終了段階を

明確に識別出来，しかも問題点の発生箇所を特定化出来るといった長所を持っている。これによって何如の工程でのレビューが不十分であったかが明確に分り，相互の索制機能が働き，より品質の高い作業の実施が可能となってくる。

(iv) きめの細かい工程分け；

システム規模のサイズによって，それぞれ異なった工程を設定しており，色々なタイプのプロジェクトに対応した形での工程分けが採用出来るように，きめの細かな配慮がなされている。

③ STEPS／C （Standardjed Technology and Enginering for Programming Support ／ Common ）

日本電気が開発したシステム開発作業やプログラム開発時の仕様作成の作業標準であり，次のような特徴を持っている。

(i) 編集主体のドキュメンテーションとなっている；

開発作業の中でかなりの負荷をおよぼしている作業としてドキュメント作成作業をあげることが出来る。そこでこのドキュメント作成作業の負荷を出来るだけ軽減できるような方策を作る必要があった。その結果，開発の実作業の中で必然的に作られるワーク・ドキュメントを出来るだけ活用し，それを編集するだけで目的とするドキュメント類が作成出来るように配慮されている。

(ii) 理解し易いプログラムの作成；

構造化プログラミング（Structured Programming）の技法を採用しているので，プログラム構造が明確となり，誰れでもが比較的理解し易い仕様となっている。

(iii) 標準パターンを提供している；

コンピュータ処理システムの中で良く使用されるプログラム・パターンを標準パターンとして整備しておき，これを必要に応じて利用出来るようにしている。これの活用によって標準化が一層推進さ

れる働きがある。

Ⓥ 人間優先の標準化が推進出来る；

標準化を無理に推進させる場合，これに関係する要員の人間性を或る程度殺した形態になる場合が多く，標準化に対する拒絶反応をおこし易かった。そこで出来るだけ人間の個性を殺したり，創造性の芽をつまないような配慮をした。

つまり定型的で作業内容の明確なものに対して標準化を実施し，最も人間らしい知的な作業に集中出来るようにした。

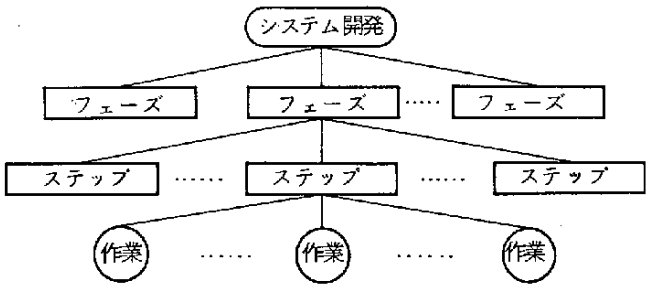
Ⓥ 融通性の高い利用方法；

ＳＴＥＰＳ／Ｃは，全体すべてを利用しなければ効果が発揮出来ないといった考え方で作られているのでなく，どの一部を採用することにしても，それなりの効果が発揮出来るような配慮がなされており，その意味ではかなり融通性の高い開発標準となっている。

⑵ 各社のシステム開発手順の考え方

① ＨＩＰＡＣＥ－ＳＰＤＳの場合

図表２－９　ＨＩＰＡＣＥ－ＳＰＤＳの作業手順階層図



ＳＰＤＳではシステム開発の作業内容を図表２－９のような３階層に分割して現在自分が行なっている作業がどの部分に相当しているかが十分認識出来るようになっており，開発要員の精神的負担の軽減の一助にもなっている。

また，階層上の作業が持つ意味は次のとおりである。

① フェイズ；次の工程に進めることが出来るか否かの意志決定をするのに利用される。

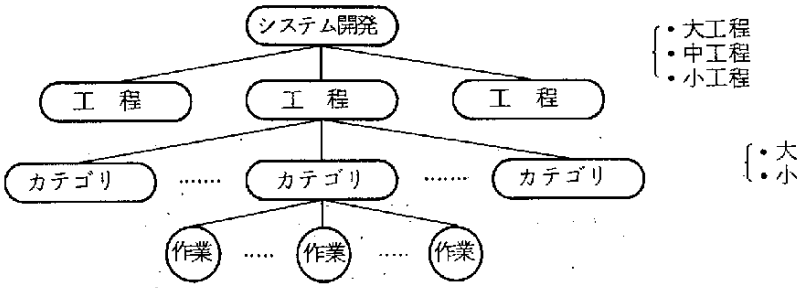
② ステップ；個々の作業状態がどのようになっているかの進捗管理の単位として利用される。

③ 作　　業；ワークシートに沿った最小レベルの作業単位で，技術上の一つの区切りを意味している。

② ＳＤＥＭの場合

マニアル上に正確には表現されていないが，マニアル上から判断すると次の図表２－１０のような構造をとっているものと思われる。

図表２－10　ＳＤＥＭの作業手順階層図

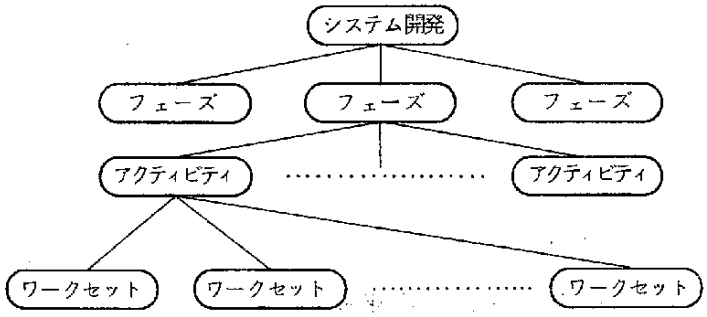


それぞれの階層が持つ意味は，ＳＰＤＳとほぼ同じ考え方をとっているものと思われる。

つまりフェイズ→工程，ステップ→カテゴリ，作業→作業といった対応関係を持っていると思われる。

③　ＳＴＥＰＳ／Ｃの場合

図表２－11　ＳＴＥＰＳ／Ｃの作業手順構造図



それぞれの階層が持つ意味はＨＩＰＡＣＥ－ＳＰＤＳと全く同一の定義付けをしており，相違点としては単に名称が異なっている程度である。

つまり，図表２－１１に示す通りフェイズ→フェイズに，アクティビティ→ステップに，ワークセット→作業に対応している。以上の３社の実例を眺めてみても，作業手順に関する階層として３階層を設定し，しかもそれぞれの階層が持つ役割においては全く同一となっている。これらから判断するに，システム開発手順といった開発の進め方には，それほど大きな相違が存在しておらず，発想上での差異はほとんどみられない。

2.3　工程での主要な作業項目とドキュメント

2.2.1で述べたように，一般の企業での開発工程の区分にはかなりのバラツキがある。これは，その企業での歴史的背景，システム規模，システム特性，開発手順（ハードウェア，ソフトウェアといった生産設備）などによって決定された結果と思う。しかし一方，フレームメーカが提唱している工程区分の状況を眺めてみると，それらの間にはそれ程大きなへだたりはないと思われる。そこで，ここでは計画段階が終了した以降の開発段階の工程を，システム設計段階，プログラム設計段階，テストラン段階と設定し，それぞれの工程で実施される基本的な作業項目や作成されるドキュメントについて3社（日電，日立，富士通）が提供している開発標準マニュアルをベースに眺めてみることにする。

2.3.1　システム設計段階

(1)　HIPACE－SPDSの場合

①　作業目的

システム計画フェーズで明確になったシステム要件を与えられた資源のもとで具体的な業務処理仕様，コンピュータ処理仕様に展開することを目的とする。

②　主要な作業項目とドキュメント

主要な作業項目	作成される主要なドキュメント
㋑ 業務処理仕様書の作成	・業務処理仕様書
㋺ 信頼性設計	・信頼性要求事項一覧表
㋩ コード設計	・コード設計書
㋥ 入力設計	・入力設計書
㋭ 出力設計	・出力設計書
㋬ オンライン入出力設計	・オンライン入出力設計書
㋣ ファイル設計	・ファイル設計書
㋠ DB設計	・DB設計書
㋡ DC設計	・DC設計書
㋢ 機械処理設計	・機械処理設計書
㋞ 性能設計	・性能設計書
㋝ テスト方式設計	・テスト計画書
㋜ 運用設計	・運用設計書
㋛ 移行方法の決定	・移行スケジュール
㋚ システム構成の見直し	

③ システム設計工程のドキュメントサンプル

C 3	コード設計	ワークシート	C 3.3	コード仕様表	作成		作成日		p.
					承認				

コード名称	部品コード	桁数	6	チェックディジット	有無	エントリ数	250

使用目的	各部品およびそのタイプを表わす。
付番方式	桁別コード

コード構成

	大分類	中分類	小分類
(1) 構成			
(2) バイト数	2	2	2
(3) 属性 1	X X	9 9	9 9
(4) 属性 2	Z Z	Z Z	Z Z

チェックディジット計算式	
チェック方式	

（コード付番方式）

構成	コード	内容	条件	備考
大分類	BL	ボルト		・対象物の名称などを憶えやすいように短縮する。
	NT	ナット		
	WS	ワッシャー		
	／			
中分類	10	SS41		・材質に対し固有のコードを付加する
	20	SUS304		
	30			
				・大きさをインチで

(2) ＳＤＥＭの場合

この手法でのこの工程の定義は，大工程として設計工程と呼んでおり，ここではシステム設計工程，システム設計レビュー工程，プログラム設計工程，プログラム設計レビュー工程の4種類の中工程を設定している。

そこでシステム設計工程に対応するシステム設計工程とシステム設計レビュー工程を抜き出して検討することにする。またこのＳＤＥＭではシステム設計工程を初期設計小工程と論理設計小工程に分けているので，それらの小工程毎に作業項目を洗いあげる。

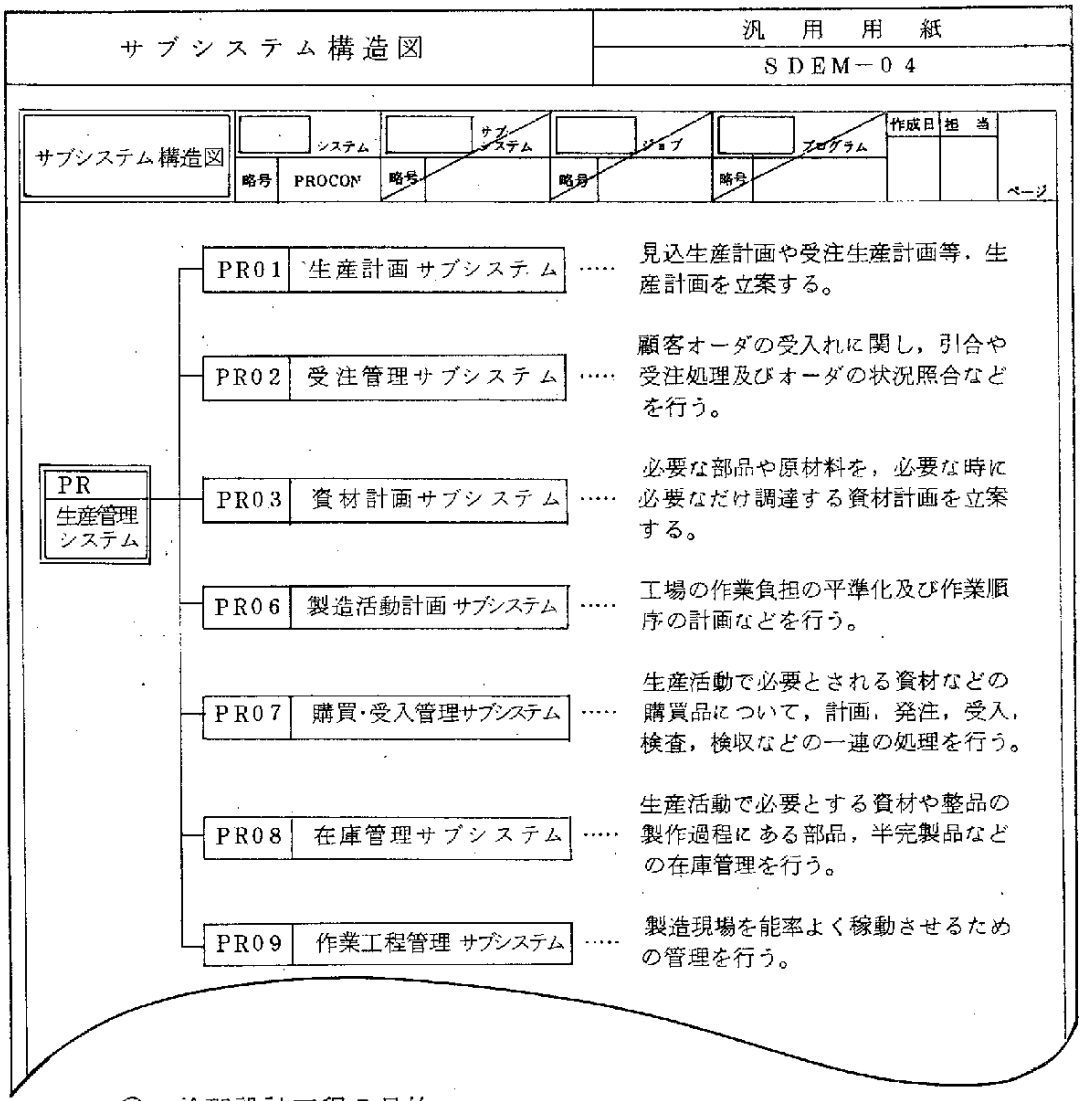
① 作業目的

ユーザ要求の収集分析結果をもとに，システムの機能を確定する。またその機能を実現するために必要な基盤（システム構成など）を明確にすることを目的とする。

② 初期設計工程の主要な作業項目とドキュメント

主要な作業項目	作成される主要なドキュメント
㋑サブシステム構造の定義	• サブシステム構造図
㋺サブシステム間の関連づけ	• サブシステム関連図
㋩サブシステムの概要定義	• サブシステム概要定義
㋥ジョブの構造化	• ジョブ構造図
㋭ジョブの関連づけ	• ジョブ関連図
㋬ジョブ概要の定義	• ジョブ概要定義
㋣コード仕様の定義	• コード仕様定義
㋠入出力データの概要定義	• 入出力帳票概要定義
㋡ファイルの概要定義	• ファイル概要定義
↓	↓
㋮初期設計書の編集	•初期設計書

③　初期設計工程のドキュメントサンプル



④　論理設計工程の目的

システム構築上必要となるすべての機能の洗い出しを行ない，その機能を一つ一つのプログラムに分割し，プログラム間の流れを明確にすることを目的としている。

⑤　論理設計工程における主要な作業項目とドキュメント

主要な作業項目	作成されるドキュメント
㋑プログラム構造の定義	•プログラム構造図
㋺プログラムの関連図	•プログラム関連図
㋩プログラムの概要定義	•プログラム概要定義図
㋥機能の詳細定義	•機能詳細定義図
㋭帳票のレイアウトの定義	•帳票レイアウト定義図
㋬画面の遷移状態の定義	•画面遷移図
㋣ディスプレイ画面の定義	•ディスプレイ画面定義
㋠ファイル／レコード仕様の定義	•ファイル／レコードの仕様
↓	↓
㋡論理設計書の編集	•論理設計書

⑥ 論理設計工程のドキュメントサンプル

記　入　例	用紙名	汎用用紙
	用紙番号	SDEM−04

プログラム構造図	生産管理 システム	製造活動計画 サブシステム	負荷山積 ジョブ	プログラム	作成日	担当	
	略号	略号	略号	略号			ページ

PRO601 負荷山積

- PRO6101 製造手配変更 … 資材計画サブシステムで計画した製造オーダを，負荷山積ジョブで使用するためにデータの編集を行う。
- PRO6102 工程展開1 … 製造手配で示される製造オーダを，品目，手順，工程の情報をもとに作業オーダレベルに展開する。この中で重要工程のみ山積用として重要工程オーダファイルに出力する。
- PRO6103 工程展開2 … 製造手配で示される製造オーダを，品目，手配，工程の情報をもとに作業オーダレベルに展開する。ここでは作業指示計画ジョブ用に展開するので，すべての工程について作業予定ファイルを出力する。
- PRO6104 前回負荷消込み … 重要工程に結合している負荷レコード及び作業明細レコードをすべて消込む。
- PRO6105 最早開始日山積 … 重要工程について，最早開始日による負荷山積を行う。また，同時に最遅開始日山積のためデータし出力する。

… 最遅開始日山積データをもとに，最遅開始日による，期間単位の山積を行う。また最遅開

(3) STEPS/Cの場合

STEPSではこの工程をシステム概要設計工程と呼んでおり，次工程であるシステム詳細設計工程の一部の作業項目も含んでいて，若干他社の工程分けの概念と異なっている。

① 作業目的

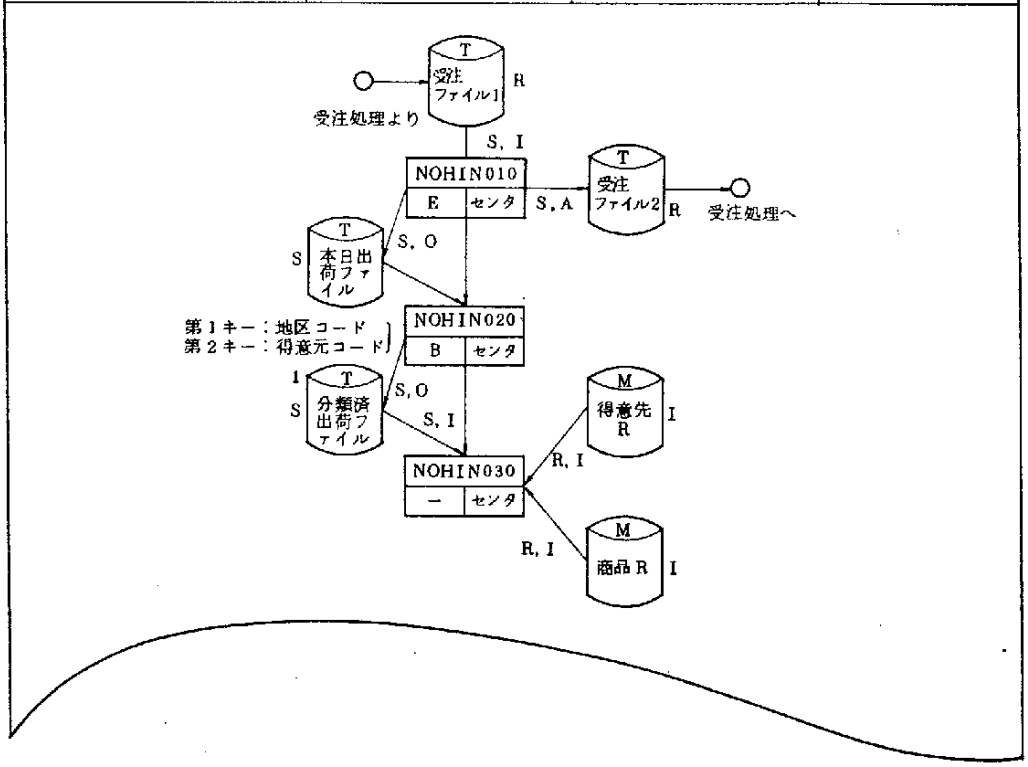
この工程では，前の工程で作成された基本構想の実現可能性を調べコンピュータの処理内容の概略および，それが人間の行なっている作業に与える影響を明らかにすることを目的としている。

② 設計工程の主要な作業項目とドキュメント

主要な作業項目	作成される主要なドキュメント
㋑人間・機械インタフェス設計	• 出力仕様，出力レイアイト 入力仕様，入力レイアイト etc
㋺データベース（ファイル）設計	• データ構造図，レコード仕様 ファイル仕様 etc
㋩データ項目（コード）設計	• データ項目仕様
㋥処理設計	• プログラム構成図
㋭業務運用設計	• 業務分掌 事務処理フロー図 etc
㋬信頼性設計	• 障害分析票 障害対策仕様 etc
㋣性能評価	• ファイル容量算定表 処理時間見積り表
㋠運用システム設計	• 機器構成図 コンピュータ運用予定 etc

③　システム設計工程のドキュメントサンプル

タイトル プログラム構成図（PND）	年　月　日	版	承認	査閲	担当	登録番号 参照番号 作成者
ジョブ名： NOHINBTH	システム名		サブシステム名			実行周期・時期： 毎日10時，17時



2.3.2　プログラム設計段階

(1)　HIPACE－SPDSの場合

①　作業目的

システム設計フェーズで分割，設定されたプログラムをさらにジョブの性能，機能としてのまとまりなどを考慮してモジュールという単位に細分化する。

また業務処理仕様書に基づいて新しい業務処理の流れの設計，マシ

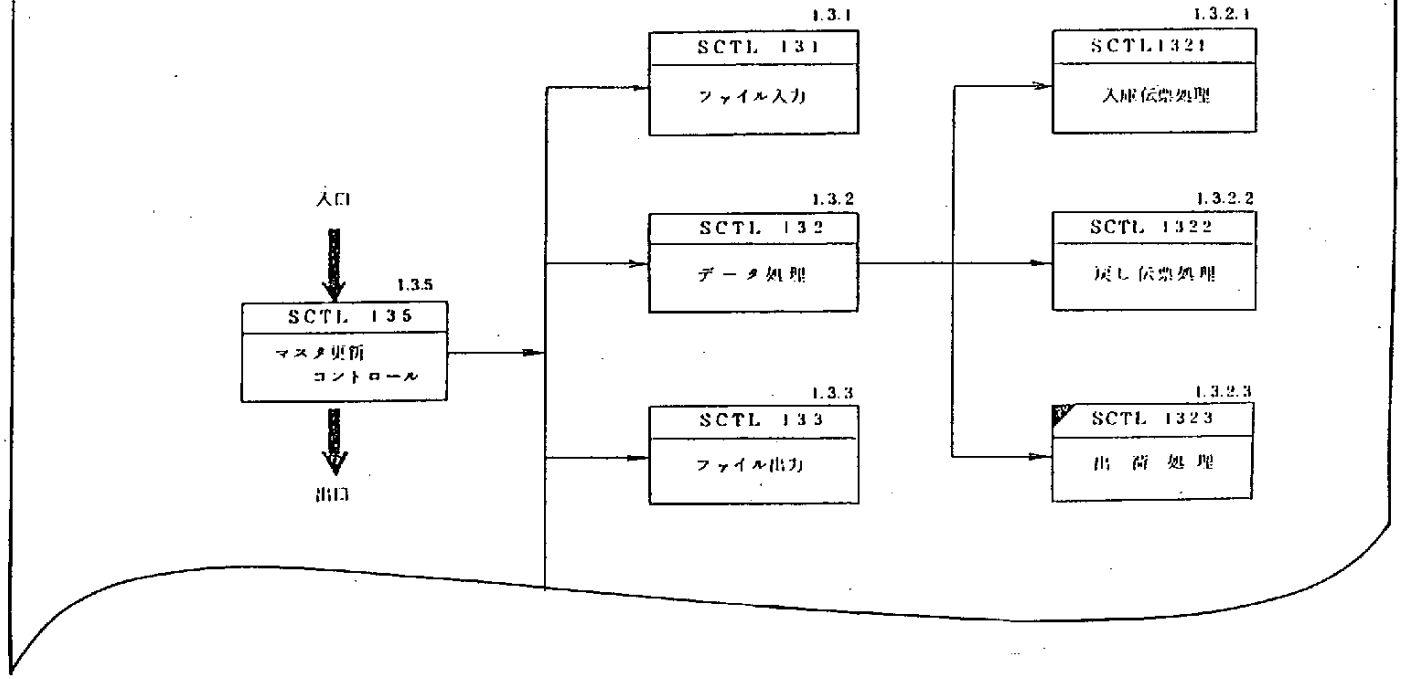
ン室の運用方法などを決定することを目的とする。

② プログラム設計の主要な作業項目とドキュメント

主要な作業項目	作成される主要ドキュメント
㈠業務処理仕様の確認	
㈡業務処理マニュアルの作成	
㈢機械処理設計の確認	• プログラム処理概要図
㈣プログラム内のモジュール分割	• プログラム機能階層図
㈤モジュール仕様表の作成	• モジュール関連図
㈥テーブル仕様表の作成	• テーブル仕様表
㈦モジュール仕様表の作成	• モジュール定義書
㈧運用規準の検討	• 運用規準
㈨運用手順書の作成	•
㈩運用支援機能の決定	
㈠㈠障害回復手順の検討	
↓	↓
㈠㈡プログラム設計レビュー	• プログラム設計レビュー報告書
㈠㈢プログラム仕様書の編集	• プログラム仕様書

③　プログラム設計工程のドキュメントサンプル

D 4	モジュール設計		ワークシート	D4.2	モジュール関連図	作成		作成日		P.
						承認				
業務名称	部品管理	項番	1.3	プログラム名称	マスタ更新		プログラム記号名称	SUK030		



(2) SDEMの場合

① 作業目的

プログラム内のモジュール構造を定義し，各モジュールの機能とインタフェースを定義する。また，プログラム内のデータ領域の関連性について定義し，さらにはモジュール内の詳細処理手順とデータ領域を定義しコーディング可能なレベル迄詳細化することを目的とする。

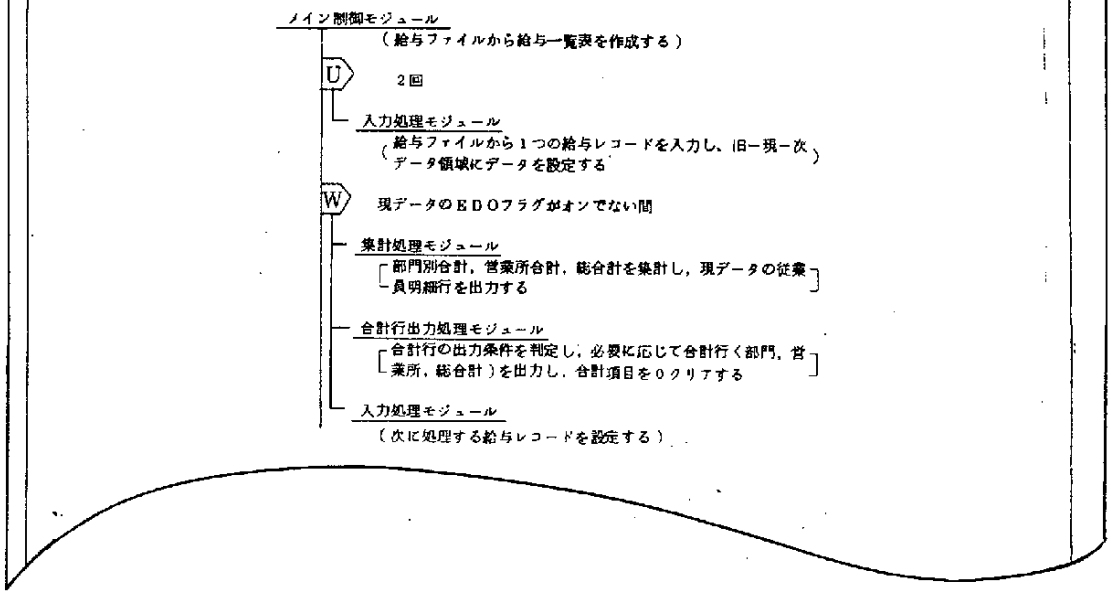
② プログラム設計工程の主要な作業項目とドキュメント

主要な作業項目	作成される主要なドキュメント
㋑プログラム外部仕様の理解	
㋺入出力データ構造の整理	
㋩主要データと機能の流れの整理	• 機能データ関連図
㋥モジュールの分割と構造の確定	• モジュール構造図
㋭モジュールの外部仕様を定義する。	•外部モジュール定義
㋬主要データ領域の定義	•データ領域定義
㋣モジュール構造，データ構造の評価	
㋠モジュール内の処理手順とデータ領域の詳細な定義	• 詳細処理手順
↓	↓
㋡プログラム設計書の編集	プログラム設計書

③ プログラム設計工程のドキュメントサンプル

記　　入　　例	用紙名	汎用用紙
	用紙番号	SDEM−04

モジュール構造図	販売管理 システム		売上管理 サブシステム		マスタ更新 ジョブ		売上マスタ更新 プログラム		作成日	担当	
	略号	HANKAN	略号	URIKAN	略号	CMUPOT	略号	UCM001			ページ



(3) STEPSの場合

STEPSの場合他の２社と比較すると，詳細設計段階の一部の作業がシステム設計段階に入り込んだ形態となっているが，基本的には詳細設計段階がプログラム設計段階に対応している。

① 作業目的

前工程であるシステム概要設計段階ではコンピュータを中心において形成されるシステムのコンピュータによる処理の概要および人間の作業変化を明確にしている。これに対してこの工程ではシステムのコンピュータによる処理内容を明確にした処理の詳細化をはかることを目的としている。

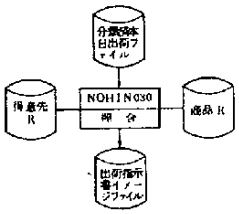
② 主要な作業項目とドキュメント

主要な作業項目	作成される主要なドキュメント
㋑コード表の作成	• コード表
㋺コード管理方法の検討	• コードの管理方法
㋩プリントレイアウトの検討	• プリントレイアウト
㋥入力媒体の仕様の検討	• 入力媒体の仕様
㋭帳票の仕様の検討	• 帳票の仕様
㋬データ作成方法の明確化	• データ作成要領
㋣原始帳票記入方法の明確化	• データ記入要領
㋠端末操作仕様の検討	• 端末操作仕様
㋡画面の検討	• ディスプレイ画面レイアウト
㋢サブスキーマ仕様の検討	• サブ・スキーマ仕様
㋧項目の整理	• 項目説明
㋐プログラム処理概要の整理	• プログラム仕様
㋒プログラム構造の検討	• プログラム構造図
㋓モジュール仕様の明確化	• 処理説明
㋁キーレコード仕様の整理	• キーレコード仕様
㋖プログラムテスト手順の検討	• テスト順序表
㋕プログラムテスト方法の検討	• テストデータ作成要領

③ プログラム設計工程のドキュメントサンプル

タイトル プログラム仕様	年 月 日	版	承認	査閲	担当	登録番号 参照番号 作 成 者
プログラム名： NOHIN030	業務名： 納品処理		処理周期： 日次（2回）			言語名： COBOL

チャート：



処理概要：
分類済本日出荷ファイルを基に得意先R，商品Rと照合し，必要なデータを得ると共に出荷数量を計算して，出荷指示書イメージファイルを作る。

得意先R……得意先名，郵便番号，住所
商 品 R……商品名

入出力，ファイルの名称	内部ファイル名	区分	特性 字数／レコード	レコード数／ブロック	編 成	アクセス	備 考
分類済本日出荷ファイル	DSFILE1	I	256	1	S	S	INP1−FILE
得意先R	DSFILE2	I	67	1	V	D	INP2−FILE
商					V	D	INP3−FILE
							OUT1−FILE

2.3.3 テストラン段階

(1) HIPACE−SPDSの場合

① 作業目的

前工程で検証された個々のプログラムをコンピュータシステムとして統合し，このシステムが一つの策として正常に動作し且つ要求される機能を充足することを確認することを目的とする。また，このシステムに関連する要員の教育訓練をすることを目的としている。

② 主要な作業項目とドキュメント

主要な作業項目	作成される主要なドキュメント
㋑作業方針	• 「テスト」作業計画書
㋺作業体制	
㋩作業スケジュール	
㋥教育計画の立案	
㋭エンド・ユーザ教育	
㋬オペレータ教育	
㋣組合せテスト計画の確認	
㋠組合せテスト準備	• 組合せテスト指示書
㋡組合せテストの実施	• 組合せテスト工程管理図
㋢総合テスト準備	• 総合テストチェックリスト
㋓総合テストの実施	• 総合テスト指示書
	• 総合テスト工程管理図
	• 総合テスト障害管理表
↓	↓
㋔テスト報告書の作成	• テスト報告書

③　テストラン工程のドキュメントサンプル

T3	組合わせテスト	ワークシート	T3.4.1	組合わせテスト指示書	作成		作成
					承認		

テスト種別		依頼者名		所要時間	３０分
組合わせテスト	総合テスト	依頼者TEL		機種	M－１５０H
	運用テスト	依頼日時	１２月１日１５時	ラン予定時間	１５時００分～１５時４０分
		JOB名称	AIR 010	備考	

依頼者特記事項	オペレータ・所見
1. 異常終了時，ＪＯＢを打切って下さい。 2. 異常終了時，テープエディット，ランダムアクセスエデットを行なって下さい。 テープエディットＪＯＢ名称は， A1RMT1～A1RMT3ランダムアクセスエディットＪＯＢ名称は， A1RDK1～A1RDK2	処理結果 正　常 異常ステップ名

(2) SDEMの場合

① 作業目的

結合テスト仕様書をベースにモジュール間，プログラム間などの結合テストを行ない，それぞれのモジュール，プログラムが目的とする機能を果たしているかをチェックすることを目的とする。また，システムテスト仕様書に基づき，システムを総合的にテストし，製品目標項目が正しく行なわれていることを確認する。

② 主要な作業項目とドキュメント

主要な作業項目	作成される主要なドキュメント
㋑性能評価，効率の測定	
㋺結合テスト	• 結合テスト総括表
㋩オペレータ教育	
㋥エンド・ユーザ教育	
㋭運用機能の確認	
㋬運用スケジュールの最終確認	
㋣入出力用紙の確認	
㋠保守手順の設定	
㋡例外処理テスト	
㋒限界テスト	
㋐操作性テスト	
㋓耐久テスト	
↓	↓
㋕製品の品質最終チェック	• テスト結果報告書

(3) STEPSの場合

① 作業目的

単体テストが終了したジョブプログラムを組み合わせ，ジョブとして或いはシステムとしてのテストを行うことを目的としている。

また，システムテストが完了した後における現行システムとの並行処理およびシステムの切りかえなどの作業を行う。

② 主要な作業項目とドキュメント

主要な作業項目	作成される主要なドキュメント
㋑備品，消耗品の準備	•備品消耗品チェックリスト
㋺システムテスト手順の検討	•テスト順序表
㋩システムテスト方法の検討	•テスト仕様及び状況一覧 •テストデータ作成要領
㋥システムテストの実施	•テスト仕様及び状況一覧
㋭業務運用の準備	
㋬コンピュータ運用の準備	•並行処理仕様
㋣運用テスト	
↓	↓
㋠システム導入の準備	•システム運用説明書 •コード説明書 •業務運用説明書

	年月日	規	承認	否認	担当	登録番号
テストデータ作成要領						参照番号
						作成者

名称	仕入業務のテスト（仕入データ）	利用プログラム		区分	☑システム ☐プログラム

データ項目	作成方法	備考
伝票№	正常なもの。 形式が誤っているもの。	
納品区分	正常なもの。 形式が誤っているもの。 コードが誤っているもの。	
訂正数量	正常なもの。 形式が誤っているもの。	
<データ量>	○正常なデータ ―――― 全件数の７０％分 ○誤っているデータ ―――― 全件数の３０％分 1項目が誤りのデータ ―――― 〃 １５〃 2項目が誤りのデータ ―――― 〃 １０〃 3項目が誤りのデータ ―――― 〃 ５〃	過去の実データを使用する。 人手で新規に作成する。

参考文献

1) 「ソフトウェアの開発・管理の効率化」研究結果報告書，昭和５５年度，各省庁電子計算機利用効率化共同研究会

2) 「情報処理サービス業の標準化に関する報告書」，昭和５６年５月，㈱日本情報センター協会

3) 「ソフトウェア生産性向上の諸方策とその実際」昭和５９年１１月，㈱ソフトリサーチセンター

4) 「ソフトウェア・エンジニアリングへの道(1)」Vol 9，№4，宮本勲・東谷秀夫；bit

5) 「これからのシステム開発Ⅰ」，Computer Report 1984/1，前田耕一

6) 「これからのシステム開発Ⅱ」，Computer Report 1984/2，前田耕一

7) 「これからのシステム開発Ⅲ」，Computer Report 1984/3，前田耕一

8) 「FACOM SDEM 概説書」（ソフトウェア開発方法論）富士通

9) 「FACOM SDEM ソフトウェア開発作業標準解説書」，富士通

10) 「STEPS/C 概要説明書」，日本電気

11) 「STEPS/C システム開発標準」，日本電気

12) 「システム開発標準手順 SPDS 概説編」，日立

13) 「システム開発標準手順 SPDS システム設計編」，日立

14) 「システム開発標準手順 SPDS プログラム設計，作成，テスト編」，日立

3. ソフトウェア開発の高度化・効率化のための方策

3. ソフトウェア開発の高度化・効率化のための方策

最近の情報化の進展につれてコンピュータ化ニーズは拡大の一途を辿っており，しかもシステム内容は益々複雑化してきている。

ところがこれを開発する側の問題として，開発作業に大きな役割を果たしている開発要員の数がそれほど増加していない事である。其の上，システム規模の増大や高度化によって，開発技術力の高度性を要求される関係上，開発要員の一層の逼迫を促している。

一方，開発の生産性自体を眺めてみても，従来の開発方法では問題ありとの考え方で，ソフトウェア工学の考え方や，各種の支援ツールの導入などによって，ソフトウェア開発での生産性向上の努力が継続的に実施されてきた。しかしこれらの努力にもかかわらず，生産性向上に対して顕著な効果を発揮出来ていないのが現状である。

その為に多くの企業ではソフトウェアのバックログ問題に頭を悩ましており，しかもコンピュータ部門の問題点として常に上位にランクされており，それだけコンピュータ部門の重要な課題となっている。

そこで，本年度では計画段階以降の開発の生産性を如何にして向上させるかについて考察することにする。ソフトウェアの生産性を向上させる為には種々の施策があるが，この章ではそれらの施策について総花的にふれ，その内容について詳細に記述することにする。

しかし，この生産性向上の方策といえども，この方法を採用することによって必ずしも同様な効果が期待出来るとは限らない。そこで開発効率向上施策を効果の面に視点を置いて分類してみると次のようにグルーピングされる。

(1) 生産性向上のため基礎的な施策

生産性を向上させる為の前提条件的なものでこれが不十分だと，(2)，(3)の施策が期待したほどの効果が発揮出来ないので非常に重要な役割を負っている。

① 標準化

② 組織化

(2) 生産性向上に即効的な効果を発揮させる施策

これらの施策でも十分な効果を発揮させる為には，種々の環境条件の整備などが必要とされる。

① 機械化

② 再利用

③ ソフトウェア・パッケージの活用

(3) 生産性向上に遅効性の効果を発揮させる施策

ここで実施される施策は，この方策を導入した事によって直ちに大きな効果を発揮するという性格のものでなく，この導入によって着実にその効果を発揮し，生産性向上に対し安定した効果を与えるものと思われる。

① 品質管理活動

② 開発環境

3.1 標 準 化

ソフトウェアの開発を考える場合，手持ちの優秀な要員をプロジェクトに投入することによって開発効率や品質向上をはかることが多い。この事は，つまり，ソフトウェア開発自体がかなり属人化していることを顕著にあらわしている。しかし最近のようにソフトウェアの規模自体が巨大化し，しかも複雑化しており，その上悪いことにそれを開発する要員の絶体的な不足と，質の問題が大きくクローズ・アップされており，今後のプロジェクトには経験の乏しい要員を多数抱えしかも，技術的に様々なレベルの外注要員を含くめた開発を余儀なくされているのが実状である。

この様な悪条件のもとで開発効率と，品質の高いソフトウェアを開発する為の最底条件として標準化の推進が必須条件となってくる。

またソフトウェアのライフサイクル論にあるように，ソフトウェアにかか

わる生産性は開発段階だけでなく，それの後の工程である運用や保守工程での生産性向上が重要な課題となってきている。

その為には開発工程での標準化がなされているかいないかが大きな影響をおよぼすことになる。この場合重要な事は，開発段階だけの開発効率や品質向上を狙うのでなく，運用や保守段階の効率や品質を向上させる標準化をはかる必要があることである。

しかし本年度は開発段階に焦点をあてているので，開発段階での開発効率を向上させる為の一つの方法論としての標準化について記述することにする。開発段階で推進すべき標準化対象項目として大きくは次のような項目があげられる。

3.1.1　ソフトウェア階層

ソフトウェア開発において，システム全体の複雑さを克服するために，全体を適切な構成要素に分割し，階層化させることは大変重要な事である。

システム → サブシステム → プログラム → モジュール

(1)　ソフトウェア階層化の必要性

ソフトウェア階層化が重要であることは，以下に述べているような大きな利点があるからである。

① 構成要素ごとに設計を並行して進められる。

② 開発されたソフトウェアが信頼性の高い，保守しやすいものになる。

③ 構成要素ごとに独立してテストができるようにし，その段階（単体テスト）でほとんどの誤りを見つけ出してしまうことができる。

④ 部分的な改造が全体に影響しない構造にして，機能の追加や削除が局所的な変更で行えるようにできる。

(2)　ソフトウェア階層化の要点

ソフトウェアを分割し，階層する場合，大きくは，システム，サブシステム，プログラム，モジュールと分けるのが一般的である。また，モ

ジュールとはコンパイル単位をさすことが多い。

モジュール分割する場合，注意すべき点として以下のようなことがいえる。

① モジュールはできるだけ他と独立になっており，インターフェースが単純であること。

② そのモジュールを使う場合に，必要な知識が少なくてすむような分割する。

③ モジュール作成に際しても必要となる知識をうまく切り分けて，開発分担を決めやすくする。

④ 最終的には，５０～２００行程度で簡単にコーディングできるレベルまで段階的に分割していく。

⑤ システム全体の外部環境（入出力データ等）をまず明らかにしてから分割していく。

3.1.2 用　語

ＩＳＯ（国際標準化機構）やＪＩＳ（日本工業規格）では，用語を始めとして多くの標準化をとりあげている。

① 情報処理用語

② 流れ図記号

③ 入出力媒体

④ プログラム用言語

⑤ データコード

⑥ 文字セット

⑦ データ通信

⑧ ネットワークプロトコル，etc.

これら以外にも提案レベルのものが数多くある。

用語は，業界や企業毎，業務内容にと統一的なものと，特殊なものが

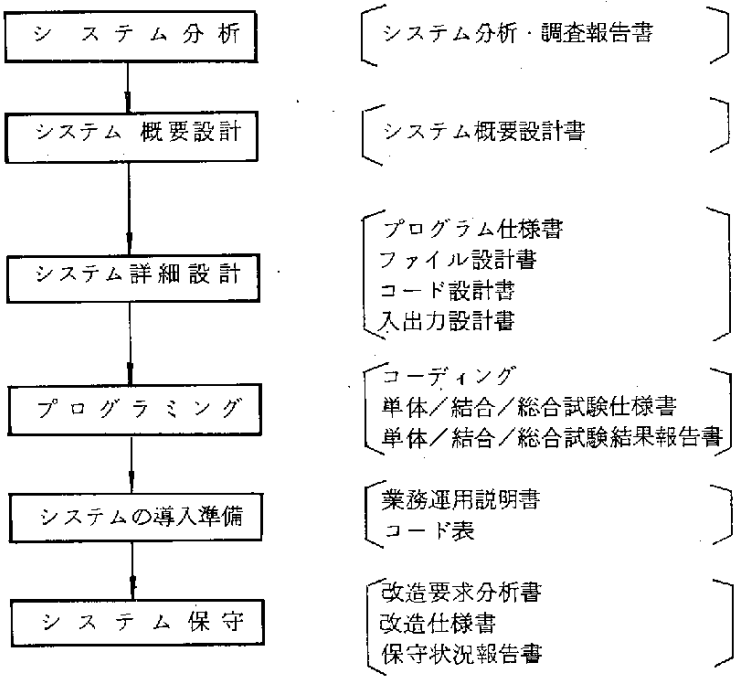
存在している。そのため，ＪＩＳ規格だけでまかなうことは無理である。

情報処理システム全体として，またソフトウェア開発の各作業段階においても，用語を始めとして，ドキュメント様式，記入要領などといったことは当然標準化しておかなければならない。

3.1.3　作業方法

(1)　システム開発工程

システム開発における各工程での作業の標準化について，日本電気㈱のシステム開発標準（ＳＴＥＰＳ）を参考にしながら，標準化の目的，項目，内容などについて述べる。またドキュメント類についても触れる。



(2) システム分析

① 目　的

要求内容を，入力，出力，処理内容，環境として明確化することであり，それら要求を正確に分析することが重要である。

例えば，マン・マシン・インタフェース，計画立案作業，作成ドキュメントなどの標準が云える。

② 標準化作業内容

システムの構築するためには，まずその業務がいかなる部分がコンピュータで処理可能か，どこまでをコンピュータ化すべきかをまず分析して見極めなければならない。そのためには，現在手もとにある情報をどんな形態の情報にすれば良いのか，どうコンピュータに処理させるのかを分析しなくてはならない。

コンピュータ化していない業務をコンピュータ化しようとする場合，現在の時点では多くの規定書等の文書類が存在するのが普通である。そういった資料をコンピュータの入出力と結びつけていき，資料の分析を行う。

入手した生の資料が重要であり，形のない情報については調査書にして残しておくべきである。

分析の段階では，生の資料だけでなくそれに対してコンピュータによって，どう飾り立てていくかを検討することが重要である。要求に対して，システム案を作るわけであるが，「何をコンピュータにさせたいのか」を把握するのが，分析の段階になると思われる。

この段階で，要求側と供給側がくい違うと先へ進んでも，振り出しへ戻らなくてはならないので，要求が上がって来たら，「こう分析しました」を文書にして確認を行うべきである。

(3) システム概要設計

① 目　的

プログラム・モジュールの独立性，処理とデータとの分離，ハードウェア／ソフトウェアの独立性などを実現する標準化でなければならない。

そのため，設計方法論やツールを利用することやソフトウェアの部品化，データ構造の標準化を推し進める。

② 標準化作業内容

システム構築のための準備作業が設計に入れる段階まで進んだら，システム概要設計に入る。

(a) 入出力設計

この段階では，まず入力と出力をさらに明確にする。例えば「エラーのチェックを行うプログラムでは，プルーフリストとエラーリストを出力する。」などである。

入出力を外見上から分類してみると，以下のものが上げられる。

ⓐ リスト類

(i) 英数カナ帳票

(ii) 日本語・罫線帳票

(iii) 単　票

ⓑ 入力用媒体

(i) パンチカード

(ii) 磁気テープ

(iii) マークシート・OCRシート

(iv) フレキシブルディスクカートリッジ

（ディスケット，フロッピィディスク等，商標が一般的）

ⓒ 入出力用機器

(i) CRT，キーボード

一般的には，上記のものがほとんどであるが，特殊業務用にCD(キャシュディスペンサ)，IDカードリーダ(暗号カード)，コンビニ

エンスストアなどで見つけるハンドスキャナ（マーク読み取り），また，設計業務等のＣＡＤ（Camputer Aided Desigh），CAE（Computer Aided Engineering)で扱うグラフィックディスプレイ，プロッタ，デジタイザ，最近ではファクシミリなどのイメージ入出力も行われている。

これらのハードウェア装置は，コンピュータシステムに存在するか直接的か間接的に接続可能であることが絶対条件である。それを踏まえたうえでシステムの構築を進めていく必要がある。

末端の入出力のほかに機能分割したサブシステム毎の入出力となる中間ファイルや，システムの中核となるデータベースなどの蓄積ファイルがある。これは主に，磁気ディスクファイルが採用されるが，ハードウェアの規模と照らし合わせた時，容量が問題となる。

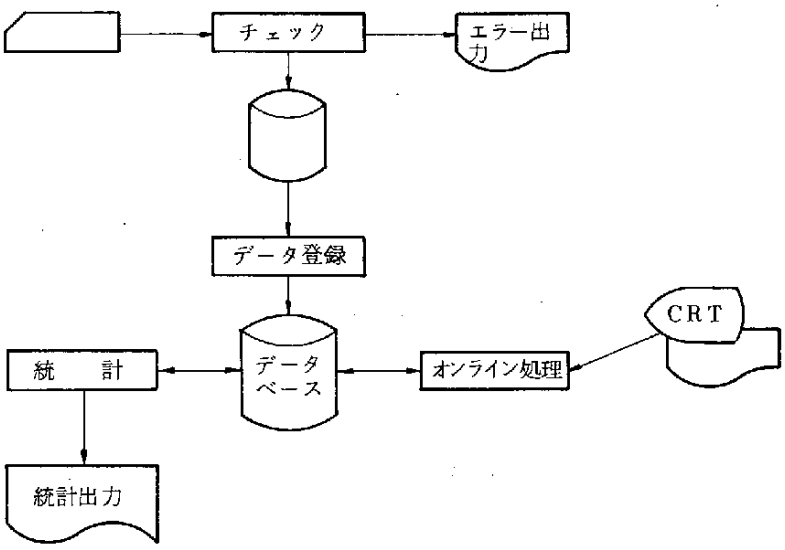
入出力方法を決める要素として，データ量がある。入出力時間が数時間にもおよぶ場合には，入出力機器や媒体等の見直しが必要となる。データベースの入出力においてはデータ構造が非常に大きな要因となるので，ファイル設計の段階で注意が必要となる。統計処理などで，データベースの全件検索を行うような場合の処理時間も計算しておく必要がある。

(b) 処理設計

入出力が決まったら，その工程をサブシステムで分割する。その過程で中間ファイルなどの必要が発生したり，あらかじめ後々のため保存しておく中間ファイルを設定する場合もある。

サブシステムの分割としては「バッチ処理」「オンライン処理」といったシステムの違う機能を中核とするもので分けたり，「統計処理」「データチェック処理」といった業務上の仕事の単位で分けたりできる。サブシステムは必ずしも，１つのプログラムから構造されるものではなくいくつものプログラムで形成されてもよい。こ

の段階でシステムの概念図が描けることになる。



この段階では，あとファイルで管理する項目やコードで管理する場合はそのコード体系を明らかにする必要がある。また，各サブシステムの機能やそれを構成するプログラムの機能分担を明らかにして，システム概要設計書にまとめる。

運用方法についても確認をとる必要がある。「オンラインを使う時には必ずコンピュータ側である操作を行う」「3年間に一度はファイルを再編成する」などの使う側で手間をかけることについてはなるべく避ける方法が望ましく，避けられない場合はシステム導入時までに詳細な手順書，運用手引書を作成する必要がある。

個人や団体の損益に絡むデータについては，機密保護が重要であり，データ量の多いものについては保全対策が必要である。

機密保護は，特にオンラインシステムで重要である。誰でもどこからでも使えるシステムでは，パスワードの二重化などをしてデータを不全なアクセスから守ることが必要である。保全対策について

は，プログラムの不完全更新対策や，磁気ディスクの物理的破壊に対してのデータ退避などの対策が必要となる。パスワードの入力方法や磁気ディスクのデータ退避についてはあらかじめ確認をとって運用手引書などに盛り込むべきである。

(c) 作成ドキュメント

システム概要説明書に盛り込むべき内容を列記してみる。

（ⅰ） システム概要図

（ⅱ） サブシステム機能設計

（ⅲ） 入出力概要

（ⅳ） ファイル概要

（ⅴ） コード設計

（ⅵ） 運用方法

（ⅶ） データベース構造設計

(4) システム詳細設計

① 目　　的

コンピュータによる処理内容の詳細について明らかにすることであり，作業結果としてまとめられるドキュメントのシートフォームを定め，作業が定型的に能率よく行えるようにする。

② 標準化作業内容

システム概要を作成し，それを要求側と確認し，合意を得るか，新たな要求が発生したらそれを盛り込むことを前提にして，システム詳細設計に入る。

システムの詳細設計の初期段階として，概要設計を踏まえたうえでプログラム仕様書作成に必要な，ファイル設計，コード設計，入出力設計を行う。これらの項目は，プログラム仕様書と一緒にすることも考えられるが，サブシステム間のみでなくシステム内で共通のファイル，コード，入出力は別の設計書として作成しても効果が高いと思わ

れる。

(a) 入出力設計

入出力設計は，概要設計段階ではシステムの概要を述べる程度で「何が入力され，何が出力されるのか」また「どんな中間ファイルや蓄積ファイルが存在するのか」ということがわかればよいが，詳細設計段階では，項目もしっかり決定しなければならない。

(i) 入出力項目

(ii) 画面レイアウト

(iii) 帳票レイアウト

(iv) カード等の入出力レコードレイアウト

(v) ファイルのレコードレイアウト

(b) ファイル設計

概要設計段階では，一般の中間ファイルや蓄積ファイルは，その存在位置（どのサブシステムでも何を目的に扱うのか）が明確になっていればよい。しかし，データベースのようなファイルの場合にはシステムの中核をなすものであり，データベース構造が他の設計要素に大きく影響してくるので，概要設計段階で構造くらいは決めておかねばならない。

詳細設計では前出のレコードレイアウト設計とともにファイルの編成も設計する。ファイル編成の種類を列記してみる。

(i) 順編成

(ii) 相対編成

(iii) 索引順編成

(iv) 各種データベース

用途に応じて，使い分けを行う必要がある。単なる中間ファイルなら順編成，キーによって検索するキーは索引順編成，複雑な検索を行うものは，データベースという具合に決定する。キーが必要な

ファイルについては，キー項目を何にするかがプログラミングにも影響をおよぼす重要な要素である。データベースについては，設計項目が多く，特別なノウハウを必要とする場合が多い。

(c) プログラム設計

概要設計でサブシステムレベルに機能分割したものを詳細設計では単体のプログラムレベルに分け各々をさらに細かく機能分割して，共通部分を見い出してサブルーチン化を図る。プログラム間インタフェースについては，プログラム仕様書以外に説明書を設け，テスト時にインタフェースの不整合がないようにしなくてはならない。

プログラム洗い出しが終ったら，プログラムに仕様書を作成する。プログラム単体の機能を洗い出し，1モジュール数＋ステップになるように，モジュール分割しモジュール分割図を作成する。サブルーチンの場合はインタフェースを明確にする。入出力のレイアウト，コード類の一覧，項目説明ももっと細かいレベルで書く。プログラムの手続き部分も詳細に書き，各単処理で入出力の概要が一目でわかるものが望ましい。オンライン処理では直接端末の操作法とかがわかってくるので，概要設計段階で手順レベルのことを決めておくのが望ましいといえる。単一プログラムの仕様書の項目を列記してみる。

（i） プログラム概要
（ii） プログラムインタフェース
（iii） モジュール分割図
（iv） 項目設計
（v） コード一覧
（vi） 画面レイアウト，帳票レイアウト
（vii） ファイル仕様
（viii） 処理仕様

(5) プログラミング

① プログラム作成

(a) 目　　的

作りやすく，理解しやすく，テストしやすく，保守しやすいプログラムを実現するため，部品化とその組み立て方式や各種のツール再利用技術，超高級言語などを活用して標準化を図っていく。

(b) 標準化作業内容

コーディングは，プログラム仕様書に忠実に行う。もし，コーディング時に仕様ミスを発見したらすみやかに仕様書にフィードバックさせるべきであり，その際他のプログラムの影響を十分考慮しなくてはならない。

ファイルのデータ項目や共通領域の設定は，コピーライブラリ等の機能を使った方が，共通性が保てるのでよい。データ定義部分ではコメントを多く活用したり，データ項目名を工夫し，そのデータ項目が何なのかを一見してわかるように配慮すべきである。手続き部分ではモジュール分割が明確にわかるようなコメントを使うべきであるし，プログラム中の手続きの階層レベルを明確にするため，コーディングの開始位置（左端）を段階的に揃えることが一般的に行われているし，望ましいといえる。コーディングの規約というべきものを持ち，多くの人がそれに従ってコーディングすることがプログラムの保守の上からも望ましい。

詳細設計→コーディング→試験までの過程はサブルーチンが先行し，結合テスト時点でタイミングを合わせるくらいの工程が最適といえる。

② プログラムテスト

(a) 目　　的

要求，設計，製造結果の正しさを証明し，後工程へのバグの流出

を防ぐため，テスト技術，テストケースの抽出，テストデータの作成，さらにはテスト結果に対する完成度の評価基準などに標準化を行う。

(b) 標準化作業内容

プログラムのテストは大別して3通りに考えられる。単体，結合，総合試験である。総合試験はどちらかというとシステムテストのレベルといえる。

単体試験は，サブルーチンのインタフェースのテストと考えてもよい。一度に結合してしまってテストしても，バグが発生した時，原因の所存をつきとめるのに時間がかかってしまう。そこでサブルーチンが仕様通りの動作をするか，テストドライバーなどのツールを使ってサブルーチンに渡した情報と戻って来た情報のチェックを行う。テストについてはあらかじめ，いろんな場合を想定してサブルーチンに渡すべき情報と戻されるべき情報のパターンを洗い出しておき，試験仕様書としてまとめ，結果を試験結果報告書としてまとめる。バグが検出され，修正を行った時には，記録にとどめておくべきである。

結合試験は，1本のプログラムテストと考えてよい。テストの完了したサブルーチン群を結合して，1本のプログラムの入出力をチェックする。これもいろんな場合を想定して，入力と出力を洗い出しておいて試験仕様書としてまとめ，結果を試験報告書としてまとめる。

プログラムの規模に応じて，単体試験を行わず，結合試験を行うこともある。この場合プログラムの入力としてはサブルーチンにもいろんな情報が渡るようなパターンを考えなくてはならない。

総合試験では，システムの入口から出口までの間のファイルインタフェースなどに注目して，チェックポイントを設け，各プログラム

が正しい動作を行うかチェックする。

(6) システム導入準備

この作業は，プログラムテストの後になるというものでもなく，システム導入の時期に合わせて，コード説明書や業務運用説明書が完成するようにしなくてはならない。

システムの設計段階で，画面や帳票のレイアウト，使用コード，端末やジョブの操作手順が決まったら，着々と作成を始めるべきである。操作者が何もわからないものとしてこれらの説明書を作成しなくてはならない。何をやりたいかが思い浮かんだら索引などから目的のページがすぐわかり，ステップ毎の説明がいもずる式にたぐれることが要求される。言葉や用語も曖昧な表現は避ける。イラストなどの目に訴えるものが効果的である。特に障害時の対処については，丁寧に書いておくことが重要で復旧が困難な状況になることがあってはならない。

① バッチ処理系の標準化項目

(i) 処理手順概要

(ii) ＪＣＬ

(iii) 入力データの形式・扱い方

(iv) 出力データの見方・扱い方

(v) ジョブの起動の仕方

(vi) コード表

(vii) エラーメッセージ

(viii) 異常終了した時の処置

② 端末処理系での標準化項目

(i) 処理手順の概要

(ii) 端末機の電源操作

(iii) 端末のキー操作

(iv) システムの起動方法

(v) 各画面の見方・入力の仕方

(vi) コード表

(vii) エラーメッセージ

(viii) 異常終了した時の処置

(7) システム保守

① 目　的

機能，性能，信頼性などを保証し，タイムリな保守作業を行うため，改造／拡張／修正作業手引，開発ドキュメントと保守ドキュメントとの分離，保守ドキュメントの自動作成または自動修正機能などによって標準化を行う。

② 標準化作業内容

システムが導入され，運用を開始するとバグ対応と改造という作業が残る。

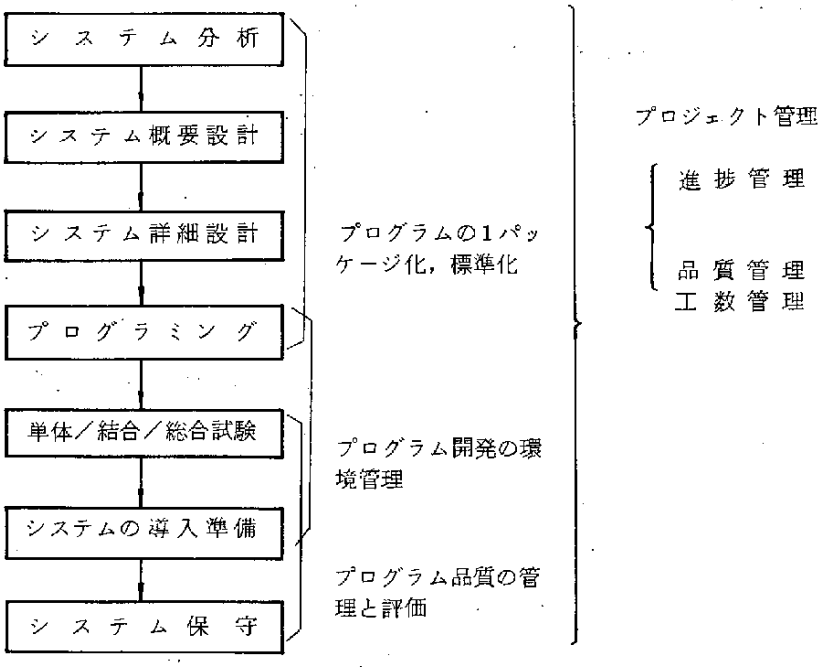
テスト段階で吸収できなかったバグは，発生した時点で速やかに対処しなくてはならない。プログラムの変更による他への影響を充分把握してから，行わなくてはならない。

後々，その対処から不都合が生じかねないので，修正箇所は明確に残しておかなくてはならない。ソースプログラム上にも，コメントとして残し保守状況報告書，障害状況報告書としても残し保守しておくべきである。

改造が要求された時には，まず改造の要求仕様を把握し，「現状システムで改造可能か」「どこまでなら改造できるのか」を仕様書などから判断し，改造仕様書としてまとめておかなくてはならない。バグの吸収より，大掛かりになるため，改造後のテストも入念に行うべきである。もともとのプログラム仕様書も改造した旨を明記しておくべきである。

3.1.4 開発管理

(1) 各作業工程での開発管理



ここでは，システム開発をいかに能率的に体系的に管理していくかを述べる。

(2) プログラムのパッケージ化標準化

システムレベルでのパッケージ化標準化は，分析の段階でパッケージ化または標準化に照準を絞れるかどうかによって決まってくる。特殊業務でどうしてもパッケージ化は不可能など，数々の要因は出てくると考えられる。標準化をすることに決定したら，類似業務の調査を行い，取扱い説明書や仕様書など，完備して保存しておくことが必要である。

プログラムの単体レベルでなら，詳細設計時にパッケージ化に的を絞れば可能である。今後，そのプログラムにインタフェースを合わせることによって効率化が図れる。

標準化するに当たっては，拡張性をも考慮して設計しなくてはならな

い。

パッケージ化，標準化しても利用しなければ意味がないので，それを流用するルートを確立することも大事である。

(3) プログラム開発の環境管理

環境管理のひとつとしては，まず環境設定がある。プログラム開発に際しては，環境が設定されていなくては，作業の進めようがないからである。現システムを開発するためにどこまで環境を設定しなくてはいけないか判断しなくてはならない。それは遅くともシステムの設計段階あたりで把握しておかないといけないだろう。開発環境が揃っていれば良いが揃っていなかったら，その環境を必要とするまでに環境を整備しなくてはならない。オンラインを扱うのに端末機がなくては何もできないし，データチェックプログラムのテストをするのにデータがなくては進まない。環境管理者はプロジェクトの進捗とつき合わせて常に先手先手に立ち廻らなくては，効率の良いシステム開発は開発できない。

環境管理としてもうひとつ大事なのが開発環境の把握である。特に大規模システムでは，プログラムライブラリ（ソースプログラム，オブジェクトプログラム，実行形式プログラム）やデータファイルが散乱し，世代管理などしていると同じプログラムもいくつも存在し，どこに何が保管されているのかわからなくなってしまう。管理者または当事者でも良いが，プログラムファイル名，ライブラリ名，プログラム名，データファイル名，データ名，さらに世代別や本番用テスト用など表形式にして管理しなくては，後々，本人でさえも忘れてしまう。いらないファイルを整理しようにも手がつけられなくなってしまう。

開発環境が破壊される恐れも考えられるので，定期的にファイルの退避を行っていくべきである。

(4) プログラムの品質の管理と評価

プログラムの一番原則として要求機能を果たすことであるが，テスト

完了時には機能が果たされていることになっている。しかし，特殊ケースの対応がなされていなかったなどの不都合がしばしば生じるようである。その点試験仕様書でチェックすべきである。

プログラムの読み易さというのも品質である。改造しようにも手がつけられないようなコーディングでは品質が良いとは言えないのである。仕様書がなくても解読できるレベルのプログラムが要求される。そのためにはコーディングの規約となるべきものを設定すべきである。

評価の要素としては「特殊ケースも配慮し仕様を満たしていること」「プログラムが見易いこと」があげられる。

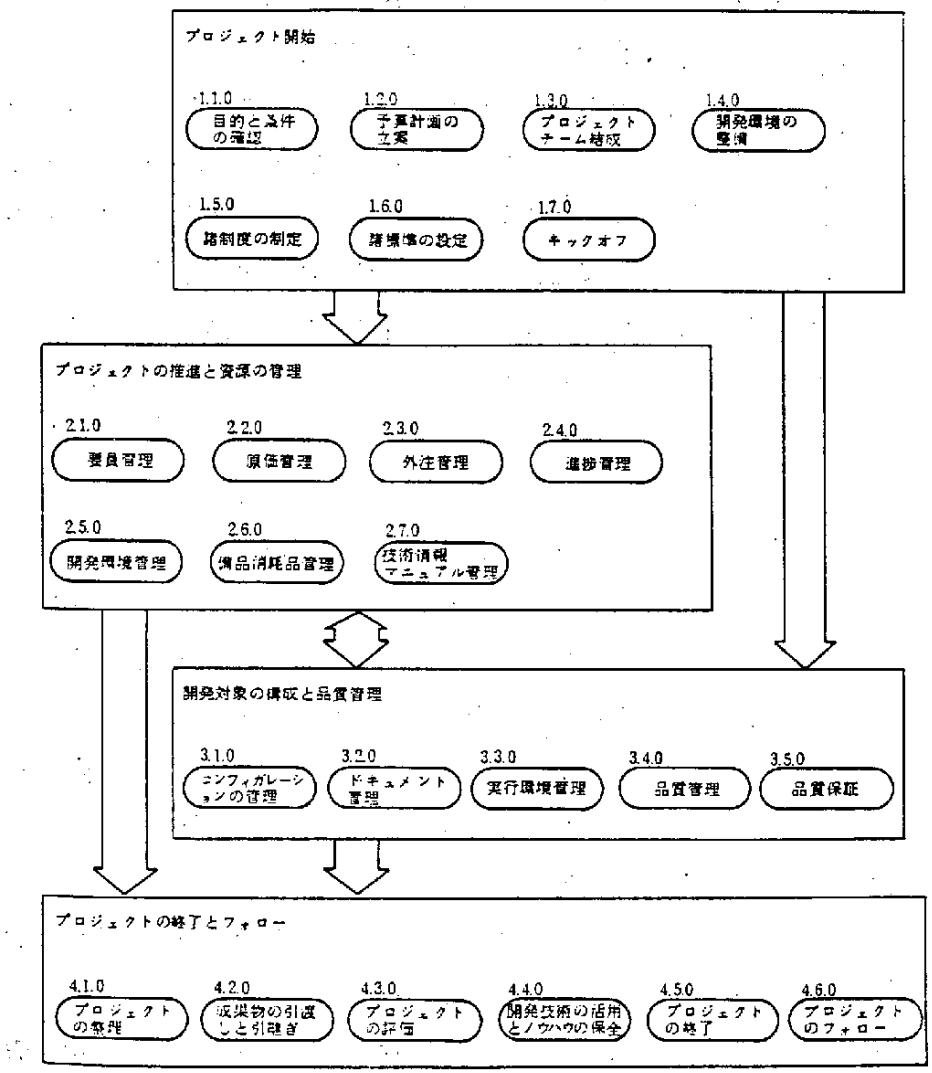
(5) プロジェクト管理

プロジェクト管理の一つとして，進捗管理がある。

まず，システム構築以前に日程を立てることから始まる。あとは時々刻々予定通りに進んでいることを確認しなくてはならない。大幅に遅れたような場合は日程を変更する必要がある。システム構築の場合，テストには多大な日時がかかることを踏まえ，設計，プログラミングは早目に終えてテストに早く入れるよう考慮しなければならない。環境設定は常に実作業より先廻りしなくてはならないので進捗と見合せて遅れないようにしなくてはならない。

次にドキュメントなどの管理がある。システム開発に沿って各種の仕様書や説明書が発生するがそれを体系的に管理し，必要時にすぐに手元に捜し出せることが必要である。また，標準化に沿ったものの流用可能な物の一元管理をしておくことによって，他で流用や，自ら流用することが容易になる。試験項目のチェックや障害状況報告書の管理（バグの吸収），改造，修正箇所の管理を行うべきである。

図表3－1　管理フェーズと管理アクティビティ



3.2　機械化・自動化

人手による作業が中心になっているソフトウェア開発は，開発要求の急増にソフトウェア技術者が追いつかない状況にきており，今後，このギャップは大きな問題になっている。

また，品質の高いソフトウェアの要求，ソフトウェアの開発規模の複雑化，

膨大化が進んでいることもソフトウェア需給ギャップの増加に拍車をかけている。

このような状況にあってソフトウェア技術者を急ぎ養成するため，学校教育のカリキュラムにコンピュータ教育を積極的に取り入れるなどの対策を講じているが，とても間に合うまでにはいきそうにない。

そのため現在並行して進められている対策として大きくクローズアップしてきていることが，ソフトウェア開発の機械化，自動化である。これによって，ソフトウェア開発の生産性が高まれば当然ソフトウェア需給ギャップも縮少されるわけである。しかも，機械化，自動化がねらいとしているところは生産性向上だけでなく，信頼性の高いソフトウェア，保守しやすいソフトウェア，汎用性が高く流用しやすいソフトウェアの開発も目的としている。

既に，このためのツール，技法などが研究レベルでなく製品として出始めてきており，また通産省，情報処理振興事業協会でも昭和６０年度から５年をかけた「ソフトウェア生産工業化システム」プロジェクトを計画している。

3.2.1　ドキュメント作成の機械化

品質・生産性が高く，運用・保守のしやすいソフトウェアを開発するためには，良いドキュメントを作成することが大変重要である。

ドキュメンテーションの機械化，自動化については，単純な作業工程においてのワードプロセッサや端末・パソコンの活用から更にわかりやすい内容でも表現するためのイメージや図，表も扱える機能を利用したドキュメンテーションが求められている。

そして，より高いレベルでのドキュメンテーションとして各種のツールがある。

ドキュメンテーションは，システムライフサイクルの各段階それぞれで重要な作業である。

通常，ドキュメンテーション用ツールは，分析・設計・製造・テスト・

運用・保守といった作業工程を機械化・自動化するツール兼ね備えていることが多い。

ここに，ドキュメンテーション用ツールについて，その用途別に説明する。

(1) ワードプロセッサ，端末

手書き，タイプ，印刷などの工程を省くことによるメリットが大きいが，それよりも，ドキュメントに対する修正や類似した仕様を作る場合にも効果を発揮する。

最近は，図・表・グラフ，さらにはイメージも扱えるようになるに従ってドキュメンテーションには欠かせないツールとなるであろう。

(2) フローチャート自動生成

以前は，ドキュメンテーション用ツールの代表的な存在であったが，最近は，フローチャートそのものがスペース効率が悪いこと，大規模・複雑化するプログラムをフローチャートで表現することの限界，ソースプログラムそのものを表現しているだけともいえることから，あまり活用されていない状況がある。

(3) ドキュメント自動生成機能

ソフトウェアの開発において，設計段階から製造，検査そして運用，保守と進むにつれて，ドキュメンテーションの内容と，実際に稼動しているプログラムとのギャップは広がる傾向にあり，そのギャップを埋めるための努力は相当な工数を必要としている。しかし，この努力を怠ったりすると，将来さらに数倍以上の工数を費やすはめになっています。

ソースプログラムを入力として各種のドキュメントを自動生成してくれる機能への要求は強いものがある。

自動生成されるドキュメントの例としては，以下のような内容である。

① 手続き機能概要一覧表

② ファイルアイテム情報

③ データ明細情報

④ 参照手続き一覧表

⑤ プログラム相関図

⑥ 手続き相関図

⑦ 参照マクロ一覧表

⑧ プログラム／参照マクロ一覧表

⑨ プログラム構成図

⑩ 手続き構成図

⑪ マクロファイルドキュメント

（NEC, COBOL／Sドキュメントエイド）

(4) ドキュメント作成支援・管理・保守機能

調査分析作業，設計作業，プログラミング作業，テスト作業の各段階を自動化するツールが既に幾つか存在しているが，それらのツールではドキュメント形式の標準化と合わせた自動生成機能を有している。

3.2.2 プログラミングの機械化

ソフトウェア開発において，プログラムの製造工程に対する機械化，自動化ツールはその適用レベルが種々である。

① 記述方式を標準化，構造化，高水準化，簡易化することによって機械化，自動化を進める。

② パッケージ化，汎用化などによりプログラムを出来るだけ作らないようにする。

③ 部品化，パターン化されたプログラムを組み合わせたりして作成する。

④ 設計段階で記述された仕様書を入力として，自動的にプログラムを生成する。

現段階にて，プログラム自動生成機能は未だ本格的に実現出来ている状

況にないが，特定の業種や小規模なソフトウェア開発においては実用化されており，近い将来，数多くのツールが発表されてくるものと期待できる。

プログラムを半自動，全自動的に生成する機能としてどのような方法が検討されているかについて紹介したい。

(1) 仕様書からのプログラム生成

設計した内容である仕様書をそのまま入力できることの効果は大きい。当然，仕様書の記述方式も細かく規定されることになる。

この方式でプログラム全てを自動生成するツールは未だ先のことになるが，一部分を生成するツールは既に利用されている。

例えば

① 入力チェック処理

② 出力レイアウトの生成

③ 画面関連の入出力処理の生成

④ 表，グラフ作成処理の生成

⑤ 入出力データ構造からデータ定義やデータ操作処理の生成

⑥ データベースアクセス処理の生成

⑦ ジョブ実行用制御カード

⑧ テスト用プログラム・データ

(2) プリコンパイラによるプログラムの生成

高級言語によるプログラミングから一歩進んでプリコンパイラ方式による記述言語によって，プログラムを生成する。

この記述方式では，人間に見易い形式で記述されている。図イメージやデシジョンテーブル形式の記述がそのまま入力となり，また出力も見易い形式になるため，設計が簡単になり，デバックが格段と容易になるといった効果が得られる。

(3) 部品化，パターン化されたものを作ったプログラムの生成

部分的に完成されたプログラムを組み合わせたり，不足な部分をコー

ディングして補ったりしながらプログラムを作成する方式である。

この方式では，どのような部品を用意するのか，それらの部品をどのように管理するのか，また部品を組み合わせる方法の標準化と機能の充実が重要である。

3.2.3 テストの機械化

ソフトウェアの検証は，プログラミング工程だけでなく，設計段階から既に検証を行いながら作業を進めなければならない。しかし，機械化となるとその中心はやはりプログラム製造，テストの段階が最も効果的であり，ツールなども用意されている。

プログラムテストの機械化について，その内容を紹介する。

(1) テスト項目の自動生成

設計記述やプログラム内容を入力して，テストすべき項目を自動生成する。これによって，検証，確認項目が重複したり，もれたりすることを防ぎ，効率の良いテストが行える。

(2) テスト網羅度測定ツール

プログラムの中で確認された機能がどの程度の割合いであるかを計測するツールである。

このツールを使うことによって，プログラムで実行されていない部分や，逆に実行頻度の高い部分がはっきりするため，きめの細かいテスト作業や，作業状況に合わせた柔軟なテストが可能である。

(3) テスト環境生成ツール

プログラムをテストする際に，インタフェースをとるべきモジュールやプログラムが未完成であるため，満足なテストが出来なかったり，プログラム製造スケジュールに大変注意を払わなければならない事態が生じることが良くある。

このようなことは，また避けられないことであるため，未だ出来てい

ない部分をシュミレートする機能を用意しなければならない。

テスト環境設定ツールでは，種々のテストを可能とする環境を用意することができる。

① 他モジュールの動きをシミュレートする。

② テストプログラムの動作モード（オンライン，TSS，バッチなど）を異なった動作モード下でテスト可能とする。

③ データベースアクセステストにおいて，本番用データベースを実際に更新せずに更新動作テストは可能とする。

④ 時間を要するオンラインテストなどを，バッチ（一括）処理方式によって，大量かつ効率的なテストを可能とする。

⑤ テスト用データベースを本番用データベースから簡単に生成できる。

(4) テスト内容モニタリング機能

テスト処理における，各種のデータの動きや処理の流れを詳かくモニタリングし，分析データとしてリストや端末へ出力できる機能である。

(5) オンライン／端末動作シュミレート機能

オンラインシステムをテストするうえで重要な内容に，時間的要因を加えたテストがある特に処理要求が集中した時のテストや，タイミングを計ったテストを行うことは大切であるが，またこのようなテスト環境を作り出すことが難かしい。

そのためのツールとしてオンラインメッセージを希望する内容・パターンそして時間的要素も加わって生成してくれる機能である。

3.2.4 開発管理の機械化

ソフトウェア開発プロジェクトの管理システムを体系化し標準化することと合わせて，管理ツールも必要である。

ここに具体的なツールと機能を挙げてみる。個々にはよく見かけるプログラムであるが重要なことは，これらが全てつながっていなければ大きな

効果が得られないことであり，その意味でのツールはそれほど多くはない。

ソフトウェア開発プロジェクト管理用ツールとして必要な要件について述べる。

① 管理者にとって，必要な情報の収集・選択・加工が個々にできること。

② 個々の管理者用のデータバンクが簡単に作れること。

③ ツールが管理者に対して管理方法や手順を押しつけるような支援ツールであってはならない。

④ 簡単に，手軽に使えること。

⑤ 演算処理機能が必要である。

⑥ グラフや表操作機能を有していること。

⑦ 情報の収集・伝達を正確かつ迅速にできる機能（通信機能など）があること。

次に具体的な管理用機能を挙げてみる。

(1) 進捗管理機能

工程，サブシステム，グループなどの単位で管理し，計画（予定）と実績の把握を行う。

① 予定実績報告

② 進捗状況報告（個別，総括）

③ プログラム進捗管理表

④ 作業能率評価

⑤ 作業進捗予測，負荷予測

⑥ 負荷予測

(2) 品質管理機能

障害の発生状況と進捗状況さらには過去の実績データや計算値をもとに，バグの発生状況・原因分析さらには品質・生産性を把握する。

① バグ発生曲線

② バグ発生状況（サブシステム，グループ別）

③ バグ措置状況

④ 原因分析表

⑤ 品質・生産性状況・評価

(3) 工数管理機能

所要工数の計画と実績を管理し，グループや担当者の生産性や能力の管理を行う。

(4) 予算管理機能

予算に対して，担当者作業工数，外注費，マシン費他の費用実績を加えて，予算管理情報を提供する。

3.3 再 利 用

ソフトウェアの生産性，品質を向上させる為の有効な手段として，既存ソフトウェアの再利用が最近注目を浴びている。その利点の1つは，再利用した部分について完全に生産活動を省略できることである。もう1つの利点は再利用ソフトウェアの品質が既に検証されていることである。

ソフトウェアの再利用は，ソフトウェア開発の始まった当初からの何らかの形で行われているが，最近特に注目されているのは，ソフトウェアの部品化の考え方である。本節では部品化に焦点を当て，ソフトウェア再利用の実態について報告する。

3.3.1 再利用ソフトウェアの形態

再利用対象とされるソフトウェアにはどのような形態のものがあるか以下に整理する。アプリケーション・パッケージについては別節で触れられるので，ここでは除外する。

(1) 独立プログラム

独立して実行できる汎用的なプログラムを蓄積しておき共同利用する。ソフトウェア開発支援ツールでは多いが，一般業務プログラムでは余り

例がない。

(2) 部　品

それ自体では独立して実行できず，加工されるか，別のプログラムに組み込まれることによって動作するプログラムを部品と呼ぶこととする。

小林要氏（富士通国際研究所）は部品を次のように分類している。

㋑ ブラックボックス型部品

部品内部をブラックボックスとして利用し，その機能が再利用の対象である。たとえば，呼び出し方式のサブルーチンなどがこの分類に該当する。

㋺ パターン型部品

部品内容を利用者に示し，利用者は個々の目的に合わせて加工して用いる。処理パターンが再利用の対象である。日本電気のSTEPSや富士通のPARADIGM などが該当する。

㋩ 手本型部品

ノウハウや定石を手本として示して，利用者にはその考え方を効果的に伝達し，広範な応用を期待する部品である。市販されているアルゴリズム集などが該当する。

また，日本電気のSEA／I．は部品を次のように分類している。

㋥ オープン部品

処理の実態が部品の引用の都度，引用場所にソースプログラムの一部として展開される。利用者は展開された部品を自由に変更することができる。

㋭ クローズ部品

処理の実態は，最初の部品引用時に内部サブルーチンの形でソースプログラムの最後に1回のみ展開される。利用者は展開された部品を変更できない。

㋬ CU部品

処理の実体はコンパイルユニットとして別に存在し，部品の引用場所には外部サブルーチン呼び出しのための命令が記述される。

以上の２つの分類方法では，前者の方がより広範囲な定義であるので，前者の分類に従ってアンケート調査を行った。その回答結果を図表３－２に示す。

独立プログラムの再利用が５１％であるのに対し，ブラックボックス型部品の再利用が７７％と高い率を示しており，各企業ともソフトウェアの部品化に具体的に取り組んでいることが分かる。

図表３－２　再利用ソフトウェアの形態別回答率

再利用ソフトウェアの形態		回答率 % (20 40 60 80)
独立プログラム		51
部品	ブラックボックス型	77
	パターン型	64
	手本型	28

3.3.2　部品化を志向した開発作業形態

部品化を志向したソフトウェア開発作業には３つの形態がある。

(1)　同一システム内での部品化アプローチ

構造化設計はソフトウェアの保守性の向上を目的として提案された技法であるが，その設計技法による開発は必然的にブラックボックス型部品を生み出している。つまり，高い独立性を持つような分割されたモジュールは同一システム内の幾つかの局面で使用することができるものが多く，そのようなモジュールは部品としての役割を持っている。アンケート調査では３６％がこのような部品化アプローチを標準化していると回答している。

また，開発作業に先だって，システム内に頻繁に存在する処理パターンを抜き出して，パターン型部品あるいは手本型部品を作成する方法も

システム内の部品化の一形態である。

(2) 部品化ソフトウェアアプローチ

これは小林要氏により後述の(3)のアプローチと共に提唱されている概念である。

この方法は，構造化設計を更に高度に押し進め，ソフトウェアを目的とする分野において一般性の高い部品の集りとして構築しようとする考え方である。

このようにしておくことにより，システムの進化に対して容易に対応することができる。また，顧客の要望に応じたソフトウェアを低コスト高品質で生産することができる。

このようなアプローチは，銀行システム，生産管理システム，財務会計システム，情報検索システムなど機能の枠組の明確な分野で有効である。

(3) ソフトウェア部品ライブラリアプローチ

以上の2つの例は一つのソフトウェアを部品に分割して生産しようとするアプローチである。もう一つのアプローチとして，一般性の高い部品を蓄積しておき，後の生産活動でそれらの部品を再利用する方法がある。これをソフトウェア部品ライブラリアプローチと呼ぶ。

本節のテーマ「再利用」にとっては，本アプローチが中心課題となる。

3.3.3 部品検索

部品ライブラリアプローチをとるに当って，まず検討しなければならないことは求める部品の検索手段である。検索のためには，部品の分類コードの標準化や検索ツールが重要なポイントとなる。

(1) 部品分類コードについて

アンケート調査では22%の企業が部品分類コードを標準化していると回答している。その分類方法は図表3－3に示すとおりである。メーカの販売している部品ライブラリの分類事例を図表3－4，図表3－5および図表3－6に示す。

図表３－３　部品分類方法と回答率

部品分類方法	回答率
業別別（給与，人事，財務，販売など）	79%
機能別（テーブル処理，コード変換，エラーチェックなど）	79%
処理パターン別（媒体変更，照合，更新，作表など）	68%
その他	9%

図表３－４
STEPS（日本電気，パターン型部品）部品分類

パターン分類	パターン数
媒体変換	1
分配	1
照合	2
更新	2
報告書作成	2
媒体変換・照合	1
更新・照合	3

図表３－５
PARADIGM（富士通，パターン型部品）の部品分類

パターン分類	パターン数
入力チェック	2
照合	3
更新	3
レポート	1

図表３－６　SEA／I（日本電気，ブラックボックス型部品）の部品分類

パターン分類	パターン数	パターン分類	パターン数
標準ファイル入出力	10	時間処理	3
テーブル処理	13	コード変換	11
フィールドチェック	3	計算	4
フィールド編集	15	単位変換	4
日付処理	17	分類他	8

(2) 部品検索ツールについて

アンケート調査によると部品検索ツールを使用している企業はわずか4％である。８０％に近い企業が部品の再利用を図っていると回答しているのに比べて，検索ツールの利用ははなはだ不十分と言える。部品再利用を図っている企業も，検索ツールを必要とする程には部品は蓄積されていないからかも知れない。

一般的に，検索用部品ライブラリには次のような情報が必要である。

① 検索キーとなる項目

部品コード，部品名称，分類コード，キーワード，登録者名，他

② 部品の属性を示す項目

登録日，言語，部品形態，部品サイズ，処理能力，他

③ 部品の機能を示す項目

使用方法，インタフェースデータ，入出力データ，使用ファイル，機能

3.3.4 再利用ツール

部品再利用に関するツールの事例を紹介する。

(1) 部品作成管理ツール

ＭＥＴＭＮＧ（日本電気）はプログラム部品の作成と管理を支援するツールであり，次の機能を持っている。

(a) 既存プログラムからの部品抽出機能

ソースプログラムを解析しプログラム構造図リストとデータ遷移トレースリストを出力する。これらの分析結果をもとに，部品化したい手続き名やデータ名を指定すると影響の及ぶ範囲がレベル付けされる。レベル番号を指定すると，その範囲のプログラム記述が部品として抽出される。抽出される部品はＣＯＢＯＬにおけるセクションとそのデータ部である。

(b) 部品作成機能

キーワードの自動挿入や日本語コード変換などの入力支援機能を用いて効率良く部品を作成できる。処理に複数個の変化が存在する場合，変化に該当する部分をマクロ記述することができる。

(c) 部品管理機能

階層化された部品メニューに従って必要な部品を検索できる。また，各階層に対して検索キーを設定できる。

(2) パターン型部品のカストマイズツール

ＳＥＡ／Ｉにはパターン型部品をカストマイズするための機能として次のようなツールが準備されている。

(a) 利用者はパターンが持つどの機能をカストマイズするかという情報を与えるだけで良い。パターン中に分散している関連個所は自動的に修正される。

(b) パターン記述の為のベース言語（ＣＯＢＯＬなど）の上に，パターンカストマイズ制御言語（ＰＣＬ）が用意されており，汎用性の高いパターンを容易に作成することができる。

(c) ＰＣＬの制御記述に従ってカストマイズ情報を対話的に入力する。ファイルの属性などの定型情報は表形式画面から入力し，処理部品などの非定型情報は行モードプロンプティングにより対話的に入力する。

(3) 部品自動組込ツール

エクスパンダ（日本電気）は部品をソースプログラムの中に組み込むためのツールである。

(a) オープン部品，クローズ部品及びＣＵ部品を統一的な呼び出し文によって組込むことができる。

(b) 部品は部品記述言語（ＰＤＬ）を用いて記述され，次の４つの記述部から構成される。

① 検索情報記述部

部品の機能，利用方法，呼び出し形式などの情報を記述する。

② パラメータ記述部

オープン部品の場合は展開形を作成するための展開時変数を宣言する。クローズ部品及びCU部品の場合はCOBOLのデータ宣言の形式で処理手続きのパラメータを記述する。

③ データ宣言記述部

部品内で必要とする作業データを記述する。

④ 処理本体記述部

プログラミング言語により処理を記述する。

(c) エクスパンダは与えられたパラメータの値によって部品を適切な位置に分配する。オープン部品では，パラメータ，展開変数，展開制御文（%IF，%DO等）により展開テキストを特殊化できる。

(4) 仕様部品からのプログラム合成システムMOTHER SYSTEM（IPA）は端末の日本語メニュー画面を介して外部仕様部品からプログラムを合成する実験システムである。

(a) 抽出，集計，分配，併合，照合，報告書データ作成のみの処理形態のプログラムを対象とする。

(b) 一群の入力レコード値が全てそろった時点で一群の出力レコード値を演算する方式のアルゴリズムのひな形があり，そのひな形にプログラム仕様をはめ込む。

(c) プログラム仕様は次の3宣言よりなる。

① プログラム宣言

プログラム名，主キー項目名，主データ，内部データ，ファイル属性などを宣言する。

② データ宣言

データ内各項目のレベル番号，項目名称，属性，初期値，値域，値域外処置を宣言する。

③ 手続き宣言

出力データと内部データの各項目値の演算を指定する。

(d) データ宣言及び手続き宣言は仕様部品として予め登録が可能である。

(e) 端末画面より仕様部品を検索し，会話的にプログラム仕様を作成できる。

(f) プログラム仕様から自動的にプログラムが合成される。合成されたプログラムの手続き部は，制御部，入力部，演算部，出力部の4部から構成される。

(5) 特定分野におけるプログラム自動生成ツール

対象分野を特定すれば要件定義からのプログラム自動生成も可能である。バンキングシステムを対象としたそのようなツールとしてBAGLES（富士通）がある。

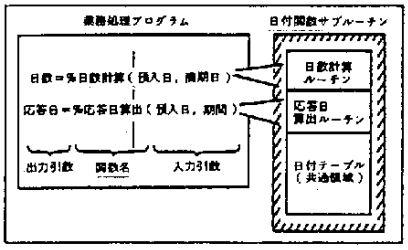
(a) 日本語で記述されたテーブル形式の論理設計仕様書からCOBOLソースプログラムを自動生成する。

(b) 条件名と論理式で条件を階層化し，処理は全て数式表現としている。

(c) 銀行業務に於ける定型機能をブラックボックス型部品として持っており，日本語による関数表現で呼び出すことができる。

(d) 共通領域の持ち回りを避け，同一の共通領域を使用する機能をまとめて一つの部品としている。この方法により，共通領域を利用者から隠ぺいすることができる。（図表3－7参照）

図表3−7　BAGLESにおける部品化



3.3.5　部品化と親和性の高い言語

部品化の考え方が取り込まれている言語の例としてSmalltalkとAdaについて紹介する。

(1)　Smalltalk（アメリカゼロックス社）はオブジェクト指向プログラミング言語の代表的な事例である。

(a)　プログラムはオブジェクトを単位として組み立てられる。オブジェクトとはデータと，そのデータに対する操作をひとまとめにしたものである。これにより，データ構造に変更があっても，それは他のオブジェクトに波及しない。

(b)　オブジェクトはデータに代表される。オブジェクトを動作させるときは，データにメッセージを送るという形で表現する。それによりメッセージに対応した手続き（メソッド）が動作する。

(c)　同じ性質を持つオブジェクトは一つのグループにまとめられる。これをクラスと言う。クラスから作られるオブジェクトをインスタンスと言う。

(d)　クラスは更に階層化することができる。上位のクラスをスーパクラスという。クラスはスーパクラスの性質を受け継ぐことができる。

(2)　Ada

Ada（米国防総省）の重要な特徴は関連するプログラムやデータがパッケージという単位にモジュール化されていることである。

(a)　パッケージは仕様部分と本体の2つの部分に分けられる。仕様部分にはデータとそれに対する演算が定義される。本体には演算がデータを操作するためのアルゴリズムが含まれる。

(b)　仕様部分と本体は別々にコンパイルすることができる。これにより，設計者はアルゴリズムに関係なく仕様部分を設計できる。プログラマはうっかり仕様部分を変えてしまうことなくアルゴリズムを変更できる。

(c) 言語の中のデータは厳重に型が決まっている。コンパイラ，別々のファイルにあるサブプログラムに対しても同じデータに互換性のない属性を設定しないかチェックする。

(d) Adaのプログラムは完全に移植できるようになっている。ユーザは新しいアプリケーションプログラムを作るとき，どのようなハードウェアで作られてきたパッケージでも自分のプログラムに組み込むことができる。

3.4 ソフトウェア・パッケージの活用

本節では，ソフトウェア開発効率化の方策の1つとして，ソフトウェア・パッケージの活用を取り上げ，その市場，利用状況，導入時の留意点について述べる。

3.4.1 ソフトウェア・パッケージの市場

(1) 日米の動向

ソフトウェアパッケージは，プログラム・プロダクトとも呼ばれるソフトウェアの"既製品"であり，その活用は生産性とニーズのギャップを埋める有効な方策であるとして注目を集めている。

パッケージを大きく分けると，システム・プログラム系とアプリケーション・プログラム系の2つになる。システム・プログラム系とは，ハードウェアを動作させるのに必要な制御プログラム，通信制御プログラム，及びプログラム開発や運転を支援するシステム開発・運用支援プログラムなどを言う。また，アプリケーション・プログラム系とは，制御プログラムの下でユーザの要求に応じたデータ処理を行うもので，金融，流通，製造などの業種毎に異なる業種別プログラム，及び科学技術計算，経営計算，経理会計等業種によらない業種間プログラムとから成る。

米国においては，１９６９年にＩＢＭ社がハードウェア価格とソフト

ウェア価格を分離したいわゆるアンバンドリングを実施して以来，パッケージの開発が盛んになった。その後，１９７０年代後半になってハードウェア価格が低下する一方で，人件費をはじめとするソフトウェア開発価格が高騰し，いわゆる"ソフトウェア危機"が現実のものになるにつれて，パッケージの市場が急速にふくらむこととなった。

米国における情報処理産業の市場規模は１９８２年に26,430百万ドルに達しており，そのうち２０％にあたる5,300百万ドルをソフトウェア・パッケージが占めている。その内訳は５３％がアプリケーション・プログラム系，４７％がシステム・プログラム系となっている。また，パッケージ市場の年間伸び率は４０％を超えており，今後も一層大きな市場になっていくと思われる。その市場をめぐってコンピュータ・メーカだけでなく，多くのソフトウェア・ハウスが競ってパッケージの開発をしており，上位１０社の売上げ合計でも全体の約１０％を占めるに過ぎないのが現状である。

一方，わが国においては，昭和５７年度における情報処理産業全体の売上高に占めるパッケージの割合はわずかに２％弱にすぎず，米国に比べるとその市場ははるかに小さい。その原因としては

(a) ソフトウェアに対してきめ細かなサービスを要求するので，パッケージを導入する場合でもかなりの手入れが要る。

(b) ソフトウェアの価値に対して正当な評価をするという考え方が確立されておらず，アンバンドリングも未だ徹底されていない。

(c) 米国におけるＩＢＭのような圧倒的なシニアを持ったコンピュータ・メーカが無いので，パッケージを大量に販売するには複数の機種で動くものを開発しなければならない。

などがあげられている。

しかし，わが国においてもユーザのパッケージに対する考え方，取り組み方が徐々に変わってきており，非常に積極的になってきた。売上高

の伸び率も年間２５％程度と情報処理産業全体の市場の伸び率１８％をかなり上回っている。

今回のアンケート調査においても，市販パッケージの購入経験の有無について，１５７社のうち６６％に当たる１０３社が購入したことありと答えている。また，市販パッケージの購入方針についても図表３－８のような結果がでており，パッケージの活用を重点的に考えているところが多いことがわかる。

反面，１５７社のうち３４％の５４社は市販パッケージの購入をしたことがないと答えている。その理由としてあげられているものを整理すると図表３－９のようになる。この表からわかるように，半数近くの会社が「自社の目的にフィットしたパッケージがない」としており，今後

図表３－８　市販パッケージの購入方針

順位	方　　針	件　数
1	自社開発より有利なものがあれば購入したい。	９５
2	使用目的に合う良いものがあれば積極的に購入したい。	７８
3	自社開発を優先して考えたい。	１６
4	多少の不便はあってもできるだけ市販パッケージを活用し，自社開発量を減らしたい。	１５
5	市購パッケージの購入はしない。	２

パッケージの品ぞろえを充実する余地が未だ残っていること，あるいは導入する側も既製品を使うという意識をもっと徹底する余地が残っていることを示していると言えよう。また１／３の会社が「機能・性能がよくわからない」「サービス・保守に対して不安」としており，パッケージ提供側のユーザに対する姿勢・体制にも改善の余地があることを示している。

(2) 主要製品及びサプライヤ

わが国では，１９７９年からパッケージの流通を目的として「汎用ソフトウェア開発準備金制度」が税制上の特別措置として設けられ，この制度の対象となるパッケージは情報処理振興事業協会（ＩＰＡ）に登録されることになっている。登録数は１９７９年度末６４７，１９８０年度末１４６３，１９８１年度末２２９７，１９８２年度末２９３２と順調に伸びている。内訳としては，言語プロセッサが最も多く６９１本（２４％），ソフトウェア開発・管理支援プログラム５１０本（１７％）などとなっている。

わが国のパッケージは，コンピュータ・メーカが開発したもの，ソフトウェア・ハウスが開発したもの，ユーザが開発しソフトウェア・ハウスが汎用化したもの，米国を中心とした海外のパッケージを購入し，場合によっては日本向けに改良したものなどがある。今回のアンケート調査結果も含め，わが国で比較的多く流通しているパッケージとそのサプライヤを図表３－１０に示す。

図表３－９　購入に消極的であった理由
（回答数　５４社）

理　　由	会社数	回答数５４社に対する割合（%） 0 10 20 30 40 50 60 70 80 90 100
使用目的に合うものがない	２９	54%
自社システムへの適応が困難	２５	46%
機能・性能がよく判らず不案	１９	35%
サービス・保守に対して不安	１８	33%
価格が希望に合わない	９	17%
他ユーザの評価がよくない	０	
その他	３	6%

図表３－１０　主要パッケージとサプライヤ
（出典；ソフトウェア流通２２）

分類		販売実績金額（累積）
システム系	ADABAS（リレーショナル型データベース管理システム）〔ソフトウェア・エージー〕	50億～55億円未満
	EASYTRIEVE（レポート作成用簡易言語）〔アシスト〕	35億～40億円未満
	MARK Ⅳ（アプリケーション開発システム）〔日本インフォマティクスゼネラル〕 FOCUS（第４世代の言語）〔フォーカス〕	25億～30億円未満
	PANVALET（ソース・プログラム管理プログラム）〔アシスト〕 UFO（CICS オンライン・アプリケーション開発ツール）〔アシスト〕 A-AUTO（コンピュータ自動運用システム）〔ソフトウェア・エージー〕 JASPOL（事務処理用簡易言語）〔日本システムサイエンス〕	10億～15億円未満
	NATURAL（第４世代プログラミング言語）〔ソフトウェア・エージー〕	８億～９億円未満
	COM-PLETE（３次元オンライン・モニタ）〔ソフトウェア・エージー〕	４億～５億円未満
	DATAMANAGER（データ・ディクショナリ）〔コンピュータアプリケーションズ〕	３億～４億円未満
	DMS/OS（ダスドスペース総合管理システム）〔シーイーシー〕 PRO/TEST（プログラム・テスティング・ツール）	１億～２億円未満
アプリケーション系	SAS（スーパーアプリケーションシステム）〔アシスト〕 CREATE-SYSTEM（クリエイト・システム）〔図形処理技術研究所〕	10億～11億円未満
	WAPS（厚生年金基金汎用システム）〔日本電子計算〕	４億～５億円未満
	FEMIS（会話型有限要素法データ生成システム）〔東洋情報システム〕 SDM（構造解析シミュレーション・パッケージ）〔日本システム技術〕	２億～３億円未満
	ADAM（会話型情報設計システム）〔東洋情報システム〕 FASPAC-I（多数企業処理型総合固定資産管理システム）〔日本ニム・アイ・エス〕 CASTER（企業財務分析診断システム）〔三井情報開発〕	１億～２億円未満
	JUSE-L-GIFS（化学プロセス・シミュレータ）〔日本科学技術研修所〕	5000万～１億円未満

3.4.2　ソフトウェア・パッケージの利用状況

(1)　ソフトウェア・パッケージに関する情報入手方法

ソフトウェア・パッケージに関する情報の入手調査によると，最も多いのが「コンピュータ導入時にメーカーから勧められた」（38.3％）というものであり，続いて「セールスマンの売り込み」（17％），「講演会・セミナー等」（14％）となっている。これをユーザの資本金別に見ると，資本金が大きくなるに従って「メーカに勧められた」というのが急激に減少し，それに代わって「新聞，雑誌，年鑑」が多くなっていく。（図表3－11）。

(2)　購入動機，理由

ソフトウェア・パッケージはソフトウェアの既製品であり，その購入動機は一般的に

ア．自社で開発するより購入した方が割安である。

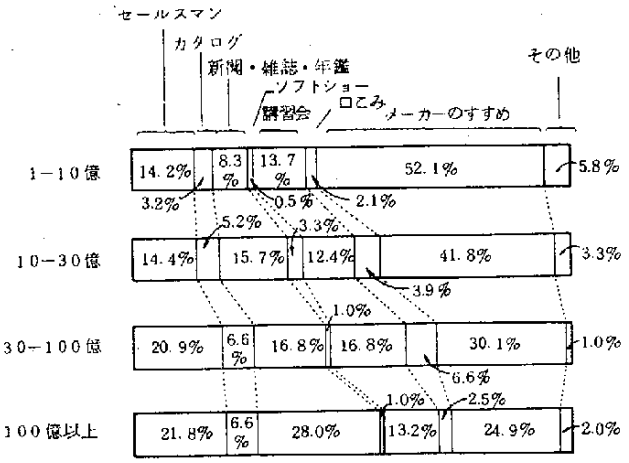
イ．新規に開発する期間がない。

ウ．自社開発では工数が不足する。

エ．自社開発より機能，性能等が良い。

といったものがある。いずれも自社開発，個別開発の限界を認識し，それに対する手段としてソフトウェア・パッケージを活用するという基本

図表3－11　ソフトウェア・パッケージに関する情報入手方法
（出典；コンピュータ白書　'83）



的動機に基づくものである。

今回のアンケート調査では，実際にパッケージを購入した会社に導入の動機，理由を上記ア～エから選択回答してもらった。その結果は図表３－１２のとおりである。この表から見ると，どの種類のパッケージについても「自社開発より安い」，「自社開発より機能，性能等が良い」という動機が多く，イやウのような止むを得ずという側面を持った動機は少ない。パッケージの導入がより積極的かつ自発的に行われるようになってきたことの現われであると考えられる。

(3) 導入後の評価

米国においては，アンバンドリング直後の一時期，ソフトウェア・パッケージが粗製乱造されたこともあって評価は一般的にあまり高くなく，その普及も遅々として進まなかった。しかし，前述したように７０年代後半に入ってソフトウェア危機が現実のものとなり，パッケージへの期待，ニーズが高まるにつれて，製品としても良いものが続々と登場して

図表３－１２　パッケージの購入動機，理由

種類	購入会社数	購入動機理由			
		ア	イ	ウ	エ
運用系／評価系	47	28 (60%)	10 (21%)	9 (19%)	21 (45%)
データベース	48	16 (33%)	9 (19%)	7 (15%)	29 (60%)
データ通信	28	12 (43%)	6 (21%)	6 (21%)	17 (61%)
言語	49	18 (37%)	6 (12%)	11 (22%)	24 (49%)
ソフトウェア開発支援	44	16 (36%)	7 (16%)	10 (23%)	21 (48%)
データハンドリング	23	9 (39%)	2 (9%)	3 (13%)	10 (43%)
科学技術計算	33	20 (61%)	4 (12%)	5 (15%)	18 (55%)
事務計算	14	11 (79%)	7 (50%)	2 (14%)	5 (36%)
合計	286	130 (45%)	51 (18%)	53 (19%)	145 (51%)

〈購入動機，理由〉

ア．自社開発より安い。
イ．開発期間がなかった。
ウ．自社開発の工数不足。
エ．自社開発より機能，性能等が良い。

（注） 購入動機，理由は複数選択方式
（ ）内は当該動機，理由をあげた会社の割合

ユーザに高い評価を得始め，その後の急速な普及，発展へとつながっていった。

一方，わが国においても当初はサプライヤ，ユーザ共に"マニュアルの未整備"，"導入準備の不足"といった不備な点が見られ，「動かないパッケージ」ということで問題にされたこともあった。しかし，近年は米国におけると同様に，ソフトウェア開発の高度化，効率化の要請が高まると共に良質のパッケージが登場し，その評価が高まりつつある。

今回のアンケート調査では，実際に購入したパッケージの評価として

図表３－１３　購入パッケージの評価

種類	購入会社数	評価			
		A	B	C	D
運用系／評価系	47	19 (40%)	19 (40%)	8 (17%)	0
データベース	48	12 (25%)	24 (50%)	15 (31%)	2 (4%)
データ通信	28	10 (36%)	17 (61%)	9 (32%)	0
言語	49	25 (51%)	20 (41%)	8 (16%)	4 (8%)
ソフトウェア開発支援	44	13 (30%)	25 (57%)	15 (34%)	0
データハンドリング	23	7 (30%)	11 (48%)	5 (22%)	0
科学技術計算	33	6 (18%)	19 (58%)	9 (27%)	0
事務計算	14	1 (7%)	8 (57%)	11 (79%)	1 (7%)
合計	286	93 (33%)	143 (50%)	80 (28%)	7 (2%)

〈評　価〉

A：大変良い
B：良　い
C：一応評価できる
D：期待はずれ

（注）　複数のパッケージを有している会社で，2つ以上の評価を選択しているところもある。
（　）内は当該評価をしている会社の割合

A．大変良い

B．良　い

C．一応評価できる

D．期待はずれ

という評価項目の中から選択回答してもらった。その結果は図表3－13のとおりである。表からわかるように，約4分の3の意見が「大変良い」「良い」となっており，期待はずれというのは約2％であった。大半は満足できるものの，まだ一部には改善の余地があるということであろう。

また，パッケージ導入後の“満足点”，“不満足点”に関するある調査によると，満足した点としては，「機能が期待通り」，「生産性の向上に役立つ」，「効果が期待どおり」といった点が評価され，不満足な点としては，「マニュアルがわかりにくい」，「コンピュータ・リソースが多くいる」，「機能変更に対する要望に応じない」などが指摘されている（図表3－14）。今後のパッケージについては，現状満足と評価されている項目はより充実し，現状不満とされている項目は改善を図っていくことが一層の普及・発展のキーとなるのであろう。

3.4.3　導入時の留意点

(1)　機能，性能の留意点

パッケージは既製品であり，既に機能，性能は固まっているため，次のような点に留意する必要がある。

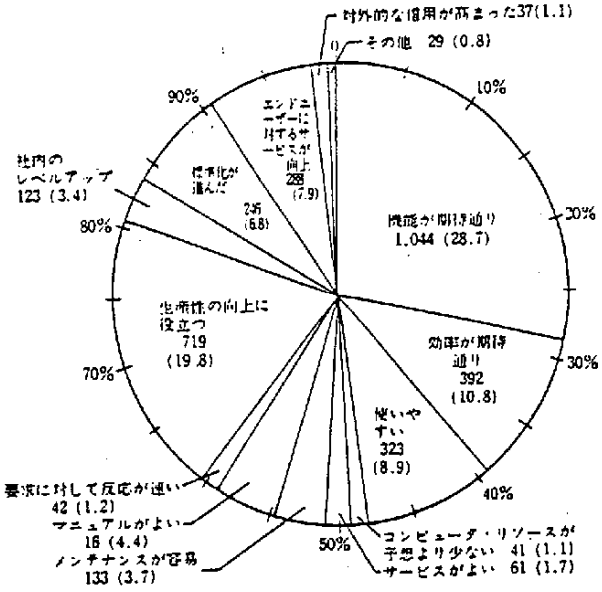
① 現在使用しているコンピュータ上での動作の可否。また，所要メモリ容量，ディスク台数などの確認

② 入出力仕様，処理内容が望んでいるものに合致していること。但し，考えている全ての条件を満足するものを求めるのは無理であり，ある程度の妥協も必要である。

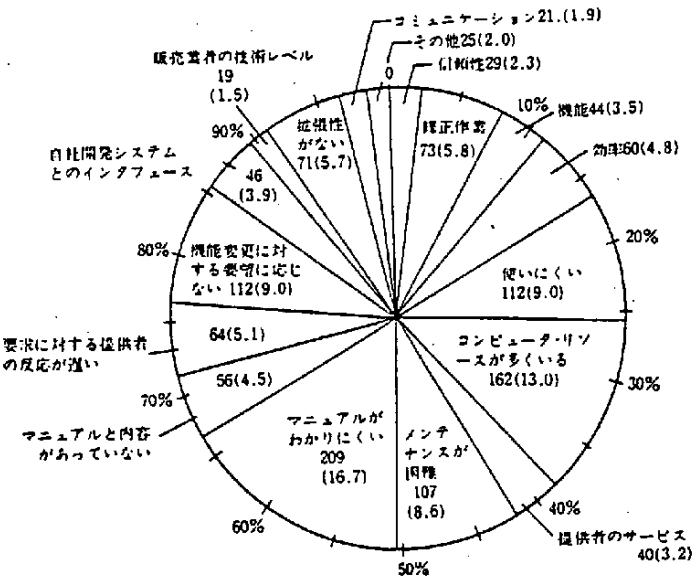
③ 現在のニーズを満足するだけでなく，将来必要と予想される機能，

図表３－１４ パッケージ導入後のユーザの評価
（出典；コンピュータ白書 '83）

(a) 導入後満足だった事項



(b) 導入後不満足だった事項



性能に対応できる拡張性を有していること。

④ パッケージ導入以前のデータ・ファイルを新しい形式または媒体に変換する必要性の有無。

(2) ドキュメントの評価

パッケージを使用する場合には，開発側の直接的なサポートは望めず，運転上の技術情報などはドキュメントに頼らざるを得ない。従来より，パッケージの場合にはこうしたドキュメントの不備が指摘されるケースも多いため，導入時にはドキュメントの内容を十分にチェックする必要がある。

(3) 保守条件の確認

パッケージは既製品であり，不特定多数のユーザが使用しているため1ユーザだけの要求で機能追加などが行われているわけははない。しかし，多くのユーザに使用され寿命の長いパッケージは，ユーザの声を常時収集し，共通的な要望から順次満たしバージョンアップしていくなど保守体

制をしっかり確立しているものが多い。パッケージ導入の際には，導入パッケージをできるだけ有効に活用していくためにも保守体制を十分チェックしておく必要がある。

また，パッケージ導入後に発見されたバグに対する修正条件の明確化も必須である。

(4) 教育・訓練内容の確認

パッケージの持つ機能を早急に引き出し，有効に活用していくためには，導入時の教育訓練が大変重要である。導入に際しては，教育・訓練の内容，期間，費用の確認をしておく必要がある。

(5) 契約条項の確認

ソフトウェア導入時には提供側との間で契約が交わされるが，その内容に関しては十分なチェックが必要である。

導入時の教育・訓練などのサービス，保守体制と内容・費用，添付ドキュメントの内容や部数，使用場所に関する制限事項，バージョンアップ版への取替の可否及び費用などについて確認し，履行上問題が発生しそうな条項については削除・訂正などを図る必要がある。

(6) 利用状況の調査

パッケージについては，利用者数を調べることによって，その製品の出来工合，品質などをある程度推察することができる。多くのユーザに使用されていればそれだけ洗練され，バグ等も少なくなっているはずである。また，実際にユーザの意見を聞くことにより，更に正確な評価をすることも有効である。

以上がパッケージ導入時に留意すべき一般的な事項であるが，実際にユーザがパッケージ導入後に不満を感じている事項に関する調査結果を見ると，図表３－１４のとおりであり，「マニュアルのわかりにくさ」，「コンピュータ・リソースがたくさん必要」，「使いにくい」，「機能変更に対する要

望に応じない」という指摘が多い。これらの項目については，今後提供側も導入側も一層考慮を払っていく必要であろう。

3.5 組織化

開発作業の効率化の為には，固有技術の高度化だけでなく，組織的に対応も重要である。本節では，ソフトウェアに於ける分業の考え方，開発パワーの増大，第三者により品質保証等について述べる。

3.5.1 開発作業の分業化

小林要氏はソフトウェア開発作業の分業化について以下のような述べている。

『ソフトウェア開発は知的活動である。組織的な知的活動を効果的に推進するには，知的労働者の能力開発が不可欠であるが，能力開発には限界がある。従って，適正な負担のもとに仕事を分担して組織的効果をねらう必要がある。

ソフトウェア開発の分業体制には2つの形態がある。一つには空間分業である。即ち，システムをサブシステム，プログラム，モジュールと階層的に分割して，それぞれを独立したグループ又は個人が分担して開発する方法である。もう一つは時間分業である。これは，開発工程によって担当者を分けて開発を推進する方法である。

ハードウェアの世界では設計と製造の区別が明確であると共に，全ての情報を図面で伝えることができる。しかし，ソフトウェアの場合は設計と製造の区別が不明確であり，全ての情報を正確に文書で伝達するには大変な労力を必要とする。従って，ソフトウェア開発のような知的活動の分業の場合は空間分業が重要である。』

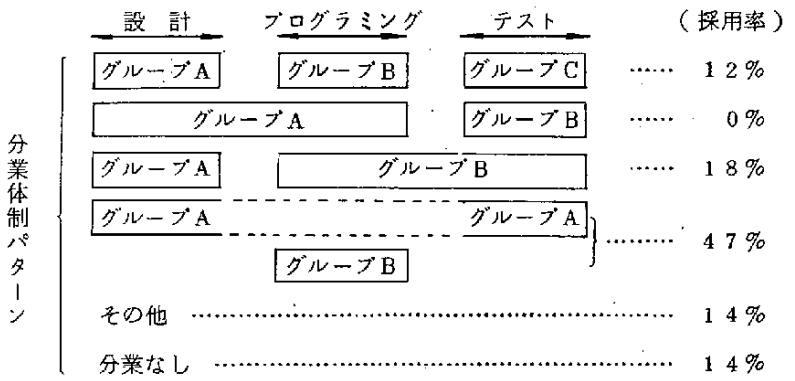
小林要氏の述べる空間分業は既に定着化しているのではないかと考えられる。更に組織的効果を上げるには効果的な時間分業を推進する必要があ

る。このような考えから各企業の時間分業体制とその目的及び問題点についてアンケート調査を行った。

(1) 分業の形態

分業体制についてのアンケート調査結果を図表３－１５に示す。本調査結果によると設計・テストとプログラミングの２グループに分けた体制をとっている企業が圧到的に多いことが分る。テストを独立グループにしているという回答が一つもないことは注目すべきデータである。

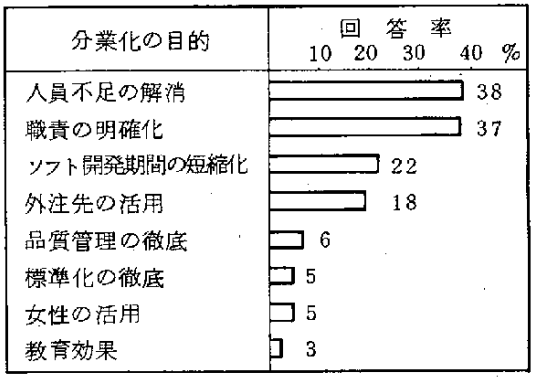
図表３－１５　分業体制の実態



(2) 分業化の目的

分業化の目的に関するアンケート調査結果を図表３－１６に示す。

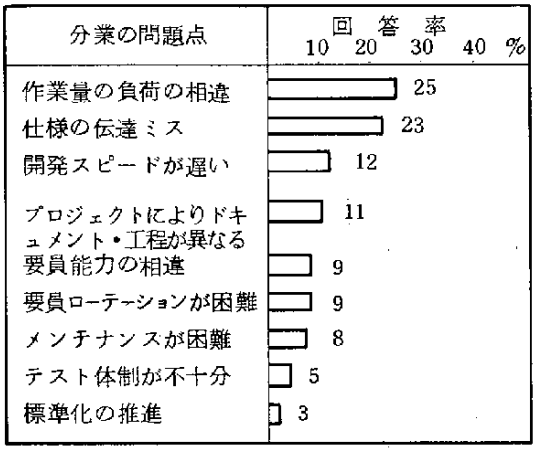
図表３－１６　分業化の目的



(3) 分業の問題点

分業の問題点に関するアンケート調査結果を図表３－１７に示す。

図表３－１７　分業の問題点



3.5.2　開発パワーの増大

コンピュータ部門のソフトウェア開発のバックログが多くの企業で深刻な問題となっている。日経コンピュータの調査では６０％の企業が１年以上のバックログを抱えていると回答している。

これらのバックログ解消のためにはアプリケーションパッケージの購入なども有効な手段であるが，それらが利用できる分野は極く限られており，大部分を内部開発に頼らざるを得ないのが現状である。

そこで，各社はどのような手段で開発パワーの増大を図ろうとしているのか調査を行った。以下，アンケート調査結果について簡単に述べる。

(1) 外注利用状況

総作業の工数のうちの派遣外注，作業請負外注の利用率の分布を図表３－１８に示す。約７０％の企業が外注を利用している。しかし，８０％の企業が外注依存率３０％以下であり，それ程外注依存率が高くないことが分る。

図表３－18　外注利用状況

作業形態＼利用率	０％	～１０％	～２０％	～３０％	～５０％	～１００％
派遣外注	３０％	１８％	１９％	１５％	１１％	７％
作業請負外注	３４％	３２％	８％	８％	９％	９％

(2)　女子プログラマ採用状況

女子プログラマの採用状況を図表３－１９に示す。８２％の企業が女子プログラマを採用しているが，女子プログラマへの依存率はそれ程高くない。

また，今後女子プログラマの採用推進を考えているかとの質問に対し，６６％が推進を考えていると回答している。このデータは女子プログラマの評価が定着していることを示していると考えられる。

図表３－19　女子プログラマ採用状況

採　用　率	０％	～１０％	～２０％	～３０％	～５０％	～１００％
回　答　率	１８％	３０％	２４％	１５％	８％	５％

(3)　在宅勤務制度

アンケート調査結果では，現在プログラマの在宅勤務制度があると回答した企業が３％（４件），将来計画していると回答した企業が９％（１４件）である。まだ多いとは言い難いが除々に普及の兆しが見えている。

また，在宅勤務の形態についての調査では図表３－２０に示す回答があった。自社退職者を中心に在宅勤務を推進する企業が多いことが特徴的である。

図表３－２０　在宅勤務形態別回答数

・作業範囲……	[6] プログラム設計	[17] コーディング	[12] 単体テスト	[0] その他（　）
・対 象 者……	[16] 自社退職者	[5] 契約社員	[1] アルバイト	[0] その他（　）
・賃　　金……	[0] ノルマ制	[17] 出来高制	[1] 固定給	[0] その他（　）
・作業環境……	[13] TSS端末利用	[4] パソコン利用	[0] ファクシミリ利用	[3] その他（　）
・そ の 他……	[　]（　）			

(4)　エンドユーザ開発の推進

調査では５７％の企業がエンドユーザ開発が推進テーマとなっていると回答している。また，日経コンピュータの調査でも，バックログ解消のための対策として，①社内ユーザによるシステム開発の推進（52%），②社内ユーザ向けソフトウェアパッケージの導入（３９％），③データベースの利用部門への開放（３８％）と，エンドユーザ開発に関連する対策が上位を占めている。

一つの事例として，花王石鹸から，コンピュータ室と現場との間で積極的に人事ローテーションを行いエンドユーザの開発能力の向上に成功していると報告されている。

3.5.3　独立した組織による品質保証と監査

ソフトウェア品質を客観的に保証することは難しく，多くの企業では，それは開発当時者の手に委ねられている。調査では独立した品質保証組織があると回答した企業は６％である。その組織の役割は図表３－２１のとおりである。

図表３－２１
品質保証組織の役割
と回答数

検収について……	7
システム監査について……	4
品質向上について……	2
運用効率について……	1
標準化について……	1

本節では，品質保証組織の役割とシステム監査の概要について述べる。

(1) 品質保証組織の役割

開発部門から独立した品質保証組織の行う検査機能は，大きく次の３つに分類することができる。

① 報告書検査

各工程の完了時にドキュメント枚数，レビュー指摘事項数，プログラムステップ数，GOTO 文比率，テスト項目数，バグ数等を報告させて，それらのデータが基準内であるかどうか検査する。異常が発見された場合は，担当者からのヒアリング，現物調査等を行う。

② ドキュメント検査

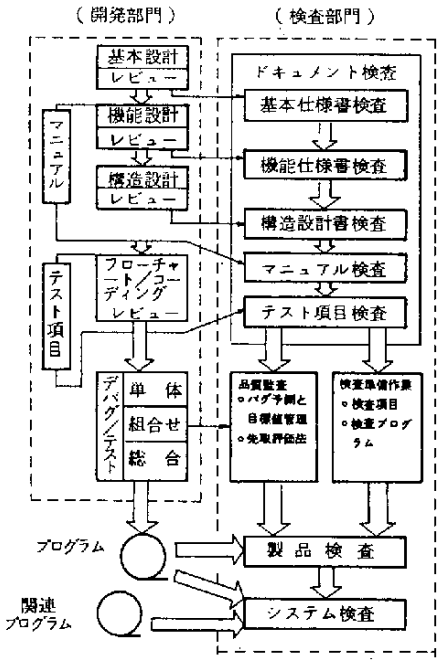
各工程の完了時に，設計ドキュメント，テストドキュメント，ソースプログラム等の全件検査を行う。検査方法には，形式的な検査と内容のレビューの２つの方法がある。

③ 実施検査

実際にテストデータを準備して被検査プログラムをテストする。

具体的な事例として日立製作所ソフトウェア工場の例を図表３－２２に示す。

図表３－２２
検査部門の役割（日立製作所）の事例



(2)　システム監査

システム監査とはコンピュータシステム及びそのシステムを利用している業務を，安全性，信頼性，効率性の面から監査することである。

(イ)　監　査　人

コンピュータ部門の管理者によってもシステム監査に類似した役割がなされているが，正式には，システム監査は開発部門から独立した第三者によってなされるべきである。開発部門から独立した社内部門によってなされる監査を内部監査と言い，社外の監査人によってなされる監査を外部監査という。昭和５７年の日本情報処理学会の調査では，自社内にシステム監査人がいると回答した企業は6.2％であり，実際にシステム監査を行ったと回答した企業は1.4％である。

システム監査人には，監査知識，業務知識以外に深いコンピュータ

知識が要求される。そのため，米国にはＣＩＳＡ というコンピュータ監査人の資格制度があり，毎年数百人の合格者が出ている。

(ロ) 監査対象

システム監査を深さのレベルで分類すると次の４つの段階に分けることができる。

① 内部統制，事務手続の監査

② システム開発，サポート，運用，ハードウェア管理の技法の監査

③ システム業務実績の監査

④ データ及びプログラムを含む問題領域の完全検証

米国におけるシステム監査分野の重要度に関するアンケート調査事例では次のような分野が重要視されている。

① 一般的オペレーションコントロール監査

② 安全統制の監査

③ 本番システムのコントロールの監査

④ アプリケーション開発の監査

3.6 品質管理活動

コンピュータ部門の抱えている多量のバックログを解消するには，ソフトウェア開発の生産性向上は焦眉のテーマであるが，そのための手段は品質を無視したものであってはならない。逆に，品質向上こそが生産性向上のための第一段階の取組みであると言われている。特に設計工程の品質向上はソフトウェアライフサイクルにおける生産性向上に大きく寄与している。あるデータでは，障害処理コストのうち設計不良の修正及び仕様変更によるコストが９５％を占めている。

設計品質を向上させるためには，検査部門による品質検査の強化やテスティングの強化だけでは対応することができない。それには，生産活動全般に渡る作業改善や担当者のモラル向上が重要な要素となる。その手段がＱＣ

サークル活動を代表すると小集団活動である。

ＱＣサークル活動は製造工業における製造部門で発達したものであるが，最近は，この活動がソフトウェア分野においても実績を上げつつある。このような品質管理活動がソフトウェア分野においてどのように取り組まれているのかその実態について報告する。

3.6.1　コンピュータ部門における品質管理活動の実態

アンケート調査では７８％の企業が会社として品質管理活動に取り組んでいると回答しているが，コンピュータ部門がその活動に参加している企業は５０％となっている。

全社の取り組みに比べコンピュータ部門の取組率は若干低いが，それでもかなり高い比率であり，コンピュータ部門における品質管理活動をいかに推進するかは重要な共通課題であると考えられる。

コンピュータ部門におけるＱＣサークル活動の実態を図表３－２３に示す。構成員数やテーマ数は製造部門における活動の状況と大差ないが，活動件数は８０％に近い企業が２年以内であり，大部分の企業において，コンピュータ部門の品質管理活動は極く最近になって開始されたことが分る。

図表３－２３　コンピュータ部門の品質管理活動の実態

サークル数	回答率
1	２２％
2	１３％
3～5	２２％
6～１０	２０％
１１以上	２３％

構成員数	回答率
3～4名	１０％
5～6名	５４％
7～8名	２５％
9～10名	９％
11～12名	ｉ％

活動年数	回答率
６ヶ月	１７％
1年	４０％
2年	１９％
3年	１５％
4年以上	８％

テーマ数	回答率
1	３５％
2	３３％
3	１５％
4	６％
5以上	１２％

3.6.2　取組テーマについて

取組テーマに関するアンケート調査結果を図表３－２４に示す。テーマにはかなりバラツキがあり様々な面から総合的に取り組んでいる状況が伺える。

図表３－２４　取組テーマ

取組テーマ	回答数
プログラム構造の標準化推進	３９
ソフトウェア生産性向上	２９
トラブルの防止	２９
ファイル管理	２４
メンテナンス工程の標準化	２２
品質管理及び向上	１６
仕様書の標準化	１６
品質管理・品質向上	１５
ＯＡ化推進	１４
検収・テスト	１１
要員教育	８

3.6.3　品質管理活動事例

財日本科学技術連盟では毎年１回ソフトウェア生産における品質管理シンポジウムを開催しており，既に４回を数えている。発表論文の件数も13，２０，２８，２８件と年々増加しており，５９年度からは発表論文の審査が行われている。テーマの件数は，４，９，１０，４件である。主にこれらの論文の中から各社の品質管理活動の事例をかいつまんで報告する。

(1) 東芝府中工場のＺＤ活動

昭和４８年度よりソフトウェア部門の小集団活動（ＺＤ運動）を行っている。

活動サイクルは６ケ月とし，半期ごとに発表会，表彰を行っている。

取組テーマは，バグ再発防止（１１％），不具合追求・分析（２２％），標準化（２３％），事例紹介（１２％），教育（６％）となっている。

(2) コンピュータサービス㈱におけるＱＣサークル活動

昭和５５年に社長指示で開始し，約６００のサークルがあり，毎年40回以上の事例発表会が開催されている。

組織的には，部課長，ＱＣサークル世話人，ＱＣサークルリーダ，ＱＣサークルメンバーの継型組織となっている。

会議には，部課長サークル会議，世話人会議，リーダ会議，ＱＣサークル会合がある。

また，教育体制も充実しており，ＴＱＣ部課長コース，ＱＣサークル世話人Ａ，Ｂコース，ＱＣサークルリーダＡ，Ｂコース等がある。

表彰制度には，最多提案提出賞，世話人功労賞，最多会合開催賞，最多テーマ完結賞がある。

(3) 日本ユニバック㈱における小集団活動

昭和５６年にＱＣサークル活動を開始し，約４６０のサークルが結成されている。しかし，他社と同じように，ソフトウェア開発が①統計的品質管理の導入が困難である。②個人ベースの作業である。③不可視性の強い頭脳作業である。④作業手順が確定していない等の性格を持っていると現場が感じており，活動進め方について今後に多くの課題があると報告されている。

品質意識の高揚運動として，ソフトウェア品質コンファレンスの開催やソフトウェア品質情報誌の発行等を行っている。

(4) 鹿島建設㈱におけるＴＱＣ推進

鹿島建設㈱は昭和５３年にＴＱＣを導入し，５７年にはデミング賞を受賞している。『草の根ＴＱＣ展開』と称し，一週間コースの全員教育を実施している。また，発表会も充実しており，ＱＣグループ発表会（月１回），ＱＣグループ指導会（週１回），ＱＣグループリーダ研修会（２ヶ月に１回）を開催している。

(5) 日本電気㈱のＳＷＱＣ

日本電気㈱のソフトウェア部門では1,０００を超すサークルを結成し，大々的な品質管理活動を推進している。

体験論文発表会を過去６回実施しているが，第６回の論文集に見る取組分野は，製品品質向上（３０％），工数削減（２８％），作業品質向上（１７％），効率化（１３％），サービス品質向上（６％）となっている。

また，目標達成手段としては，次のような手段が使われている。

① 方法論の強化（レビュー方法，管理方法，作業方法）
② ドキュメントの作成（規定用紙，手引書等）
③ ツールの開発（データ収集／分析，ジェネレータ，ソフト開発支援，デバック／検査）
④ 標準化（作業手順，ドキュメント）
⑤ 適用／流用（既存ソフト，ツール）

(6) 旭化成工業㈱のＱＣサークル活動

現在約３０のＱＣサークルが活動している。活動は１年のうち８ヶ月間を１サイクルとしている。その間，各サークルは週１回半日程度の会合を約２０回繰り返す。活動成果は，次のＱＣストーリに則った活動実施報告書により報告される。

① テーマを取り上げた理由
② 目標（いつまでに，何を，どのように）
③ 現状の把握（パレート図，ヒストグラム，グラフ等を添付）

④ 問題点の要因（特殊要因図を添付）

⑤ 要因のデータ分析（パレート図，ヒストグラム，グラフ等を添付）

⑥ 対　策

⑦ 実施結果と効果

⑧ 標準化，歯止め

⑨ まとめと将来計画

(7) バグ根本原因分析事例

富士通㈱で実施している高信頼性運動の中の一つの活動事例としてバグ根本原因の分析事例を紹介する。

本事例では，５０件のバグの根本原因を探り１００件のバグの予防を行うことに成功している。

本事例の特徴は，バグを統計的に取扱うのではなくバグ一つ一つについてもその根本原因を追究し，努力しなくてもバグを無くすことのできる設計の技（ワザ）を見つけることにある程度成功したことである。

その成果をバグゼロのための７つの設計原理として次のようにまとめている。

①単純原理　②一様原理　③対称原理　④階層原理　⑤透過原理

⑥明証原理　⑦安全原理

3.7 開発環境

ソフトウェアを開発する作業は一般的に知的で根気の要するものであると言われている。最近では開発技法や支援ツールが整備されつつあるものの，ソフトウェア開発が基本的に人間の能力に依っている状態を脱し切っていないのが現状である。ソフトウェア開発の生産性を向上させるためには，開発に携わる人間がその能力を最大限に発揮できるように，作業環境についても改善に務めることが大切である。また，最近では開発要員の不足が深刻な問題として取り上げられることが多いが，優秀な要員の確保のためにも優れた

作業環境を整備する意義がある。

3.7.1 作業環境の推移について

(1) 従来の作業環境（マシン室型）

ソフトウェア開発作業は，融通の効かない機械を相手にする辛気な仕事であり，言い換えれば正確さと集中力，忍耐力が要求される仕事である。たった一つの単純なミスがシステムの大混乱を招くこともあり，ソフトウェアの技術者はそうしたプログラムの脆弱さを知り，完全を期して作業する。納期が間近ならばプログラムの機能を崩壊させるバグを追って徹夜に近い状態が幾日も続いたりすると言った状態が四六時的一般的に有り得たのである。

将来のソフトウェア開発者の作業環境としては，オフィスにおける設計作業が終われば，デバッグのためにマシン室内かデバッグ室（マシン室近傍；マシン中心に室温・換気等が調節されていることが多い）が中心であった。しかも，開発者はデバッグのためにコンソールを中心にカードリーダやラインプリンタの間を走り回っていると言うような状態が多かった。

(2) 最近の傾向（オフィス＆端末型）

最近では，ハードウェア価格の低下によりデバッグマシンが潤沢になったことや自動運転技術，遠隔運転・診断技術及びデータベース技術，会話型デバッグ技術等々の技術革新により，作業環境が改善されている。

アンケート調査結果によれば８６％の企業が会話型デバッグシステムを導入している。今やソフトウェアの開発は，計算機室に入って機械操作をするようなことはなく，一般的なオフィス環境下で会話型デバッグ端末機を使用する作業へと移行しつつある。

しかし，一般的オフィスとは違い，①開発者が精神集中し易く，思考し易い，②大規模ソフトウェア開発のための分担開発やミーティングが

し易い，③端末を縦横に使用し易い，といったような配慮が必要である。

このような特徴を踏まえて，広さ，照明，湿湿度・換気，机・椅子，間仕切り・会議コーナ，研修室，休憩室，騒音等の作業環境の要素について考察する。

3.7.2　作業環境に関するアンケート結果

ソフトウェア開発者の作業環境として，どの様な工夫がされているかアンケート調査結果（図表３－２５）から見てみる。作業環境の配慮のポイントを採光，換気，机の配列，その他について実施の有無を聞いたところ次の結果を得た。

図表３－２５　作業環境の配慮のポイント

採　光	（７２％）
換　気	（６２％）
机の配列	（６１％）
騒音の遮断	（５０％）
その他	（　７％）

いずれも高い実施率であるが，特に高い採光は作業が緻密で根気を要するものであり，ドキュメントリストあるいは端末（ワークステーション）を長時間見ることが多いからであろう。騒音の遮断５０％も高い配慮率であるが，これは設計作業や端末機操作中の思考を妨げないための配慮と思われる。

また，具体的な工夫・改善の実例を自由に記入して貰った結果については，図表３－２６の通りである。

図表３－２６　具体的な工夫・改善の内容

ディスプレイ面の反射防止対策	(41%)
騒音対策	(31%)
机及び椅子	(15%)
端末装置の設置場所	(10%)
採　光	(7%)
室内の整理・整鈍	(5%)

図表３－２７　端末普及状況

要員数区分	企業数	要員数	端末台数	１人当り端末台数
１０人以下	27	225	148	0.66
11～20人	26	564	343	0.61
21～30人	19	557	287	0.52
31～50人	24	1,053	428	0.41
５１人以上	25	4,091	2,070	0.51
計	121	6,490	3,276	0.50

ディスプレイ，騒音対策，端末の設置場所と導入した端末の配置，使用環境について腐心していることが特徴的である。因なみに，端末の普及は図表３－２７の通り一人当たり 0.5 台となっている。

3.7.3　望ましい作業環境の具体的例

米国においては，ソフトウェア開発者は殆んど個室が与えられており，通常 3 m × 3 m が一般的の様である。

ＩＢＭ社サンタテレサラボラトリの例では，窓付の 3 m × 3 m の個室と１５の個室に一つの割合で会議室がある。

米国と日本の差は，土地の価格，経済規模の差，国民性の違い等があり，一概にこのレベルに近づけるべきだとは言い切れないが，個人のスペースの確保，ミーティングスペースの設置等各社とも種々の工夫をしている。机の配列等について 2, 3 の例を上げれば，次の様な例も見られる。

Ａ．寺小屋式（教室式）

Ｂ．部屋の周囲の壁にデバック端末を設置し机は部屋の中央に寺小屋式に配置する。及びその逆の形態。

Ｃ．デバッグ室の机は，フリーに使用できるようにする。

Ａのメリット；スペースの有効利用，精神集中がし易い。

Ｂのメリット；Ａのメリットおよび空端末が見つけ易く，端末操作に集

中し易い。また，端末までの移動距離が少なくてすむ。

Cのメリット；机の有効利用

このように，机の配置法，端末の置き方一つで，ソフトウェア開発者の精神集中やスペース及び端末の有効利用等が可能となった例も見受けられる。

参 考 文 献

1) 小林要：部品化ソフトウェアの考え方，情報処理学会第２９回全国大会，１９８４.９.１１

2) 平倉一郎他：ＳＥＡ／Ｉに基づくソフトウェア開発支援システム，同上

3) 清水達男他：ＳＥＡ／Ｉにおける部品作成管理ツール，同上

4) 大島正敬他：ＳＥＡ／Ｉに基づくソフトウェア開発支援システム，同上

5) 西村高志：MOTHER SYSTEMによるソフトウェアの設計と製造，情報処理，Vol. 25. №11, １９８４

6) 富士通：ファコムジャーナル，Vol. 10 №12，１９８４

7) 特集オブジェクト指向プログラミング，日経エレクトロニクス，１９８４.８.１３

8) 和田英一：Adaの概要，情報処理，Vol. 22 №2，１９８１

9) 小林要：ソフトウェア製品生産における知的分業，情報処理学会第２回ソフトウェア工学シンポジウム，１９７９.１２.１２

10) バックログ問題解決の道を探る，日経コンピュータ，１９８２.１２.２７

11) システム部門からの転出，他部門からの転入によりＯＡ化を実現，日経コンピュータ，１９８３.５.３０

12) 奈良隆正：ソフトウェアの品質保証活動，日本科学技術連盟，第３回ソフトウェア生産における品質管理シンポジウム，１９８３.９.１３

13) 松尾明他：システム監査人への道，日経コンピュータ，１９８４.１.２３

14) 日本能率協会：システム監査コーステキスト

15) 蛯原秀：産業用ソフトウェア品質向上活動の一事例，日本科学技術連盟第２回ソフトウェア生産における品質管理シンポジウム，１９８２.９.２７

16) 西本安弘：CSKにおけるQCサークル活動の導入と推進，日本科学技術連盟第３回ソフトウェア生産における品質管理シンポジウム，１９８３.９.１３

17) 前田裕：ソフトQCの現状と課題，同上

18） 和田忠昭：TQCの推進とプログラムの計算スピード向上事例，同上
19） 萩原徳望他：ソフトウェア品質管理グループ活動，情報処理学会第２９回全国大会，１９８４．９．１１
20） ソフトウェア品質管理への挑戦：日経コンピュータ，１９８４．１０．１
21） 日野克重：ソフトウェア障害の分析と予防，日本科学技術連盟第４回ソフトウェア生産における品質管理シンポジウム，１９８４．９．１８
22） 青山義彦：ソフトウェア生産の諸問題とその生産技術の動向，日立評論，Vol．６２，１９８０．１２

4. 主な開発技法と開発支援ツール

4. 主な開発技法と開発支援ツール

本章ではソフトウェア開発のための技法と支援ツールについて述べる。

ここで技法と呼ぶものは，ソフトウェア開発を進めるにあたっての指針となる方法論ないしは各工程における具体的な手法を指し，それら技法を作業手順やドキュメント仕様として集大成した開発標準の類はここには含めない。

また，開発支援ツールというのは，技法をコンピュータのプログラムに実現したものを言う。手書きのフォームやチャートなどもツールに含めて考える場合もあるが，本章で取扱う範囲はプログラム化したものに限定している。

4.1 開発技法

4.1.1 設計技法

「要求分析・定義」工程と「設計」工程とは論理的には別物と考えられるが，実際には両者の境界は判然と区別し難い場合が多い。それゆえ本項では，要求分析・定義技法も広義の設計技法の中に含め，併せて論ずることにした。

一般にシステムには，データの流れと制御の流れがある。このいずれに着目して分析を行うかによって，要求分析・定義技法を２つのタイプに分類することができる。データ中心型の技法は事務処理システムなどの要求記述に適しており，データの流れ図を利用する方法や，情報代数と呼ばれる形式的な言語を用いる方法などがある。制御中心型の技法は実時間システムなどの要求記述に適しており，これにはグラフ・モデルが利用される。本項では両者の代表的なものとして，データの流れ図によるＳＡＤＴと，グラフ・モデルの一例であるペトリ・ネットによる方法を取り上げて説明する。

狭義の設計のための方法論を考える場合，２つの観点がある。１つは設計の進め方の問題で，全体から部分へ進めるいわゆるトップダウンのアプ

ローチか，部分から全体へ進めるボトムアップのアプローチかという観点である。IBMの総合的ソフトウェア開発技法体系であるIPT（Improved Programming Technologies：効果的プログラム開発技法）などではトップダウン・アプローチが推奨されているが，両者の優劣についての決着は未だついていない。いずれにしろ，いずれか一方だけに限定しようとするのは無理があるようで，問題の性質に応じて，両者のバランスを考えながら折衷的な方法で開発を進めるのが現実的と考えられる。

もう1つの観点は設計のよりどころとして，データの流れに着目するか，あるいはデータの構造に着目するかという立場の違いによるものである。前者をデータフローによる設計法，後者をデータ構造による設計法と呼ぶ。データフローによる設計法にはマイヤーズの複合設計法やヨードン，コンスタンティンの構造化設計法などがあり，データ構造による設計法にはジャクソン法やワーニェ＝オア法などがある。本項では，この第2の観点から見た，これらの技法について解説する。

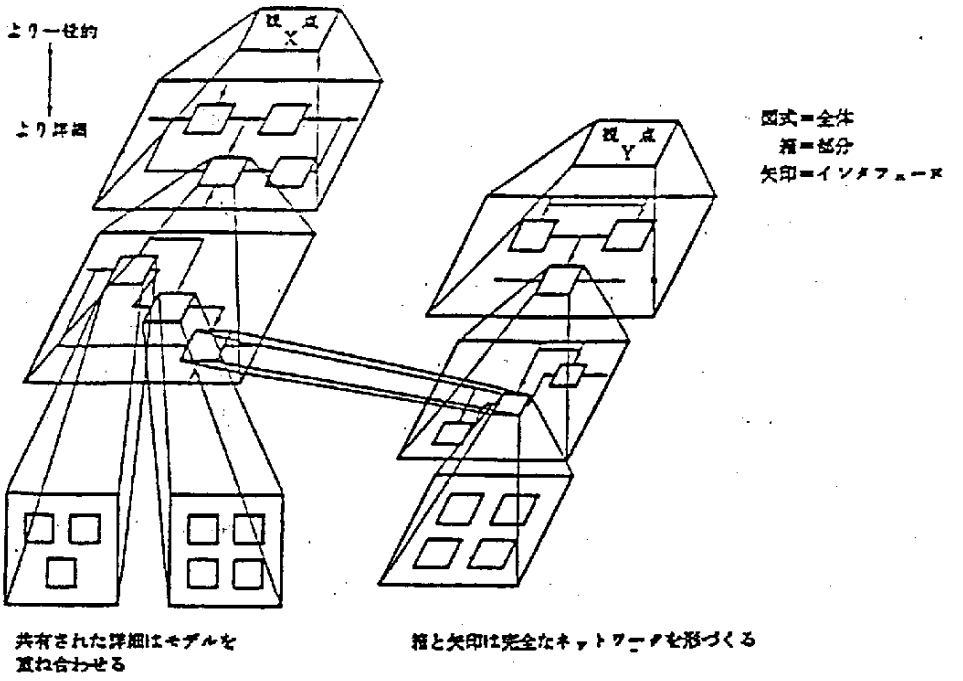
(1) SADT（Structured Analysis and Design Technique）

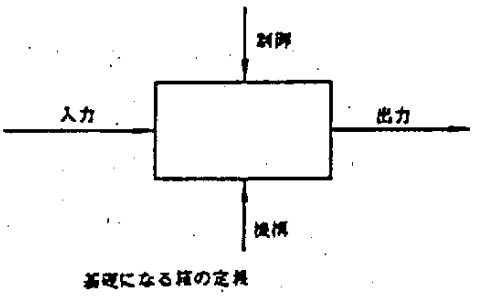
これはSof Tech社で開発された，データ中心型のシステム分析・要求仕様定義のための技法である。

その基本的な考え方は，問題をトップダウンにモジュール化して分析して行って，それを階層的な構造を持ったモデルとして図式的に表現することである（図表4－1（a））。そのためにはシステムを「動作」と「物（データ）」の2つの異った面から捕え，2つの相補的なモデルを作成する。すなわち，動作分解では「動作」を箱で表わし，それらを関連づける「データ」を矢印で表わすが，データ分解では逆に「物(データ)」

図表４－１　ＳＡＤＴダイアグラム

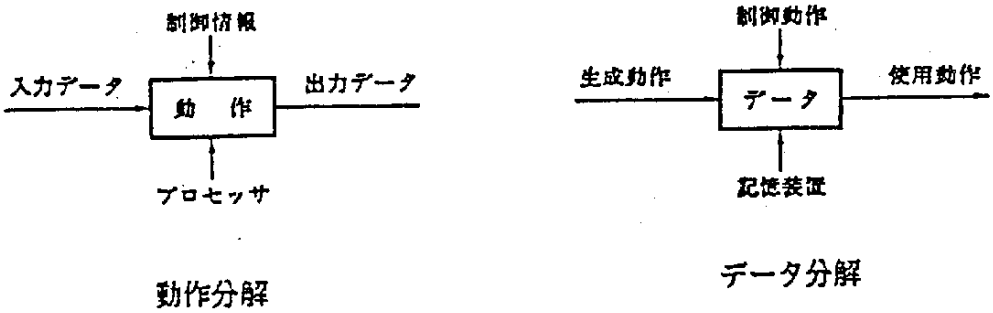
（ａ）　階層構造





基礎になる箱の定義

（b）　動作分解とデータ分解



動作分解　　データ分解

（国井利泰編「bit 増刊，ソフトウェア・プロダクト工学」（共立出版）より転載）

を箱で，「動作」を矢印で表わす（図表４－１（b））。最後にこれら２つのモデルを突き合せて，互いに不足したり，矛盾したりするところがないかを調べる。

なお，SADT図の矢印の構造は箱のあいだの制御関係を表わすもので，フローチャートのように制御や処理の流れを表わすものではない。

(2)　ペトリ・ネット・モデル（Petri net model）

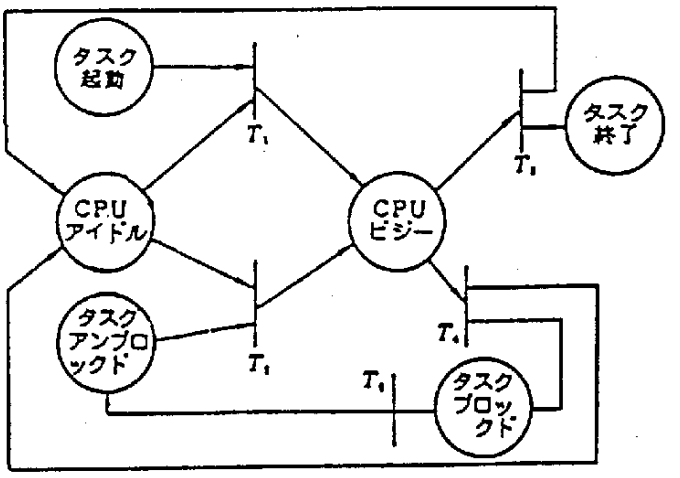
これはGRCで開発された，制御中心型の要求分析・定義のための技法である。

ペトリ・ネットは図表４－２に示したように２種類の節（node）を持った有向グラフである。円形の節は「条件／場所」を，棒線の節は「事象／推移」を表わし，矢線は「処理／作用」を意味する。事象はそこへの入力条件がすべて満たされたとき，「可能にされた」と呼んで付点を付けて表わす。可能にされた事象を次々と選択して行くことにより，可能な事象系列のシミュレーションが生成できる。作用の推移や非同期動作を表わすのに便利な図法である。

(3)　複合設計（composite design）

従来のモジュラー・プログラミングやトップダウン設計法は，プログラムの階層的モジュール化の必要性を認めながら，モジュール分割の方

図表４－２　ペトリネット・モデルの例



（宮本勲著「ソフトウェアエンジニアリング：現状と展望」（TBS出版会）より転載）

法を明確に示すことができなかった。これに対し，マイヤーズ（Glen J. Myers）はモジュールの独立性を高めるようにモジュール分割を行うべきことを主張し，それを実行するための具体的な方法論として提唱したのが複合設計法である。

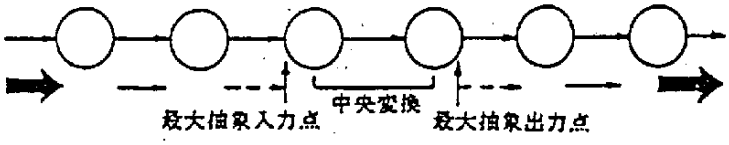
マイヤーズはモジュールの独立性を測る尺度として，モジュールの強度とモジュールの結合度という２つの概念を定義した。モジュールの強度とはモジュール内部の関連性の強さを示すもので，弱いものから強いものの順に，暗合的強度，論理的強度，時間的強度，手順的強度，連絡的強度，情報的強度，機能的強度の７つのタイプに分類される。一方，モジュールの結合度はモジュール間の関連性の度合いを示すもので，強いものから弱いものの順に，内部結合，共通結合，外部結合，制御結合，スタンプ結合，データ結合の６つのタイプに分類される。マイヤーズによれば，モジュールの独立性を高めるためにはモジュールの強度を最大に，モジュールの結合度を最小にするのが望ましく，それゆえ上記の分

類に従えば，機能的強度のモジュールをデータ結合するのがベストということになる。

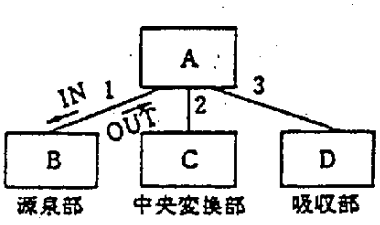
しからば，ベストのモジュール分割を行うにはどうしたら良いか。その分割の手順を実際に与えるのが複合分析（composite analysis）と呼ばれる手法である。複合分析を行うためには，まず問題の中の主要な入力データと出力データの流れに着目する。その主要な入力データの流れを問題の構造に沿ってトレースして行くと，データが段々と抽象化されていって，それ以後はもはや入力データと呼べなくなる点に到達する。これを最大抽象入力点と呼ぶ。同様の分析を出力データについても（出口から逆方向に）行うと，最大抽象出力点が得られる。これらの中間が，入力データを出力データに変換する部分であり，中央変換と呼ぶ。このようにして，問題の構造が3つの主要な部分，すなわち，源泉（source）部分，中央変換（transform）部分，吸収（sink）部分に分けられる。これらの各部分がそれぞれ1つの機能を果すモジュールになる（図表4－3)。以下，各モジュールについて同様の分析を繰返し，さらに下

図表4－3　複合設計法

（a）　問題構造と主要入出力データの流れ



（b）　プログラム構造とインタフェースの定義



最初のプログラム構造

	IN	OUT
1	なし	最大抽象入力データ
2	最大抽象入力データ	最大抽象出力データ
3	最大抽象出力データ	なし

モジュール間のインタフェースの定義

（国友義久著「効果的プログラム開発技法」（近代科学社）より転載）

位のモジュールへと展開して行く。モジュールの論理が直感的に理解できるようになったところが，分析の終点である。

このようにして，複合分析は問題の構造をトップダウンで機能展開して行く。結果として機能の階層構造が得られるが，これがそのままプログラムの階層構造に対応する。

しかし，上記の手順はあくまで分析の指針を与えるものであり，誰がやっても必ずベストの分割ができるとは限らない。また問題によっては，機能的強度やデータ結合が実現し難い場合もある。要はできるだけ強度が強く，結合度の弱いモジュール分割を行うように心懸けることである。

(4) 構造化設計（structured design）

ヨードン，コンスタンティン（Ed Yourdon Larry L. Constantine）らによる構造化設計は，複合設計の考え方をさらに押し進めて形式的な度合いを強め，データ構造による設計法の考えも取り入れたものである。

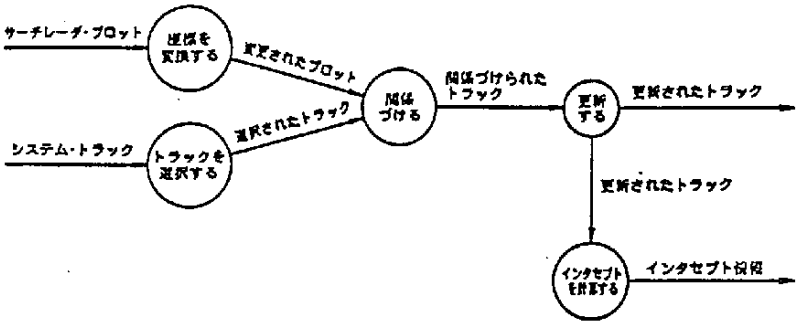
設計過程として初期，中間，最終の3つの過程に分割する。初期設計では主なシステム機能とデータの流れだけを考え，中間設計では概念的なものと実現段階の双方，たとえばデータベースやシステム制御などを考え，最終設計では実現段階の具体的な設計を行う。

設計の補助手段としてバブル図や構造図などの表現法を用いる（図表4－4）。バブル図（bubble chart）はデータの流れを表わすもので，複合設計では素朴に使われていた記法を1つの図法として洗練させたものである。円形のバブルでデータの変換処理を，バブルに出入りする矢印で入出力データを表わす。バブル図に示されたデータの流れを実現するためのモジュールの階層構造を表わすには，複合設計と同様な階層図を用いる。

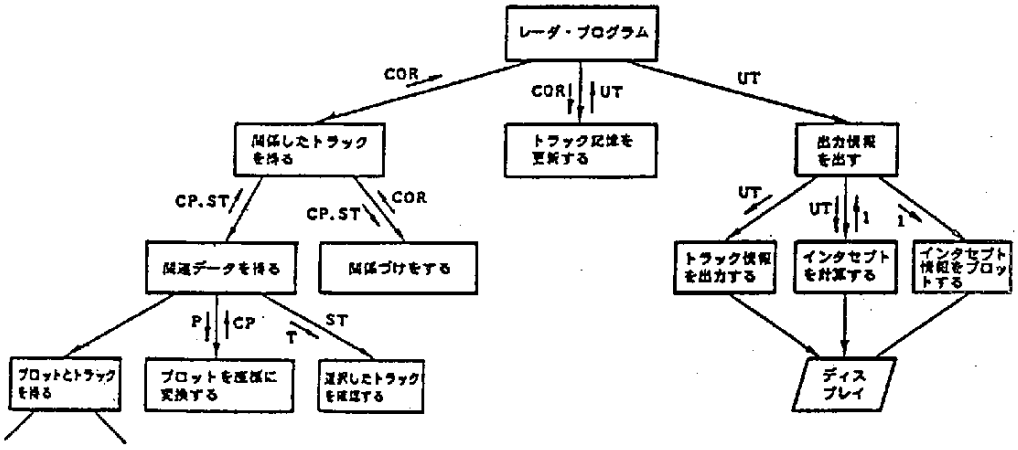
複合設計で複合分析と呼んでいた手法は，ここでは変換分析（transform analysis）と呼ばれる手法に翻訳され，源泉／吸収といった言

図表４－４　構造化設計法

（ａ）　バブル図の例



（ｂ）　構造図の例



（宮本勲著「ソフトウェアエンジニアリング：現状と展望」（TBS出版会）より転載）

葉の代りに導入（afferent）／導出（efferent）という用語が使われている。

さらに変換分析の代替的な手法としてトランザクション分析（trans-

action analysis）という方法も提案されている。データの流れ図に記述されたデータ項目の中に，それを評価した結果，いくつかの異ったパスの流れが生じる場合がある。このようなデータ項目をトランザクションと呼び，このような流れをトランザクション・フローと呼んで，上で述べた変換フローと区別する。トランザクション分析はこのトランザクション・フローを分析するための手法であるが，データの流れのどの部分でいずれの分析方法を用いるべきかの基準が，もう１つ明確でない嫌いがある。

(5) ジャクソン法（Jackson method）

ジャクソン（Michael Jackson）が発表した設計法で，いわゆるデータ構造による設計法の代表的なものである。

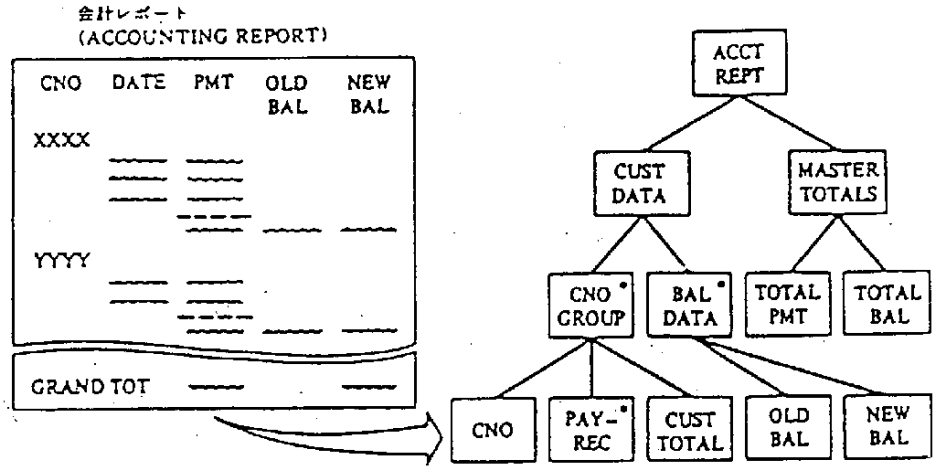
ジャクソンの主張によれば，ソフトウェア（プログラム）の構造は，それが対象とする社会の組織の構造を反映するものでなくてはならず，それゆえ組織で必要とされるデータそれ自身の構造と相似の構造を持たなければならない。異った構造のプログラムはたとえ動いたとしても，それは単に悪いプログラムというに止まらず，誤ったプログラムであるというのである。

設計の手順はまず，対象とするシステムで用いる入力と出力のデータ構造を考え，それらを構成する各要素ごとに入出力の対応関係を調べる。プログラムはその要素ごとに入力から出力への変換を行うようなものであり，従ってその構造は入出力双方のデータ構造に対応する形となる(図表４－５（a），（b））。

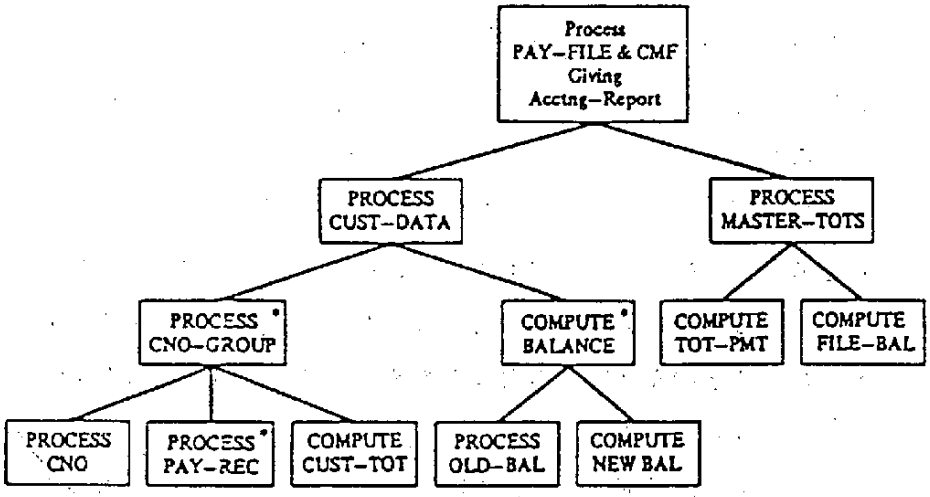
しかし，実際に遭遇する問題では，入力と出力のデータ構造がうまく対応するような例はあまりなく，むしろ何の関係もないのが普通である。ジャクソンはこれを「構造の不一致」もしくは「衝突」（structure clash）と呼んでいるが，その場合の解決策としては，中間にもう１つのデータ構造を導入し，それと入出力それぞれのデータ構造をつなぐ別々

図表４－５　ジャクソン法

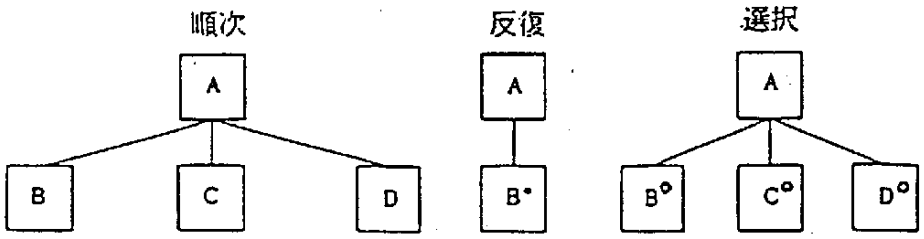
（ａ）　システム出力



（ｂ）　結果として得られるプログラム構造



（ｃ）　３つの基本構造



（ロジャー・Ｓ・プレスマン著，岸田孝一監訳，岡田正志訳
「ソフトウェア・エンジニアリング序説」（ＴＢＳ出版会）より転載）

のプログラムを考えることでデータ構造の不一致を吸収しようとした。中間データ構造の導入は確かに構造不一致の問題を解決はしたが，そのために効率の低下という問題が新たに生じた。そこでジャクソンはさらに，中間データ構造を省いて直接２本のプログラムをリンクするプログラム転化（program inversion）や，それより複雑化した複合縫合（multithreading）といった方法を提案した。

ジャクソンの処理階層はいわゆるモジュールの概念とは必ずしも一致しない。従って，概要設計と詳細設計の区別も明確でなく，設計は全体から部分へ連続的に進められる。処理階層の最下位のレベルは，構造化プログラミングでいう順次，選択，反復の３つの基本構造に帰着されるので（図表４－５（ｃ）），そのまま疑似コードを使って滑らかにコーディングへ移行することができる。

(6)　ワーニェ＝オア法（Warnier-Orr method）

これはワーニェ（Jean-Dominique Warnier）によって開発され，オア（Kenneth T. Orr）らによって拡張された方法である。基本的な考え方はジャクソン法に良く似ているが，数学的に厳密な理論に基づいて組立てられている点は他の設計法に見られない特色と言えよう。

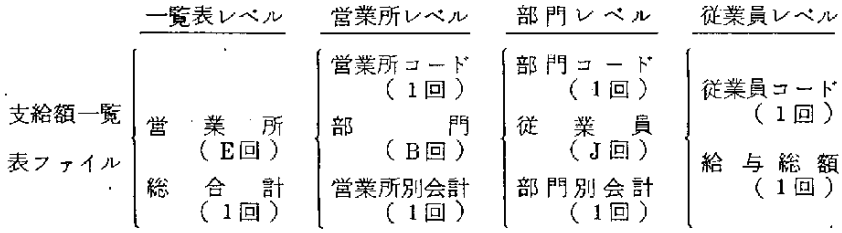
この方法は３つの要素から成っている。すなわち，プログラム設計の

ためのLCP（Logical Construction of Programs），システムのデータを組織化し，そのデータを保有するプログラムを一貫したシステムとして組織化するためのLCS（Logical Construction of Systems），および論理的に正しい階層構造を持つシステムを設計するための構造化システム設計（Structure Systems Design）とである。3者とも方法論としてのアプローチの仕方は原則的に同じであるので，ここではＬＣＰについて説明する。

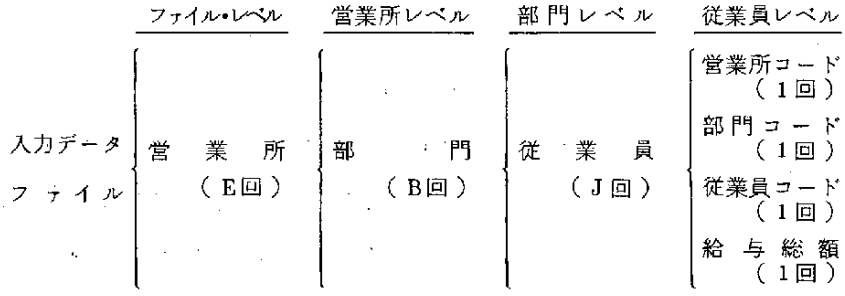
まず初めにワーニェ図を用いて入出力のデータ構造を仕様化する（図表４－６）。ワーニェ図はジャクソンの階層図と同様に階層と手続きを同時に表わせる。このデータ構造に対応するものとして処理の階層構造

図表４－６　ワーニェ＝オア法(1)

ワーニェ図の例



出力データ構造図



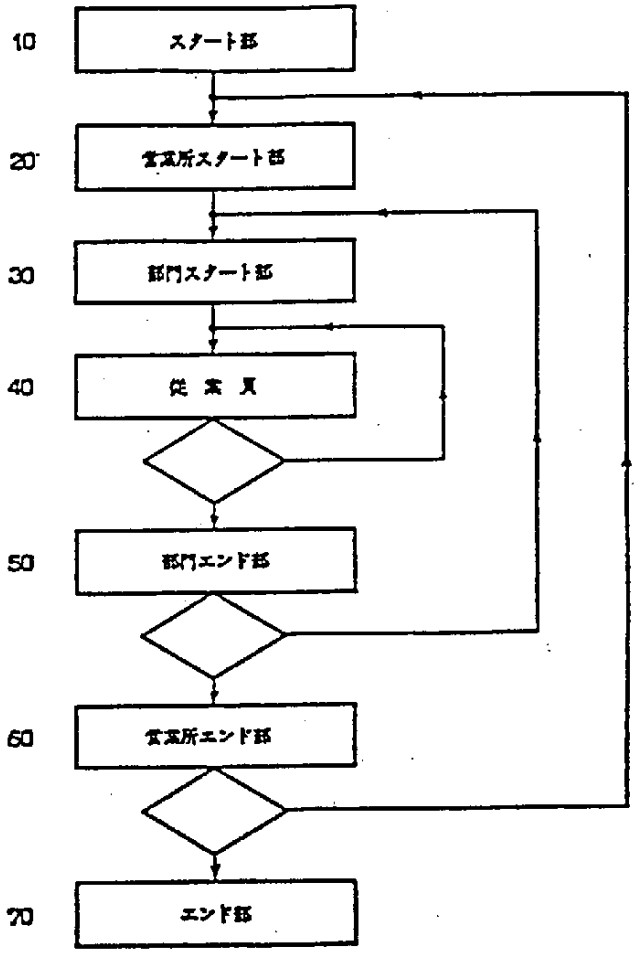
入力データ構造図

	プログラム・レベル	営業所レベル	部門レベル
プログラム	スタート部（1回）	営業所スタート部（1回）	部門スタート部（1回）
	営業所（E回）	部門（B回）	従業員（J回）
	エンド部（1回）	営業所エンド部（1回）	部門エンド部（1回）

プログラム構造図

図表４－６　ワーニェ＝オア法(2)

ワーニェ図から得られたプログラム・フローチャート



（「図形入力を加味したプログラミング技術，昭和５６年度」（ソフトウェア産業振興協会）より転載）

が決まる。次に，このワーニェ図による処理階層の表現を通常のフローチャートの記法に変換する（図表４－６）。いわゆる構造的な記述法を用いないでフローチャートを用いている点はやや旧式な感じもするが，ワーニェ法の特徴はこうして作られたプログラムの論理構成から，さらに詳細な命令を作り出すための詳細構成（detailed organization）という技法を開発していることである。

ワーニェはさらに，複雑な論理を単純化するための技法や，既存の非構造的プログラムを再構造化するための技法なども提案している。

この方法もジャクソン法と同様に，モジュールの概念は希薄である。

データ構造による設計法は事務処理のようにファイル変換を主とするようなプログラムを設計するのに適している。反面，科学技術計算や制御プログラムのようにデータ構造が重要でなく，制御やデータの流れが複雑なものはデータフローによる設計法の方が優れている。

以上，主な開発方法を述べてきたが，一種の連続的統合であるプロトタイピングも最近よく用いられている。これには，いわゆる使い捨て的なものと，将来システムの一部に組み込まれるものの２種がある。いずれにしても，目標とするシステムのモデムを作って，種々の検証をしようとするものである。

4.1.2　設計技術技法

設計技術技法は設計技法の一部であることが多く，後者と切り離しては論じ難いものである。すでに前項でも，このような技法のいくつか（バブル図やワーニェ図など）について説明した。

本項では設計記述技法の中でも，比較的独立して論ずることのできるプログラムの論理の記述法を中心に話を絞った。

(1) フローチャート

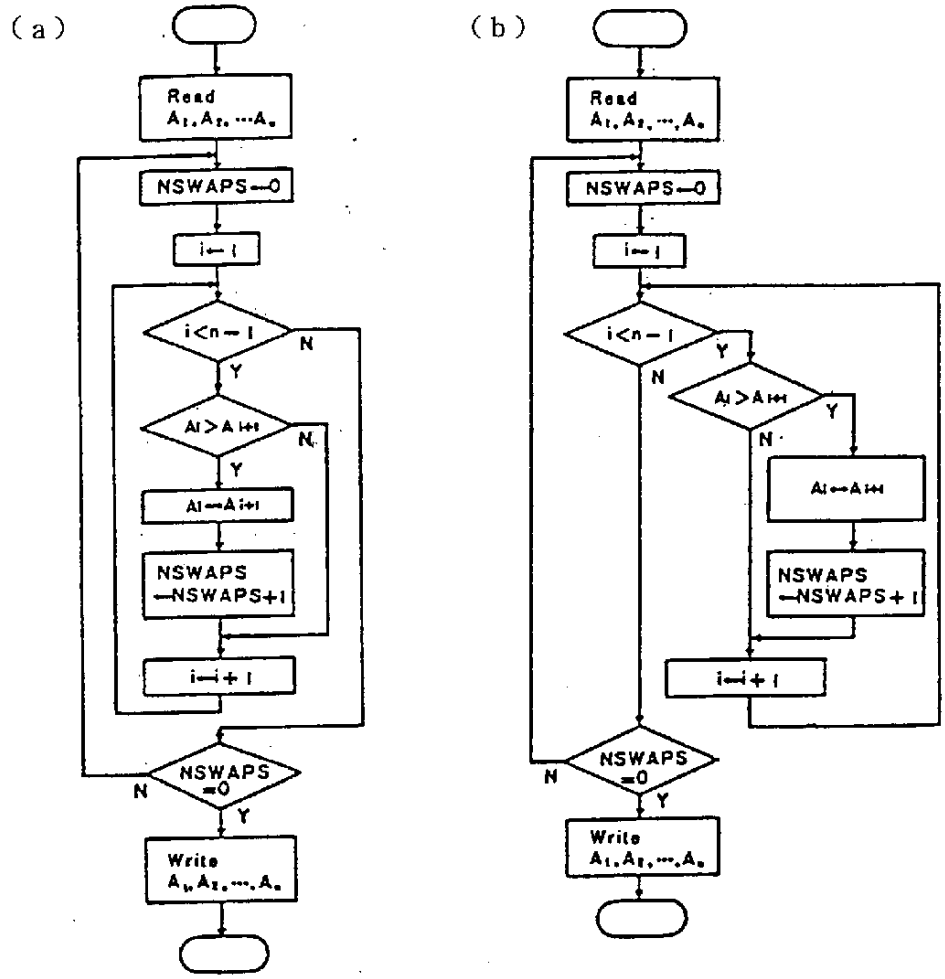
フローチャートは1940年代後半にゴールドスタインとノイマン(H. H. Goldstine, J. von Neumann)によって考案されたと言われている。以来，四半世紀にわたって，プログラムの論理を記述する唯一無二の技法として君臨して来た。1970年代にタイクストラ(Edsgar Dijkstra)が構造化プログラミングを提唱するに及んで，この記法の欠点がいろいろ指摘されるようになり，学界での評価はとみに低落しているが，ソフトウェア開発の現場ではまだ根強く使用され続けているようである。

フローチャートは良く知られているように，処理を四角形で，分岐(YES／NO)を菱型で表わすが，それらの順序(制御の流れ)を示す矢線を上下左右どこにつないでもかまわないのが大きな特徴である。その自由自在さのゆえに，「プログラムの構造が見え難い」，「同一の論理が書く人によって異った表現になる」などの批判を浴びる破目となった反面，初心者にとっての学習が容易であり，幾多の批判にもかかわらず，しぶとく生き延びてきた原因の1つになっていると思われる。図表4－7には同じ論理(交換ソース)が異った表現で表わされた例を示す。

ディクストラの構造化プログラミングの目的は，プログラムを階層的な構造にして，理解し易く，かつ個人差の生じ難いプログラムを作ることであり，そのために(処理の)順次，選択，反復という3種類の基本要素(図表4－8)のみを使ってプログラムを組み立てることを主張した。

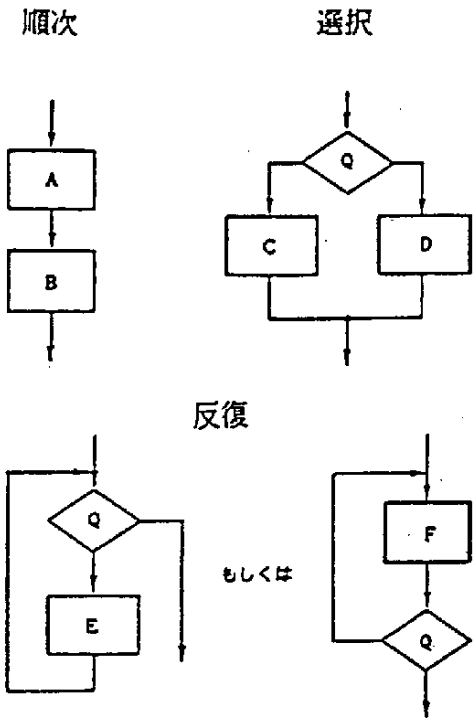
フローチャートを用いて構造化プログラミングを実現することは，もちろん可能である。しかし，そうしても論理が複雑になるとプログラムが見難くなる欠点は解消されず，また，ともすると非構造的なプログラムになる危険性が多い。そこで構造的プログラミングの表現に適した，

図表４－７　フローチャートによる記述例（交換ソース）



＊同一の論理が（ａ），（ｂ）のように異った形で表現される

図表４－８　構造化プログラミングにおける３つの基本要素



以下のような記述法が種々考案されて来た。

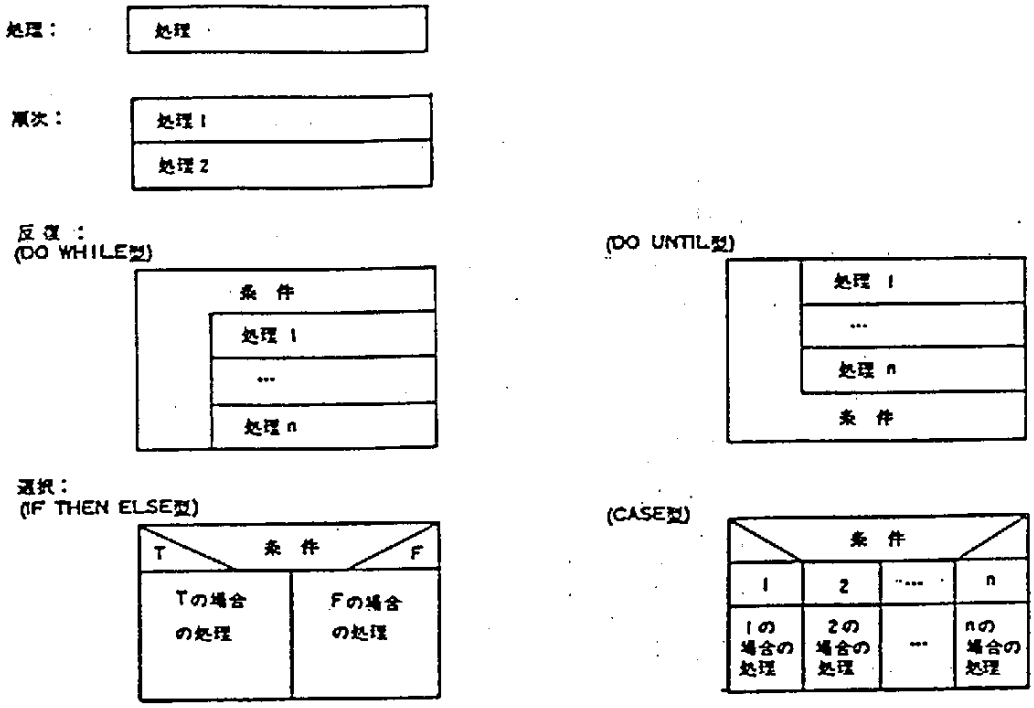
⑵　ＮＳチャート（Nassi-Shneiderman chart）

フローチャートに代る図法として，最も早く登場したのがＮＳチャートである。これは1974年頃ナッシとシュナイダーマン（I. Nassi, Ben Shneiderman）によって考案されたもので，２人の頭文字を取って命名された。またその形からボックス・ダイアグラムということもある。

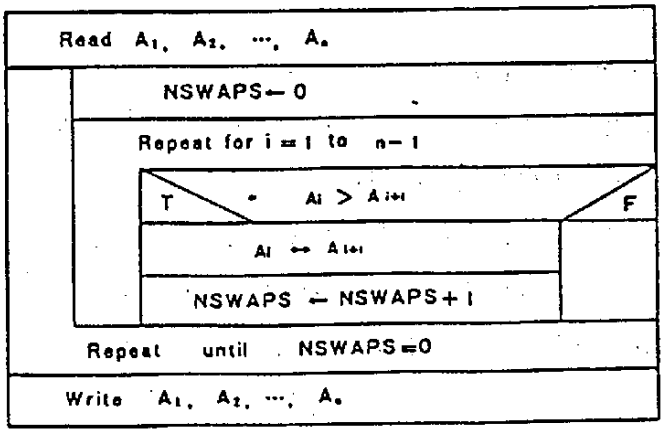
ＮＳチャートの特徴は制御の流れを示す矢印がいっさいないことで，図表４－９（ａ）に示したような基本図形の連続したブロックで処理の流れを表わす。ＮＳチャートに限らず，新しい図法に共通して言えることは，処理が上から下へ流れるのが原則で，その逆はないことである。ＮＳチャートはプログラムの制御構造が見易く（ただし，モジュールの構成はそれほど見易くない），図がコンパクトに表わせるのが利点だが

図表４－９　ＮＳチャート

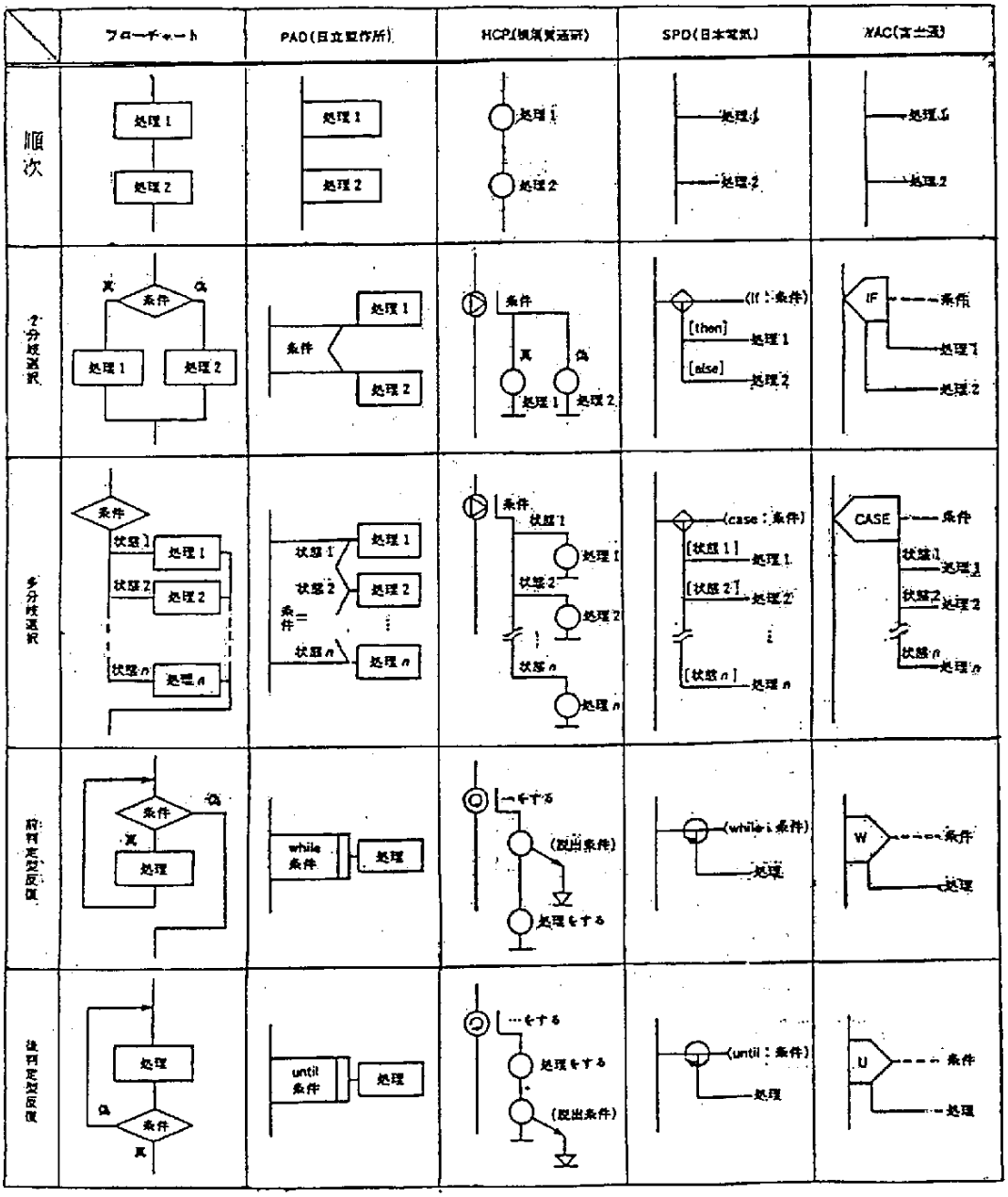
（ａ）　表記法



（ｂ）　記述例（交換ソート）



図表４－１０　主な木構造チャートの記法



（日経コンピュータ，１９８４．１．９号「プログラム制御構造の新しい表現技法：木構造チャートが普及期に」より転載）

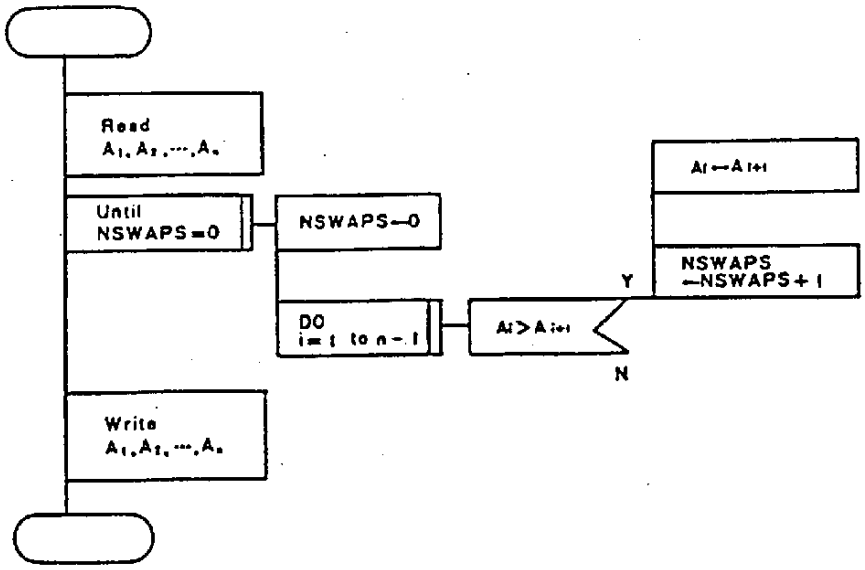
（図表４－９（ｂ）），記述スペースの取り方や，修正の手間など，書き難い難点があり，有名な割りには普及度がいまひとつという感じである。しかし，オランダがＮＳチャートの改良版であるＰＳＤを国内規格として採用しているほか，西ドイツでも規格を進めていると伝えられている。

のちにチャピン（N. Chapin）はＮＳチャートとフローチャートを折衷したチャピン・チャートを発表している。

ＮＳチャートの欠点を克服するための記法として考え出されたのが，いわゆる木構造チャートと言われるもので，その代表的な例を図表４－１０に示し，以下(3)～(5)および(7)に解説した。

(3) PAD（Problem Analysis Diagram：問題分析図）（図表４－11）

図表４－１１　PADによる記述例（交換ソート）

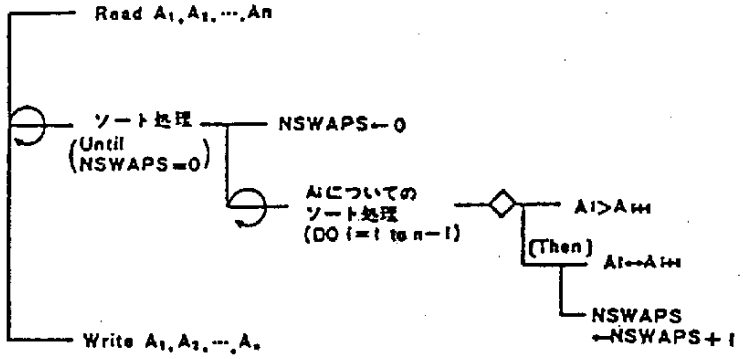


これは日立製作所の二村良彦氏らによって考案されたもので，ＮＳチャートのように基本図形を密着して結合させるのではなく，線でつないだ点が異っている。このため，ある意味でフローチャートと同様な記述の自由さが得られ，ＮＳチャートよりも書き易くなった。しかし，この線はフローチャートのような制御の流れを表わすものではなく，縦線は処理の順序を，横線はモジュール間の従属関係を表わすが，それによってプログラムの階層構造もいっそう見易くなっている。

処理をフローチャートのように箱の中に書くため，あまり細かな記述のできないことが欠点といえば言えよう。

(4) ＳPD（Structnred Programming Diagram ：構造化プログラム図）（図表４－１２）

図表４－１２　ＳPDによる記述例（交換ソート）



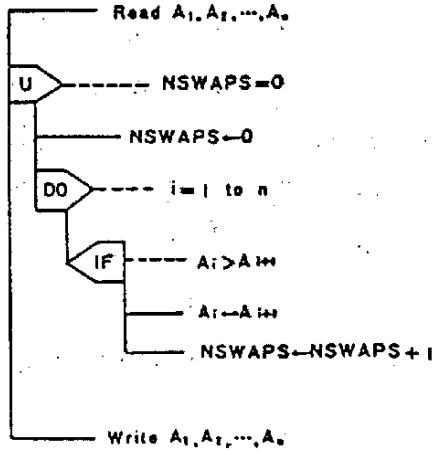
これは日本電気のソフトウェア開発標準ＳTEPＳの一環として開発された図法である。一口に言うと，箱を取り除いたＰＡＤという感じで，ＰＡＤに比較すると箱に制限されないだけ自由な記述が可能である反面，視覚的な理解という点で一歩を譲るようである。

(5) ＹACⅡ（Yet Another Control chart）（図表４－１３）

これは富士通で開発されたものであるが，ＳＰＤのバリエーションといった感じが強い。選択と反復を表わす記号が異るほかは，ＳＰＤに酷

似している。

図表４－１３　YACによる記述例（交換ソート）



(6)　HIPO（Hierarchy plus Input-Process-Output）（図表４－１４）

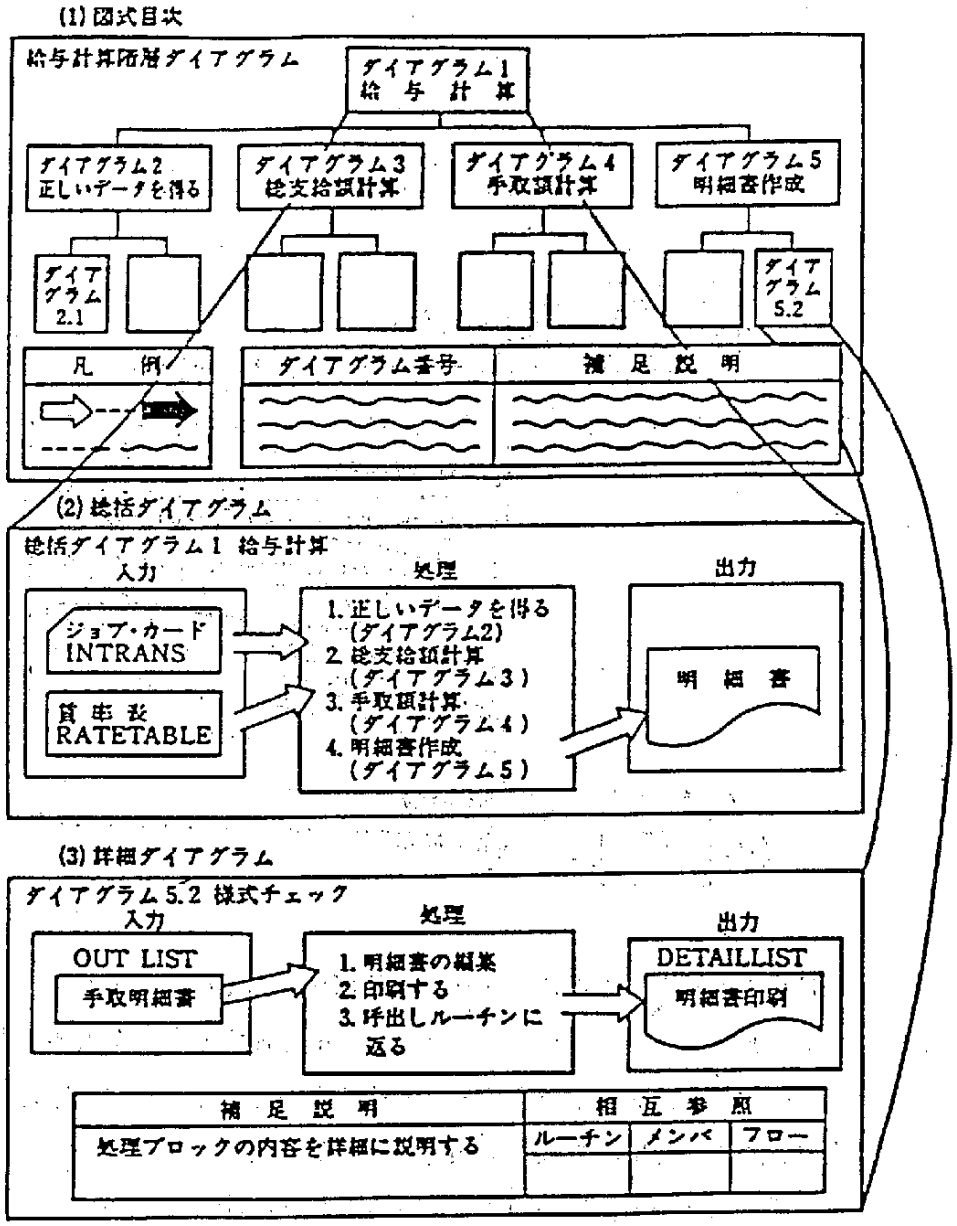
上記の図法はすべてプログラムの論理を記述するためのものであったが，コーディングを行うためには論理だけでなく，データの構造や流れについての情報も必要である。これらを記述するための代表的な図法としてＨＩＰＯがある。

これはＩＢＭのＩＰＴの中に含まれているもので，英語の正式名称が示すとおり，システムの階層（Hierarchy）を表わす図式目次と，各モジュールにおける入力＝処理＝出力（Input-Process-Output）を表わすＩＰＯダイアグラムとから成る。

ＩＰＯダイアグラムは図に示したように紙面を縦に３つの部分に分け，左に入力データ，中央に処理，右に出力データを書き，データの流れを太い白抜きの矢印で示す。更新データのように入出力双方を伴う場合の扱い方に問題がありそうである。

処理の記述法はとくに規定していないが，ＮＳチャートやＰＡＤ，ＳＰＤのような木構造チャート，あるいは後述する擬似コードなどを用いて

図表４－１４　ＨＩＰＯ



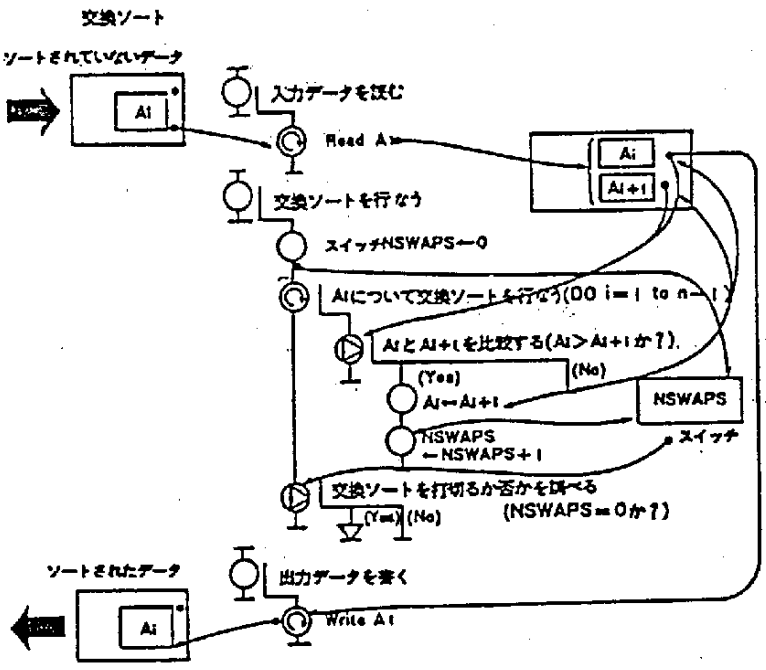
総括ダイアグラムと詳細ダイアグラムを合わせてIPOダイアグラムと呼ぶ

（国友義久著「効果的プログラム開発技法」（近代科学社）より転載）

記述すれば良い。（フローチャートの適用も不可能とは言い切れないが，かなり工夫がいりそうである。）

(7) HCP（Hierarchical ComPact chart：階層化コンパクト・チャート）（図表４－１５）

図表４－１５　HCPによる記述例（交換ソート）



ＮＳチャートや木構造チャートとＨＩＰＯのＩＰＯダイアグラムを併用することによって，処理とデータの双方を記述することが可能であるが，これを１つの図表にすっきりとまとめて表わそうとしたのが電電公社・横須賀通研の花田収悦氏らが開発したＨＣＰである。

処理の記述の方法は，いうならばＳＰＤ風の開いた（つまり，箱で囲まない）記述法によっているが，ＳＰＤよりも記号の種類が豊富であり，多彩な表現が可能である。

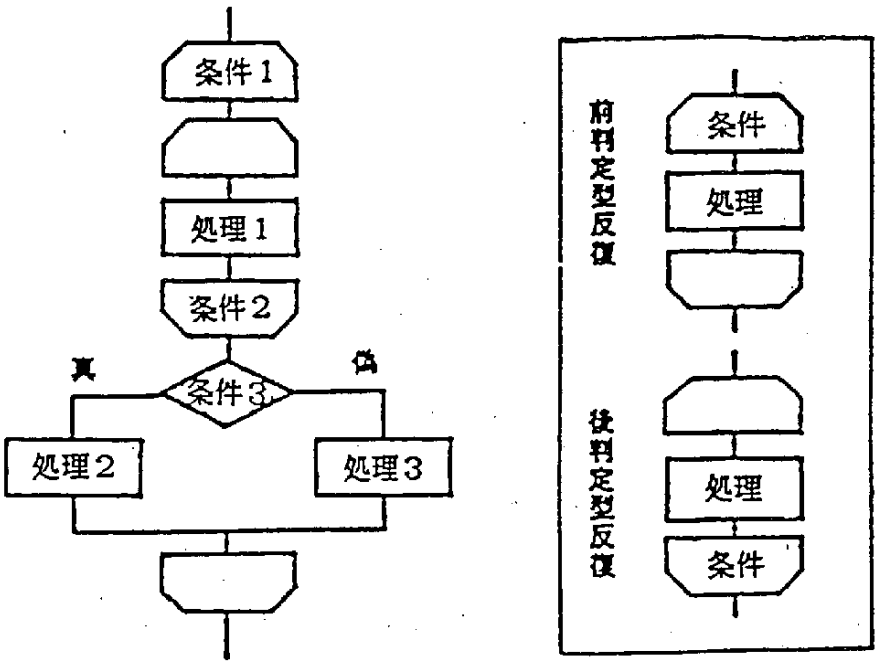
データの表現については，ＨＩＰＯではデータを入力と出力の２種類に

分けているのに対し，ＨＣＰではインタフェースデータとワークデータ（つまり外部とやりとりするデータと内部だけで用いるデータ）に分けて考える点が異る。

上述のように，記号の種類が豊富であり，かつデータ記述が可能な点は，電子交換という電電公社特有のソフトウェア開発業務を意識した図法であることを窺わせる。しかし，表現力の高さとは裏腹に，図が複雑になってくると，制御構造を表わす線とデータの流れを表わす線が錯綜して，著しく見難くなるといった問題もあり，この辺はもうひと工夫欲しいところである。

(8) 新プログラム・フローチャート（図表４－１６）

図表４－１６　新プログラム・フローチャート



（日経コンピュータ，１９８４．１．９号「プログラム制御構造の新しい表現技法：木構造チャートが普及期に」より転載）

木構造チャートとは異り，フローチャートそのものに反復構造を表わす記号を導入して，構造化プログラミングに適した記法に改良しようという試みがＩＳＯによって進められている。しかし，反復が入れ子になると見難くなるなど，まだ改良の余地がありそうである。

(9) 疑似コード（pseudo-code）あるいはプログラム設計（／記述）言語（PDL：Program Design（／Description）Language）（図表４－17）

図表４－17　疑似コードによる記述例

```
在庫データ照会
    データ宣言
    初期設定
    DO WHILE トランザクションあり
        妥当トランザクションを得る
        IF 妥当トランザクション THEN
            該当在庫レコードを得る
            IF 該当在庫レコードあり THEN
                IF トランザクション＝A THEN
                    入出庫活動表作成
                ELSE
                    在庫状況表作成
                    IF 在庫量＜最低保有承 THEN
                        発注メッセージ印刷
                    END IF
                END IF
            ELSE
                エラー・メッセージ印刷
            END IF
        ELSE
        END IF
    END DO
終り
```

（国友義久著「効果的プログラム開発技法」（近代科学社）より転載）

上記の(1)～(8)がすべて，多かれ少なかれ図による記法であったのに対し，いっさい図は用いず，文章だけで処理を記述しようとするのが疑似

コードである。これはまた，プログラム設計言語あるいはプログラム記述言語とも呼ばれる。歴史的にはやや古く，NSチャートと同じ1974年頃にIBMのIPTの中の1手法として発表されている。

その基本的な考えは，処理の手順をそのまま日常用いている国語（に良く似た言語）で表わそうというもので，タイプライターひとつで自由に記述・変更でき，プログラム・コードへの移行も容易である。

基本的な制御構造はキーワードを用いて表わし，プログラム自体の階層構造は字下げで表わすが，図形を使用しないだけに，複雑な構造になると見難くなるのが欠点である。

一般に日本やヨーロッパでは図式表現が好まれ，アメリカでは疑似コードが良く使われているようである。

⑩ 決定表（decision table）（図表4－18）

図表4－18　決定表

条件スタブ	条件エントリ
行動スタブ	行動エントリ

（a）　決定表の形式

a＝0？	Y	Y	N	N
b＝0？	Y	N	Y	N
x＝b/a			X	X
不　定	X			
不　能		X		

（b）　制限エントリ型の決定表

a	= 0	= 0	≠ 0
b	= 0	≠ 0	
x=	不 定	不 能	b／a

（c） 拡張エントリ型の決定表

a = 0 ?	Y	Y	N	N
b = 0 ?	Y	N	Y	N
x =	不 定	不 能	b／a	b／a

（d） 混合エントリ型の決定表

（「図形入力を加味したプログラミング技術，昭和５６年度」
（ソフトウェア産業振興協会）より転載）

いろいろな条件が複雑に入り組んでいて，それらの条件の組み合せで処理の内容が決るような場合には，決定表を用いるのが便利である。決定表の歴史は古く，これからソースコードを生成するプロセッサも開発されている。

決定表の構成は図に示すように，大きく４つの部分に分けられている。左上の条件スタブには検査すべきすべての条件を質問の形式で書き，左下の行動スタブには上に挙げた条件の組み合せで可能になるすべての行動のリストを書く。右上の条件エントリには条件スタブの質問の答え(である条件）を，右下の行動エントリには答えの組み合せに対して取るべき行動を，それぞれマトリックスの形で記入する。条件の答えをＹＥＳ／ＮＯで書く制限エントリ型，式または文章で書く拡張エントリ型，それらを混用する混合エントリ型の区別がある。

決定表は制限の流れを表わすことができないのが大きな欠点で，フロ

ーチャートや木構造チャートにとって代るものではなく，むしろ補助的に使うべきものと言えよう。

4.1.3 レビュー技法

レビュー（ミーティング）は，主として開発の主要局面の区切りごとに，プロジェクトの状況把握，今後の計画の見通しと軌道修正の方策，プロジェクトの成果物の技術的評価などを目的として行われる検討会のことである。レビューそれ自体は，従来でも各企業・組織などで多かれ少なかれ実行されて来たもので，とくに目新しくはないが，ＩＢＭのＩＰＴの中に組み込まれている構造化ウォークスルーやインスペクションは，その目的や実行手順などを明確に規定して，レビューの１手法としての位置を確立した点に大きな意義がある。

(1) 構造化ウォークスルー（structured walkthrough）

従来のレビューがとかく技術的な検討をなおざりにし，失敗の責任のみを追及するといった，魔女狩り的イメージに片寄り勝ちだった点を反省し，出席者全員による民主的な技術検討会を目的としたのが構造化ウォークスルーである。

これが従来のレビューと異る主な点は次のようなものである。

- レビューを受ける人（レビューイ）が会を主催し，スケジュールする。会の出席者もレビューイが選び，問題点を指摘する能力のある人（人数は４～５人）に限る。
- 管理者は原則として出席しない。これはウォークスルーの結果を個人の評価材料にしないためである。
- レビューイはレビューする立場の人（レビューア）に事前に資料を配付しておく。レビューアは会の前までに資料の内容を十分に理解しておかなければならない。
- エラーの発見を中心に行い，その場では訂正はいっさい行わない。見

つかったエラーは文書化しておき，後日レビューイがまとめて訂正作業を行う。

- ウォークスルーの時間はできるだけ短時間（1～2時間）にする。時間内に終らないときは，日を改めて再実行する。

(2) インスペクション（inspection）

構造化ウォークスルーがかなりインフォーマルな検討会であったのに対し，もっと形式的な色彩を強めたのがインスペクションである。

インスペクションも技術的な検討会という意味ではウォークスルーと違いはないが，後者と比較して以下のような特徴を持つ。

- エラーを見つけるだけでなく，開発の各局面の終りに達成しておかなければならない完成基準（exit criteria）を定義しておき，その局面での出力（成果物）が基準を満すか否かをチェックする。
- 発見されたエラーの修正や完成基準の確認に対して正式の承認を必要とする。承認を与える責任者としてモデレータを設ける。
- インスペクションに対するすべての責任はモデレータが持つ。モデレータは出席者の選定やスケジュールを行う。モデレータは検証されるべき出力の開発者ではなく，また開発チームのメンバーでなくても良い。
- インスペクションは6つのステップから構成される。すなわち，計画，概要説明，準備，ミーティング，再作業（エラー修正），フォローアップである。
- ウォークスルーでは準備作業として資料の事前配布が行われたが，インスペクションではさらにこれを強化して，参加者全員に対して資料の概略説明（overview）を行うほか，エラーのチェックリストなども配布する。
- ウォークスルーではエラー修正はすべて開発者の責任で行われたが，インスペクションではモデレータが開発者と連絡を密にして，エラー

修正が正しく行われたかどうかをフォローアップする。

・インスペクション技術の向上のために，インスペクション実行のたびにエラーのタイプや発生頻度などのデータを収集・分析して後の用に役立てる。

4.1.4 テスト技法

テストは大きく静的解析と動的解析に分けられる。静的解析はプログラムの実行を伴わないもので，これにはプログラムのエラーや構造のまずさなどを検出する静的エラー検出や，実際の数値データの代りにシンボリックな変数値を用いて，出力を評価したり，パス選択の様子を調べるシンボリック実行などが含まれる。

動的解析はプログラムを実行させてその正しさをチェックするもので，通常「テスト」と呼ぶものはこの動的解析を指す。以下，本項で述べる技法は，主としてこの狭義のテストに関するものである。

テストの戦略としてはモジュールの上位から下位に進めるトップダウン・テスト，下位から上位に進めるボトムアップ・テスト，上下位同時に行う一斉テストおよびビッグバン・テストがある。

テストの段階としては上記のいずれの戦略を取った場合でも，基本的には，単体テスト，結合テスト，システム・テストの3段階に分けて実行されるようである。（場合によっては，ある段階がさらに分割されたり，省略されたり，2つの段階が1つにまとめられたりして，段階の数が変化することもあり，またそれぞれの段階が異った名前で呼ばれたりすることもあって，上記の分類は必ずしも確定的ではないが，これらは上記のバリエーションと考えて差し支えない。）

単体テストで評価すべき特性としては，モジュール・インタフェース，局所データ構造，重要な実行パス，エラー処理パス，およびこれらすべてに影響する境界条件，の5つが挙げられる。結合テストではモジュール間

のインタフェースがテストされる。上記のテストの戦略は主としてこの段階に関係する問題である。システム・テストはいくつかのソフトウェアを1つのシステムに組み込む場合の統合的なテストのことで，ソフトウェア工学の範囲からはやや離れるのでここでは論じない。

テスト・ケースの設計法としては，プログラムの内部構造を調べて論理が正しいか否かをテストするホワイトボックス・テストと，プログラムの内部構造や動作には関知せず，仕様書通りの動き方をするか否かだけをテストするブラックボックス・テストがある。前者の代表的なものに論理網羅，後者の代表的なものに同値分割，境界値分析，原因＝結果グラフなどの技法がある。

また，エラー予測のための技法としては，バグ管理図や埋め込みモデル（キャプチュア・リキャプチュア）などがある。

以下，テストの戦略，テスト・ケースの設計法，エラー予測のための技の3つの場合に分けて，個々の技法について説明する。

A．テストの戦略

(1) トップダウン・テスト

トップダウン・アプローチによるテスト法で，上位のモジュールから下位のモジュールへとテストを進めて行く。

この方法では主要な制御構造からテストされて行くし，最初の段階からインタフェースの検証が行えるので，致命的なエラーがあとになって見つかるといった危険性が少ない。また下位モジュールのテストには作成済みの上位のモジュールが使用できるため，システムの本番稼働時と同じ環境下でテストでき，インタフェースのエラーを減らすことができるばかりでなく，エラーの発見も容易になる，などの利点がある。

しかし，上位のモジュールをテストする時点で，まだ下位のモジュールが作成されていないので，下位モジュールが返すべきデータをシ

ミュレートするためのスタブが必要になるのが欠点である。複雑なプログラムの場合，要求をうまく満すようなスタブを設計するのはかなり大変な作業である。

(2) ボトムアップ・テスト

ボトムアップ・アプローチによるテスト法で，上記の方法とは逆に，下位のモジュールから上位のモジュールへテストを進めて行く。

ボトムアップ・テストの長所・短所は，トップダウン・テストのそれらをそのまま裏返した恰好になる。たとえば，この方法ではスタブを必要としない代りにテスト・モジュールを制御するためのテスト・ドライバーを作成する必要がある。テスト・ドライバーはシミュレーションのプログラムであるから，実際の稼働時とは似て非なる環境しか実現することができない。従って，本当に稼働時と同じ状況でテストできるのは，すべてのモジュールが完成する最後の段階でしかないことになる。それまではインタフェースを仮定したままテストを進めなければならないので，エラーが発見されたときにその原因を見つけることが容易でない。

しかし，最初にモジュール単位でテストされるから，具体的なイメージが摑み易く，いろいろなパスを試すことができて，徹底的なモジュール・テストが行えるのは利点である。

設計法と同様にテストにおいても，トップダウンとボトムアップの優劣の決着はにわかにはつけ難い。一般にプログラム構造の上位レベルのテストに対してはトップダウン，下位レベルのテストに対してはボトムアップを組み合せた複合アプローチが最良とされている。

(3) 一斉テスト

上記の2つのテストは順序の違いこそあれ，いずれもモジュールを作成するそのつど，テストを実行する点では一致している。一斉テストはモジュールがすべて完成してから，それらを結合したプログラム

を1つのモジュールと見て，一度にテストしてしまう方法である。いわばテストの手抜きであり，うまくテストに合格した場合はコストが少なくて済む。しかし，実際には多かれ少なかれエラーが存在することは免れないから，結局は上記のような方法で部分的に攻めて行かなければならないことになり，かえってコストが高くつく恐れが多い。

あまり推奨できない方法である。

(4) ビッグバン・テスト

一斉テストの改良版ともいうべきもので，各モジュールについて単体レベルのテストを行ったあと，一斉テストを実行するものである。従来，最も多く行われて来た方法であるが，単体テストのためにスタブ，テスト・ドライバーの両方が必要である。

B．テスト・ケースの設計法

(1) 論理網羅（logic coverage）

すべての論理パスをテストすることは理論的には可能であっても，実際にはパスの数が天文学的な数になってしまうため，事実上不可能である。従って，単体テストにおいても，テストされるパスはいくつかの重要なものだけに限られる。

論理網羅はパス・テストの基準をより完全なものに向上させるためのテスト手順を与える手法である。網羅の基準を弱いものから強いものの順に示すと，次のようになる。

① 命令網羅（statement coverage）

すべての命令を少なくとも1回は実行するようにテストすること。

② 判断網羅（decision coverge）

すべての判断について真と偽の結果を少なくとも1回はとるようにし，かつ入口点が少くとも1回は呼ばれるようにテストすること。

③ 条件網羅（condition coverage）

または分岐網羅（branch coverage）

判断におけるあらゆる条件で，すべての可能な結果を少なくとも1回はとるようにし，かつ入口点が少なくとも1回は呼ばれるようにテストすること。

④ 判断／条件網羅（decision／condition coverage）

1つの判断でのそれぞれの個別条件で，すべての可能な結果を少なくとも1回はとるようにし，かつそれぞれの判断が可能な結果を少なくとも1回はとるようにし，さらに入口点が少なくとも1回は呼ばれるようにテストすること。

⑤ 複数条件網羅（multiple-condition coverage）

それぞれの判断における条件付き結果の可能なすべての組み合せと，すべての入口点が少なくとも1回は試されるようにテストすること。

なお，IF（A OR B）THEN ……のような場合，IFのあとの（ ）全体を「判断」，（ ）の中の“A”とか“B”とかを「条件」と呼ぶ。

(2) 同値分割（equivalence partition）

これはブラックボックス・テストを行う場合，できるだけ少ないテスト・ケースで，できるだけ多くのエラーを発見するような，適切なテスト・ケースのサブセットを見出すための手法である。

そのためにはテスト・ケースをいくつかの同値クラスというものに分割し，その同値クラスの中ではどれか1つの要素についてテストすれば，他の要素については同じテスト結果が得られるようにする。それゆえ，テストは異った同値クラスについてのみ実行すれば良い。この方法ではまず入力の同値クラスを見分け，次いでそれを利用してテスト・ケースを定義する。

(3) 境界値分析（boundary value analysis）

一般にエラーは入力と出力の同値クラスの端およびその付近（これ

を境界条件という）の値で起り易い性質を持っている。同値分割ではテスト・ケースの値として同値クラスのどの値を選んでもかまわなかったが，境界値分析は境界条件にある値を選ぶことによって，エラー発見の効率を高める工夫をしている。境界値分析はこの境界条件を見出すための手法を与えるものである。

(4) 原因＝結果グラフ（cause-effect graph）

同値分割や境界値分析では入力状況の組み合せを明確にすることが容易でない。原因＝結果グラフは入力条件（原因）とそれに対応するアクション（結果）の因果関係を，見易く表現するための図法で，これはエラー発見効率の高いテスト・ケースを選択するのに役立つ。

このグラフで使用する基本的な記号の意味を図表４－１９に，その使用例を図表４－２０に示す。例では１～９の９つの原因から，その組み合せによって３１～３４の４つの結果が生じることを表わしている。２１～２３は中間的な条件である。このグラフから4.1.2 (4)に述べた決定表に変換し，それに基づいてテスト・ケースを作成する。

C．エラー予測のための技法

(1) バグ管理図

図表４－２１に示したように横軸に工数を取り，縦軸にテスト項目の消化件数と発見したバグ数を表示した図表である。元来，未解決バグ数がどれだけ残っているかを見るための管理図であるが，発見バグ数の曲線を外挿することによってバグの総数（従って，未発見のバグ数）を推定することができる。

バグ数の曲線は近似的にゴンペルツ曲線に従うことが知られている。これによると時間（工数）ｔにおけるバグ数Ｎは次式で与えられる。

$$N = K \cdot a^{bt}$$

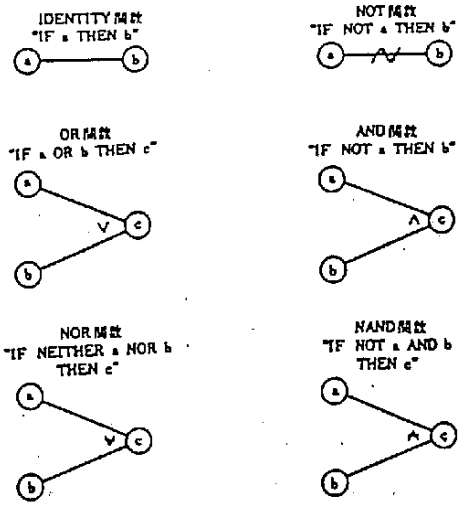
ただし，　Ｋ：バグ総数（予測），

$0 < a,\ b < 1$：曲線の形状を決定する定数，

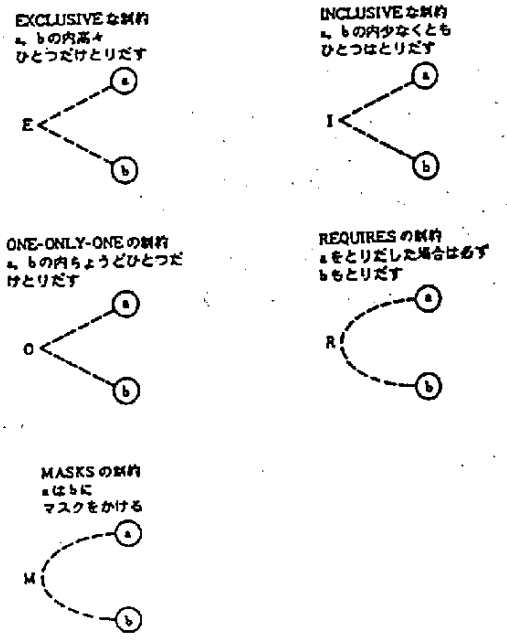
である。

図表４－１９　原因＝結果グラフで用いる記号

（ａ）　原因＝結果の基本的な関係

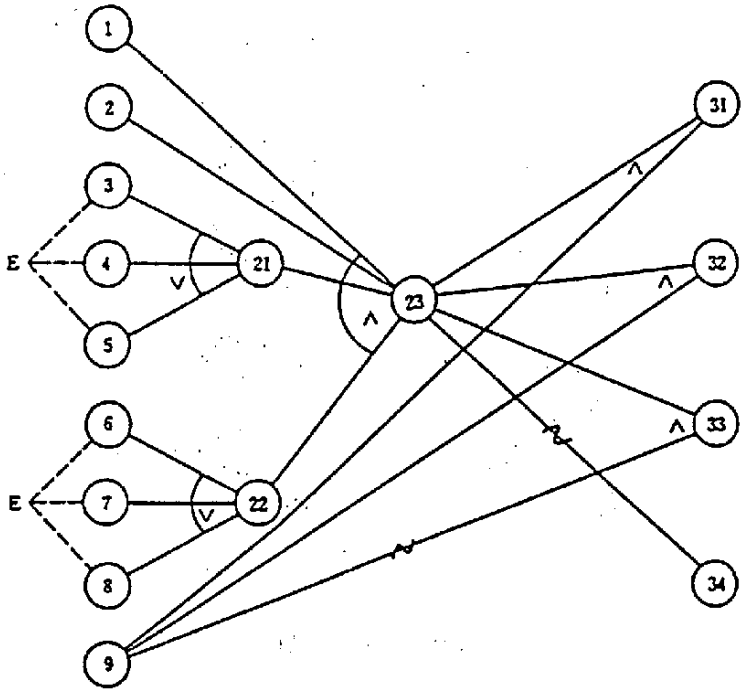


（ｂ）　原因＝結果の制約



（宮本勲著「ソフトウェアエンジニアリング：現状と展望」（ＴＢＳ出版会）より転載

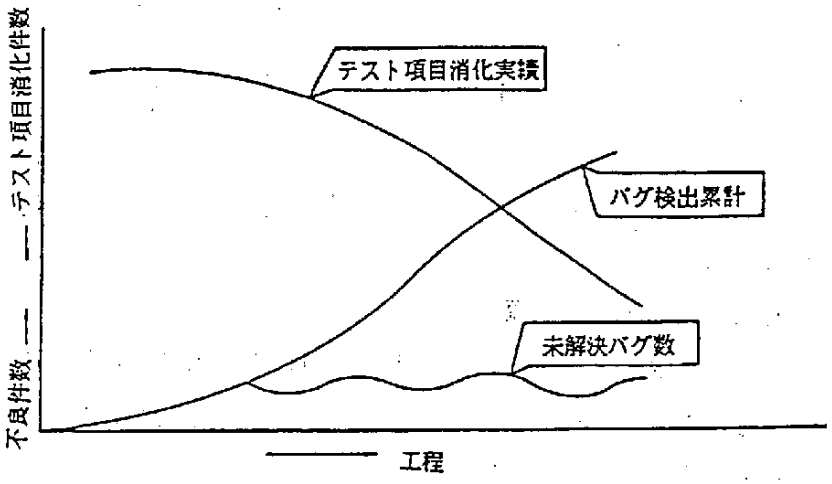
図表４－２０　原因＝結果グラフの例



原因 1.　「C」と１個以上の空白のあとにある．最初の空白でない文字は「/」である．
原因 2.　コマンドには，ちょうど2個の「/」の文字がある．
原因 3.　String1の長さが1である
原因 4.　String1の長さが30である．
原因 5.　String1の長さが2−29である．
原因 6.　String2の長さが0である．
原因 7.　String2の長さが30である
原因 8.　String2の長さが1−29である．
原因 9.　現在みている行に，String1が含まれる．
結果31.　書き換えた行が印刷された．
結果32.　現在みている行の最初のstring1がstring2でおきかえられた．
結果33.　NOT FOUNDと印刷された．
結果34.　INVALID SYNTAXと印刷された．

（宮本勲著「ソフトウェアエンジニアリング：現状と展望」（TBS出版会）より転載）

図表４－２１　バグ管理図



(2)　エラー埋め込みモデル（error seedingmodel）またはキャプチュア・リキャプチュア（capture・recapture）

故意に作ったバグをプログラムの中に埋め込み，テストによって検出された埋め込みバグの割合から，プログラム中に潜在する総バグ数を推定する方法である。

総バグ数Ｎとテストによって検出されるバグ数ｎの比は，埋め込んだバグ数Ｍとテストによって検出される埋め込みバグ数ｍの比に等しいと仮定されるから，次式が成立つ。

$$N = M \cdot n / m$$

従って，ある時点における埋め込みバグ数を除いた潜在バグ数ｌは

$$l = N - M - (n - m)$$

で与えられる。

これから信頼度９５％のＮの範囲を計算すると，次のようになる。

$$\frac{m}{n} - 1.96\sqrt{\frac{N-n}{N-1} \times \frac{m/n(1-m/n)}{n}} \leqq \frac{M}{N}$$

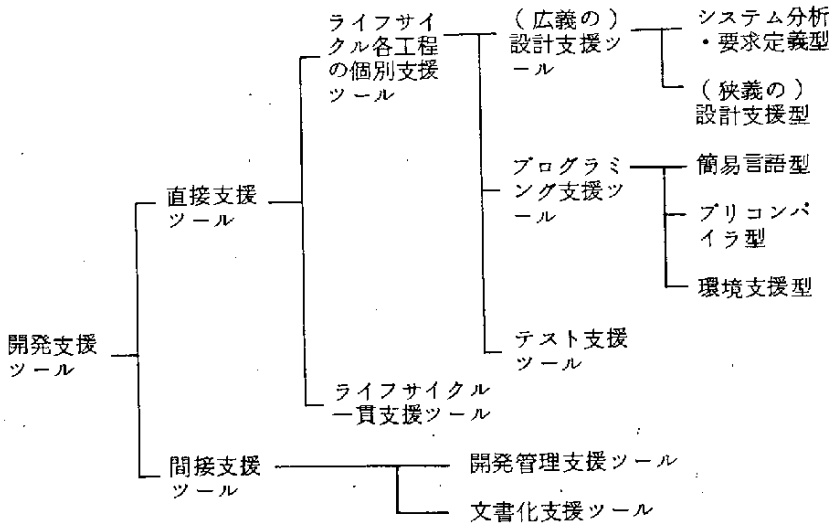
$$\leqq \frac{m}{n} + 1.96\sqrt{\frac{N-n}{N-1} \times \frac{m/n(1-m/n)}{n}}$$

4.2 開発支援ツール

4.2.1 開発支援ツールの分類

一口に開発支援ツールと言っても，その目的・用途・規模など千差万別で，これらをきれいな形に分類して示すのは容易ではないが，ここでは一案として図表4－22のように分類してみた。

図表4－22　開発支援ツールの分類



まず，開発支援ツール全体を直接支援と間接支援とに分ける。直接支援とはソフトウェア開発の作業そのものを支援するツールのことで，これにはライフサイクルの各工程を個々に支援するものと，全工程を一貫して支

援するものとがある。

従来の支援ツールは，ほとんど前者の個別支援型のツールに限られていたが，これは工程別に（広義の）設計支援，プログラミング支援，テスト支援の３つに分類される。保守・運用支援というものもライフサイクルの上からは考えられるが，これは間接支援の方に含めて考える。

広義の支援ツールの中にはシステム分析・要求定義を主とするものと，狭義の設計を主とするものとがある。前者に属するものではＩＲＷ社のＳＲＥＭや，ミシガン大学ＩＳＤＯＳプロジェクトのＰＳＬ／ＰＳＡ，イギリスSystem Designer社のCOREなどが有名である。狭義の設計支援ツール，特に基本設計段階を支援するツールはあまり見当らないが，ここではその数少ない一例として，ＰＲＩＤＥの方法論をツール化したＡＳＤＭを挙げておく。

プログラミングとテストの工程を支援するツールは多数あり，従来の支援ツールの大半はこの分野に集中している。テスト支援ツールのさらに詳細な分類は4.2.4で行うので，本項ではプログラミング支援ツールについてのみ説明を加える。

プログラミング支援ツールには，非手続き的な簡易言語を用いて直接オブジェクトコードを生成する簡易言語型と，あるプログラム言語のソースコードを生成するプリコンパイラ型とがある。前者の代表的な例としてＭＡＲＫⅣ，ＥＡＳＹＴＲＩＥＶＥ，ＮＡＴＵＲＡＬなどがあり，後者の代表的な例としてはＨＹＰＥＲＣＯＢＯＬ，ＪＳＰ-ＣＯＢＯＬ，ＣＯＲＡＬ，ＪＡＳＰＯＬ，ＳＰ-ＦＯＲＴＲＡＮなどがある。

上のような区分とは別に，ＵＮＩＸのＰＷＢ（Programmer's Work Bench）のようにプログラマの作業環境を整備・向上させることにより開発支援を行おうとする環境支援型のツールがある。これは本来，ライフサイクル全体に適用される一貫支援型のツールであるべきものであるが，現状では僅かの例外を除いて，プログラミング～テスト工程の支援に止まっ

ている。

従来のツールが個々バラバラで，有機的なつながりがないため，きわめて使い難いものであった点を改善し，ライフサイクル全体を首尾一貫して支援しようというのが一貫型の支援ツールである。しかし，あらゆる分野に適用できるオールラウンド型の一貫支援ツールというのはまだ実現されておらず，たとえば日本電気のSEA／Iでは主に事務処理用のCOBOLプログラムを対象とするなど，ある特定分野に適用を絞ったツールが多い。この中で，東洋情報のCADAPはドキュメントの作成支援を主眼とすることで，オールラウンド型のソフトウェアCADを意図している点は注目に値する。

間接支援型のツールとしては開発管理支援と文書化支援のツールがある。

4.2.2 設計支援ツール

ここでは要求分析・仕様ツールと狭義の設計支援ツールから代表的なものを選んで解説する。

(1) SREM（Software Requirements Engineering Methodology）

これはTRW社がアメリカ陸軍のBMDATC（弾道ミサイル防衛高等センター）のために開発した自動化要求分析・定義技法／ツールである。もっと正確に言うと，その実体は要求記述言語RSL（Requirement Statement Language）と支援ツール群REVS（Requirement Engineering and Validation System）とから成り，それを統一する方法論がSREMである。

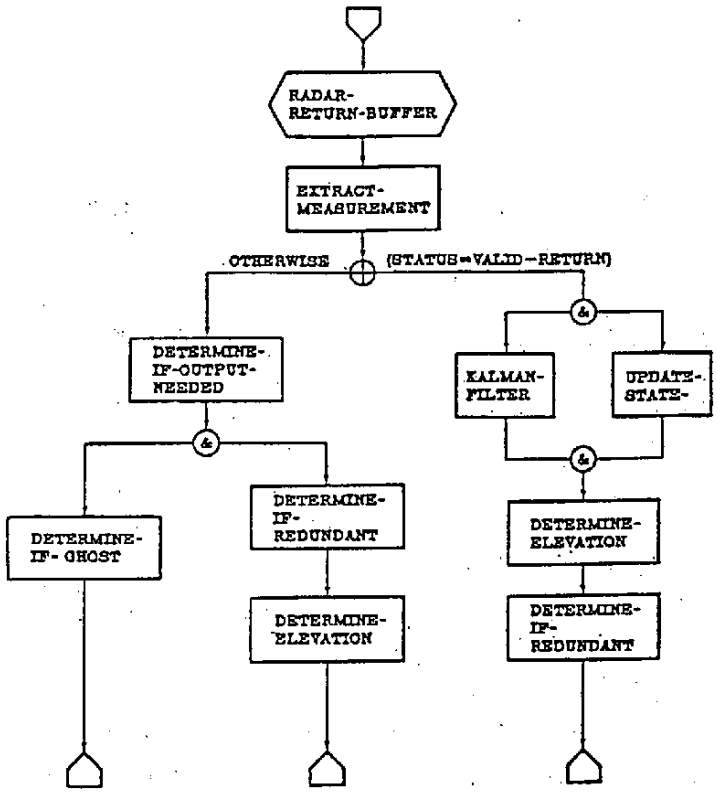
SREMはもともと，実時間システムのための要求定義技法として開発されたところから，要求を「処理の流れ」として把えるが，RSLはそれを記述するのに「要素」，「関係」，「属性」，「構造」の4つの基本的概念を用いる。要素は要求を記述するための対象ないしは概念を表わし，関係はそれらの相互関係を，属性は要素が持つ性質を表わす。構

造は要素を使って組み立てられる処理の流れのモデルのことで，全体の構造はR－NET（Requirements-Net）と呼ばれるネットワークで表わす。図表４－２３にはR－NETとそれに対応するRSLの記述例を示した。

REVSはRSLで記述する仕様を作成・保守・検証・文書化するためのシステムで，①RSLの処理，②会話型R－NET処理，③要求仕様

図表４－２３　SREM

（a）　R－NETによる構造の記述例



（b）　ＲＳＬによる記述

```
R-NET PROCESS-RADAR-RETURN.
    STRUCTURE
        INPUT-INTERFACE RADAR-RETURN-BUFFER
        EXTRACT-MEASUREMENT
        DO(STATUS=VALID-RETURN)
           DO UPDATE-STATE AND KALMAN-FILTER END
           DETERMINE-ELEVATION
           DETERMINE-IF-REDUNDANT
           TERMINATE
        OTHERWISE
           DETERMINE-IF-OUTPUT-NEEDED
           DO DETERMINE-IF-REDUNDANT
                DETERMINE-ELEVATION
                TERMINATE
           AND DETERMINE-IF-GHOST
                TERMINATE
           END
        END
    END.
```

（国井利泰編「bit 増刊，ソフトウェア・プロダクト工学」
（共立出版）より転載）

の解析と文書化，④シミュレーション機能，⑤拡張ＲＳＬ処理，の５つのツールから構成されている。REVSは，要求仕様は最初から明確に決められるものではなく，検討・修正を重ねて次第に固まって行くものであるという考えに立っており，そのため仕様を繰返し修正・入力できるようになっている。

ＳＲＥＭは実際に要求仕様を行う際の手順を定めているが，それは次のようなものである。

① 解釈と翻訳（interpretation and translation）

システム仕様を分析して，詳細なデータ記述と処理段階に変換する。

② 分解（decomposition）

要求機能の詳細仕様を決定し，その一貫性や完全性を静的および動的解析によってテストする。

③　割り付け（allocation）

性能要求を仕様化して，それによって拘束される処理のパスを決定し，それらに性能要求を割り付ける。

④　実現可能性の実証（analytical feasibility demonstration）

シミュレーションによって，要求のシステムが実現できるか否かを検証する。

上記①，②の手続きでは主としてRSLが，③，④では主としてREVSが使われる。

(2)　PSL／PSA（Problem Statement Language／Problem Statement Analyser）

これはミシガン大学におけるISDOSプロジェクトで開発され，CADSATと呼ばれる大規模システムの一部に組み込まれている要求分析・仕様ツールである。

構成はSREMと似ており，要求記述言語PSLと，PSLで記述された仕様を分析したり，それから文書を作成したりするツールPSAとから成る。

要求定義は，①システム入出力の流れ，②システム構造，③データ構造，④データ導出，⑤システム・サイズ，⑥システム・ダイナミックス，⑦システムの性質，⑧プロジェクト管理，の8つの局面について行う。

情報システムの記述は，その「構成部分」（オブジェクトと呼ばれる），それぞれの「性質」，およびそれらのあいだの「関係」の3つの基本的要素で表わし，それらをすべて2項関係としてPSAデータベースに格納する。

PSAはこれらの関係を基にして①構文，②入出力の完全性，③データ構造定義の完全性，④データ構成要素間の論理構造の矛盾，⑤処理ブロックの親子関係，に関してチェックを行い，種々のドキュメントを作成する。

(3) ＡＳＤＭ

これはPRIDEに基づいた設計支援ツールの総称で，その中核を成すのはシステム・ディクショナリ／ディレクトリのＩＲＭと，システム設計自動化ツールのADFである。

ＩＲＭは既存のシステムに関する情報をすべて管理するもので，情報要求，組織，システム，サブシステム，人間のプロシージャ，コンピュータ・プロシージャ，プログラム，モジュール，コール，人間のファイル，コンピュータ・ファイル，入出力，レコードの１３のコンポーネントとそれらの相互関係として定義する。

ADFはユーザの要求に対して，どのようなシステムを構築したら良いか，再利用できる既存のシステムがあるか，サブシステムの分割はどのようにするのか，などを教えてくれるシステムである。まずユーザの要求をまとめて，出力，レコード，データの定義を作成し，デザイン・パラメータと一緒にADFに入力すると，分析サブシステムがデータ・フロー分析を行って，出力分析，エラー分析，現状システム分析，新システム分析，保守用トランザクション分析，などの形で結果を出力する。この結果に必要ならば手を加えてデータを再入力し，レポーティング・サブシステムを起動すると，システム，サブシステム，プロシージャの3種類のフローチャートのほか，用語集，設計マニュアル，ユーザ・マニュアル，コンピュータのラン・ブック，などが出力される。これらの出力を基にして，プログラムやモジュールのコーディングが可能である。

4.2.3 プログラミング支援ツール

このジャンルに属する支援ツールは非常に数が多い，ここではデータフロー型とデータ構造型のプリコンパイラから，それぞれ代表的なものを１つずつ選んで紹介する。

(1) HYPER COBOL

これは富士通が開発した事務処理用COBOLプログラム作成のためのプリコンパイラで，データフローによる設計法に基づいている。

このシステムでは標準ユニットと呼ぶいくつかの機能モジュールを用意し，それらをSA（ストリーム・エリア）と呼ぶバッファを介して結合することにより，プログラムを作成する。標準ユニットには，たとえば入力データの正当性をチェックするCHECK，入力データを編集・選択するEDIT，2つの入力データを指定されたキーに従って突き合わせ，更新・追加などを行うMATCH，入力データをキー順にソートあるいはマージするSORTやMERGE，入力データ群から出力形式の指定に従ってレポート作成するREPORTなどがある。

利用者はこれらの標準モジュールにパラメータを入力するだけで良く，フローチャートなどを用いたプログラム設計が不要になった。工数の削減率は約50％と報告されている。

(2) JASPOL

日本システムサイエンス社が開発したJASPOLは，やはり事務処理用のCOBOLまたはPL／Iのプリコンパイラであるが，こちらはワーニェ法やジャクソン法などのようなデータ構造による設計法に基づくものである。

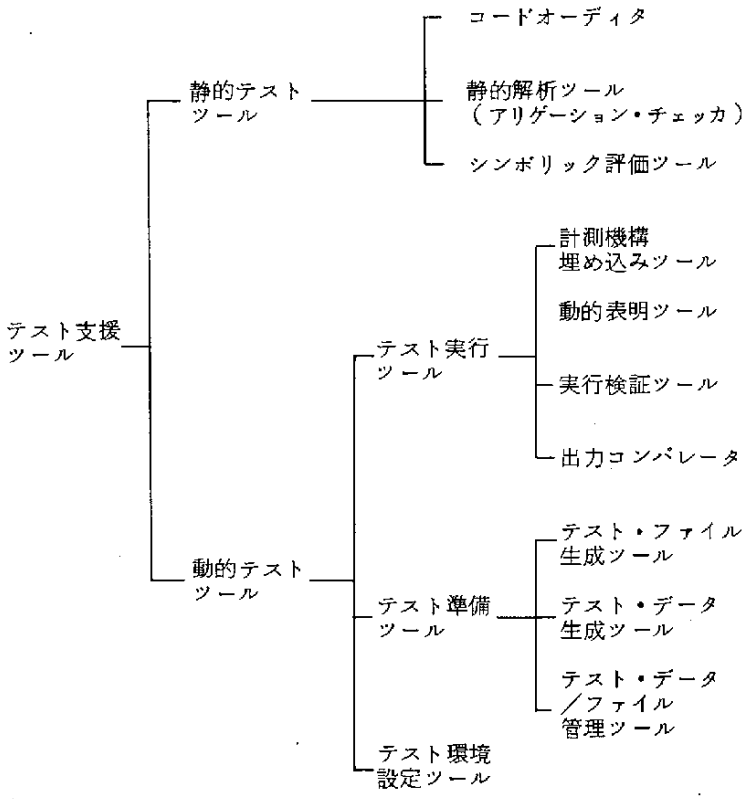
このシステムでは入力のデータ構造と出力のデータ構造の対応関係を標識などで宣言的に表明することによって，プログラムの構造を自動的に生成する。

JASPOLを用いた場合，従来の方法に比べて，ステップ数は⅓に，開発作業工数は¼に，デバック回数は⅓に減少するという報告がある。

4.2.4　テスト支援ツール

テスト支援ツールを用途別に分類すると図表４－２４のようになる。このジャンルに属するツールの数は非常に多い，ここでは個々のツールを挙げることはせず，種類別に簡単な説明を与えるに止める。

図表４－２４　テスト支援ツールの分類



(1)　プログラミング標準チェッカ

プログラミング規約に従って，プログラミングの記述の誤りをチェックする。

(2) 静的解析ツール

アリゲーションのチェックを行う。具体的には

・変数の初期値のセットの有無

・変数のスコープルール

・プログラムの制御構造

・サブルーチンのパラメータ

・サブルーチン・コールの順序

などをチェックする。

なお，アリゲーション（allegation：弱い表明）とは後述の「表明」ほど強くなく，満足しなくても誤りではないが，満足することが望ましいプログラムの状態を言う。

(3) シンボリック評価ツール

プログラムをコンパイル・実行してテストするのではなく，ソースプログラム上でシンボリックなデータを与えて評価する。つまり，実際の入力値でなく，"A"とか"B"のようなシンボリック値を用いる。

このツールはプログラ内のパスを選択する機構，そのパスをシンボリックに評価（解析・実行）する機構，および変数のシンボリックやパス述語などのシンボリック出力を生成する機構が必要である。

(4) 計測機構埋め込みツール

プログラム中に埋め込まれた計測機構により，実行中のプログラムの振舞いに関する以下のような情報を出力する。

・全実行ステートメントの実行回数と実行割合

・分岐ステートメントの実行回数と実行割合

・サブルーチンコールの実行回数と実行割合

・各サブルーチンの実行回数

・サブルーチン間の相対的な実行時間関係

・特定変数の最初の値，最後の値，最大値，最小値

など。

(5) 動的表明ツール

プログラム中に埋め込まれたアサーション（assertion：表明。その時点において満足されるべきプログラムの状態の表現）を，そのプログラムが満足しているか否かを実行中にチェックする。

(6) 実行検証ツール

テスト担当者の指示により，プログラムにスタグ，ドライバーなどのテスト用の装備を施して，そのプログラムを実行し，実行中のデータの収集を行う。そして，実行後に収集されたデータの分析評価を行ったり，テストの網羅率（カバレージ）の評価を行う。

(7) テスト・ファイル生成ツール

プログラムのファイル定義文を調べ，テストに用いるためのダミーファイルを作成する。場合により，生成したファイル内に「典型的な値」を格納することがある。

(8) テスト・データ生成ツール

プログラム構造を分析して，プログラムの実行パスを抽出し，そのパスが実行されるような入力データの組を生成する。

(9) テスト・データ／ファイル管理ツール

実行テストに用いられるテスト・データ，テスト・ファイルを一括管理し，効率の良いテストを可能にする。

(10) 出力コンパレータ

新旧2つのプログラムの出力を比較する。

(11) テスト環境設定ツール

実際のセンタ・マシン，端末装置の動作をシミュレートする模擬センタ，模擬端末や，大型マシンを用いてシステムの動作環境を設定し，その上で被テストプログラムを実行して，実行時のデータを収集する。

4.2.5 開発管理支援ツール

開発管理支援ツールとしては，PRIDE，SDEM，HIPACEなどの開発標準の一部として組み込まれているものや，一般のプロジェクト管理用のものなど数多くあるが，ここではとくにソフトウェア開発用に作成されたツールについて述べる。

(1) PMP（Project Management Program）

メルコム・オキタック・システムズが開発した日程管理，工程管理のためのツールである。

特徴としては以下の諸点が挙げられる。

① 多重プロジェクトが管理できる。

② 複雑なネットワークが取り扱える。

③ ネットワークをライブラリに登録して再利用できる。

④ 出力としては，時間分析，進捗状況，資源山積みにおける各種報告書のほかに，ネットワーク図の表示が可能である。

(2) PCMP（Project Cost Management Program）

日本電子開発が開発した原価管理（開発費用管理）のためのツールである。

特徴としては原価管理を原価計算，見積り支援，要員管理支援の3つの機能として把え，プロジェクト群管理，階層管理，多次元管理の考え方から総合的に管理する。

プロジェクト群管理ではプロジェクト相互の関係を把握して，それらを共存的に管理し，階層管理では各管理レベル（階層）ごとに適合した計画・管理を行い，多次元管理では日程，費用，要員などの複数管理要素によって，目的に応じた多面的な計画・管理を行う。

以上，本章では，ソフトウェア開発に用いる開発技法と開発支援ツールについて紹介した。アンケート結果によると，最も良く用いられている技法，ツールの上位10は次のとおりである。

HIPO, IPT, MARKⅣ, DANVALET, SDEM, PFD, 構造化チャート, STEPS, PRIDE, COBOL／S。

参 考 文 献

1) 「bit増刊, ソフトウェア・プロダクト工学」国井利泰編, 共立出版

2) 「ソフトウェアエンジニアリング：現状と展望」宮本勲著, TBS出版会

3) 「効果的プログラム開発技法」国友義久著, 近代科学社

4) 「ソフトウェア・エンジニアリング序説」ロジャー・S・プレスマン著, 岸田孝一監訳, 岡田正志訳, TBS出版会

5) 「図形入力を加味したプログラミング技術」昭和56年度, ソフトウェア産業振興協会

6) 「プログラム制御構造の新しい表現技法：木構造チャードが普及期に」1984.1.9号, 日経コンピュータ

5. 要員管理のための具体的方法

5. 要員管理のための具体的方法

5.1 コンピュータ要員の情報処理教育の実態

本節では情報処理要員がますます不足しつつある傾向の中で発生しつつある問題点を述べる。特にコンピュータ要員の構成がどのような実態になっているか，その推移傾向について調査結果を示す。このような実態の中で情報処理教育がどのように行われているか，一般的な教育，専門技術教育，教育に関する費用，要求される能力，教育の事例等について記述する。

また，要員管理の問題はそれぞれの企業の特殊性があって一概には述べられないが，アンケート調査を中心としその他資料も参考にして記述することにする。なお情報処理部門の高齢化は一般産業でいわれているのと異なり35歳停年説に言われるように比較的若い年齢における問題であるという点に特徴がある。

5.1.1 コンピュータ要員の推移傾向

本項では，コンピュータ要員の構成と要員の過不足の実態を分析し，その実態を述べることとする。

(1) コンピュータ要員の構成

通産省の「情報処理実態調査」によると，技術者全体のうち，プログラマー３８％（昭和５５年３７％，５１年３４％），システムエンジニア２２％（昭和５５年２０％，５１年16％），その他４０％（５５年４３％，５１年５０％）となっており，システムエンジニアの比率が年々高くなっている。オペレーション・パンチャーは年々比率が低くなっていて，また日本システミックスリサーチセンターの「システム部門の要員のアンケート調査結果（図表５－１，図表５－２）によると，過去５年間の増加の大きかった職種はシステム設計要員とプログラミング要員で，管理職，システム・コンサルタント要員も増加傾向にある。

また，減少の大きかったのはキー・エントリ要員，オペレーション要員，プログラミング要員となっている。プログラム要員の絶対数不足の中で，増と減の大きい両方の職種にあげられているのは，社内要員を外注化に転換したものと思われる。次に，今後5年間の増加が大きいと考えられる職種はシステム設計要員，システム・コンサルタント要員，高度コンピュータ技術者，プログラミング要員の順になっている。一方，減少が大きいと考えられる職種はプログラミング要員，オペレーション要員，キー・エントリ要員の順になっている。

プログラミング要員に今後更に外注化が進むことを意味しているものと推測される。

図表5－1　過去5年間増減の大きかった職種

	増　加	減　少
管　理　職	18％	2％
システム・コンサルタント要員	15％	1％
システム設計要員	36％	13％
高度コンピュータ技術者	9％	7％
プログラミング要員	33％	24％
運用管理要員	4％	14％
オペレーション要員	5％	25％
キー・エントリ要員	0	30％
一般事務要員	0	6％

N＝118

（出所）日本システミックスリサーチセンター「システム部門マネージメントの実態分析」

図表５－２　今後５年間増減の多い職種

	増　加	減　少
管　理　職	１％	４％
システム・コンサルタント要員	４３％	１％
システム設計要員	５６％	４％
高度コンピュータ技術者	２９％	１％
プログラミング要員	２４％	１５％
運用管理要員	８％	１４％
オペレーション要員	２％	１７％
キー・エントリ要員	０	１７％
一般事務要員	０	５％

（出所）日本システミックスリサーチセンター「システム部門マネージメントの実態分析」

(2)　コンピュータ要員の過不足

コンピュータ要員の不足傾向は著しく，通産省の「情報処理実態調査」によると，技術者の不足は必要に対してシステムエンジニア３３.２％，上級プログラマー２８.２％，初・中級プログラマー７４.４％，技術者全体で６５.６％となり，要員の充足が問題となっている。

これを昭和５５年度と比較するとシステムエンジニア（１.２％），上級プログラマー（５.８％）が充足状況が悪化し，初・中級プログラマーが３.９％良化している。

また，日本ＤＰ協会の「コンピュータ部門と実務担当者の実態と意識に関するアンケート調査結果」によっても同じような結果が出ており，そのＤＰ協会のまとめによると，

「低成長下で急激に人を増やせないこと，要員の育成にプログラマーで１～２年，システム設計者で５～６年，システム・コンサルタントの育成には更に多くの年月を要する事，を考えると，コンピュータ部門のシステム・コンサルタント，システム設計者，プログラマーの不足は恒常

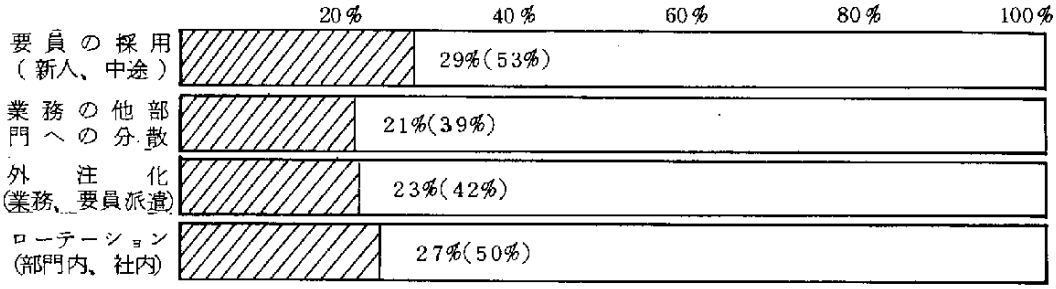
的なもので，今後共，まず，解消しないものと考えられる。これに対し，各企業に要員不足の対策として図表５－３に示すように要員の新規採用，システム開発の利用部門への分散，外注化により解決しようとしている。」と述べている。

上記以外の対応策としては標準化，自動化等による生産性向上策があげられる。

図表５－３　今後の要員・過不足の対策

N＝４３２人で７９６回答

（　）内は回答人数割合



（出所）日本データ・プロセッシング協会「コンピュータ部門と実務担当者の実態と意識に関する調査研究報告書」

5.1.2　要員に関する諸問題

コンピュータ要員の最大の問題は，要員の確保の面にある事は前節で述べた通りであるが，本節では，利用部門からコンピュータ部門へのローテーションの実態，処遇の問題，高齢化の問題，教育の問題に焦点をあてて，概要を述べる。

(1)　ローテーションの問題

職種にかかわりなく，他部門からの配置転換が困難であると５４.３％の企業が回答している。職種別には，プログラマーについて６１.６％，システムエンジニアについて５３.６％が配置転換が困難であることをあげている。（'８３コンピュータ白書）

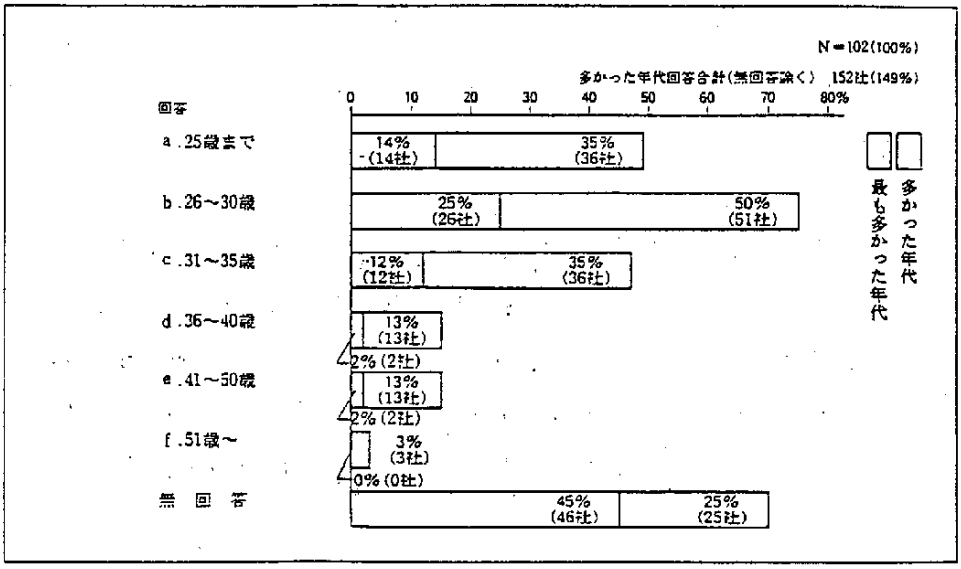
ＮＩＸのアンケート調査によると，コンピュータ部門は一般の男子社員のローテーション先として，一般の部門と全く同じ考え方であると回

答したものはわずか２０％であった。

あとは「若干」または「かなり」異っていると回答している。次に他部門からコンピュータ部門への転入者の年代は図表５－４に示す通り，最も多い年代は２６～３０才で，次いで２５才まで，３１～３５才迄の順になっている。これはほぼ，常識通りの結果であるが，若い年代でのローテーションでないと技術の進歩についていけないことを示しているものと思われる。しかし，３５才迄ならば，個人の適性，職種によっては配置転換が可能であるともいえよう。転入者が最初に担当した職種は図表５－５に示す通り，過去５年間ではプログラマー，システムエンジニア，管理者，オペレーターなどが多く，今後はシステムエンジニア，プログラマー，システム・コンサルタントの順番で多いと予測している。

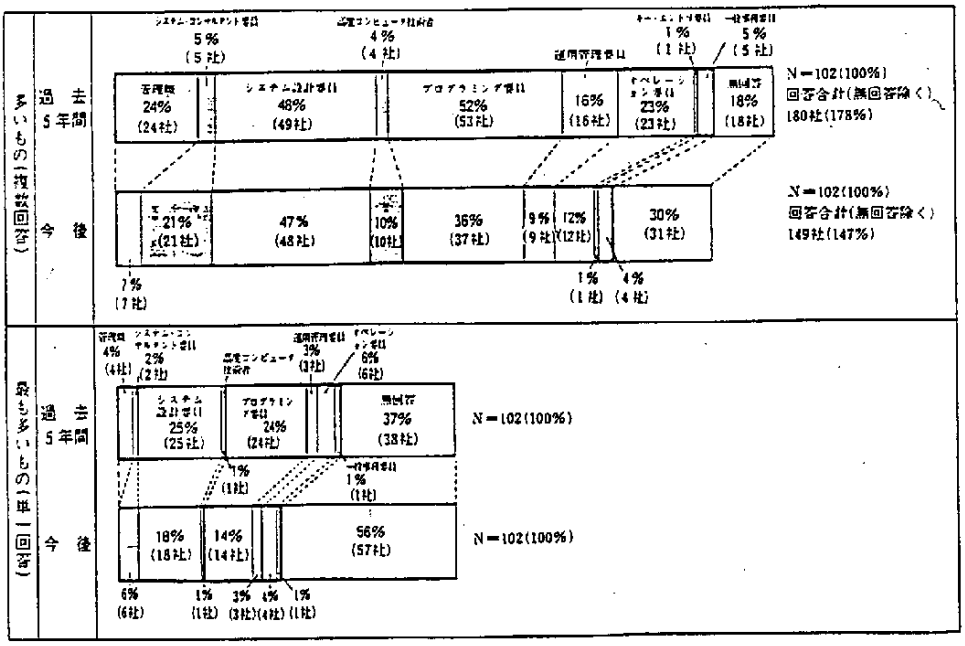
今後，減少する初期担当職種は管理職（２４％→７％），プログラマー（５２％→３６％），オペレーター（２３％→１２％）であり，今後増加する初期担当職種はシステム・コンサルタント（５％→２１％），高度コンピューター技術者（４％→１０％）となっている。

図表５－４　過去５年間の男子転入者の年代層



（出所）日本システミックスリサーチセンター「システム部門マネージメントの実態分析」

図表５－５　男子転入者が最初に担当した職種



（出所）日本システミックスリサーチセンター「システム部門マネージメントの実態分析」

これは個人の資質に左右され，特に教育の困難なシステム・コンサルタント，高度コンピュータ 技術者を他部門より配置転換させ，プログラマーは外注化により充足させようとの意図が伺える。オペレーター職は自動化，外注化により他部門よりの転入者が減少するとみているものと思われる。

(2)　処遇の問題

昇進，給与，労働時間が他部門と比べてどうかという問題について，ＤＰ協会のアンケート調査結果（図表５－６）によると，昇進，給与は「他部門とほとんど変わらない」と回答している。労働時間については６０％以上の人が「他部門と比べて長い」と回答している。バグログを大量にかかえている現状からすると，この状態は今後も続くものと思われる。

図表５－６　他部門との比較・処遇問題

処遇問題	不利である	やや不利である	他部署と変らない	やや有利である	有利である	合計
昇進の有不利	3.2	12.7	77.0	6.1	1.1	100.0
給与の有不利	0.8	4.0	88.2	6.1	0.8	100.0

処遇問題	短い	少し短い	他部署と変らない	やや長い	長い	合計
労働時間の長短	1.1	2.5	36.1	30.6	29.7	100.0

（出所）日本データプロセシング協会「コンピュータ部門と実務担当者の実態と意識に関する研究報告書」

スペシャリストとしての地位が確立していないという問題については，ＮＩＸ社のアンケート調査結果（図表５－７）によると，「専門職位制度を採用しているか」の設問に対し，一部実施と全面実施を合計しても２０％強しかならず，８０％弱の企業は「実施していない」と回答している。しかし，「実施していない」と回答した企業の６０％が「必要と思う」と回答しており，コンピュータ要員の年令限界との関連で，今後注目を要する課題である。

図表５－７　専門職位制度の実施状況

（必要と考えられる職種に対し）全面的に実施している	１１％
（必要と考えられる職種に対し）一部実施している	１０％
必要と思うが実施していない	４７％
考えていない	２８％
その他	１％
無回答	３％

Ｎ＝１０２

（出所）日本システミックスリサーチセンター「システム部門マネージメントの実態分析」

(3)　コンピュータ要員の年令限界

日本ＤＰ協会の「コンピュータ 部門と実務担当者の実態と意識に関するアンケート調査結果（図表5－８）（１９８３年度）によると，６０

%の企業でコンピュータ要員の高齢化が問題になっていると回答している。その中で，高齢化から起る問題点の主要なものとして，以下の3つをあげている。（図表5－9）

① 年齢に見合う役職やポストが準備出来ない。

② 技術革新に円滑な対応ができない等，知的能力面から業務に支障を生じる。

③ 実際に作業をする人が少なくなる。

この3つの中で最も深刻なのが①のポストの問題で全体の30%以上を占めている。次いで②の能力問題が16%，③の作業上の問題が14%で，体力面はそれ程大きな問題になっていない。

図表5－8 高齢化の問題

問題である	26%
ある程度問題になっている	34%
どちらともいえない	22%
ほとんど問題でない	8%
問題となっていない	10%

（出所）日本データプロセシング協会「コンピュータ部門と実務担当者の実態と意識に関する研究報告書」 N=456

図表5－9 高齢化の影響

知能面から業務に支障がある	16%
体力的な面から業務に支障がある	10%
モラールが低下する	6%
ポストが準備できない	32%
作業担当者が少なくなる	14%
業務遂行能力が向上する	5%
質の高いシステム作りができる	15%

（出所）日本データプロセシング協会「コンピュータ部門と実務担当者の実態と意識に関する研究報告書 N=477

(4) 教育の問題

今回のアンケート調査結果によると，「専門知識を持った要員教育」を５０％の企業がコンピュータ 部門の課題としてあげ，システム・コンサルタント，システム設計者，高度なコンピュータ 技術者の育成，確保が如何に難かしいかを示している。

一方，ＮＩＸ社のアンケート調査結果（図表5－１１及び図表５－１２）によると，要員育成に組織的に対応している企業は全体の半数しかなく，育成手段もＯＪＴに頼っている企業が多い。キャリア・パス制度，目標管理制度，要員の評価制度など，本格的な能力開発制度を重視している企業が少ない。これは専門知識を持った要員の確保が難かしいこと，今後のコンピュータ 部門が多種，多様な役割を担うことを考えると，重要な課題を含んでいるといえる。

図表５－１１　要員育成の担当組織

（単位：％）

独立した担当組織（課，係，グループ）がある	4
専任の担当（１～Ｎ名）がいる	7
兼任の担当がいる	２１
必要に応じプロジェクトを編成して検討を行なう	１４
要員育成問題に対する組織的対応はしていない（各管理の責任）	４９
そ　の　他	4
無　回　答	2

（出所）日本システミックスリサーチセンター「システム部門マネージメントの実態分析」　N＝102

図表５－１２　要員育成手段

（単位：％）

部門外とのローテーション	３９
部門内のローテーション	３９
キャリア・パス制度など職種到達目標の明示	３１
目標管理制度	２７
要員の評価制度	１３
ＯＪＴ	５９
集合教育	１７
グループによる研究会制度　勉強会	３４
システム部門内ＱＣ（的）活動	２２
業務の標準化，マニュアル化	５７
そ　の　他	6
無　回　答	３２

（出所）日本システミックスリサーチセンター「システム部門マネージメントの実態分析」

5.1.3　情報処理教育の実態

本節では我が国における情報処理教育の実態を分析しその状況を述べることとする。

(1)　一般的な情報処理技術の教育

ソフトウェア作成の業務量の増大に対し前述したような要員不足から，この補充のためには要員教育を充実せざるを得ない。そこで一般的な情報処理技術の教育はどのようになっているのかをみる。

一般的な情報処理技術の中味を分類すれば次のように分けられる。

- コンピュータの入門教育
- 要求定義技術
- システム設計技術
- プログラミング技術
- システム開発手順全般にわたる標準的な技術

今回の調査研究のアンケート結果によると上記のうち，コンピュータ入門教育とプログラミング技術については大部分が実施しているが，その他については比較的少ないことがわかった。

システム設計技術とシステム開発手順全般にわたる標準的な技術については約５０％が実施しているが，要求定義技術については１９％のみ実施している。

そこでこれらを詳細に見ると，まずコンピュータ 入門教育については全体の９７％が実施しており，平均実施期間は2.2週間である。３週間以上のものが２６％もあるということは，かなりの企業が入門教育に力を入れていることがわかる。

一番多いのが一週間の期間で４４％であった。

次にプログラミング技術についての教育は９０％が実施している。これに対してはかなりの期間をさいており，平均５週間である。この中で２週間以内の企業が３５％あり，一番多いものが４週間以上である。

次に，システム開発手順全般にわたる標準的な技術については，全体の半分が行っている。実施期間は平均2.2週間であるが，一番多いのが１週間の５３％である。４週間以上の企業が１３％ある。

システム設計技術については，４６％が実施している。平均3.8週間をかけており，一番多いのが一週間の４８％である。４週間以上かけているのが２１％である。

最後に要求定義技術については１９％のユーザーしか実施していないので一番問題となるものであろう。これは平均2.6週間実施しており，１週間実施が一番多く，６１％を占めている。

また，教育の方法をみると，社内（教育部やベテランＳＥ）による教育によって実施しているところが多く，７６％を占める。また，同時にメーカーのユーザー教育コースを活用している場合は６８％であり，これは一般的にどのユーザーでも積極的に活用していることがわかる。

さらに各種のセミナーを活用している例も多く，アンケートでは５９％がなんらかの形で利用していた。またビデオテープを活用した教育，即ちビデオツールの活用も約２０％のユーザーが活用していた。

以上のことから，一般的な情報処理教育はコンピュータの入門教育とプログラミング技術については大部分のユーザーが行っているが，システム設計技術，システム開発全般にわたる標準的な技術については約半分が，また要求定義技術については非常に少ないのが現状である。教育の方法については，社内のベテランによるもののほか，メーカーによる教育の活用は大部分のユーザーが行っている。

(2) ソフトウェアに関する専門技術の教育

ソフトウェアの専門技術に関する教育はＯＳ等の基本ソフトウェア技術に関する教育やオンライン・データベースについての教育が重点的に行われている。

専門教育を次の５つの分野に分類した。

- ＯＳ等の基本ソフトウェア技術
- オンライン・データー通信技術
- データベース技術
- マネージメントサイエンス技術
- ＣＡＤ／ＣＡＭ技術

この５つの分野についてアンケート調査の結果から分析する。

まず，ＯＳ等の基本ソフトウェア技術については８１％が実施している。実施期間は５５％が１週間を前提としている。また，４週間以上行っているのが１４％あった。また１週間未満が１６％となっている。

次に，オンライン・データー通信技術の教育については７０％が行っている。期間は１週間が一番多く５０％，次に２週間が２２％となっており，４週間以上行うのも９％あった。オンライン・データー通信技術はユーザーにとって重要な技術なので，実施するユーザーが多いものと

考えられる。

次にデータベース 技術については６８％が実施している。１週間が全体の４６％，２週間が２２％，１週間未満が１７％である。一方，４週間以上のものも１０％ある。

次に，マネージメントサイエンス技術については９％とごく少なく，しかも期間は１週間以内が９０％である。最後に，ＣＡＤ／ＣＡＭ技術についてはわずかに５％しか実施しておらず，これは特に専門分野の技術のためと思われる。

教育を実施するやり方の面についてみると，まず，ＯＳ等の基本ソフトウェア技術については，メーカーのユーザー教育コースを利用する場合が一番多く８９％である。

次に，社内（教育部，ベテランＳＥ）による教育が６０％であり，この２つの方法が一番広く活用されており，社外の専門指導員を呼ぶ（１５％），ビデオツールの活用（１４％）等は実施の仕方が低くなっている。

次に，マネージメントサイエンス等の高度な技術の教育のやり方についてみると，各種セミナーを活用することが５３％と一番多く，次にメーカーのユーザー教育コースの利用が３１％と多い。社内の専門家による場合が１３％と実施が減り，社外の専門家によるもの（１０％），ビデオツールの活用（５％）と順に少ない比率となっていく。また，アプリケーションの教育の面についてみると，社内で教育する場合が圧倒的に多く８２％を占める。次に多いのは，メーカーによるユーザー教育コースを利用するのが２３％，各種セミナーの活用が２０％となっている。アプリケーションの教育はやはり，ユーザー自身の社内熟練者による教育が一番多い。

(3) 情報処理技術者試験対応の教育

情報処理技術者試験に対応する教育については，一般ユーザーの場合はほとんど行われていない。わずかに第二種対応が９件，第一種対応が

7件，特種対応が5件というアンケート結果であるから，全体の中の比率としては極めて低い。これはソフトウェアハウス等の専門会社であればともかく，一般企業においてはこのような結果になるのは当然と考えられる。また，これを教育方法の面からみると，通信教育の活用の55％は情報処理技術者試験のためであるというアンケート結果となっている。即ち，通信教育でまかなうケースが多い。一方，社内教育の7％がこの情報処理技術者試験の目的となっており，比率は低い。

(4) コンピュータ 要員の教育費用

コンピュータ 要員に対して，どの程度費用が投入されているかについてみる。

アンケート結果によると，要員一人当り投入費用は年間平均102千円である。件数の比率でみると，年間101千円以上は19％であり一番多いが，次に多いのは26～30千円の16％であり，46～50千円が13％，6～10千円が12％となっている。

IBM社の「IBM研究会アンケート調査結果」（1983年度）によると，投入費用について，ほぼ同じような結果が出ている。

これによると，アナリスト／プランナについては112千円（希望としては186千円投入したい），システム・プログラマー137千円（希望194千円），適用業種プログラマー76千円（希望133千円），オペレーター53千円（希望90千円）となっている。これにより，ほぼ，1人当り年間投入費用は100千円前後が平均的な投入費用であるといえる。

次に，教育研修に要した費用はどの部門で負担しているかをみると，89％がコンピュータ 部門で負担していると回答している。重複回答はあるが，この他では人事部門とするのが12％，教育部門とするのが8％という実態である。

要員教育のために，どの程度の比率の時間をかけているか（要員の全

作業時間に対する比率）をみてみる。

アンケート結果によると，平均6.5％となっている。件数としてみると作業時間の５％という回答が４４％で最も多く，次に１０％と回答したものが１８％となっている。平均的に５％から最大１０％程度とみるのが現状の実態であろう。

⑸ 要求される能力

コンピュータ 要員に期待される能力（資質）とコンピュータ 部門に要求される能力（知識・技術）について述べる。コンピュータ 要員に期待される能力についてアンケート調査によると次のような結果となった。

① プログラミング担当者

プログラミング担当者に要求される能力として第一にあげられているのは「正確性」である。これについては挙げられる各種能力の中で１４５件回答のうち８７％を占めている。次に多いのは「根気の強さ」で４７％，次いで「迅速性」（４０％），「論理的な思考能力」（３２％），「体力」（１６％）と続いている。外部との交渉力，視野の広さ，指導力などはあまり重視されない結果となっている。

② システム設計者

システム設計者に要求される能力は，まず第一に「論理的な思考力」が９７％で，これ以外は割合は比較的に少なく，「視野の広さ」が３９％，「正確性」が３７％，「根気の強さ」が１９％，「協調性」が１８％となっている。

③ プロジェクトリーダー

プロジェクトリーダーについては，最も多いものが「指導性」で８７％，次いで「外部との接渉力」が５０％であり，あとは「管理力」，「視野の広さ」，「論理的な思考能力」等が求められている。

④ 管理者

管理者はまず第一に「管理能力」が要求されており（９０％），次いで「外部との接渉力」（５７％），「視野の広さ」（４３％），指導性（３５％）等が求められている。

次に，コンピュータ 部門に対する能力（知識・技術）面から検討する。要求される能力を「基本ソフト」，「データベース 技術」等の項目に分類し，これに対して「現在貴社が保有している技術」，「最も重要と思われる技術」，「今後伸ばして行きたい技術」にわけてアンケートした。この結果，まず「現在保有している技術」では「不充分」としているのが「マネージメントサイエンス」（６８％），「ニューメディア及びＯＡ機器利用技術」（５５％），「要求定義技術」（３９％），「データベース技術」（４２％），「通信技術」，「プロジェクトマネージメント技術」，「一般的な要求改善技術」などが３７％が不足しているという結果となった。

一方，「まずまず保有している又は充分保有している」という面からみると，「適用業務知識」について８７％が「まずまず又は充分技術を持っている」としており，これはユーザーという立場からの回答であるからうなずける結果である。

また，「コンピュータシステム開発技術」についても８４％が保有しているとしている。これは，日常システム開発をする必要性に迫られているからこれらの基本的な技術は身につけているという事で当然といえよう。次に多いのは，「ＯＳなどの基本ソフトウェア」で７９％となっている。続いて「プロジェクトマネージメント技術」（５８％），通信技術（５８％），データベース技術（５８％），一般的な業務改善技術（５７％）等となっている。

「充分持っている」という技術は比率としては少ない。

一番比率の高い「適用業務知識」でさえ２３％程度という結果となっている。

次に「最も重要と思われる技術」についてみると，「データベース技術」，「通信技術」が６５％，「コンピュータシステム開発技術」が５２％，「要求定義技術」が５０％，「ニューメディア及びＯＡ機器利用技術」が４１％，「適用業務知識」が３８％となっている。また，同じ最も重要と思われる能力で，3年後についてはどうかというアンケートに対しては，「ニューメディア及びＯＡ機器利用技術」が急激に増大して６５％に，また，「通信技術」が７２％に増大している。さらに「マネージメントサイエンス」も１８％から３８％へ重要性が増大するとしている。

一方，「適用業務の知識」や「ＯＳなどの基本ソフトウェア」等は3年後には重要性が減少してくるものとしている。

次に，今後伸ばして行きたい技術についてみると，「ニューメディア及びＯＡ機器利用技術」，「通信技術」，「データベース技術」の3つが高い比率を占めており，その他については比率が低くなっている。

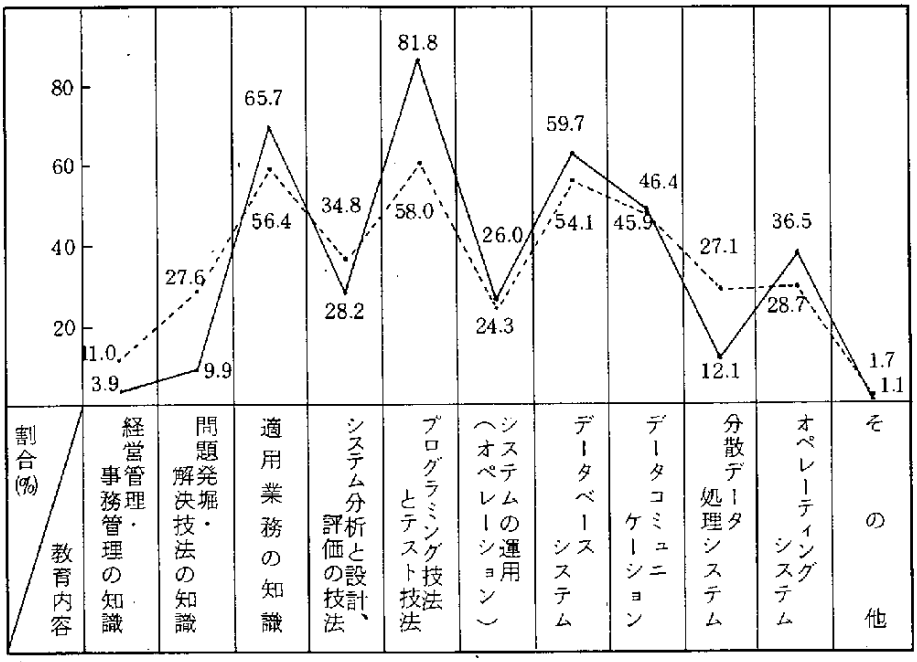
(6) 将来の教育内容について

現在の教育内容と今後必要と考える教育内容について，ＩＢＭ社が調査した結果があるのでこれを参考にしたい。

これは中規模以上のコンピュータ・ユーザ に調査したものであるが，プログラマについて適用業務プログラマとシステム・プログラマに関してそれぞれ調査したものである。

図表５－１３は適用業務プログラマについて現在実施している教育と将来必要と思われる教育内容について示している。

図表５－13　教育実施内容と将来必要と思う教育内容
―適用業務プログラマー―



●――●：現在実施している教育内容　○-----○：将来必要と思われる教育内容

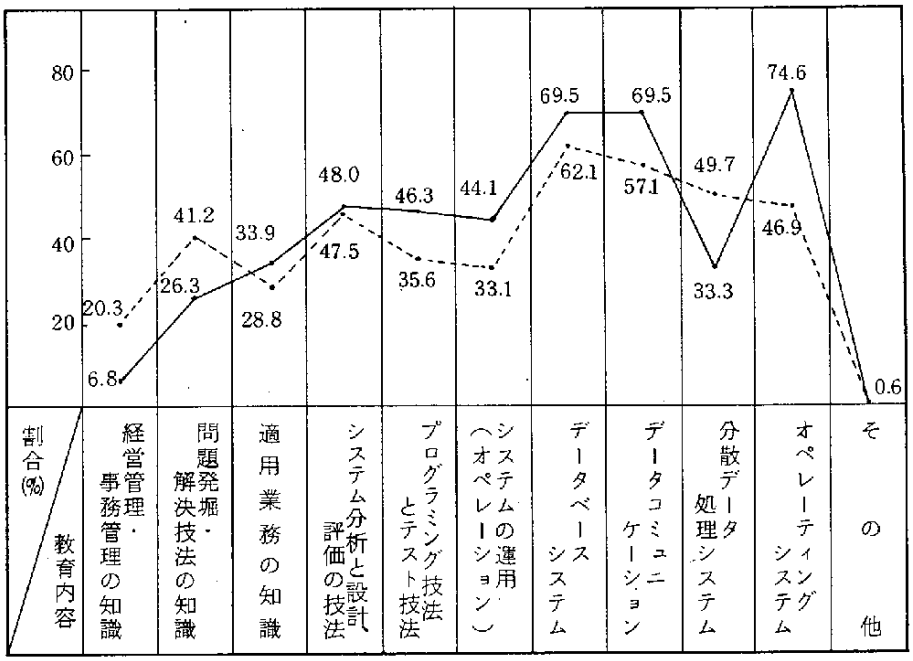
（「ＩＢＭ研究会アンケート調査結果」－1983年度）

当然ながら「プログラミング技法とテスト技法」へのニーズが高く（８１.８％），これに「適用業務の知識」，「データベース・システム」などのニーズが続いている。

将来必要とする教育内容で現在実施しているものよりニーズが高いものは「問題発掘・解決技法の知識」，「分散データ処理システム」，「経営管理・事務管理の知識」などである。

次に，システム・プログラマの場合をみると，次の図表５－１４のようになっている。

図表５－14　教育実施内容と将来必要と思う教育内容
―システム・プログラマー―



●――●：現在実施している教育内容　●-----●：将来必要と思われる教育内容

（「ＩＢＭ研究会アンケート調査結果」－1983年度）

「オペレーティングシステム」（７４.６％），「データベースシステム」（６９.５％），「データ・コミュニケーション」（６９.５％）などの実施率が高くなっている。将来必要と思う教育内容としては「問題発掘・解決技法の知識」，「分析データ処理システム」，「経営管理・事務管理の知識」などが現行よりも比率が高くなっており，将来のニーズが強いことを示している。

(7)　企業内教育の事例

本項では情報処理に関する企業内教育の事例を述べる。最初に国の機関の教育について国家公務員研修を例に述べ，次に地方自治体について，（財）地方自治情報センターにおける研修を例に述べる。

一般企業については紙数の制限もあり，他に詳細に記述したものが多数出版されているので割愛した。

①　国家公務員の情報処理研修

国家公務員を対象に行なう情報処理関係の研修は全省庁向けに行なうものは総務庁が企画し，実施している行政管理セミナーが大部分を占めており，それ以外では科学技術庁が主催するもの等がある。

総務庁が主催する行政管理セミナーにおける情報処理関連のコースは，次のようになっている。

- 行政事務管理分析コース

  行政事務の管理改善や施策策定のための問題発掘および解決案作成のための各種技法とその関連知識の習得を目的として，情報システムの開発・利用部門の職員を対象として実施する。

- プログラマーコース

  電子計算機による情報処理に関するマネジメント思考，システム分析および設計等について，基礎的な知識，技術の習得を目的としてプログラミング担当者を対象として実施する。

- データベースコース

  データベースシステムの開発および利用等に関する専門的な知識と技術の習得を目的として，データベースシステムの計画開発に従事する者で，プログラミングシステム設計に関し，2年以上の経験を有する者を対象とする。

- システムエンジニアコース

  情報処理に関するマネジメント思考，システム分析および設計の技術等の修得を目的として，システムの分析，設計に従事する者で，ソフトウェア全般に関する知識と2年以上のプログラミング経験を有する者を対象とする。

- 運用管理者コース

  効果的なオペレーション管理やデータベース等のシステム資源の総合的な管理によるコンピュータ室運用の効率化についての知識の習得を目的として情報処理システムの運用管理業務に従事する人を

対象とする。

- システム評価コース

コンピュータシステムの評価の必要性と科学的な評価技法の修得を目的とし，コンピュータシステムの評価作業の従事者を対象とする。

- プランナコース

効果的な情報システム計画のための各種技法とシステム開発プロジェクトの効果的な管理についての知識の習得を目的とし，コンピュータシステムの評価作業に従事し，システム設計に関する2年以上の経験を有する者を対象とする。

- マネジメントサイエンスコース

オペレーションズリサーチの主要手法，意志決定のための予測の意義と各種予測手法について，その目的，適用方法，実施効果についての知識の習得を目的として，予測計画作業に従事する者を対象とする。

- 電子計算機入門コース

日常発生する非定型的な業務処理のためのコンピュータ利用方法についての知識の習得を目的として，情報システム利用部門のコンピュータ利用者を対象としている。

- 研修指導者養成コース

情報処理要員の実践的知識体得のためのＯＪＴ技術および効果的な自省庁情報処理研修推進のための指導者として必要な知識の修得。

以上が，総務庁が主催する全省庁を対象とする研修カリキュラムの内容であるが，この他に科学技術庁が，（財）日本情報処理開発協会に委託して実施する電子計算機プログラミング研修（初級・上級）があり，これは国立研究機関等で電子計算機を使用する業務に従事している職員を対象としてプログラミング技術の修得を行なわせるものも

ある。

② 地方自治体の研修

地方自治体における研修は，各自治体内で独自に実施してはいるが，（財）地方自治情報センターが地方自治体におけるコンピュータによる情報処理を推進する機関として教育を実施しているので，このカリキュラムの要点を中心に説明する。

- 管理者コース

管理者を対象にコンピュータ活用に必要な知識の向上のため，情報機器の開発，利用状況およびコンピュータ利用拡大に伴なう諸問題を理解するとともに，行政におけるコンピュータの効果的な利用について認識を深める研修である。

- 運営管理者コース

コンピュータ部門の主務者を対象に，情報処理システムの開発およびコンピュータの導入計画や推進並びに情報処理に関する研修等に携わる際に必要な基礎知識や管理技法等を習得する研修である。

- 初級コース

初心者を対象にコンピュータの基礎知識の習得を目的としており，コンピュータの仕組みや働き，システム設計の知識やプログラミング，オンラインシステムの初歩知識等コンピュータを活用する上に必要な最少限度の基本事項について演習，実習を中心にした研修。

- 上級コース

コンピュータの実務経験のある職員を対象に能力の拡充を目的としており，効率的なデータ処理システムの開発や運用に必要な知識および技術を習得する研修。

- 適用業務コース

行政事務のコンピュータ処理に当たって必要なシステム開発技術や実務に密着した技術および専門知識を先発団体の利用事例を中心

に研修する。

- 計画策定コース

  地域社会の諸問題を分析し，計画を支援するシステム設計に役立つ，経営科学（応用数学）に関する基礎知識および実務知識を習得する研修。

- 委託処理研修

  コンピュータ委託処理の基本的留意事項，技術上・法律上の諸問題，委託活用のポイント等について研修する。

5.2 コンピュータ要員の年令限界

本節では，コンピュータ要員の年齢限界を取り上げ，コンピュータ要員の年齢構成の実態，職種別年令限界に関する考察，高齢化対策の現状と将来，コンピュータ要員の将来に対する所見について述べる。

なお，本節で取り上げる高齢化とは40歳以上を対象としている。

5.2.1 コンピュータ要員の年齢構成

本項では，一般企業を中心としたコンピュータ要員の職種別年令構成の実態について述べ，併せて，高齢化が本当に進んでいるのかという問題についても言及する。

(1) 職種別年齢構成

コンピュータ要員をどのような職種に分類し，どの範囲迄把えるかは，問題のある所だが，今回は調査研究のアンケート結果に合わせ，以下の職種区分について，分析を行なう。

- 管理者
- プロジェクトリーダー
- システム設計者
- プログラマー
- 教育担当者

・　オペレーター

アンケート結果（図表５－１５）によると，コンピュータ部門の量的主役は２０代，３０代である。しかし，２０代の占める比率は年々減少しており，逆に３０代，４０代以上の比率は増加している。これを職種別に見ると，管理者は，４０代以上，リーダーは３０代後半，システム設計者は３０代前半，プログラマー，オペレーターは２０代がそれぞれ中心となっている。

教育担当者は２０代，３０代前半に多いものの，割合各年代に散らばっている。

これらを詳細に見ると，管理者は４０才以上が７４.２％と圧倒的に多く，３４才以下は4.5％となっている。これは一般の管理者の状況と比較しても大きな差異はないものと思われる。

プロジェクトリーダーは，３０代が７０.５％と圧倒的に高く，２９才以下はわずか1.７％となっている。これは経験を要する職種の性格からいって当然の結果といえる。システム設計者は３０代が６９.３％とその中心を占め，２０代は２２.６％となっている。これは昭和５０年迄の状況が２０代と３０代がほぼ同じ比率であった事と比較すると，着実に年齢層があがってきている。

プログラマーは２９才以下が７３.１％，３４才迄で８９.９％を占め，若年傾向は変らない。教育担当者は２９才以下３０.２％，３１～３４才迄３１.８％，３５～３９才迄１９.８％，４０才以上１８.２％と各年代に散らばっている。これはニューメディア及びＯＡの教育等，利用部門のトップ層からコンピュータの部門内教育迄，教育の対象部門，対象範囲が拡がってきているためと思われる。オペレーターは２９才以下が６４.４％と最も高いが，プログラマーよりこの年代の比率が低くなってきている。また，３０代が１９.５％に対し，４０代以上も１６.１％と急速にその比率を高めている。オペレーターの２０代の減少と４０代以上の増

加は，外注化の促進と高齢者対策のため，他部門より，４０代以上の転入者が増大している結果と思われる。

職種別に年齢構成の若い順（３４才以下の比率の高い順）からいえば，プログラマー（８９.７％），オペレーター（７６.４％），システム設計者（６４.１％），教育担当者（６２.０％），プロジェクトリーダー（２５.１％），管理者（４.５％）となっている。昭和５０年度と比較し，変化したのはプログラマーとオペレーターの順位が逆転したことである。これは従来のオペレーターから，プログラマーへというジョブ・ローテーションが崩れ始めたことを意味し，オペレーション経験が要員教育の必須の条件でなくなりつつある事を意味していると思われる。

図表５－１５　コンピュータ要員の年齢構成

N＝１５４　　（単位：％）

	29歳以下	30～34歳	35～39歳	40歳～停年	停年退職者の再雇用	合　計
プロジェクト管理者	0.2	4.3	21.3	73.7	0.5	100.0
プロジェクトリーダー	1.7	23.4	47.1	27.2	0.6	100.0
システム設計者	22.6	41.5	27.8	7.6	0.5	100.0
プログラミング担当者	73.1	16.6	6.4	3.4	0.5	100.0
教育担当	30.2	31.8	19.8	18.2	—	100.0
オペレータ	64.4	12.0	7.5	15.7	0.4	100.0
コンピュータ要員トータル	47.0	23.0	19.0	(再雇用含む) 11.0		100.0

⑵　高齢化の進展状況

過去のコンピュータ要員の年齢構成と比較すれば（図表５－１６，図表５－１７及び図表５－１８），確実に２０代の年齢層が減し，３０代，４０代の年齢層が増加している。高齢化へのきざしが見えてきている。

図表５－１６　システム設計者，年齢構成，年次別比率

（単位：％）

年次＼年齢	29才以下	30～39才	40才以上
５１年	６９.０	３０.０	1.０
５５年（男子）	４５.１	５１.４	3.６
５９年	２２.６	６９.３	8.１

（出所）５１年　情報化対策委員会情報処理技術者問題調査部会調査
５５年　財団法人労働科学研究所調査
５９年　日本情報処理開発協会調査

図表５－１７　プログラマー，年齢構成，年次別比率

（単位：％）

年次＼年齢	29才以下	30～39才	40才以上
５１年	９５.０	3.８	1.５
５５年（男子）	９４.４	5.６	０
５９年	７３.６	２３.０	3.９

これを職種別に見ると，システム設計者の年齢の上昇傾向が顕著であり，次いでオペレーターが上昇傾向にある。プログラマーは若干上昇しているものの，その若年傾向は変わらない。これは，プログラマーは外注化の傾向が強くなっていることと，システム設計者の社内要員確保のため，プログラマーからシステム設計者へのジョブ・ローテーションが活発に行なわれているものと思われる。

全体としては，高齢化方向に進みつつあるが，労働白書の従業員全体の年齢構成の統計と比較すれば（図表５－１９の通り），４０代以上の比率はコンピュータ部門の要員が圧倒的に低く，そういう意味においては，高齢化はそれ程進んでいないといえる。

図表５－１８　オペレータ，年齢構成，年次別比率

（単位：％）

年次＼年齢	29才以下	30～39才	40才以上
５１年	９８.０	２.０	０
５５年（男子）	９１.８	８.０	０.３
５９年	６４.４	１９.５	１６.１

図表５－１９　就業労働者全体とコンピュータ要員の年齢構成の比較

（単位：％）

要員＼年齢	29才以下	30～39才	40才以上	
コンピュータ要員	４７	４２	１１	（５９年度）
就業労働者	２８	２８	４９	（５７年度）

（出所）コンピュータ要員は情報処理開発協会
就業労働者は総理府統計局「労働力調査」

5.2.2　コンピュータ要員の年齢の限界に対する考察

コンピュータ部門が，一般に若い年齢層によって占められていることにより，プログラマーの３５才定年説等，コンピュータ要員の年齢限界説が唱えられているが，本当にそうなのであろうか。この項では，年齢限界がいわれる背景，コンピュータ部門の役割の変化による年齢限界，職域への影響，職種別の年齢限界について述べることとする。

(1)　年齢限界の叫ばれる背景と実態

年齢限界説の根拠となっている側面１つづつに考察を行なう。

①　能力の低下

能力が低下し，技術革新の激しい変化についていけない。また，迅速性，正確性，根気がなくなる。これらが年齢限界説の最大の根拠になっている。これについて，図表５－９に示されているようにＤＰ協会のアンケート調査によると，知能面から業務に支障があると回答し

ている企業は１６％にしか過ぎず，逆に質の高いシステム作りができると回答している企業が１５％にものぼっている。

これは高齢者が個々の技術では劣るものがあっても，総合力では充分対応できる事を意味しているものと思われる。

② 体力の低下

若年層に比較し，間違いなく体力は劣ろえ，休日，徹夜が続けば，業務遂行への影響もでてこよう。ＤＰ協会のアンケート調査結果（図表５－６）によると，他部門に比較し労働時間が「やや長い」と「長い」を合計すると，６０％以上を占めており，労働条件は決して良くない事を示している。

しかし，アンケート調査結果（図表５－９）によると，体力的な面から業務に支障があると回答しているのは，１０％に過ぎない。

現在の年齢構成では，まだ問題になる程の支障はきたしていないと言える。

③ 付加価値生産性の低下

この問題は特にプログラミングの職種において問題となっている。プログラミング技術は，若年層の得意とする分野で，高齢者になる程，賃金の上昇に反比例し，付加価値生産性は落ちていく傾向にある。

④ モラールの低下

人事の停滞，マンネリ化，若い人々と同一業務の分担等によりモラールの低下をきたす。また高齢者の多い職場は活力が失なわれ，モラールの低下をきたす。

⑤ 処遇面

一般企業であれば，労働時間が長い事を除けば昇進，給与面では他の部門とほとんど変わらない。しかし，ＤＰ協会のアンケート調査(図表５－９）によると，高齢化から起る最大の問題はポストが準備できないと多くの企業が回答しているが，これは他の部門も同様であり，人事政策全体の問題として考えていく必要がある。

⑥ 人材育成面での配慮

企業サイドより見た場合，同一職種に長く携さわるより，ジョブ・ローテーションにより色々な業務を経験させ，人材育成を図りたい。

事業戦略面からコンピュータ部門経験者を他分野で活躍させたいとの要請が強まっている。

(2) コンピュータ部門の役割の変化による年齢限界への影響

コンピュータ部門の期待される役割が従来の単なるコンピュータ化，及びシステム開発運用部門から，全社の業務改善，効率化の推進，経営の意志決定サポート，全社情報ネットワークの構築，データベースの構築，ＯＡの推進等と，高度化し，多岐にわたってきている。

これにより必要とされる技術，能力が単にプログラミング技術，システム設計技術のみでは対応出来なくなっている。技術面では，データベース技術，通信技術，ニューメディア及びＯＡ機利用技術の重要性がまし，能力面では利用部門との調整，コンサルティング能力，即ち，幅広い業務知識，強い指導性，調整能力，人間関係能力，総合判断力が重要となってくる。業務によっては，コンピュータ技術よりも重視される場面も増えてこよう。

これらは，一方，コンピュータ部門の職域領域の拡大を持たらしている。即ち，データベース，通信技術を中心とした高度コンピュータ技術者の需要の増大，ニューメディア，ＯＡを中心とした教育担当者の需要の増大，全社の業務改善，ＯＡの推進のための社内コンサルタント（システム・コンサルタント）の需要の増大等を持たらしている。

これらの職域の多くは広い視野に基づいたリーダーシップ，調整能力の発揮が必要で，付加価値生産性の高い高年齢者向きの仕事といえよう。

(3) 職種別年齢限界

ここではシステム・コンサルタント，システム設計者，高度コンピュータ技術者，プログラマーについて年齢限界の考察を行なう。

① システム・コンサルタント

全社の業務改善，ＯＡの推進等社内コンサルタントとして，利用部門との調整能力，コンサルティング能力，人間関係能力，幅広い業務知識力，総合判断力を必要とする。このような能力は高齢者が得意とする所で，２０代及び３０代前半の若年層では難かしい。

年齢限界はほとんどないといっても良い。

② システム設計者

システム設計者の期待される資質は論理的思考能力，視野の広さ，正確性が上位となっており，若年層が最も得意とする迅速性，体力，根気の強さは下位の方に属しており，資質面からは年齢限界はないといえる。

要求される知識，技術力は業務知識の深さ，システム設計技術が中心であり，業務知識は経験年数が大きくものをいうし，システム設計技術は割合不変的で陳腐化しにくい。むしろ，経験年数が知識力，技術力を高めるといえる。次に，若年層との仕事の差別化については，システム設計のベテランは当然高度なシステム設計を担当するのが一般的で，充分差別化ができ，この面でのモラールの問題もない。

むしろ，高度なシステム設計がなく，若年層と同じレベルのシステム設計を分担させた場合に，マンネリ感がでるものと思われる。

上記の様な点を考慮した育成策，仕事の与え方をすれば，能力面，付加価値生産性の面でも，年齢限界はないといえる。むしろ，職位として専門職としての地位を如何に確立するかが最大の課題である。

③ 高度コンピュータ技術者

データベース技術，通信技術等，最も技術革新の激しい分野であり，技術の吸収力，新規技術を取り入れる革新性が要求される。この面からみると若年層向きといえるが，特に技術指向が強い職種なので，技術指向の強い人ならば年齢制限はないといえる。しかし，一般企業の

中で，こういった幅の狭い分野のスペシャリストとして育成していって良いのかという事になる。他部門にローテーションした時の対応力がなくなってしまう恐れがある。

専門職としての地位が確立していない場合には，つぶしが効かなくなり，問題が残る。

④ プログラマー

プログラマーとして期待される資質は，正確性，根気の強さ，迅速性，論理的な思考能力となっており，実務面では更に細心の注意力と集中性を必要としており，若年層が得意する資質が要求されており，高齢者向きでない。

一方，技術面では，プログラムの標準化，簡易化，自動化により，高度な技術専門家としての地位を失ないつつあり，技術集約型というより労働集約型の色合いを濃くしている。

付加価値生産性の面からいっても若年層とベテランの間で極端な差がなく，ベテランを活用した場合には，当然，付加価値生産性が低くなる。一方，仕事の質的面でも若年層と差をつけにくく，モラール面で問題がある。また，人材育成面でも，広く会社の業務を見る事ができず，仕事の幅を拡げることが難かしい。３５才定年説は肯づける。

むしろ，今後の標準化，簡易化，自動化を考えると年齢限界を更に引き下げても良い。

5.2.3 コンピュータ要員の将来に対する総合所見

前述のように，コンピュータ部門の役割が変化し，戦略的な役割が増大し，活動領域は大幅に拡大し，高質化している。その活動領域は経営課題と密着し，総合判断力，指導力，人間関係能力等，総合力を必要とし，付加価値の高い領域となっている。まさに高齢者に適した分野といえる。

しかし，その前提となる能力は若い時からの積み重ねの教育と訓練が必

要である。

本項では，高齢化対策の基本は，若い時からの予防策であるとの考えに立ち，そのための施策を中心に述べる。

最後に，高齢化対策全体を通しての将来に対する総合所見を述べる。

(1) これからの高齢化対策

高齢化問題は，人材能力開発の長期育成策の反映である。とうが立ってからあわてても遅すぎる。結局若い時から陳腐化しないように，予防策をとる事が大切である。

その基本は，

- この会社は何のためにあるのか
- 会社の成長のためにコンピュータ部門は何をするのか
- そのためにどんな能力を持つのか

を常に把握し，各人が絶対に陳腐化しないように，経営戦略とリンクした長期育成計画をたて，充分に能力を発揮できる仕組みを作りあげることである。いいかえれば，要員育成方針の明確化，その育成方針にあった教育体系の整備，各人の能力把握と育成策を明確化する人材点検の仕組，業務目標管理の4つがうまくリンクして動くようにする事である。

(2) 高齢化の具体的施策

高齢化対策に有効と思われる重要施策について説明する。

① キャリア・パス制度

今回のアンケート調査によると，キャリア・パス制度を取り入れている企業は9％と非常に少ない。各人を陳腐化させず，能力を最大限に発揮してもらうには，各人の能力を定期的に把握し，キャリア・パスと照らして，正常な道を歩んでいるか等をチェックし，今後の育成策と育成方向を明確にしてあげる事は人材育成の基本といえる。

これにより，計画的なローテーションも可能となる。また環境の変化に対しての迅速な育成策もたてられる。

② 教育制度の整備

コンピュータ部門の教育制度を整備しておかないと，環境の変化，技術革新に対し，迅速なる教育体系へのフィードバックができずに，気がついた時は既に高齢化問題が発生しているという状況になりかねない。ここで重要な事は，各年代層別の教育体系をしっかり作りあげる事である。

③ 目標管理制度の活用

目標管理制度を採用している企業は５８％と半数以上にのぼっているが，本人の自己管理に任せているのでは，高齢化対策に何の意味も持たない。目標管理設定における各人の育成方向とテーマの擦り合せが重要で，この擦り合せがうまくいった時に，始めて本人の能力開発に大きく貢献する。

④ ローテーション制度の確立

今回のアンケート調査結果で，高齢化対策として，他部門へのローテーションをあげている企業が最も多かったが，当然である。

高齢化対策としては非常に有効な手段であるといえる。部内外を問わず，計画的にローテーションが行なわれている時は，本人の育成に寄与する。しかし高齢化し，有効活用の場がなくなり，仕方なく行なうローテーションは後向きの施策であり，真の高齢化対策とはならない。あくまでもその人の成長を促進し，本人が能力を最大限に発揮できるものでなくてはならない。このためには，人材点検，キャリア・パス計画に基づいた計画的なローテーションが必要となる。

⑤ 専門職位制度

前項で述べた通り，高齢化から起こる最大の問題は，「年齢に見合う役職やポストが準備できない」であると指摘されている。

この解決策として，専門職位制度が脚光をあびてくるわけだが，単なる資格制度になっては意味がない。真に専門職にふさわしい資格条

件をそなえた人に与えられてこそこの制度も生きてくる。

⑥　外注化の推進

付加価値の小さい業務（オペレーション，プログラミング）は出来るだけ外注化を促進し，社内の要員は付加価値の高い業務に早くシフトし，高齢者にふさわしい業務を遂行出来るような体制を作りあげておくことが大切である。

⑦　女子社員の活用

これもほぼ⑥と同様の目的である。

(3)　まとめ

コンピュータ部門が単なる従来のプログラミング技術を中心としたコンピュータ化部門から，経営の意志決定サポート，全社の業務改善等活動領域を拡大している。この活動領域の拡大につれて，従来のプログラミング能力，システム設計能力に加え，利用部門との調整，コンサルタント能力が重視されてきている。又これらの能力をより多く有している高齢者の活動領域も拡がる。

一方，プログラミング技術については標準化，簡易化，自動化により，プログラマーの専門技術者としての地位は急速に落ちてきている。

コンピュータ要員の年齢限界については，プログラマーを除き，それほど問題にする必要がなく，高齢化に伴ないスペシャリストの地位を如何に確立するかが大きな課題となりつつある。プログラマーは年齢限界説通り，更に，その年齢限界を意識的に引き下げる必要がある。

高齢化対策として，現在は「要員のローテーション」を各社考えている。しかし，今後高齢化がますます進む状況を考えると，経営戦略とリンクした長期育成策をたて，要員を陳腐させない事が大切である。高齢化問題は，長期育成策の反映といえる。

5.3 モラール向上対策

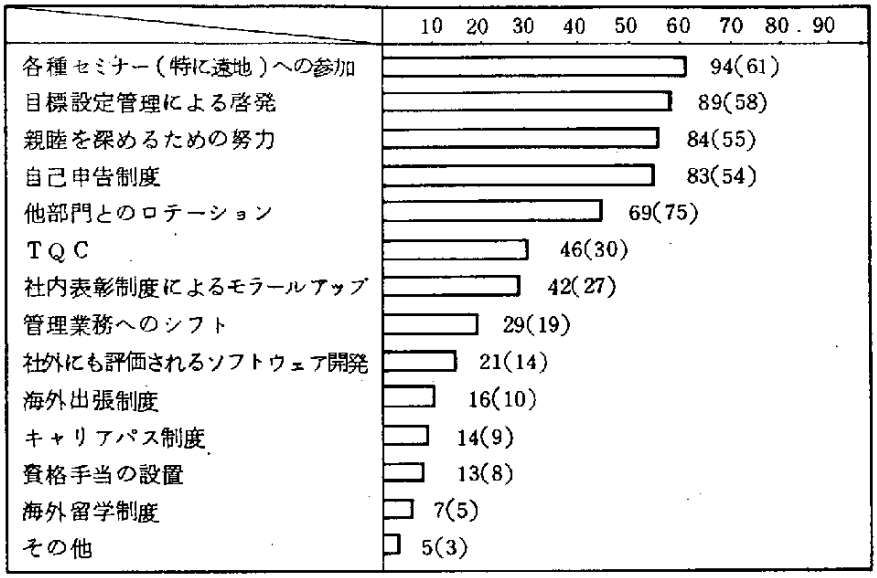
コンピュータ部門のモラールやモチベーションを向上させるための施策は益々重要な要素となってきた。それはコンピュータ部門に課せられる仕事の増大とともに組織の拡大が進み，それに対して個々に割り当てられる職務も限られた一部分を担当させられるという，いわゆる企業化が進んできたからである。個々の担当者は全体の一部しか担当しておらず，自分が何をやりどのように貢献しているかがわからなくなり，ひいてはモラールの低下が，生じ易い傾向になりつつあるといえる。

また，コンピュータ部門の仕事は個々の働きも重要であるが，システムの建設，運用については，チームで共同しなければ達成できない傾向にある。

システムとして充分な機能を発揮し，建設し，さらに運用をすすめていくためには，チームワークの上手な結成が不可欠である。ともすれば，自己の枠の中にとじこもり勝ちなコンピュータ作業の諸要素の中に，チームワークをうまく機能させる必要があり，これらの問題をどのように処理したらよいかを検討する。

まず，アンケートの回答からみると，１５３件の回答のうち，一番多い実施事項は「各種セミナー（特に遠地）への参加」であり，９４件（６４％）となっている。次に多いのが「目標管理による啓発」で８９件（５８％），となっている。三番目が「親睦を深めるための努力」が８４件（５５％）及び「自己申告制度」が８３件（５４％）となっている。アンケートの結果は，図表５－２０に示されている。この表にみられるようにその他の項目としては，キャリア・パス制度をとり入れている企業（９％），海外出張制度（１０％），管理業務へのシフト（１９％）などがある。

図表5－２０　モラール向上対策の実施状況



Ｎ＝153

図表５－２１　モラール向上対策の数

対策の数	件数	％
1	５件	3
2	３１〃	２０
3	３６〃	２３
4	３４〃	２２
5	１９〃	１２
6	１５〃	１０
7	６〃	4
8	８〃	5

Ｎ＝154

次に，これらの対策を複数件実施しているケースをみると，これを図表５－２１に示す。

これによると，対策の件数は２件～４件を実施している企業が一番多く（２０％～２３％），次いで５件が１２％，６件が１０％となっている。

７件および８件実施している企業は４％および５％となっている。これをみると，ほぼ多くの企業が２～４件の対策を実施していることがわかる。

アンケートの回答の中で，一番多かったのは，「各種セミナー（特に遠地）への参加」であるが，これは一番採用しやすくまた，実施しやすいものであり，一般的に広く実施されているといえる。

しかし，注目されるのは二番目に多いものとして，「目標設定管理による啓発」である。ソフトウェアの作成は，システムが大規模化し分業化しつつある現在にとって，個人の労苦はどこで生かされているかが見失われ勝ちであり，これがモラールの低下の大きな要因になりつつある。そこで目標設定管理を充実する事により，個人の向上と達成感の充実をはかることが必要となる。Ｃ・Ｌ・ヒューズ「目標設定」（小野豊明・戸田忠一訳）によれば，個人の目標設定システムを次のステップに分けている。

(1) 特定の目的確立

業務計画遂行計画についての人事管理システムの一部として目標の設定を行う。

(2) 目標の重要度 ― モチベーションの検討

目標を文書等で表現させ公約させ，慎重な検討と動機づけを行なう。

(3) 行動のための計画

実現のための計画は自分で立てさせる。

(4) 業績基準および測定基準

仕事が完了する以前に評定尺度および目標達成の期日を決定しておく。

(5) 予期される問題点の排除

(6) 必要な援助

個人一人では達成することはできない。誰でも援助を必要としているの

で上長及び関連部門は常に援助する。

(7) 実際の業績の測定と評価

目標管理はモラールの向上のためには，重要な手法である。何よりも個人の目標に対する達成感の充足が必要とされている。達成感の充足は適切な目標管理が前提となっており，この手法の一層の普及が望まれる。

参考文献

1) 情報システム要員の養成，確保等に関する調査研究報告書（行政管理庁行政管理局　昭和59年3月）

2) 我が国情報処理教育の現状（（財）日本情報処理開発協会情報処理研修センター　昭和59年3月）

3) IBM研究会アンケート調査結果（日本IBM　1983年度）

4) 人を動かす（D．カーネギー，山口博訳，創元社）

5) 目標設定（C．L．ヒューズ，小野豊明，戸田恵一訳，ダイアモンド社）

6) 行政情報システム研究所：情報処理分野における中高齢者雇用の開択に関する調査研究報告書　昭和56年3月

7) 日本データ・プロセシング協会：コンピュータ部門と実務担当者の実態と意識に関する調査研究報告書・昭和58年3月

8) 日本システミックスリサーチセンター：システム部門のマネジメントノウハウ第2集　昭和59年7月

9) 関西生産性本部：メカトロニクス革新と今後の産業・労働政策，昭和58年3月

10) 日本情報処理開発協会編：コンピュータ白書　1983年

11) 西田耕三：高齢化問題への企業戦略　昭和55年2月

12) 松島静雄：高齢化社会の労働者　1983年2月

13) 産業労働調査所：能力開発事例集　昭和59年3月

6. ソフトウェア開発段階のチェックリスト

6. ソフトウェア開発段階のチェックリスト

6.1 基本設計・詳細設計

1.	完全性に関するチェックリスト
①	決定待ち（TBD：To Be Determined）の項目がなくなっているか。
②	存在しない機能，入力，出力への参照がなくなっているか。
③	仕様で書くべき項目が全て記述されているか。
④	機能が抜けがないか。
⑤	プロダクトとしての抜けがないか。設計していないところはないか。
2.	一貫性に関するチェックリスト
①	仕様の構成要素内や要素間で，表記法，用語，記号等に矛盾がないか。
②	仕様間で表記法，用語，記号等に矛盾がないか。
③	仕様上の記述項目が，要求分析書，システム分析書等と対応がとれているか。
④	文字，記号の書き誤まりはないか。
3.	実現可能性に関するチェックリスト
①	システムの機能を実行するために，利用者に満足のいく手段を与えているか。
②	システムが使いやすくなるように設計されているか。
③	満足できるシステムが受容範囲内の資源で実現できるか。
④	予想される運用上の要求の増加に対し，費用対効果の面から対応できるか。
⑤	開発されるソフトウェアが性能，信頼性，拡張性，保守性，操作性，移植性の面で一応満足できると思われるか。
⑥	高いリスクを有していないか。
4.	テスト容易性のチェックリスト
①	ケースの分析が容易にできるように設計されているか。
②	あいまいな記述がないか。
③	定量的扱いが厳密になされているか。
5.	信　頼　性
①	入力誤りに対するチェック機能はあるか。

②	入力の組合せ誤りに対するチェック機能はあるか。
③	入力の値の範囲は明確か。また，範囲のチェック機能はあるか。
④	操作手順誤りに対するチェック機能はあるか。
⑤	媒体のセット誤りに対するチェック機能はあるか。
⑥	ハードウェア障害発生時の障害検出機能はあるか。
⑦	ハードウェア障害発生時の再試行機能，または自動回復機能はあるか。
⑧	ハードウェア障害発生時の縮退運用が可能か。またこのときの切り離し，再構成の単位が明確か。
⑨	ハードウェア障害に対する予防策（２重化など）が検討されているか。
⑩	ソフトウェア障害による誤りを早期に検出する自動チェック機能はあるか。
⑪	故障したソフトウェア部分の自動再ロード，あるいは再起動機能はあるか。
6.	機　密　保　誤
①	システムの使用資格に対するチェック機能はあるか。
②	特定データの使用資格に対するチェック機能はあるか。
③	特定機能の使用資格に対するチェック機能はあるか。
④	使用資格違反をチェックアウトして，システム管理者にその旨を知らせる機能はあるか。
⑤	システム内の重要データについて，暗号化する機能はあるか。
7.	操　　作　　性
①	すべての操作手順が明確にされており，これらに対応するコマンド及びメッセージが明確になっているか。
②	システムの起動，終了に関する操作が明確にされているか。
③	システムの中断，再開，状態監視などの制御機能が明確にされているか。
④	システムの状態変更，構成変更の機能が明確にされているか。
⑤	コマンドの形式は標準化されているか。
⑥	コマンドの種類が多すぎることはないか。
⑦	コマンドの投入回数は多すぎないか。
⑧	メッセージの形式は標準化されているか。
⑨	異常発生時には必らずメッセージが出力されるようになっているか。
⑩	応答要求につきメッセージの場合の応答方法は明確か，また標準化されているか。

⑪	入力の形式は標準化されているか。
⑫	入力の必要量は最少限となっているか。
⑬	入力の各オペランドのキーワードおよび，パラメータは理解しやすい表現となっているか。
⑭	ソフトウェアの機能と，入力制御文との対応は明確か。
⑮	入力の組合せ，入力の順序に関する規定が明確にされているか。
⑯	入力のデフォルト値，または属性が設定されているか。
⑰	入力の形式，および入力値，属性の範囲に関する規定が明確になっているか。
⑱	入力方法の代替手段が考慮されているか。
⑲	出力の形式は標準化されているか。
⑳	ヘディング情報は妥当か。すなわち，タイトル，ページ番号，出力日付などの必要な情報が入っているか。
㉑	出力種別が明確になっており，ユーザが必要なものだけを選択できるようになっているか。
㉒	出力の単位，精度等はユーザにとって適切か。
㉓	エラーメッセージ等，システムからのメッセージと出力データの区別は明確か。
㉔	出力の代替手段があるか。
㉕	会話型システムあるいはオンラインシステムにおいて，応答時間が適切か。
8.	性　　能
①	性能の目標値が明確に規定されているか。
	ⓘ　応答時間
	ⓙ　単位当りの資源使用料
	・ステップ数
	・主記憶使用料
	・２次記憶使用量
	・各装置に対する入出力回数
	・チャネル負荷
	・回線の負荷
	ⓚ　障害時のシステム再開始，ジョブ再開始までの回復所要時間

②	性能目標値に対し，ユーザの了解がとれているか。
③	既存システム以上の性能目標になっているか。
④	負荷の経年変動を考慮に入れた性能目標値か。
⑤	性能目標設定条件の妥当性は検討されているか。
	㋑ ハードウェア条件
	㋑ 機器名称・型名
	㋺ システム構成
	㋩ 機器の性能
	・平均命令実行時間
	・チャネル転送速度
	・入出力速度
	・主記憶容量
	・２次記憶容量
	・回線速度
	㋥ ネットワーク構成
	㋭ 伝送制御方式
	㋺ ソフトウェア条件
	㋑ 使用するOS
	㋺ 関連する他のソフトウェア
	㋩ 入力データ条件
	㋑ 平均データ量
	㋺ ピークデータ量（日，月，年のピークを考慮）
	㋩ １日当りの総データ量
	㋥ データの種別と種別ごとの比率
	㋭ 操作時間および思考時間
9.	拡　張　性
①	ユーザの長期のシステム拡張計画を把握しているか。
②	ハードウェアの拡張に対する配慮がなされているか。
	㋑ 負荷の増加率は明確か。また，これに伴い，ハードウェアの増設計画は明確か。

	㈡ 新機種の導入計画はないか。
	㈢ 計算機構成変更の計画はないか。
③	拡張による動作環境への影響レヴューはなされているか。
	㈠ ハードウェアに関する互換性の検討
	① 中央処理装置，主記憶装置，周辺装置，端末などのインターフェース
	② これらのハードウェアの最大，最少接続法，または，接続法に関する互換性
	㈡ ファイルに関する互換性の検討
	① ファイルの形式及びレコードの形式
	② ファイル及びレコードのアクセス法
	㈢ ユーザ・インターフェースに関する互換性の検討
	① 入出データおよび入力制御文の形式
	② コマンドの形式
	③ 出力リストの形式
	④ 出力メッセージの形式
	⑤ その他の入出力操作に関する互換性
	㈣ 他のソフトウェアとのインターフェースに関する互換性
	① プログラミング
	② プログラム・ライブラリの形式
	③ OSとのインターフェース
10. 保守性（テスト容易性）	
①	テスト基準が定められているか。
②	当該テスト基準についてユーザの了解を得ているか。
③	テスト仕様は単体テスト仕様書，あるいは結合テスト仕様書等のドキュメントによって，明確に規定されているか。
④	テスト手順は明確に規定されているか。
11. 移　植　性	
①	計算機の種類やOSの影響を最小にするように，標準的な高級プログラミング言語で書かれているか。

②	入出力インターフェースは論理的インターフェースになっており，ハードウェアの相違による影響を受けないようになっているか。
③	ファイルの形式は標準的な形式を用いているか。
④	ユーザインターフェースは論理的なインターフェースになっており，ハードウェアやOSの相違による影響を受けないようになっているか。

6.2 プログラム設計，プログラミングのチェックリスト（COBOLを対象とする。）

(1) 性能に関するチェックリスト	
①	実行頻度の高い副プログラムの実行速度が最適化されているか。
②	一時的な事象が共通して使用するエリアは繰り返して使用するようになっているか。
③	定数は実行文ではなく，VALUEで定義しているか。
④	処理方法や出力形式の選択ができるようなオプションを持っているか。
⑤	意味のある定数や繰り返し記述する定数に数値を直接書いていたいか。
(2) 信頼性に関するチェックリスト	
（完全性に関すること。）	
①	本質的に不定な操作（例えば，0での割算，負数の平方根等）に対する保護対策が講じられているか。あるいはそれらの不定条件に対し，適切なコメントが書かれているか。
②	ループや複数の遷移のインデックス・パラメータの幅は使用前にテストされているか。
③	プログラム内には，ライブラリにあるもの以外の全てのサブプログラムが含まれているか。
④	ダミーのサブプログラムがそのまま残されていないか。
⑤	必要に応じて，中間結果等が得られるようになっているか。
⑥	エラー発生の際に，明瞭で手助けになるエラーメッセージを表示するようになっているか。

⑦	プログラム中にまとまった処理の部分がある場合は，コメントが，なされているか。
⑧	プログラムは，メモリを0クリアしてから使用しているか。
⑨	ファイルの定義は入力，出力が明確化されているか。
⑩	入力データは，使用する前に妥当性チェックを行っているか。
⑪	特定なシステムだけに有効なシステムルーチンまたは，ライブラリーにプログラムが依存していないか。
⑫	全ての，あるいは主要な論理パスが正しく実行されているか。
（無矛盾性に関すること。）	
①	1つの変数名が異なる意味を表わすのに使われていないか。
②	1つの実体に対しては1つの名前が定義されているか。
③	ファイル名，レコード名などは，異なるプログラム間で同一の仕様（名前）になっているか。
④	プログラム内では，物理的あるいは数学的な定義の表現は，それぞれ1つになっているか。
⑤	機能的に同じ算術式表現は，常に同様の構造になっているか。
⑥	同一のプログラム出力を識別するのに常に同じラベルが使われているか。
⑦	異なるプログラム出力を識別するのに異なるラベルが使われているか。
⑧	ソースリスト上では，一貫したインデンテーションを行っているか。
⑨	プログラム中のコメントやフォーマットのスタイルは，全てのモジュールで一貫しているか。
⑩	配列の全ての要素は機能的に関係を持ったものか。
⑪	予約語を使用していないか。
（正確性に関すること。）	
①	正確な値を得るためのコーディングができているか。特にコンピュータの特性を配慮しているか。
②	明らかなエラー（予約語のスペリングミス等）がないように，コーディングされているか。
③	全ての配列は正しく宣言されているか。
④	中間結果等が大きな値となるような場合，その対処を行っているか。

（そ　の　他）	
①	プログラムはチェック・ポイント／リスタート機能を持っているか。
②	プログラムは，パラメータが指定されない場合に備えて，既定値を与えているか。
③	致命的でないエラーは，回復が行われるようになっているか。

6.3　保守性に関するチェックリスト

（理解性に関すること。）	
①	変数は，その物理的あるいは，機能的属性をうまく表現したものになっているか。
②	手続名は適切に記述されているか。
③	プログラムの各モジュールは，次の記述を行う見出しブロックを含んでいるか。
	ⓘ　プログラム名，機能
	ⓘⓘ　作成日，作成者
	ⓘⓘⓘ　正確さの要求
	ⓘⓥ　目　　的
	ⓥ　制限や制約
	ⓥⓘ　修正変更の履歴
	ⓥⓘⓘ　入力と出力
	ⓥⓘⓘⓘ　処理方法の説明
	ⓘⓧ　仮定，前提条件
	ⓧ　障害発生時の回復処理手続き
④	個別に認識すべき処理には，その目的が明確になるように適切な説明がついているか。
⑤	判断点と継続する分岐は適切に説明されているか。
⑥	異常終了の条件は全て説明されているか。
⑦	プログラムは，冗長なコード・ブロックではなく，機能別にモジュール化がなされているか。
⑧	変数名のクロス・リファレンスリストがあるか。
⑨	サブルーチンの呼び出し関係を示したマップがあるか。
⑩	算術表現式は解りやすくコーディングされているか。
⑪	1行にはたかだか1つの代入文，あるいは実行文が書かれているか。

⑫	そのプログラムが呼んでいるプログラム及び呼ばれているプログラムが，記述されているか。
⑬	コメント文の形式は全モジュールで統一されているか。
⑭	ある手続きに制御を渡す分岐文が簡単に見つかるようになっているか。
⑮	次元の小さい配列で十分であるにもかかわらず，わざわざ次元の大きい配列を使用していないか。
⑯	算術式の中に繰り返して，同じ表現が現われていないか。
⑰	繰り返し実行しない命令コードは全ループ外に置いてあるか。
⑱	後で変更が予想される部分については，それが変更された場合に影響を受ける部分についての情報が書かれているか。
⑲	止むを得ずプログラムを変更した場合，それが明確に解るようになっているか。
⑳	全てのコードは制御が到達可能か。
㉑	同じコードを何度もくり返していないか。つまり，モジュールとして独立した方がよい部分はないか。
㉒	変数がどこの計算式で値を代入されるかを簡単にさがせるようになっているか。
㉓	ラベル付き共通ブロックには，関係のあるデータだけが含まれているか。
㉔	モジュール間の関連が解るようにコメントされているか。
㉕	コーディング形式は，全モジュールで一貫しているか。

参　考　文　献

1) 「ソフトウェア・デザイン・レビュー」菅野文友監修

2) 「ソフトウェアエンジニアリング」宮本勲著

3) 「ソフトウェア開発のマネジメント」菅野孝男著

昭和 60 年 3 月発行

発行所 財団法人 日本情報処理開発協会
東京都港区芝公園3丁目5番8号
機械振興会館内
Tel (434)8211(代表)

印刷所 株式会社 タケミ印刷
東京都千代田区神田司町2－16